Services Science

LNCS 11236

Claus Pahl
Maja Vukovic
Jianwei Yin
Qi Yu (Eds.)

Service-Oriented Computing

16th International Conference, ICSOC 2018
Hangzhou, China, November 12–15, 2018
Proceedings

Springer

Claus Pahl · Maja Vukovic
Jianwei Yin · Qi Yu (Eds.)

Service-Oriented Computing

16th International Conference, ICSOC 2018
Hangzhou, China, November 12–15, 2018
Proceedings

Springer

Editors
Claus Pahl
Free University of Bozen-Bolzano
Bolzano, Italy

Maja Vukovic
IBM Research
Thomas J. Watson Research Center
Yorktown Heights, NY, USA

Jianwei Yin
Zhejiang University
Hangzhou, China

Qi Yu
Rochester Institute of Technology
Rochester, NY, USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-030-03595-2 ISBN 978-3-030-03596-9 (eBook)
https://doi.org/10.1007/978-3-030-03596-9

Library of Congress Control Number: 2018960420

LNCS Sublibrary: SL2 – Programming and Software Engineering

Preface

The service-oriented computing (SOC) paradigm has established itself as an interdisciplinary methodology that comprehensively addresses modern software development needs. Service orientation combines established best-practice principles of modularization and encapsulation, allowing architectures of large-scale software systems to be designed, analyzed, and deployed successfully. The services paradigm enables the right level of abstraction to achieve loosely coupled, independent, and reusable software artifacts with well-defined interfaces. It has evolved into a technology to embody and express the ubiquity of software that transforms our society from an industrial, production-centric economy into a digital, service-centric economy. The International Conference on Service-Oriented Computing (ICSOC) is the premier international forum for academics, industry researchers, developers, and practitioners on all topics related to services and service-oriented computing. ICSOC fosters cross-community scientific innovation and excellence by gathering experts from various disciplines, such as business process management, distributed systems, computer networks, wireless and mobile computing, cloud computing, IoT and edge computing, cyber-physical systems, networking, scientific workflows, services science, data science, management science, and software engineering. Since the first edition in 2003, the ICSOC conference has grown to become the top international forum in service-oriented computing for academics, industry researchers, developers, and practitioners to report and share the latest research results and innovations.

ICSOC 2018, the 16th event in this series, took place in Hangzhou, Zhejiang, China, during November 12–15, 2018. Following in the ICSOC tradition, it featured three outstanding keynotes given by Schahram Dustdar (Technical University of Vienna, Austria), Liming Zhu (Data61, CSIRO, Australia), and Jeff Zeng (Alibaba Group, China), a research and industry presentations track, a panel session, as well as workshops, tool demonstrations, tutorials, and a PhD track.

This volume contains the proceedings of ICSOC 2018, the 16th International Conference on Service-Oriented Computing. As with previous editions, this year's call for papers generated substantial interest from the community. A total of 273 research and industry paper submissions were received from countries across all continents. Each paper submission was carefully reviewed by at least three members of the Program Committee (PC), followed by discussions moderated by a senior PC member who made a recommendation in the form of a meta-review. The PC consisted of 172 world-class experts in service-oriented computing and related areas (151 PC members and 21 senior PC members). The ICSOC 2018 program featured 40 full papers (acceptance rate of less than 15%) and 23 short papers. The selected papers cover a wide variety of important topics in the area of service-oriented computing, including foundational issues on service discovery and service-systems design, business process modeling and management, economics of service-systems engineering, as well as emerging concerns such as services in the cloud and edge, social networks, IoT, and data analytics.

We would like to express our gratitude to all individuals, institutions, and sponsors that supported ICSOC 2018. This high-quality program would not have been possible without the expertise and dedication of our PC members, and in particular our senior PC members, as well as the untiring efforts of the external reviewers. We are also grateful for the guidance and commitment of the general chairs, Michael Sheng, Zhaohui Wu and Xiaofei Xu, the workshop chairs (Xiao Liu, Michael Mrissa, Liang Zhang), the PhD symposium chairs (Djamal Benslimane, Aditya Ghose, Zhongjie Wang), the demonstration chairs (Antonio Bucchiarone, Wei Zhang, Ying Zou), the panel chairs (Athman Bouguettaya, Jian Yang), the publicity chairs (Bin Cao, Qiang He, Yanjun Shu, Paul de Vrieze), the publication chair (Qi Yu), the finance chair (Bernd J. Krämer), the industry chairs (Min Fu, Lijie Wen), the sponsorship chairs (Dongjin Yu, Jun Shen), the Web chairs (Hai Dong, Adnan Mahmood), the local organization chair (Shuiguang Deng), and the entire ICSOC Steering Committee. All of them helped make ICSOC 2018 a great success. Finally, we would like to thank in particular all authors who submitted papers to the conference, and we congratulate those authors whose papers appear in these proceedings. These papers reflect the quality of the current state of the art in service-oriented computing research and practice. We hope that you find these papers interesting and stimulating.

November 2018

Claus Pahl
Maja Vukovic
Jianwei Yin

Organization

General Chairs

Michael Sheng	Macquarie University, Australia
Zhaohui Wu	Zhejiang University, China
Xiaofei Xu	Harbin Institute of Technology, China

Program Chairs

Claus Pahl	Free University of Bozen-Bolzano, Italy
Maja Vukovic	IBM Research, USA
Jianwei Yin	Zhejiang University, China

Workshop Chairs

Xiao Liu	Deakin University, Australia
Michael Mrissa	University of Pau and Pays de l'Adour, France
Liang Zhang	Fudan University, China

PHD Symposium Chairs

Djamal Benslimane	Lyon University, France
Aditya Ghose	University of Wollongong, Australia
Zhongjie Wang	Harbin Institute of Technology, China

Demonstration Chairs

Antonio Bucchiarone	FBK, Italy
Wei Zhang	Macquarie University, Australia
Ying Zou	Queen's University, Canada

Panel Chairs

Athman Bouguettaya	University of Sydney, Australia
Jian Yang	Macquarie University, Australia

Local Organization Chair

Shuiguang Deng	Zhejiang University, China

Publicity Chairs

Bin Cao	Zhejiang University of Technology, China
Qiang He	Swinburne University of Technology, Australia
Yanjun Shu	Harbin Institute of Technology, China
Paul de Vrieze	Bournemouth University, UK

Publication Chair

Qi Yu	Rochester Institute of Technology, USA

Financial Chair

Bernd J. Krämer	FernUniversität, Germany

Web Chairs

Hai Dong	RMIT University, Australia
Adnan Mahmood	Macquarie University, Australia

Industry Chairs

Min Fu	Alibaba, China
Lijie Wen	Tsinghua University, China

Sponsorship Chairs

Dongjin Yu	Hangzhou Dianzi University, China
Jun Shen	University of Wollongong, Australia

Steering Committee Liaison

Jian Yang	Macquarie University, Australia

Steering Committee

Boualem Benatallah	UNSW, Australia
Fabio Casati	University of Trento, Italy
Bernd J. Krämer	FernUniversität, Germany
Winfried Lamersdorf	University of Hamburg, Germany
Heiko Ludwig	IBM, USA
Mike Papazoglou	Tilburg University, The Netherlands
Jian Yang	Macquarie University, Australia
Liang Zhang	Fudan University, China

Senior Program Committee

Boualem Benatallah	UNSW, Australia
Athman Bouguettaya	The University of Sydney, Australia
Carlos Canal	University of Malaga, Spain
Fabio Casati	University of Trento, Italy
Flavio De Paoli	Università di Milano Bicocca, Italy
Schahram Dustdar	TU Wien, Austria
Xavier Franch	Universitat Politècnica de Catalunya, Spain
Aditya Ghose	University of Wollongong, Australia
Mohand Said Hacid	University of Lyon, France
Cesare Pautasso	University of Lugano, Switzerland
Barbara Pernici	Politecnico di Milano, Italy
Gustavo Rossi	UNLP, Argentina
Antonio Ruiz-Cortés	University of Seville, Spain
Michael Sheng	Macquarie University, Australia
Stefan Tai	TU Berlin, Germany
Zahir Tari	RMIT University, Australia
Samir Tata	IBM Research, USA
Antonio Vallecillo	University of Malaga, Spain
Mathias Weske	HPI/University of Potsdam, Germany
Jian Yang	Macquarie University, Australia
Liang Zhang	Fudan University, China

Program Committee

Marco Aiello	University of Stuttgart, Germany
Pedro Álvarez	University of Zaragoza, Spain
Vasilios Andrikopoulos	University of Groningen, The Netherlands
Alvaro Arenas	Instituto de Empresa Business School, Spain
Ebrahim Bagheri	Ryerson University, Canada
Luciano Baresi	Politecnico di Milano, Italy
Thais Batista	UFRN, Brazil
Moez Ben Haj Hmida	National Engineering School of Tunis, Tunisia
Salima Benbernou	Université Paris Descartes, France
Djamal Benslimane	University of Lyon, France
Sami Bhiri	Telecom SudParis, France
Domenico Bianculli	University of Luxembourg, Luxembourg
Walter Binder	University of Lugano, Switzerland
Juan Boubeta-Puig	University of Cádiz, Spain
Omar Boucelma	Aix-Marseille University, France
Lars Braubach	Hochschule Bremen, Germany
Antonio Brogi	University of Pisa, Italy
Antonio Bucchiarone	Fondazione Bruno Kessler, Italy
Christoph Bussler	Oracle Corporation, USA
Cristina Cabanillas	Vienna University of Economics and Business, Austria

Wing-Kwong Chan	City University of Hong Kong, Hong Kong, SAR China
Francois Charoy	University of Lorraine, France
Faouzi Ben Charrada	University of Tunis El Manar, Tunisia
Sanjay Chaudhary	Ahmedabad University, India
Liang Chen	Sun Yat-Sen University, China
Shiping Chen	CSIRO, Australia
Lawrence Chung	The University of Texas at Dallas, USA
Carlos E. Cuesta	Rey Juan Carlos University, Spain
Edward Curry	Insight Centre, Ireland
Hoa Khanh Dam	University of Wollongong, Australia
Florian Daniel	Politecnico di Milano, Italy
Bruno Defude	Telecom SudParis, France
Shuiguang Deng	Zhejiang University, China
Nirmit Desai	IBM T.J. Watson Research Center, USA
Hai Dong	RMIT University, Australia
Khalil Drira	LAAS Toulouse, France
Yucong Duan	Hainan University, China
Joyce El Haddad	Paris Dauphine University, France
Abdelkarim Erradi	Qatar University, Qatar
Rik Eshuis	Eindhoven University of Technology, The Netherlands
Onyeka Ezenwoye	Augusta University, USA
Noura Faci	Université Lyon 1, CNRS, France
Marcelo Fantinato	University of São Paulo, Brazil
Zhiyong Feng	University of Posts and Telecommunications, China
Pablo Fernandez	University of Seville, Spain
Joao E. Ferreira	University of São Paulo, Brazil
George Feuerlicht	University of Technology Sydney, Australia
Marios-Eleftherios Fokaefs	York University, Canada
Xiang Fu	Hofstra, USA
Walid Gaaloul	Telecom SudParis, France
N. D. Gangadhar	M S Ramaiah University of Applied Sciences, India
G. R. Gangadharan	IDRBT, India
Claude Godart	University of Lorraine, France
Mohamed Graiet	ISIMM, Tunisia
Sven Graupner	HP Labs, USA
Daniela Grigori	Paris Dauphine University, France
Georg Grossmann	University of South Australia, Australia
Nawal Guermouche	Université de Toulouse, France
Amin Haller	Australian National University, Australia
Jun Han	Swinburne University of Technology, Australia
Chihab Hanachi	IRIT Laboratory, Toulouse University, France
Qiang He	Swinburne University of Technology, Australia
Richard Hull	IBM Research, USA
Fuyuki Ishikawa	National Institute of Informatics, Japan
Pooyan Jamshidi	Columbia University, USA

Karthikeyan Ponnalagu	Robert Bosch India, India
Mu Qiao	IBM Almaden Research Center, USA
Manfred Reichert	University of Ulm, Germany
Wolfgang Reisig	Humboldt-Universität zu Berlin, Germany
Stefanie Rinderle-Ma	University of Vienna, Austria
Colette Roland	Université Paris 1 Panthéon Sorbonne, France
Diptikalyan Saha	IBM Research India, India
Iman Saleh	University of Miami, USA
Stefan Schulte	TU Vienna, Austria
Aviv Segev	KAIST, Republic of Korea
Lionel Seinturier	University of Lille, France
Mohamed Sellami	ISEP, France
Jun Shen	University of Wollongong, Australia
Ignacio Silva-Lepe	IBM, USA
Sergey Smirnov	SAP, Germany
George Spanoudakis	City University London, UK
Eleni Stroulia	University of Alberta, Canada
Yehia Taher	University of Versailles-St-Quentin-en-Yvelines, France
Monica Vitali	Politecnico di Milano, Italy
Guiling Wang	North China University of Technology, China
Jianmin Wang	Tsinghua University, China
Jianwu Wang	University of Maryland, USA
Xianzhi Wang	Singapore Management University, Singapore
Yan Wang	Macquarie University, Australia
Zhongjie Wang	Harbin Institute of Technology, China
Ingo Weber	Data61, CSIRO, Australia
Jun Wei	Institute of Software, Chinese Academy of Sciences, China
Matthias Weidlich	Humboldt-Universität zu Berlin, Germany
Lijie Wen	Tsinghua University, China
Jin Xiao	IBM T.J. Watson Research Center, USA
Hanchuan Xu	Harbin Institute of Technology, China
Lina Yao	UNSW, Australia
Sira Yongchareon	Auckland University of Technology, New Zealand
Jian Yu	Auckland University of Technology, New Zealand
Qi Yu	Rochester Institute of Technology, USA
Gianluigi Zavattaro	University of Bologna, Italy
Uwe Zdun	Vienna University of Technology, Austria
Weiliang Zhao	Macquarie University, Australia
Yan Zheng	Aalto University/Xidian University, Finland
Zibin Zheng	Sun Yat-sen University, China
Zhangbing Zhou	China University of Geosciences, China
Olaf Zimmermann	University of Applied Sciences of Eastern Switzerland (HSR FHO), Switzerland
Floriano Zini	University of Bologna, Italy
Christian Zirpins	Karlsruhe University of Applied Sciences, Germany

Andrea Zisman	City University London, UK
Ying Zou	Queen’s University, Canada

Additional Reviewers

Nabil El Ioini	Free University of Bozen-Bolzano, Italy
Ilenia Fronza	Free University of Bozen-Bolzano, Italy
Martina De Sanctis	Fondazione Bruno Kessler, Italy
Mahesh Babu Jayaraman	Ericsson Research, India
Ramamurthy Badrinath	Ericsson Research, India
Rafiqul Haque	Cognitus R&D, UK
Frédéric Camps	University of Toulouse, France
Nicolas Seydoux	University of Toulouse, France
Ahlem Rhayem	University of Sfax, Tunisia
V. S. Yerragudi	Ramaiah University of Applied Sciences, India
Jonathan Sid-Otmane	Sorbonne Université, France
Pierre Sutra	Telecom SudParis, France
Donatella Firmani	Università Roma Tre, Italy
Silvia Bonomi	Sapienza Università di Roma, Italy
Umberto Grandi	University of Toulouse 1 Capitole, France
Qianli Xing	Macquarie University, Australia
Feng Zhu	Macquarie University, Australia
Qi Wang	Macquarie University, Australia
Burkhard Hoppenstedt	Ulm University, Germany
Michael Stach	Ulm University, Germany
Sebastian Steinau	Ulm University, Germany
Kevin Andrews	Ulm University, Germany
Yu Zhao	Queen’s University, Canada
Guoliang Zhao	Queen’s University, Canada
Taher Ahmed Ghaleb	Queen’s University, Canada
Ehsan Noei	Queen’s University, Canada
Weishi Shi	Rochester Institute of Technology, USA
Minxun Zheng	Rochester Institute of Technology, USA
Moayad Alshangiti	Rochester Institute of Technology, USA
Sana Sellami	Aix-Marseille Université, France
Shushu Liu	Aalto University, Finland
Chen Liu	North China University of Technology, China

Abstract of Keynotes

Engineering the Smart Fabric of IoT, Services, and Systems

Schahram Dustdar

Distributed Systems Group, TU Wien, 1040 Vienna, Austria
dustdar@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at

Abstract. This presentation explores the integration of people, software services, and things with their data, into a novel resilient ecosystem, which can be modeled, programmed, and deployed on a large scale in an elastic way. This novel paradigm has major consequences on how we view, build, design, and deploy ultra-large scale distributed systems and establishes a novel foundation for an "architecture of value" driven Smart City.

In particular, this keynote talk addresses three novel paradigms for designing the service-oriented information systems of the future: Elastic Computing, Social Compute Units, and Osmotic Computing. These three paradigms serve as a foundation for future large-scale distributed systems. Furthermore, we will discuss our responsibilities as computer scientists, technologists, and researchers for creating technologies, which benefit society in a positive way, thereby strengthening the new fabric of interconnected people, software services, and things into a novel resilient ecosystem.

Keywords: Internet of Things · Smart Cities · Cloud Computing

Distributed Trust: How Data-Driven Applications, AI and Blockchain is Impacting Service Oriented Computing

Liming Zhu[1,2]

[1] Data61, CSIRO, 13 Garden Street, Eveleigh, NSW, Australia
`liming.zhu@data61.csiro.au`
`http://data61.csiro.au/`
[2] School of Computer Science and Engineering,
University of New South Wales, Kensington, NSW, Australia

Abstract. A key premise of service oriented computing is about trusting the computing behind the service interface. This often relies on trusting the entities running the services and the algorithms behind the services. These assumptions are being challenged. Trust in institutions is at its historical low while our life is increasingly decided by complex data-driven algorithmic learnings that humans do not fully understand. This talk will discuss the emerging of distributed trust and how it may help improve trust in services and deal with black-box algorithmic decision making. The talk will also discuss the role of consumer data rights (a new legislation Australian government is introducing with Data61 being the standard setter) in driving new trustworthy service oriented applications.

Deep Insight of End to End E-Commerce Business Management

Abstract. In the context of E-Commerce eco-system, there are hundreds of millions of consumers, thousands of businesses and shops, and hundreds of delivery people. Alibaba Group, as one of the main E-Commerce providers, cooperates with tens of thousands of software vendors to provide all necessary software services to support the business. With the booming of eco-business, more ecological roles in E-Commerce businesses emerge. For instance, Alibaba Group has expanded its business scale from Taobao Software to several business units, with 10000 plus technical staff. Large E-Commerce businesses such as Alibaba Group need to support a large number of applications and business modules, and cater for hundreds of business requirements and independent changes on a daily basis. As such, there are several changes: (1) we lack a requirement management mechanism from a full business chain perspective, resulting in low cooperation efficiency; (2) The business and the platform are not well separated, which makes it unable to support self-development of the business; (3) The business customization is performed from the system's perspective, instead of the full business chain's perspective, thus bringing much inconvenience to developers who need to understand hundreds of systems during one round of business customization; (4) there is a lack of cross-market, highly reusable and transplantable business assets.

In light of the above-mentioned challenges, we develop a product named Halo. The main ideas of Halo are as below:

(1) Separation among businesses. Each business is assigned with a unique "identity mark", which is used for loading and executing the business customization according to its identity, grouping and routing the business sets based on their identities, monitoring and guaranteeing businesses with intended identities. This ensures that the change of a business will not affect another business.
(2) Scenario-based business capability SDKs. Based on the E-Commerce eco-system of Alibaba Group, we extract several business capabilities, each of which is mapped with a business customization SDK from the full business chain's perspective. All developers need to do is to understand and apply these SDKs, which can be automatically deployed into the targeted physical applications.
(3) Dynamic business deployment using containers. For large-scale distributed applications, the use of containers is able to realize the rapid, incremental and dynamic deployment and publishing of business customization packages.

(4) Clear business definitions. Based on the separation of the management domain and execution domain, we can define and present the visualization of the businesses. Based on the same business execution framework, we can realize the dream of “define once, execute at all places”, and make sure that the business semantics inside the full business chain are consistently understood by all the staff.

Based on those design ideas and decisions, we started developing Halo since 2015. Until now, Halo has been successfully applied in upgrading the systems of several core business platforms, including the trading platform, the commodity platform, the marketing platform and the fund platform. Compared to the old mechanism, the use of Halo significantly reduces the threshold required for developing the business customization, and increases development efficiency to a large extent. Meanwhile, with the scenario-based business capability SDKs, it is easier to achieve high reusability of business logics (e.g. pre-sale, e-certificate, virtual commodities, payment on arrival of goods, etc.) in a cross-market and cross-region manner.

Short Bio

Jeff Zeng is a senior staff engineer at Alibaba Group. He is now leading the core trade platform which is the foundation of Alibaba’s key business. Jeff is also in charge of Alibaba financial platform and business innovation by applying block chain technology. Prior to this, he was a Global development Director of Small and Medium Enterprise at SAP. He has been working on global enterprise product development in the Internet industry for about 10 years. Jeff received his master degree from East China Normal University, and bachelor degree from Shanghai University.

Contents

Microservices

Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments

Jinjin Lin, Pengfei Chen(✉), and Zibin Zheng

School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
linjj23@mail2.sysu.edu.cn
{chenpf7,zhzibin}@mail.sysu.edu.cn

Abstract. Driven by the emerging business models (e.g., digital sales) and IT technologies (e.g., DevOps and Cloud computing), the architecture of software is shifting from monolithic to microservice rapidly. Benefit from microservice, software development, and delivery processes are accelerated significantly. However, along with many micro services running in the dynamic cloud environment with complex interactions, identifying and locating the abnormal services are extraordinarily difficult. This paper presents a novel system named "Microscope" to identify and locate the abnormal services with a ranked list of possible root causes in Micro-service environments. Without instrumenting the source code of micro services, Microscope can efficiently construct a service causal graph and infer the causes of performance problems in real time. Experimental evaluations in a micro-service benchmark environment show that Microscope achieves a good diagnosis result, i.e., 88% in precision and 80% in recall, which is higher than several state-of-the-art methods. Meanwhile, it has a good scalability to adapt to large-scale micro-service systems.

Keywords: Microservice · Kubernetes · Root cause analytics
Cloud computing

1 Introduction

Nowadays, driven by the emerging business models (e.g., digital sales) and IT technologies (e.g., DevOps and Cloud computing), the architecture of software is shifting from monolithic to microservice [20] architecture rapidly. With microservice architecture, an application is decoupled into many loosely distributed fine-grained services with complex interactions. Usually, these services are connected by some light-weight network protocols such as REST and RPC protocols. But each of them has simple and independent functions following the SRP (Single Responsibility Principle) [20]. The microservice architecture has enabled software systems with new properties such as strong scalability, agile development, fast delivery, and so on. Even though, performance problems are not uncommon

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 3–20, 2018.
https://doi.org/10.1007/978-3-030-03596-9_1

in microservice systems due to external (e.g., configuration changes) and internal (e.g., software bugs) impairments [7], which brings significant impacts on enterprise revenues. According to [14], Amazon experiences 1% decrease in sales for additional 100 ms delay in response time per request while Google reports a 20% drop in traffic due to 500 ms delay in response time.

To keep microservices running continuously and reliably, it is necessary to detect undesirable performance problems and pinpoint potential root causes. However, it is notoriously difficult to achieve that in microservice environments due to the following challenges:

- **Complex network dependencies**. With a microservice architecture, an application is decoupled into many fine-grained components with an extraordinarily complex network topology. Moreover, to connect different micro services wrapped in a container, an overlay network such as *flannel* is always adopted, which further increases the complexity of performance diagnosis.
- **Continuous integration and delivery.** A microservice system is evolving all the time with continuous integration and delivery technologies. According to a DevOps report from Puppet [1], an enterprise may have 1600 updates a year. That means the anomaly detection and root cause diagnosis procedure should adapt to these changes in order to achieve better results.
- **Dynamic run-time environment.** A microservice system often runs in a containerized environment where the states of containers change frequently. The highly dynamic environment exacerbates the difficulty of performance diagnosis.
- **A large volume of monitoring metrics.** Since so many services co-exist in micro-service systems, the volume of monitoring metrics (e.g., response time) of these services is very large. The paper [23] states that Netflix, Uber and OpenStack has 2,000,000 metrics, 500,000,000 metrics, and 17608 metrics respectively to monitor. How to pinpoint the root causes = from these data is a challenging problem.

Extensive studies have been done to resolve performance diagnosis problems in distributed systems. However, they either (e.g., X-Trace [12], Roots [15]) require to modify the source code of applications or platforms to obtain the service dependencies or cannot adapt to the dynamics of microservice environments (e.g., CauseInfer [7]). In order to address the aforementioned challenges and shortages of previous work, we propose Microscope, a novel system to identify performance problems and infer root causes with causal graphs towards microservice systems. It basically comprises three procedures, namely data collection, causality graph building, and cause inference. Once an anomaly is detected in front-end services, a causality graph which denotes the anomaly propagation paths is constructed automatically. Then the cause inference procedure is triggered to pinpoint root cause with the causal graph. The causality graph is built with no need domain knowledge and instrumenting the application. Microscope leverages a conditioned graph traversing algorithm to locate the root causes rather than a brute-force search, which significantly reduces the volume of performance metrics to process. Moreover, Microscope works in real-time mode to adapt to the dynamics of microservice environments. We developed a prototype system and validated its effectiveness in a microservices benchmark, namely Sock-shop [2], managed by Kubernetes. The results show Microscope achieves an average 88% precision for root cause identification outperforming several state-of-the-art methods. Meanwhile, it can be applied in a large distributed system without a significant accuracy lose.

The contributions made by this paper are threefold:

- We propose a novel service dependency discovery method through capturing and parsing the network-related system calls, which works automatically to capture the real service instances dependency in real time.
- We provide a **parallelized** service causality graph building method based on the service dependency and service impact graph co-located in a single machine. We can precisely pinpoint the root causes at service instance level with this causality graph.
- We design and implement a prototype of Microscope to infer the root causes of performance problems without domain knowledge and application instrumentation, and achieve a high precision and recall with a low cost.

The rest of this paper is organized as follows. Section 2 presents the system overview and formulation. Section 3 elaborates the details of Microscope. In Sect. 4, we will evaluate Microscope in the controlled environment. And in Sect. 5 we will compare our work with previous work. We discuss the advantages of Microscope and Sect. 6 concludes this paper.

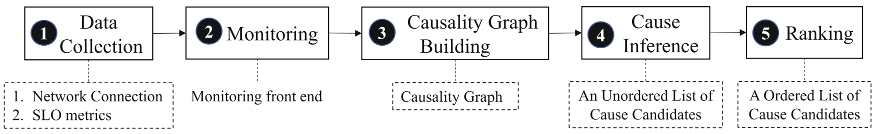


Fig. 1. Workflow of Microscope

2 System Overview and Formulation

Figure 1 shows an overview of Microscope. For data collection, Microscope mainly collects two types of data: network connection information between two service instances and SLO (Service Level Objective) metrics of each service instance. To diagnose system anomalies, Microscope continually monitors the SLO metrics of the front end within a sliding time window. When an SLO violation is detected, the root cause analysis is triggered. In service causality graph building phase, Microscope uses the network connection information and SLO metrics to build a causality graph. Then the cause inference engine starts from the front end and traverses the entire causality graph along with the directed edges. After that, Microscope gets a list of possible root cause candidates. Finally, Microscope calculates a score for each candidate and ranks the candidates with the score.

3 System Design

3.1 Data Collection

In the data collection part, Microscope mainly collects two types of data, namely network connection information between two service instances and SLO metrics of each service instance.

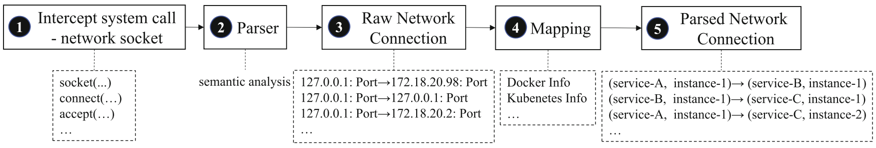

1 Intercept system call - network socket
2 Parser
3 Raw Network Connection
4 Mapping
5 Parsed Network Connection
socket(...)
connect(…)
accept(…)
…
semantic analysis
127.0.0.1: Port→172.18.20.98: Port
127.0.0.1: Port→127.0.0.1: Port
127.0.0.1: Port→172.18.20.2: Port
…
Docker Info
Kubenetes Info
…
(service-A, instance-1)→ (service-B, instance-1)
(service-B, instance-1)→ (service-C, instance-1)
(service-A, instance-1)→ (service-C, instance-2)
…

Fig. 2. Workflow of capturing network connection information

Network Connection Information. This type of data is used for causality graph building. Network connection is an important information to represent the real service dependencies. However, most of time, the transmission between client and server is bidirectional. Hence, we will get an opposite connection direction when we observe in different hosts. For example, when we observe in host (192.168.1.2) which sends a request network package, we get a connection (192.168.1.2)→(192.168.1.3). But in host (192.168.1.3) which sends a response network packet, we will get (192.168.1.3)→(192.168.1.2).

In order to address this issue, we introduce a novel method to capture the network connection information by monitoring and intercepting system calls related to network socket, such as *socket()*, *connect()*, *send()* and *recv()*. A network socket is an internal endpoint for sending or receiving data within a computer network. Each socket function relates to a socket variable. When we intercept a socket system call, it parses the socket variable and parameters of this function to get a client IP and a server IP according to the semantic meaning of function name. For instance, if a *connect()* function is intercepted, the local address of socket will be parsed as client IP and the peer address will be parsed as server IP. Another situation is that if a *accept()* function is intercepted, the local address will be parsed as server IP and the peer address will be parsed as server IP. So we can get the direction of network connection and the real dependency between two services.

The complete workflow of capturing network connection information shows in Fig. 2. After the socket parsing, we get the raw network connection whose ends are host IPs rather than service IPs. To know the corresponding service of a host IP, we extract some information from infrastructures such as Docker and Kubernetes. Combining with this information, we map the raw network connection(IP:Port) to parsed network connection ((service name, service instance)). Finally, we get a bunch of records which describe the network connection information from a service instance to another service instance.

SLO (Service Level Objective) Metrics. This type of data is used for detecting whether a service instance is abnormal and ranking root cause candidates. According to our observations, most cloud-native applications that internally generates performance metrics such as throughput for monitoring and maintenance. If these data are not internally available, we can also crawl the service logs to that end. For example, the spring boot framework provides a plug-in of service log for monitoring. Therefore, we can easily get SLO metrics from cloud-native applications in microservice environments. In this paper, we will use a unified SLO metric, namely service request latency which is the service calling time, which exposed by the services themselves. In the future work, we will explore more SLO metrics in microscope for improving the effectiveness. Although it is simple, it works well in Microscope.

3.2 Service Causality Graph Building

In this section, we describe the details of the service causality graph building. The definition of the causality is that given two variables X and Y, we say X is a cause of Y if the changes of X can affect the distribution of Y but not vice versa, denoted by X→Y. In other words, X is a parent of Y, denoted by $X \in pa(Y)$. In a collective variable, if all the parents of Y have been determined, the distribution of Y will be determined and not affected by other variables. In this paper, X and Y represent the SLO metrics of each microservice. Note that in the causality graph, it is not allowed two variables cause each other. So the causality graph can be represented as a DAG (Directed Acyclic Graph).

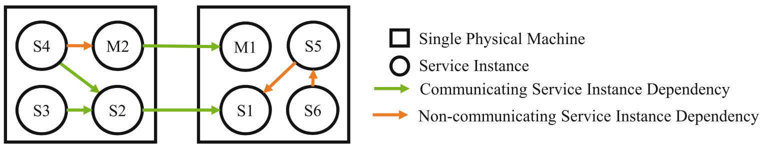

S4
M2
S3
S2
M1
S5
S1
S6
Single Physical Machine
Service Instance
Communicating Service Instance Dependency
Non-communicating Service Instance Dependency

Fig. 3. An example of service causality graph, where S1-S6 and M1-M2 represent unrelated applications with several service instances.

Microscope constructs a causality graph based on the communicating service instance dependencies and non-communicating service instance dependencies. Each node in the causality graph represents a service instance, and the directed edge between two nodes represents a direct “cause-effect” relation between two service instances. Figure 3 shows an example of causality graph generated by Microscope.

Communicating Service Dependency. The first type of dependency represents a dependency relation between two communicating service instances via network. In previous studies, [7,23], they also use the network connection information to construct causality graph. But the direction of dependency between two service instances are determined by some statistical methods such as Granger Causality [13] and PC-algorithm [16]. So their results strongly depend on the quality of data. Compared with these work, Microscope captures the directed connection information to represent the communicating service instance dependency relation without any statistical error.

To construct communicating service dependency, Microscope uses the *parsed network connection* data directly which already contains two service instances and the direction of dependency in each record. For example, (service-A, instance-1)→(service-B, instance-1) represents (service-A, instance-1) is a cause of (service-B, instance-1). Microscope uses the data collected in the last 10 min, so the causality graph can be updated dynamically, which can exclude some inactive service instances and improve the precision of root cause inference. However, Microscope can also use data for longer periods of time and save static communicating service dependencies, which reduces the cost of building this type of dependencies repeatedly.

Non-communicating Service Dependency. Due to local resource sharing, the service may interfere with other services running in the same node, which is called "non-communication service dependency". For example, if a service instance occupies all the CPU resource of a physical machine, the response time of the other service instances in the same node, especially for computation-intensive ones, is likely affected. Therefore, the SLO metrics changes of other co-located services are also responsible for changes in the SLO metrics of the current concerned services. We construct such relations by a statistical approach.

In one physical machine, the anomalies of Service **A** and Service **B** may be caused by a common anomaly of Service **C**. To model these relations, we adopt causal statistics [22] rather than the pair-wise correlation. Considering the large volume of SLO metrics in microservice environments and the light-weight requirement, we design our algorithm on the basis of PC-algorithm [16], which is more computationally efficient than Bayesian network approaches [11]. To obtain such a DAG, we first construct a skeleton of the DAG, namely an undirected graph. Then we orientate the skeleton with D-separation [16] rules. A causal Markov condition [22] is used to produce a set of independent relations amongst more than two variables and to construct the skeleton of a causality graph. It is defined as

Definition 1. *Given a DAG, $G = (V, E)$, for every $v \in V$, v is independent of the non-descendant of v given its direct $pa(v)$.*

In this paper, we leverage a conditional cross-entropy based metric G^2 [22] to qualitatively test whether X is dependent on Y given $\mathbf{Z}$, where X, Y and $\mathbf{Z}$ are disjoint set of variables in V, X and Y are single variables, but $\mathbf{Z}$ can be a set of variables. We choose G^2 instead of other methods like Gaussian independence test [16] as it does not need any assumption on the distribution of each variable. G^2 is defined as

$$\begin{aligned} G^2 &= 2mCE(X,Y|\mathbf{Z}) \\ &= \sum_z P(z) \sum_x \sum_y P(x,y|z) log(\frac{P(x,y|z)}{p(x|z) \cdot p(y|z)}), \end{aligned} \tag{1}$$

where m is the sample size, $CE(X,Y|\mathbf{Z})$ is the conditional cross entropy of X and Y given $\mathbf{Z}$. As stated in [22], under independence hypothesis, the metric G^2 follows a χ^2 distribution with a degree of freedom equals to

$$(N_X - 1)(N_Y - 1) \prod_{Z' \in \mathbf{Z}} N_{Z'}, \tag{2}$$

where N_X, N_Y and $N_{Z'}$ represent the number of values of variable X, Y and $Z^{'}$ respectively. Hence, via a χ^2 test, we can decide whether the independence hypothesis is accepted. If the *p-value* exceeds the significance level ξ, namely *p-value* $> \xi$ ($\xi = 0.02$ in this paper), the independence hypothesis is accepted otherwise rejected. If X is independent of Y given $\mathbf{Z}$, then $I(X,Y|\mathbf{Z}) = 1$.

PC-algorithm begins with a completely connected undirected graph, then facilitates G^2 to capture all the independence relationships within all variables in pair-wise manner. The following work is to determine the causal directions using D-separation [16] rules, which is demonstrated in our previous work [7] in detail. Due to the limited space of this paper, we cannot show the details of this PC-algorithm. Please refer to the paper [16] for the details of PC-algorithm. On the basis of PC-algorithm, we construct a parallelized algorithm to construct non-communicating service dependencies in micro-service environments more efficiently. When the cardinality of $\mathbf{Z}$ equals 0, namely $|\mathbf{Z}| = 0$, each pair of X and Y is completely independent. Therefore, we leverage "MapReduce" [10] approach to test their independence relations in parallel. Then, we get another undirected graph with significantly reduced edges. If $|\mathbf{Z}| > 0$, the independence tests of X and Y given $\mathbf{Z}$ are independent any more as they share the intermediate results. Under such a condition, we implement a multi-core parallel algorithm in one node. That means we conduct one independence test per core. After the parallelization of PC-algorithm, Microscope can construct the causal relations in real time.

Once an anomaly is detected in the front-end services, the construction procedure of non-communicating service dependency is triggered. We first group all the micro services based on physical machines. Let $(S_1, S_2, \cdots, S_n)$ denote n services, $N_1, N_2, \cdots, N_m$ denote m machines, (S_i, N_j) denote service i locates on machine N_j, so all services are separated into m groups. For each group, we leverage parallelized PC-algorithm to construct service dependency relations. The inputs are time series of SLO metrics of micro services. In this paper, we leverage the mean response time as the SLO metric. Other SLO metrics are also adoptable. 200 data points starting from the abnormal moment are adopted to construct the service dependency graph. The reason why we choose 200 is stated in the sensibility analysis of Sect. 4. One point we observed from our experiments is that some causal directions calculated by PC-algorithm are not consistent with the service dependencies obtained by network analysis. In this scenario, we trust

Algorithm 1. The *parallelized* PC-algorithm

```
Input: The significance level $\xi$ used to test the conditional independence; the response time metrics
    of micro services, $\mathbf{R}=\{R_1, R_2, R_3, \cdots, R_n\}$, set the maximal cardinality of $\mathbf{Z}$ as 3; the number
    of CPU cores $c$;
Output: non-communicating service dependencies DAG, $G$
 1: $/ ** $ Construct the skeleton of $G ** /$
 2: Form the complete undirected graph $G^u$ based on $\mathbf{R}$
 3: i=-1
 4: repeat
 5:     i=i+1
 6:     if $i == 0$ then
 7:         $/ **$ Map process $** /$
 8:         Select one pair of $(X, Y)$ from all the combinations
 9:         if $I(X, Y) == 1$ then
10:             Record $(X, Y)$
11:         end if
12:         $/ **$ Reduce process $** /$
13:         Collect all pairs of $(X, Y)$ calculated by each Map process
14:         Remove the edges $X - Y$ recorded by $(X, Y)$ from $G^u$
15:         Update $G^u$
16:         $/ **$ Calculate service dependencies by multiple process on one machine $** /$
17:     else
18:         for each $c_i \in (1, 2, \cdots, c)$ do
19:             Fork one process on one machine to conduct in independence tests
20:             for each $X \in \mathbf{X}$ do
21:                 for each $Y \in adj(G^u, X)$ do
22:                     $/** adj(G^u, X)$ represents the set of metrics which are adjacent to $X$ in $G^u$. $**/$
23:                     repeat
24:                         Choose $\mathbf{Z} \subseteq adj(G^u, X) \setminus \{Y\}$ with $|\mathbf{Z}| = i$
25:                         if $I(X, Y|\mathbf{Z}) == 1$ then
26:                             Remove $X - Y$ from $G^u$
27:                             Update $G^u$
28:                             Make the separate set $\mathbf{S}(X, Y) = \mathbf{Z}$
29:                         end if
30:                     until edge $X - Y$ is removed or all $\mathbf{Z}$ with $|\mathbf{Z}| = i$ have been chosen.
31:                 end for
32:             end for
33:         end for
34:     end if
35: until $|adj(G^u, X)| \leq i, \forall X$ or i == 3
36: The skeleton,$G^s = G^u$
37: $/ **$ Orient the directions in $G^s$ with D-Separation rules$* * /$
38: for all pairs of nonadjacent variables $X, Y$ with common neighbor $Z$ do
39:     if $Z \notin \mathbf{S}(X, Y)$ then
40:         Replace $X - Z - Y$ in $G^s$ with $X \rightarrow Z \leftarrow Y$
41:     end if
42: end for
43: Orient $Y - Z$ as $Y \rightarrow Z$ whenever there is an arrow $X \rightarrow Y$
44: Orient $X - Y$ as $X \rightarrow Y$ whenever there is chain $X \rightarrow Z \rightarrow Y$
45: Orient $X - Y$ as $X \rightarrow Y$ whenever there are two chains $X - Z \rightarrow Y$ and $X - L \rightarrow Y$
46: Finally, output $G$
```

the result obtained by the latter as it is the ground truth. Therefore, to avoid this scenario, we preset the connections and directions of edges that can be obtained by network analysis in the undirected graph.

For the sake of clarity, we show the pseudo code of our algorithm in Algorithm 1. The computational complexity of Algorithm 1 is dominated by the DAG skeleton construction procedure. The worst case is bounded by $O(nmax\{p^q, p^2\})$ [17], where n is the data length of each metric, p is the number of metrics, q is maximal size of the adjacent sets, i.e., the cardinality of $\mathbf{Z}$, $|\mathbf{Z}|$. When q is large, the complexity increases exponentially. However, from the real data, we observe that q always stays at a low level $q < 5$. Hence this complexity is affordable. We set $q = 3$ in this paper. The constructed non-communicating service dependencies will be merged with communicating service dependencies to form the final service causal graphs.

3.3 Cause Inference

We summarize the process of cause inference in Algorithm 2. Microscope continually monitors the SLO metrics of the front end. When an SLO metric is detected as abnormal, the cause inference is triggered. Then the cause inference engine starts from this abnormal node in the causality graph and traverses the causality graph along the opposite direction of edges, which represents the dependency between two service instances. When a node is abnormal, the cause inference engine will check its neighbors. If all the neighbors are normal, the current node will be added to the set of root cause candidates and the engine stops traversing its children. If there exist one or more abnormal children, the

Algorithm 2. The *cause inference* algorithm

```
Input: An original abnormal service instance, rootNode; A causality graph DAG, G;
Output: A ordered list of root cause candidates
 1: // Find root cause candidates
 2: $stack \leftarrow Stack(); candidates \leftarrow List()$
 3: $stack.push(rootNode)$
 4: while $stack$ is not empty do
 5:    $node \leftarrow stack.pop()$
 6:    // $adj(G, X)$ represents the neighbors which are adjacent to $X$ in $G$.
 7:    if $adj(G, node)$ is empty then
 8:       $candidates.append(node)$
 9:       continue
10:    end if
11:    $children \leftarrow List()$
12:    for each $neighbor \in adj(G, node)$ do
13:       if $neighbor$ is abnormal then
14:          $children.append(neighbor)$
15:          $stack.push(neighbor)$
16:       end if
17:    end for
18:    if $children$ is empty then
19:       $candidates.push(node)$
20:    end if
21: end while
22: // Scoring for each candidates
23: $candidatesScore \leftarrow Dict()$
24: for each $candidate \in candidates$ do
25:    $candidatesScore[candidate] \leftarrow scoring(rootNode, candidate)$
26: end for
27: return keys of $candidatesScore$ sorted by value
```

cause inference engine will continue to traverse these abnormal children. When the traversal is finished, the engine gets a set of root cause candidates. Then the engine calculates a ranking score for each root cause candidates. Finally, the engine gets an ordered list of root cause with ranking score and the top one in the list is considered to be the real root cause. In this paper, we use *three-sigma rule of thumb* to detect if a service instance is abnormal and use the correlation between the front end and the abnormal service instances as the ranking score.

To detect anomalies, we use a *three-sigma rule of thumb*. The so-called three sigma rule of thumb expresses a conventional heuristic, that is, almost all values are considered to be within the three standard deviations of the mean. Therefore, it is empirically useful to treat the possibility of 99.7% as near certainty. In mathematical notation, this fact can be expressed as $Pr(\mu - 3\sigma < x < \mu + 3\sigma) \approx 0.9973$. If a value of SLO metric is not within the three-sigma interval of the last 10 min, we think this service instance is abnormal. Although it is simple, it works in this paper and can adapt to system changes in real time.

To sort the root cause candidates and get the most possible real root cause, we calculate the pearson correlation coefficient of SLO metrics between the front end and each candidate as the ranking score. According to our observations, if two service instances have a strong dependency relationship, the curves of service request latency of them are very similar. With this method, Microscope has the ability to diagnose real root causes even when several system faults happen at the same time.

4 Experimental Evaluation

Experiment Settings. Microscope is evaluated in a self-constructed distributed system. The controlled system contains four client physical servers that host the benchmark. Each physical server machine has a 12-core 2.40GHz CPU, 64GB of memory and runs with Ubuntu 16.04 OS. We evaluate Microscope in Kubernetes platform. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications, which is one of best platform for developing and running micro services.

Data Collection. To capture the network connection information, we develop several tools from scratch. The captured network connection information is recorded to a local log file. For forwarding and centralize log files, we use Filebeat which offers a lightweight way to forward and centralize logs and files. Meanwhile, we use elasticsearch to save all the network connection information harvested by Filebeat from each physical nodes. To collect service request latency metrics, we use Prometheus, an open-source systems monitoring and alerting toolkit, to monitor the services instances. The sample interval in service request latency metrics is 1 s.

Benchmark. Sock-shop [2] is a microservices demo that simulates the sale of socks of an e-commerce website which is a widely used micro-service benchmark designed to help demonstrate and test microservices and cloud-native technologies. It provides some key properties (e.g. Polymorphism) that a micro-service system should have. It contains 13 services in the form of microservices. In each services instance, we configure the CPU resource limited to 1 GHz and the memory resources limited to 1GB. The replicas of service instances are set to 1–3. The total service instances are 36. Furthermore, sock-shop contains a load generator, which defines user behavior, to simulate the query per second(QPS) on a website with simultaneous users, so we adopt this load generator to generate the workload and configure the load to keep the QPS about 5000.

Fault Injection. Our work focuses on locating the root cause service instances. The service instance in Kubernetes is a Pod which contains one or more containers. To mimic the real performance problems, we inject faults to containers in Pods. We inject the following faults: (1) CPU exhausting: we use stress [3], a deliberately simple workload generator for POSIX systems, to exhaust CPU resources in injected containers; (2) NetworkJam: we use “tc”, a traffic control tool in Linux, to delay the network packets; (3) ContainerPause: for simulating the status of hangup of a service instance, we pause the container with“PAUSE” command of Docker. We do not kill containers because Kubernetes will recreate a new replicated Pod immediately. For evaluating the effectiveness of Mircoscope, each fault mentioned above will be injected in each service instances more than 5 times and last 1 min. The total number of fault injections is 240.

Evaluation Metric. We use the following metrics to evaluate the effectiveness.

- *Precision at top K*(PR@K) indicates the probability that the root cause appears in the top K of ranking list if the cause inference is triggered. It is important to capture the root cause at a small value of K, thereby resulting in lesser number of service instances to investigate. Here we use K=1,2,3.
- *Recall at top K*(Recall@K) is the fraction of real cause that has been retrieved at top K of the ranking list over the total amount of fault injections. Here we use K=1,2,3.

4.1 Effectiveness Evaluation

Microscope strongly relies on the causality graph, Fig. 4 illustrates a causality graph obtained by Microscope in about 10 min when running a load generator. Different colors represent different applications. Via comparison with the

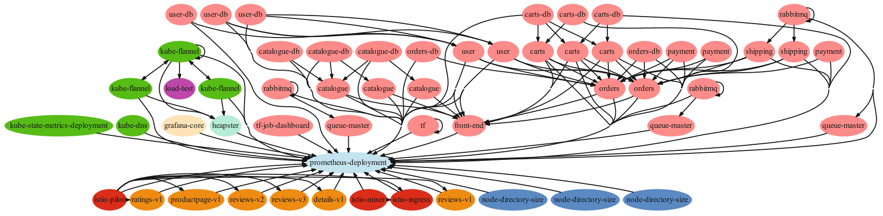


Fig. 4. The sample causality graph generated by Mircoscope in our system

ground truth, we find that all the relations shown in Fig. 4 are indeed the service call relations without any exception, which demonstrates that Microscope can build a reasonable causality graph in real time without domain knowledge and instrumenting the application.

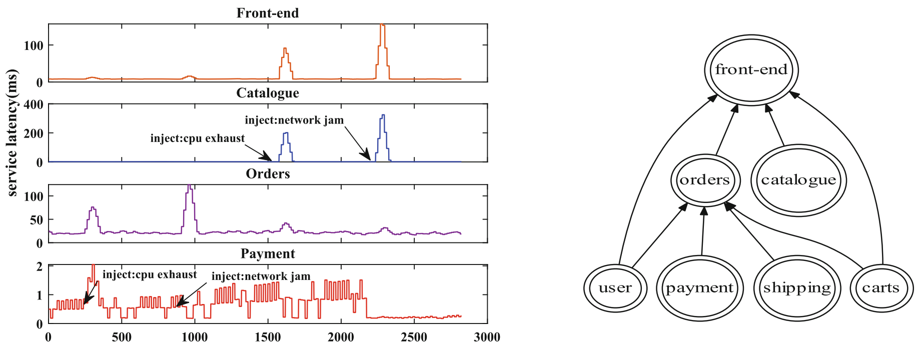


Fig. 5. A view of service latency and causality graph of Sock-Shop

Figure 5 shows the curves of service request latency of four service instances when 4 faults are injected respectively and the dependent relations between service instances of sock-shop. To simplify the description, we keep only one replica of each service. The curve of front-end and catalogue services in Fig. 5 show that if a service is abnormal, it does affect other services which it depends on. But it is not the case for the injected faults in payment service. The orders which the payment depends on is strongly affected. However, it has a very subtle effect on the front end. This is because Kubernetes has a load balancing mechanism. So the cause inference may not be triggered and the precision and recall are low on these types of services.

Figure 6 demonstrates the results of PR@1 and recall on several services of Sock-shop. From Fig. 6, we observe that most of the PR@1 fall in the range 70%–100% in different services and faults, except the shipping and payment service. One of the exceptions is we lack the results of shipping and payment

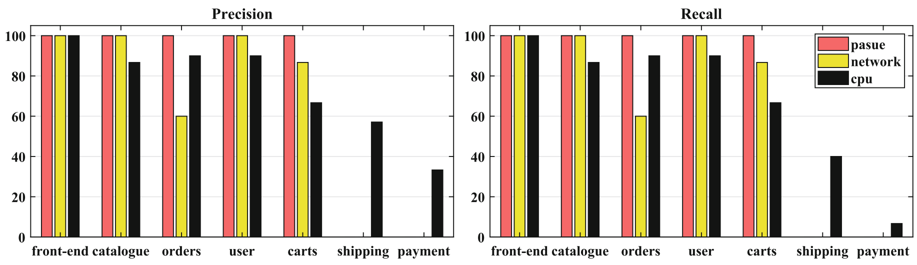


Fig. 6. The results of PR@1 and Recall@1

with respect to network and container pause. This is because (i) the service latency is collected by the process in the Docker container, (ii) these services don't request any other services, so the service latency is returned immediately from the process without passing through the container network, (iii) we inject the network delay to block the network in container rather than the process. So the fault injection doesn't work on these services. The other one of exceptions is the results of CPU exhausting on shipping and payment. This is because (i) the cause inference may not be triggered which mentioned above, (ii) both shipping and payment are not computation-intensive, which the usage of CPU in these service instances is the only 5 mHz. So even though we use *stress* tool to exhaust the resources of CPU, these service instances still have a good response time. Significantly, we get 100% at the fault of container pause, because it causes the service latency missing and it's a strong feature to diagnose.

Table 1. Performance

	Catalogue	Front-end	Carts	Orders	User	Shipping	Payment
CPU exhausting							
PR@1	86.7	100	66.7	90.0	90.0	57.1	33.3
PR@2	93.3	100	66.7	90.0	100	57.1	33.3
Recall@1	86.7	100	66.7	90.0	90.0	40.0	6.7
Recall@2	93.3	100	66.7	90.0	100	40.0	6.7
Network jam							
PR@1	100	100	86.7	60.0	100	-	-
PR@2	100	100	86.7	60.0	100	-	-
Recall@1	100	100	86.7	60.0	100	-	-
Recall@2	100	100	86.7	60.0	100	-	-
Container pause							
PR@1	100	100	100	100	100	-	-
PR@2	100	100	100	100	100	-	-
Recall@1	100	100	100	100	100	-	-
Recall@2	100	100	100	100	100	-	-

Table 1 demonstrate the performance of Microscope in Sock-shop on different service and fault. It shows that Microscope can achieve an average 80%–95% precision and recall for those computation-intensive and network-sensitive services.

4.2 Comparisons

To validate the effectiveness of Microscope thoroughly, we compare it with several state-of-the-art methods including TAN [9], NetMedic [18], Sieve [23], Roots

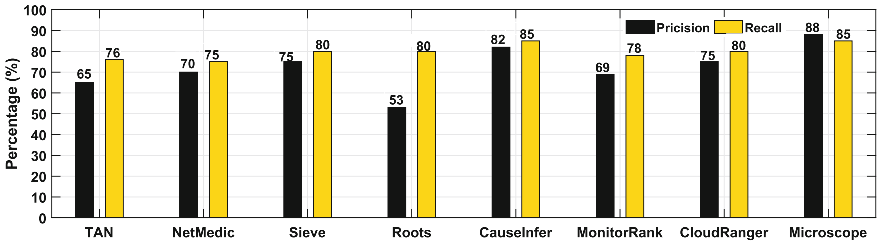


Fig. 7. The comparison results in PR@1 and Recall@1

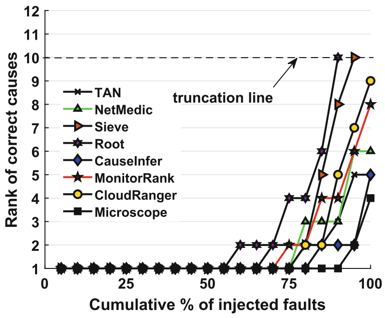


Fig. 8. Rank comparison result

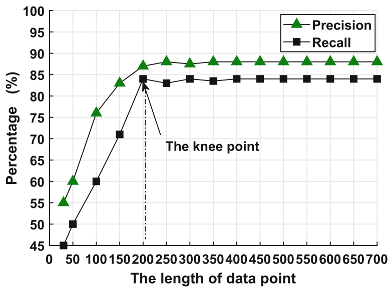


Fig. 9. The Diagnosis Result VS data length

[15], CauseInfer [7], MonitorRank [19], and CloudRanger [24]. To compare with TAN, we replace our service dependency construction approach with Tree Augmented Bayesian Network approach; To compare with NetMedic, we leverage NetMedic's state correlation approach to estimate service dependencies; To compare with Sieve, we adopt *sysdig* to obtain the static service call graph, a bi-directed graph then leverage Granger Causality to obtain the dynamic service dependencies with response time metrics; To compare with Roots, we implement the four root cause identification approaches mentioned in Roots. But since Microscope cannot track each request, we leverage the aggregated response time of requests instead of the response time of one single request to identify root causes; To compare with CauseInfer, we capture the network packets and leverage lag correlation to find service dependencies; To compare MonitorRank, we use a random walk approach to find the root causes which has been implemented in our previous work [24]; To compare with CloudRanger, we leverage PC-algorithm to construct service dependencies with response time metrics of micro services. Figure 7. shows the comparison result in PR@1 and Recall@1. From this figure, we observe that Microscope achieves a significantly better result, 3% higher than CauseInfer, 13% higher than CloudRanger, 35% higher than Roots, in PR@1. Roots identifies the service which contributes the most variance of the abnormal service as the root cause. However, in our experiments, we observe that Roots always finds the first upstream service as the culprit rather

than real root causes. That is why its result is not very good. Similarly, MonitorRank also puts the first upstream service in the first rank. Compared with CauseInfer, CloudRanger, and Sieve, Microscope constructs the service dependencies by analyzing the network connection events rather than calculating the statistical correlations, which is closer to the ground truth. From the perspective of the rank of correct causes, Fig. 8 shows the comparison result between different approaches when $rank <= 10$. From Fig. 8, we observe that Microscope can find 88% of injected faults at Rank 1, which outperforms other approaches.

4.3 Discussion

Overhead. Table 2 shows the overhead of Microscope. Data Collection module takes about 8% CPU utilization as we collect the network connection information and service latency. Overall, Microscope is a light-weight tool for monitoring and pinpointing the abnormal service instances.

Table 2. The overhead of Microscope

System module	CPU Cost
Data collection(Network connection)	$8\% \pm 2\%$ CPU utilization(Single cpu core)
Data collection(SLO metrics)	$4\% \pm 1\%$ CPU utilization(Single cpu core)
Causality graph building	10 s(Single physical node)
Cause inference	2 s(Single physical node)

Sensibility. To evaluate the sensibility to the data length in constructing non-communicating service instances dependency, we conduct several experiments. Figure 9 illustrates the changes in the data length increasing. From this figure, we observe that Precision and Recall stay at a low level when the length of data is lower than 200. Because the service causal graphs calculated by PC-algorithm are not precise enough. However, after "the knee point", the service causal graphs keep constant and the diagnosis results do not change anymore.

Scalability. Microscope is easy to scale up whether we add new service instances or machines in a large distributed system. In order to test the scalability of the system, we deployed more replicas of services instances with Sock-shop. From Fig. 10, we can see that Microscope has only 7% accuracy degradation from 36 service instances to 120 service instances, showing good scalability.

Indeed, the micro-service system provides some fault-tolerance mechanisms such as circuit-breaker, which aim to avoid cascading errors. Also some real microservice infrastructures in kubernetes provide features such as scalability, restart or load balance policies. Considering these, Microscope will resolve these problem in the future work.

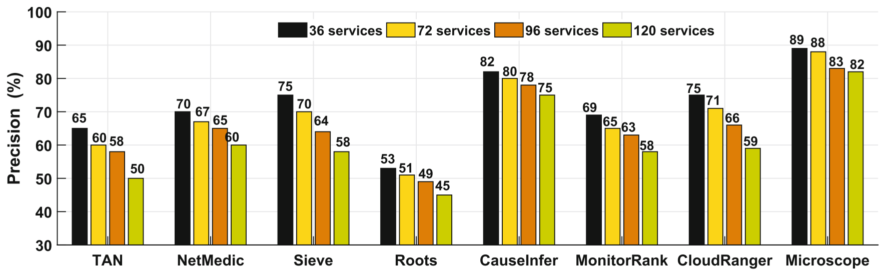


Fig. 10. PR@1 vs serivce instances number

5 Related Work

RCA(Root cause analysis) in large distributed systems with many services is a frustrating task. To pinpoint the root causes of performance problems, a large number of researchers are dedicated to this area. In the following, we present the relative work briefly.

Trace Based Work. Many famous tools fall into this category such as Magpie [4], X-trace [12], Pinpoint [6] and The Mystery Machine [8]. These tools can accurately record the execution path of the program and locate the error by detecting source code or binary code. It's helpful to debug distributed applications. However, deploying these tools is a daunting task and requires administrators to understand the code well. Compared to them, Microscope doesn't need instrumenting the source code and domain knowledge. So it can be easily deployed and used. Although Microscope does not detect true software bugs, it does provide some hints. Roots [15] is a near real-time monitoring and diagnostics framework for web applications deployed in a PaaS cloud without instrumenting the application code. However, Roots only consider web applications and must modify the front-end request server(typically a software load balancer) by instrumenting the request identifier to an HTTP request, which hinders it to be widely used.

Signature Based Work. These methods employ a supervised learning algorithm to classify performance anomalies under several typical scenarios. CloudPD [21] uses a layered online learning approach to deal with the higher occurrence of faults in clouds; Fingerprint [5] provides the basis for automatic classification and identification of performance crises in a data center; All of these methods require labeled data or problem tickets and have limited generalization on new anomalies. While Microscope is an unsupervised method, so it's able to capture new anomalies and have a good generalization.

Dependency Graph Based Work. In recent years, performance diagnosis based on dependency graphs has become a surge. Using graph models, we can not only understand the problem propagation path but also infer the root cause. Sieve [23] infers metrics dependencies between distributed components of the system by using Granger Causality tests. CauseInfer [7] automatically builds a two-layered hierarchical causality graph and the dependency direction is determined by a lag correlation. This method strongly depends on data and the dependent direction is generated by some statistical method. Compared to them, Microscope uses the network connection information, which is real dependencies, to build the causality graph. TAN [9] is used to infer performance issues at a metric level, but it is not effective enough due to a lack of causality.

Finally, compared to much excellent performance monitoring and analysis tools such as Splunk, CaleCd, and IBM Tivoli Gardens, Microscope provides more advanced analysis capabilities, such as causality graph building and root cause inference techniques, without the need for human intervention.

6 Conclusion and Future Work

This paper designs and implements Microscope, a novel system for helping operators and developers with pinpointing the root causes in microservices environments. Microscope automatically builds a causality graph without domain knowledge and instrumenting the application and infers the root causes along the directed edges in the graph. The experimental evaluation shows that Microscope not only can achieve a high precision and recall for performance diagnosis but also is lightweight and can scale up readily in large distributed systems.

As part of future work, we plan to explore using more types of SLO metrics and methods of ranking candidates to improve the effectiveness and generalization ability of Microscope to adapt more different types of services. We will make Microscope more lightweight in order to work in real-time and validate Microscope in some real microservice systems with more kinds of faults.

Acknowledgment. The work described in this paper was supported by the National Key R&D Program of China (2018YFB1004804), the National Natural Science Foundation of China (61722214) and the Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme 2016.

References

1. https://puppet.com/blog/2017-state-devops-report-here
2. https://microservices-demo.github.io/
3. https://people.seas.harvard.edu/~apw/stress/
4. Barham, P., Donnelly, A., Isaacs, R., Mortier, R.: Using magpie for request extraction and workload modelling. In: OSDI, vol. 4, pp. 18–18 (2004)
5. Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H.: Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of the 5th European conference on Computer systems, pp. 111–124. ACM (2010)
6. Chen, M.Y., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.: Pinpoint: problem determination in large, dynamic internet services. In: Proceedings of International Conference on Dependable Systems and Networks (DSN 2002), pp. 595–604. IEEE (2002)
7. Chen, P., Qi, Y., Zheng, P., Hou, D.: Causeinfer: automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In: INFOCOM, 2014 Proceedings IEEE, pp. 1887–1895. IEEE (2014)

8. Chow, M., Meisner, D., Flinn, J., Peek, D., Wenisch, T.F.: The mystery machine: end-to-end performance analysis of large-scale internet services. In: Proceedings of the 11th symposium on Operating Systems Design and Implementation, pp. 217–231 (2014)
9. Cohen, I., Chase, J.S., Goldszmidt, M., Kelly, T., Symons, J.: Correlating instrumentation data to system states: a building block for automated diagnosis and control. In: OSDI, vol. 4, pp. 16–16 (2004)
10. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
11. Ellis, B., Wong, W.H.: Learning causal Bayesian network structures from experimental data. J. Am. Stat. Assoc. **103**(482), 778–789 (2008)
12. Fonseca, R., Porter, G., Katz, R.H., Shenker, S., Stoica, I.: X-trace: a pervasive network tracing framework. In: Proceedings of the 4th USENIX conference on Networked systems design implementation, pp. 271–284. USENIX Association (2007)
13. Granger, C.W.: Investigating causal relations by econometric models and cross-spectral methods. Econom.: J. Econom. Soc. **37**, 424–438 (1969)
14. Ibidunmoye, O., Hernández-Rodriguez, F., Elmroth, E.: Performance anomaly detection and bottleneck identification. ACM Comput. Surv. (CSUR) **48**(1), 4 (2015)
15. Jayathilaka, H., Krintz, C., Wolski, R.: Performance monitoring and root cause analysis for cloud-hosted web applications. In: Proceedings of the 26th International Conference on World Wide Web, pp. 469–478. International World Wide Web Conferences Steering Committee (2017)
16. Kalisch, M., Bühlmann, P.: Estimating high-dimensional directed acyclic graphs with the PC-algorithm. J. Mach. Learn. Res. **8**(Mar), 613–636 (2007)
17. Kalisch, M., Bühlmann, P.: Robustification of the PC-algorithm for directed acyclic graphs. J. Comput. Graph. Stat. **17**(4), 773–789 (2008)
18. Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., Bahl, P.: Detailed diagnosis in enterprise networks. ACM SIGCOMM Comput. Commun. Rev. **39**(4), 243–254 (2009)
19. Kim, M., Sumbaly, R., Shah, S.: Root cause detection in a service-oriented architecture. In: ACM SIGMETRICS Performance Evaluation Review, vol. 41, pp. 93–104. ACM (2013)
20. Newman, S.: Building Microservices, 1st edn. O'Reilly Media Inc., Sebastopol (2015)
21. Sharma, B., Jayachandran, P., Verma, A., Das, C.R.: CloudPD: problem determination and diagnosis in shared dynamic clouds. In: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–12. IEEE (2013)
22. Spirtes, P., et al.: Causation, Prediction, and Search. MIT press, Cambridge (2000)
23. Thalheim, J., et al.: Sieve: actionable insights from monitored metrics in distributed systems. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, pp. 14–27. ACM (2017)
24. Wang, P., et al.: Cloudranger: root cause identification for cloud native systems. In: Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2018). IEEE (2018)

Architecture-Based Automated Updates of Distributed Microservices

Fabienne Boyer[1](✉), Xavier Etchevers[2], Noel de Palma[1], and Xinxiu Tao[2]

[1] UGA/LIG, Grenoble, France
{fabienne.boyer,noel.depalma}@univ-grenoble-alpes.fr
[2] Orange Labs, Paris, France
{xavier.etchevers,xinxiu.tao}@orange.com

Abstract. Microservice architectures are considered really promising to achieve devops in IT organizations, because they split applications into services that can be updated independently from each others. But to protect SLA (Service Level Agreement) properties when updating microservices, devops teams have to deal with complex and error-prone scripts of management operations. In this paper, we leverage an architecture-based approach to provide an easy and safe way to update microservices.

Keywords: Microservices · Dynamic update
Architecture-based reconfigurations

1 Introduction

To facilitate agile development and operations (*devops*), many companies, including established ones such as Netflix [1] and Uber [2], are switching to a microservice architecture for their Cloud applications. With this approach, applications are designed as loosely-coupled services deployed on distributed PaaS (Platform-as-a-Service) sites and running in their own full-stack [3].

The key property that is expected from microservices is the notion of independent replacement and updatability. Especially, microservices exhibit independent lifecycles: they can be deployed and updated independently from each others. The objective is to favor reactivity of small development teams, each team being in charge of developing and evolving its own set of microservices through simple and fast processes.

Such an objective is attractive, but the reality is much more complex because microservices are often associated to SLA properties regarding availability, performances, and resource costs [4,5]. To keep these properties at update time, devops teams follow complex strategies. Typically, the well-known *BlueGreen* strategy [6] intends to update a microservice with zero downtime, but requires deploying and starting all the new microservices before stopping and uninstalling the old ones. In comparison, the *Canary* strategy [7,8] minimizes the resources used at update time, at the expense of a reduced availability: microservices are updated *in-place* (new instances taking the place of the old ones), in an incremental manner to slowly transfer the load from the current to the new version.

Using *strategies* to update microservices is considered relevant [9], but so far, the process is managed manually or only automated through using scripts [10]. Scripts provide flexibility but their imperative form limits their ease of use. When devops teams are provided with application-independent scripts, they have to determine what script can be applied to process a given update. Furthermore, they must check that the current state of their application meets the requirements of the chosen script. This is cumbersome and error-prone as most update scripts encompass complex pipelines of PaaS commands. When update scripts are designed specifically for a given application, they can be used in a much easier and safer way, but the price is that devops teams have to compose these scripts, facing the usual coding and debugging challenges.

This paper advocates switching from a script-based to an *architecture-based* approach to automate microservices updates: instead of scripts processing PaaS commands, update strategies are defined as sequences of elementary changes being applied on an *architectural model* of a microservice application. Simply put, this architectural model (also known as model@runtime [11]) reflects how microservices are deployed on PaaS sites and how they are configured. Compared to scripts, the benefits of the proposed approach are the following:

- *ease of use*: to update a microservice application, devops teams simply give as input the desired target architecture, along with the strategy to follow, without having to deal with low-level PaaS commands.
- *preview*: any update can be processed on the architectural model without being applied on the effective system, allowing to preview its result in terms of architectural changes.
- *control*: all stages of an update can be observed on the architectural model. Moreover, at any stage an update can be stopped and resumed with a new target architecture and/or strategy.
- *robustness*: failures occurring at update time are supported.

Leveraging an architecture-based approach raises two main challenges: (i) determining an architectural model encompassing microservices deployed on heterogeneous PaaS sites and (ii) defining a strategy-driven update protocol relying on this architectural model. This paper describes how these challenges were addressed to provide an update framework that can add, remove, migrate, split, or scale microservices as well as upgrade their code or change their configuration across distributed and potentially heterogeneous PaaS sites.

The remaining of this paper is structured as follows. Section 2 summarizes the background. In Sect. 3, we present the architectural model of the proposed update framework. Section 4 describes the strategy-driven update protocol and Sect. 5 focuses on the robustness aspect. An evaluation is given in Sect. 6 and we conclude in Sect. 7.

2 Background

2.1 Microservice Patterns

There is no standard definition for the microservice concept, but common patterns guiding the development of distributed applications on Cloud platforms [12]. We summarize hereafter the patterns that impact the processing of updates.

Microservices are independently deployable modules that run as self-contained units encompassing an operating system along with the necessary runtimes, frameworks, libraries, and code. For improving scalability and availability, each microservice can involve multiple redundant and distributed instances in production.

Microservices communicate through lightweight protocols such as reliable asynchronous bus [8]. They also interact through their provided and consumed services, often exposed as Web services accessed through REST communications. Each instance of a microservice may expose a service through registering a remote API along with a given *route* (*url*) at a registry (usually a per-application registry) such as Consul[1], Apache ZooKeeper[2] or Netflix Eureka[3].

Microservices tolerate the unavailability of the services they access. Two main design patterns are used to this end. Firstly, microservices intend to be stateless through keeping and retrieving any data through an external server, typically a (per-microservice) database. Thereby, any available service instance can be used to execute a given task. Secondly, microservices use smart proxies to access services provided by others microservices [13]. Smart proxies manage the cases where an accessed service is unavailable. Most commonly, depending on the SLA properties of the accessed microservice, a proxy may either (i) select another service instance, (ii) wait for the service to be restored, or (iii) produce a by-default reply, following the circuit-breaker pattern [14,15]. By supporting the unavailability of the services they access, microservices get independent lifecycles—they can be deployed, started, scaled, stopped independently from each other.

2.2 Dynamic Update of Microservice Applications

To update microservices, a first approach is using the Command Line Interface (CLI) provided by PaaS sites [16–18], Although powerful, the CLI is a low-level interface that may be challenging to use directly as most applications are composed of many microservices distributed across PaaS platforms [10].

A second approach relies on using frameworks striving towards continuous delivery features. For instance, Spinnaker [19], AWS CodeDeploy [20] and IBM UrbanCode [21] allow to deploy and update distributed microservices on heterogeneous PaaS platforms. Updating an application goes through a script-based approach where devops teams specify a pipeline of low-level operations to execute. The main limitation is certainly that it is an imperative approach. First,

[1] https://www.consul.io/.
[2] https://zookeeper.apache.org/.
[3] https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance.

devops teams often have to compose the scripts, which faces the usual coding and debugging challenges. Second, they need to check that scripts are compatible with the current state of the updated microservices. Third, they have to make sure that applying such scripts will produce the desired target architecture. Fourth, in case of failures, script-based approaches are usually not idempotent, which requires either to rollback-restart the entire update process, or to analyze the failure to determine how to restart forward, potentially requiring to adapt the scripts.

Another framework, push2cloud [22], allows to update microservices deployed on a single CloudFoundry PaaS site. Recently, push2cloud investigated an approach allowing to express a desired target architecture. However, only mono-site architectures are supported. Moreover, strategies are defined as pipelines of low-level CloudFoundry-specific operations. Finally, failures are fixed and managed manually.

From an academic perspective, [23] automates the deployment of microservices according to a desired target architecture. However, the approach is constrained to using their own dedicated language [24] to program microservices and does not consider update strategies. [10] aims at helping devops teams to manage consistent refactorings, by using a model of a microservice application that covers both architectural and functional aspects. [25] provides an autonomous tool to troubleshoot and repair microservice applications using canary testing and version-aware routing techniques. [26,27] investigate on synthetizing a target architecture for cloud components, considering capacity constraints and conflicts, but they provide no actual mechanism to update running microservices. Other architecture-based approaches for managing reconfigurations [11] are interesting but do not consider any specifics of microservices. In particular, works such as [28–30] have introduced formalisms for automating deployment processes, but they consider components having dependent lifecycles and focus on the management of their dependencies.

3 Architectural Model

With the proposed framework, devops teams update a microservice application by simply giving as input the desired target architecture and the strategy to follow. The strategy may be chosen among pre-defined ones or newly defined.

From a practical point of view, this framework should be launched on a machine having a network connection towards the PaaS sites hosting the microservices to update. Once launched, devops teams can invoke a *pullArchitecture* command to get the current architecture of an application, and a *pushArchitecture* command to update an application towards a given target architecture, following a given strategy, as depicted in Fig. 1.

With both commands, the architecture of an application is expressed through an *architectural model* specifying how microservices are deployed on PaaS sites

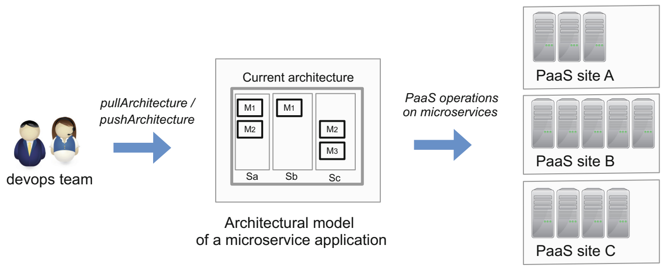

devops team
pullArchitecture / pushArchitecture
Current architecture
M1
M2
M1
M2
M3
Sa
Sb
Sc
Architectural model of a microservice application
PaaS operations on microservices
PaaS site A
PaaS site B
PaaS site C

Fig. 1. Operational view

and how they are configured (Listing 1.1[4]). To consider both PaaS-common and PaaS-specific configuration attributes, a microservice is expressed as an extensible set of (attribute, value) pairs[5].

```
Architecture = (String app_id, Set<PaaS−Site> sites);
PaaS−Site = (String site_id, Set<Microservice> services);
Microservice = (String ms_id, Set<String attribute, String value>);
```

Listing 1.1. Architecture model

The architectural model can be introspected and reconfigured through the *PaaSOperations* interface (Listing 1.2). This interface provides four canonical operations allowing to add, get, modify, or remove microservices, following a CRUD (Create, Read, Update, Delete) approach [31]. For any PaaS targeted by the framework, a specific implementation of this interface should be provided, mapping canonical operations towards PaaS-specific operations[6].

```
interface PaaSOperations{
 Set<Microservice> get(String app_id);
 int add(String app_id, Microservice m);
 int remove(String app_id, Microservice m);
 int modify(String ap_id,Microservice m, Set<String attribute, String value>);
 }
```

Listing 1.2. PaaS introspection and reconfiguration interface

Using the *PaaSOperations* interface, our framework can reconfigure a microservice application towards a desired target architecture through the steps given in Listing 1.3. As an example, let's consider a target architecture upgrading a microservice M, deployed on two *CloudFoundry* PaaS sites S_a and S_b, towards a new version $V_{0.2}$ (current version being $V_{0.1}$).

[4] app_{id}, $site_{id}$ and ms_{id} respectively identify a microservice application, a PaaS site, and a microservice.

[5] PaaS-common attributes include *name, code-version, code-path, route, instances-number, lifecycle-state* (STARTED, STOPPED, *etc.*)).

[6] So far, we mapped this interface for the Cloud Foundry and Heroku platforms.

```
reconfigure(String app_id, Architecture Acurrent, Architecture Atarget) {
 1: compute an architectural diff[27] between Acurrent and Atarget and
    determine the reconfiguration operations (add, remove, modify) to process at each PaaS site
 2: map the reconfiguration operations towards their PaaS−specific implementation
 3: execute the PaaS−specific operations in parallel at each PaaS site
}
```

Listing 1.3. Core reconfigure function

At *step 1*, the *reconfigure* function determines that the following operation should be processed at S_a and S_b:
→ *modify(M,{("code-version", "0.2"),("code-path", "https : //gitX/M/M*$_{0.2}$*.jar")})*
At *step 2*, the *CloudFoundry* implementation of the *PaaSOperations* interface maps this operation as follow:
→ *cf push M –var version=V*$_{0.2}$ *-p https : //gitX/M/M*$_0$*.2.jar*
At *step 3*, this *push* operation is executed in parallel at site S_a and S_b. Notice that such upgrade induces downtime, as the *push* operation stops M, loads its new version of code, and then restarts it, a process taking several minutes in average. The way to avoid downtime is using strategies.

4 Strategy-Driven Updates

A strategy forces an update process to follow a particular path of intermediate architectures, protecting SLA properties throughout the update process, as illustrated in Fig. 2. For instance, still considering the previous upgrade case, the *BlueGreen* strategy would protect the availability property through reaching a first intermediate architecture where M is started in both versions $V_{0.1}$ and $V_{0.2}$, before reaching another architecture where M in version $V_{0.1}$ is stopped.

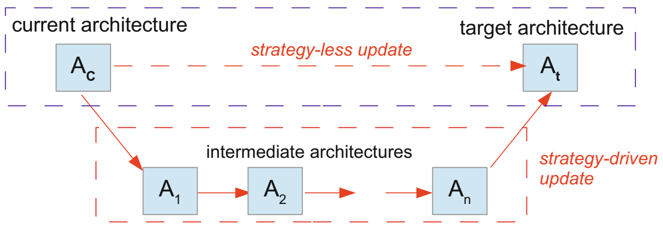


Fig. 2. Strategy-driven update

To account for strategies, we design the *pushArchitecture* command as a *fix-point* (Listing 1.4). At each step, the fix-point requests the strategy to compute the next intermediate architecture along the update path, and then reconfigures the application accordingly.

A strategy is simply defined as a sequence of *transitions*, each transition managing elementary updates. To compute the next intermediate architecture, a strategy goes through its transitions incrementally, until finding one that can evolve the current architecture closer to the target architecture (see *Strategy* and *Transition* definitions in Listing 1.4).

```
pushArchitecture(String app_id, Architecture Atarget, Strategy strategy) {
   Architecture Acurrent, Anext;
   Acurrent = pullArchitecture(app_id, Atarget.sites);

   while (Acurrent.differ(Atarget)) { ////////////// update fix-point
         // compute next intermediate architecture
         Anext = strategy.nextArchitecture (Acurrent, Atarget);
         if (Anext == null) exit("target unreachable");
         // reconfigure towards Anext
         reconfigure (app_id, Acurrent, Anext);
         Acurrent = Anext;
   }
 }

abstract class Strategy {
       // Sequence of transitions (to define in subclasses)
       List<Transition> transitions;

       // compute the next intermediate architecture to reach
      Architecture nextArchitecture(Architecture Acurrent, Architecture Atarget) {
            // process transitions until finding one moving closer to the target
             for each Transition tr in transitions {
                      Architecture Anext = tr.process(Acurrent, Atarget);
                      if (Anext != null) return Anext;
               }
             return null;
      }
 }

interface Transition {
      // returns null if the transition does not allow moving closer to the target
       Architecture process(Architecture Acurrent, Architecture Atarget);
 }
```

Listing 1.4. Strategy-driven updates

```
class AddRemoveStrategy extends Strategy {
      // manages additions then removals of microservices,
      List<Transition> = new List(Tadd, Tremove);
}

class Tadd implements Transition {
      Architecture process (Architecture Acurrent, Architecture Atarget) {
      // get microservices added in Atarget compared to Acurrent
      List<Microservice> additions = Atarget.minus(Acurrent);
      if (additions != null) {
       // return an architecture including current miroservices plus the ones to add
            Architecture Anext = Acurrent.clone();
            Anext.add(additions);
            return Anext;
      } else return null;
}}
...
```

Listing 1.5. Example of strategy and transition definitions

For instance, a transition *Tadd* managing the additions of microservices behaves as follow. Comparing the current and target architectures, it determines if new microservices have to be deployed. If yes, it delivers a next architecture containing the current microservices plus the new microservices to deploy. Symetrically, a transition *Tremove* determines if there are microservices to undeploy. If yes, it delivers accordingly an architecture containing the current microservices minus those to undeploy.

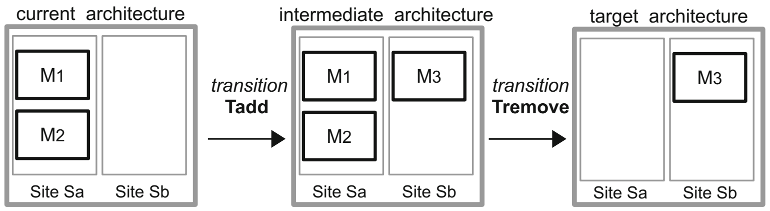


Fig. 3. Using the basic AddRemove strategy

Listing 1.5 illustrates defining a strategy (named *AddRemoveStrategy*) composed of the *Tadd* and *Tremove* transitions. Figure 3 depicts the behaviour of this strategy when updating an elementary application composed of two microservices (M_1, M_2) deployed on a site S_a. The target architecture only contains the microservice M_3 on S_b.

– At the first step, the update fix-point processes the first transition (*Tadd*) of the strategy, that delivers the intermediate architecture composed of the current microservices plus M_3 deployed on S_b. The application is then reconfigured towards this intermediate architecture.
– At the second step, the fix-point processes again the transition *Tadd*, that has no more changes to perform. It then processes the next transition (*Tremove*), that removes microservices not appearing in the target architecture (M_1 and M_2 on S_a). The application is then reconfigured towards this architecture and the fix-point terminates because the target has been reached.

Notice that transitions may apply changes over several steps of the fix-point. Let's consider a transition scaling up microservices horizontally as follow. For each microservice to scale, new instances should be deployed and started one by one[7]. Each time it is processed, this transition returns a next architecture in which every microservice to scale has one more instance. When all microservices have reach their target number of instances, it simply returns null.

5 Update Robustness

Two main kinds of failures may interrupt an update process, letting the application in an arbitrary architectural state: first, the framework may faces a hardware failure or a software one, for example a strategy raising an exception when computing the next architecture; second, a microservice may fail when reconfigured

[7] This pattern is required for microservices that do not support having several instances started concurrently.

on a PaaS site, for example it fails to start. With our approach, this is not a problem since any update process may be stopped at any time and later re-started as a fresh update process. This *kill-restart* capability relies on the following main properties.

– *Runnability*. Whatever the current architectural state for a microservice, it can always be introspected and reconfigured by the PaaS site hosting it (even a failed microservice can be restarted by PaaS operations).
– *Idempotence*. Transitions compare the current and target architectures to determine the changes to process. Once a change has been done, a transition just does not do it again. Thus it is always possible to restart an update process that just failed.

Notice that re-starting an update process offers a way to change the target architecture and/or the strategy (Fig. 4), allowing the devops team to fix some microservice configuration or to rollback to a previous architecture for the updated application.

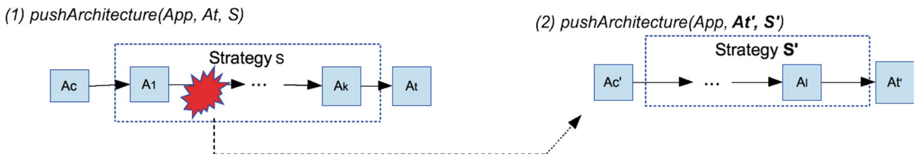

(1) pushArchitecture(App, At, S)
Strategy s
Ac
A1
...
Ak
At
(2) pushArchitecture(App, At', S')
Strategy S'
Ac'
...
Al
At'

Fig. 4. Management of failures at update time

6 Evaluation

We evaluate our framework on the ease of programming strategies, the ease of updating microservices, and the ability to protect SLAs at update time.

6.1 Ease of Programming Strategies

Let's consider first the *BlueGreen* strategy that updates an application without downtime – through installing, starting, and testing the new version (called the green one) before uninstalling the current version (called the blue one). Once the green environment is ready, incoming requests should be routed to it. We defined such strategy with four transitions.

```
class BlueGreen implements Strategy {
    List<Transition> = new List(Tadd, Tupdate, Tswitch, Tremove);
}
```

Listing 1.6. BlueGreen Strategy

In short, *Tadd* deploys microservices newly defined in the target architecture. *Tupdate* deploys the green version of the microservices that are modified in the target architecture (associating them to a temporary route (i.e., url) for testing purposes). *Tswitch* switches from the temporary route to the regular one for green microservices deployed at the previous step. Finally, *Tremove* removes microservices that are no longer defined in the target architecture. The code of the *Tupdate* and *Tswitch* transitions is shown hereafter (*Tremove* is quite similar to *Tadd* given in Sect. 4). Altogether, these four transitions required only 54 lines of code.

```
class Tupdate implements Transition {
    Architecture process(Acurrent, Atarget) {
        Architecture Anext;
        // get microservices modified in At compared to Ac
        List<Microservice> modified = getModified(Ac, At);
        if (updates != null) {
            Anext = Ac.clone();
            for each Microservice m in modified {
                // deploy green version for the microservice to modify
                Microservice mgreen = m.clone();
                mgreen.route = m.get("temporary-route");
                mgreen.set("role", "green");
                mgreen.set("blue-version", m.get("id"));
                Anext.add(mgreen);
            }
        }
        return Anext;
    } // end of process method
}

class Tswitch implements Transition {
    Architecture process(Acurrent, Atarget) {
        Architecture Anext;
        // get green versions of microservices in current architecture
        List<Microservice> greens = getGreens(Ac);
        if (greens != null) {
            Anext = Ac.clone();
            for each Microservice m in greens {
                    // remove blue version of the microservice
                     Anext.remove(m.get("blue-version"));
                    // assign the regular route to the green version
                    m.set("route", m.get("regular-route"));
                    m.set("role", "blue");
                }
        }
        return Anext;
    } // end of process method
}
```

Listing 1.7. Tupdate and Tswitch transitions

Additionally to the BlueGreen strategy, we programmed a dozen of other classical update strategies[8], some summarized in Table 1. Altogether, they only required programming about fifteen transitions and each strategy was only composed a few transitions (see column named nT). Overall, all transitions were easy to program: (1) they were following similar patterns, essentially comparing the current and target architectures to determine the next architecture, (2) they only required a few lines of code (less than 30).

[8] The code is available at https://github.com/tao-xinxiu/prototype-template-engine.

Table 1. Some strategies programmed

Name	Description	nT
Straight	Reach target directly (no intermediate architecture)	1
CleanRedeploy	Remove all in current, deploy target, one microservice at a time	2
BlueGreen	Reach target, creating green versions then removing old (blue) versions for microservices to update	4
BlueGreenByGrp	As BlueGreen, but processes at most k microservices at a time per site	4
Canary	Reach target, incrementally stopping and restarting instances for microservices to update, site by site	6
CanaryBySite	As Canary, but all instances in parallel on a site	3
CanaryByInst	As Canary, but all sites in parallel	6
Mixed	Reach target, creating one new instance for any microservice to update (for test pupose) then apply Canary strategy for pending instances	5

6.2 Ease of Updating Microservices

We report here on using the proposed framework to deploy and update a microservice application composed of three microservices deployed on two PaaS sites. To perform the initial deployment, we simply use the *pushArchitecture* command with the *Straight* strategy and the desired initial architecture (shown in the left part of the Fig. 5) as target.

Then, to update the application towards the target architecture shown in the right part of the Fig. 5, upgrading M_1 and removing M_3, we decide to use the *BlueGreen* strategy to avoid downtime. The framework allows us to follow step by step the update, through the path of intermediate architectures. When at the architecture 1, we perform some tests, checking that the newly deployed microservice M_1' runs properly on S_a and S_b. Note that external client requests are still routed to M_1, as M_1' is assigned a temporary route.

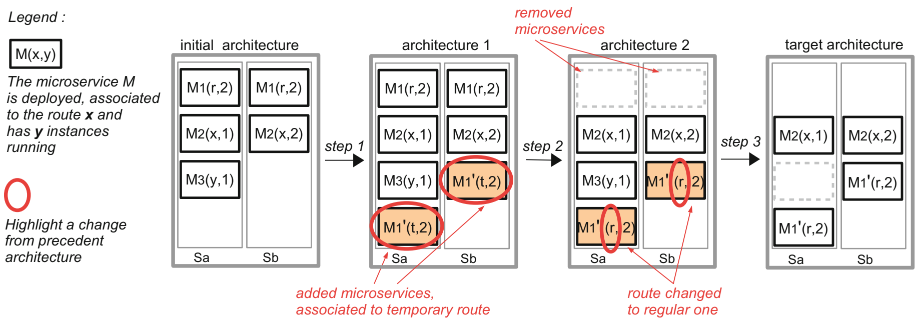

Legend :
M(x,y)
The microservice M is deployed, associated to the route x and has y instances running
Highlight a change from precedent architecture
initial architecture
M1(r,2)
M2(x,1)
M2(x,2)
M3(y,1)
Sa
Sb
step 1
architecture 1
M1'(t,2)
step 2
removed microservices
architecture 2
M1'(r,2)
step 3
target architecture
added microservices, associated to temporary route
route changed to regular one

Fig. 5. Elementary application update through the BlueGreen strategy

Once the final target architecture has been reached, we want to continue with a new update, upgrading M_1' towards a new version M_1'' on S_a and S_b. Due to a faulty configuration, M_1'' fails to run properly on S_b, automatically stopping the update. Here we only have to fix M_1'''s configuration and re-issue the same *pushArchitecture* request to continue the upgrade of M_1' towards M_1''.

However, M_1'''s still fails to run properly on S_b. To fix the problem, we decide to launch a new *pushArchitecture* command with the initial architecture as target, which rollbacks the partial updates we just tried. This results in the automatic un-deployment of M_1'' on S_a and S_b.

Finally, to consider framework failures, we process updates along with a script that randomly kills the framework. This time we consider longer processes, updating one hundred microservices. Each time the framework is killed, whatever the current state of the application, we just have to re-launch it and re-issue the last *pushArchitecture* command to pursue the update towards the desired target.

6.3 Protecting SLA Properties: Real-Life Application Usecase

We used our framework to update a complete clone of a cross-canal order capture application in live production at Orange, focusing on the simultaneous update of two microservices (*(C)atalog* and *(E)ligibility*) that were deployed redundantly over three distributed CloudFoundry (version 2.75.0) PaaS sites (S_0, S_1, S_2). The two microservices are about 10000 lines programmed in Java/Angular. CloudFoundry ran on Cloudwatt [33] on top of OpenStack [34], under VMs with medium flavor (4GB/2VCPU/50GB disk). The update to perform included a code upgrade for the *Catalog* microservice and configuration changes for the *Eligibility* microservice. We experienced with four strategies, comparing their metrics: duration of the whole update process, downtime of the microservices during the update (evaluated with *Apache Jmeter*), and resource consumption (representative of the billing costs for the update). Each experiment was performed 30 times. Results are given in Fig. 6.

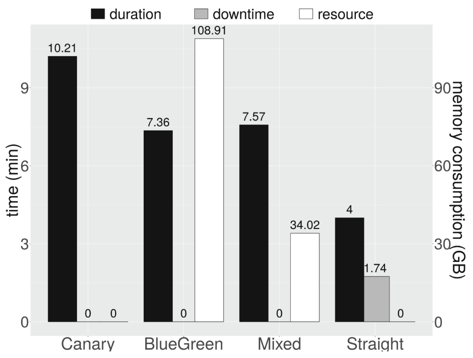


Fig. 6. Update metrics

The *Canary* strategy (whise behavior is shown in Fig. 7) does not involve additional cost in terms of resource usage. It ensures that any deployment of a

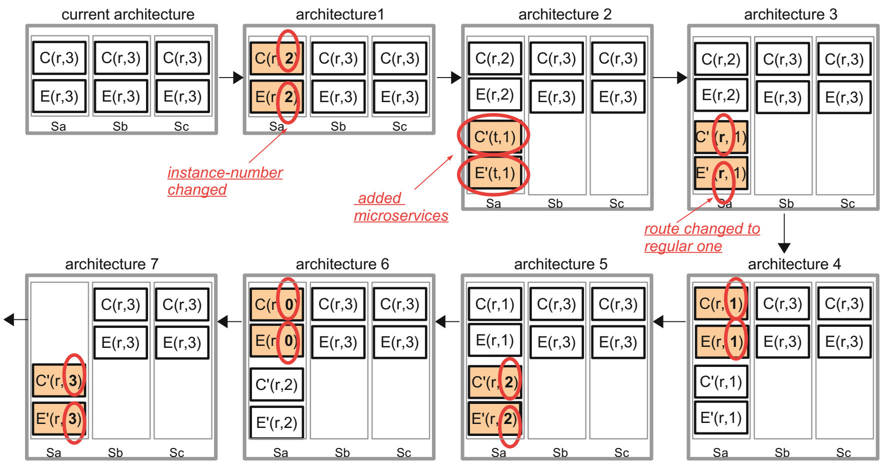


Fig. 7. Real application update through the Canary strategy

new microservice instance is preceded by the removal of a current one. In the version we used, the first new instance being deployed is associated to a temporary route to allow testing before continuing the update. Notice that in case of high request load, some client requests may not be served as the microservices have one less instance running during the update. Overall, this strategy involves 21 intermediate architectures to reach the target, 7 per site, sites being updated sequentially. Its processing took about 10 min.

In comparison, the *BlueGreen* strategy updates the two microservices through creating the three new (green) instances before removing the three current (blue) ones, on all sites in parallel. Accordingly, it ensures zero downtime but consumes nearly the double of memory during the update. The duration is about 7 min, corresponding to the time required to create the new microservices instances (which includes uploading their code), to switch their route, and remove the three blue instances, on one site.

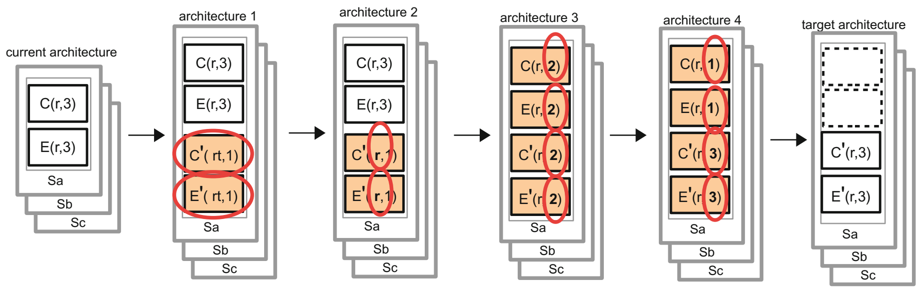


Fig. 8. Real application update through the Mixed strategy

The *Mixed* strategy (Fig. 8) updates microservices instance by instance, creating a new instance before removing an old one, across all sites in parallel. This strategy takes approximately the same duration as *BlueGreen*, with no downtime. It further limits the update cost in terms of resource usage since it uses only one extra instance for each microservice while the *BlueGreen* doubles the number of instances per site.

Finally, the *Straight* strategy delivers the shortest update duration (4 min), as it reaches the target without going through intermediate architectures. The duration corresponds to the time needed to update both microservices on one site, the three sites being updated in parallel. This strategy does not consume any additional resources but induces the largest downtime as the microservices are stopped before their new version is uploaded, recompiled, and restarted.

7 Conclusion

With the proposed framework, devops teams update microservices through specifying target architectures and chosing strategies. They can follow an update step by step, with the opportunity to change the strategy or adapt the target architecture at each step, which is key to handle failures gracefully. Since it permits to pre-compute a path of intermediate architectures, the architecture-based approach leveraged in this paper opens up an interesting perspective: predicting how well a strategy will protect SLA properties during an update.

Acknowledgements. We would like to thank the Orange company for its valuable collaboration in the presented work. This work was partially supported by the FSN HYDDA and the EU FEDER STUDIO VIRTUEL projects.

References

1. Mauro, T.: Adopting Microservices at Netflix: Lessons for Architectural Design. https://goo.gl/DyrtvI
2. Hoff, T.: Lessons Learned From Scaling Uber To 2000 Engineers, 1000 Services, and 8000 Git Repositories. https://goo.gl/1MRvoT
3. Amundsen, M., McLarty, M., Mitra, R., Nadareishvili, I.: Microservice Architecture - Aligning Principles, Practices, and Culture. O'Reilly Media, Sebastopol (2016)
4. Fowler, S.: Production Ready Microservices. O'Reilly, Sebastopol (2016)
5. Carnero, C.: Microservices From Day One: Build Robust and Scalable Software From the Start. Apress, New York City (2016)
6. Fowler, M.: Blue-Green deployment (2010). https://martinfowler.com/bliki/BlueGreenDeployment.html
7. Sato, D.: Canary update strategies (2014). https://martinfowler.com/bliki/CanaryRelease.html
8. Tarvo, A., Sweeney, P.F., Mitchell, N., Rajan, V., Arnold, M., Baldini, I.: Canaryadvisor: a statistical-based tool for canary testing (demo). In: International Symposium on Software Testing and Analysis, ISSTA 2015, pp. 418–422 (2015)

9. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, 1st edn. Addison-Wesley Prof, Boston (2010)
10. Sampaio, A.R., et al.: Supporting microservice evolution, In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 539–543 (2017)
11. Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.): Models@run.time. LNCS, vol. 8378. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08915-7
12. Newman, S.: Building Microservices - Designing Fine-Grained Systems, 1st edn. O'Reilly, Boston (2015). http://www.worldcat.org/oclc/904463848
13. Montesi, F., Weber, J.: Circuit breakers, discovery, and API gateways in microservices, CoRR, vol. abs/1609.05830 (2016)
14. Nygrad, M.T.: Stability patterns. Release It!: Design and Deploy Production-Ready Software, 1st edn. Pragmatic Bookshelf, Raleigh (2007)
15. Fowler, M.: Circuit Breaker (2014). https://martinfowler.com/bliki/CircuitBreaker.html
16. Cloud foundry. http://www.cloudfoundry.com/
17. Heroku. https://www.heroku.com/
18. Openshift. https://www.openshift.com/
19. Spinnaker. https://www.spinnaker.io/
20. AWS CodeDeploy. https://aws.amazon.com/codedeploy/
21. IBM UrbanCode. https://developer.ibm.com/urbancode/
22. Push2Cloud website. https://push2.cloud/
23. Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J., Montesi, F.: Self-reconfiguring microservices. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 194–210. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30734-3_14
24. Jolie. Official Web Site. http://www.jolie-lang.org/
25. Rajagopalan, S., Jamjoom, H.: App-bisect: Autonomous healing for microservice-based apps. In: 7th USENIX Conference on Hot Topics in Cloud Computing, HotCloud 2015 (2015)
26. Di Cosmo, R., Lienhardt, M., Mauro, J., Zacchiroli, S., Zavattaro, G., Zwolakowski, J.: Automatic application deployment in the cloud: from practice to theory and back. In: 26th International Conference on Concurrency Theory (CONCUR 2015), vol. 42, Madrid, Spain, pp. 1–16 (2015)
27. Di Cosmo, R., Eiche, A., Mauro, J., Zacchiroli, S., Zavattaro, G., Zwolakowski, J.: Automatic deployment of services in the cloud with Aeolus blender. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 397–411. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_28
28. Lascu, T.A., Mauro, J., Zavattaro, G.: A planning tool supporting the deployment of cloud applications. In: 25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, pp. 213–220 (2013)
29. Fischer, J., Majumdar, R., Esmaeilsabzali, S.: Engage: a deployment management system. In: 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2012), pp. 263–274. ACM (2012)
30. Etchevers, X., Coupaye, T., Boyer, F., Palma, N.D., Salaün, G.: Automated configuration of legacy applications in the cloud. In: IEEE/ACM Conference on Utility and Cloud Computing (UCC 2011), pp. 170–177 (2011)
31. Martin, J.: Managing the Data Base Environment, 1st edn. Prentice Hall PTR, Upper Saddle River (1983)

32. Kramer, J., Magee, J.: The evolving philosophers problem: dynamic change management. IEEE TSE **16**(11), 1293–1306 (1990)
33. Cloudwatt. https://www.cloudwatt.com/fr/
34. Openstack. https://www.openstack.org/

Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems

Adambarage Anuruddha Chathuranga De Alwis[1], Alistair Barros[1](✉), Artem Polyvyanyy[2], and Colin Fidge[1]

[1] Queensland University of Technology, Brisbane, Australia
{adambarage.dealwis,alistair.barros,c.fidge}@qut.edu.au
[2] The University of Melbourne, Parkville, VIC 3010, Australia
artem.polyvyanyy@unimelb.edu.au

Abstract. We present heuristics that help to identify suitable consumer-oriented parts of enterprise systems which could be re-engineered as microservices. Our approach assesses the key structural and behavioural properties common to both enterprise and microservice systems, as needed to guide a microservices discovery process and coherently assess restructuring recommendations. Building upon existing business object and system structural definitions, we present heuristics for two fundamental areas of microservice discovery, namely function splitting based on object subtypes (i.e., the lowest granularity of software based on structural properties) and functional splitting based on common execution fragments across software (i.e., the lowest granularity of software based on behavioural properties). A prototype analysis tool was developed based on the defined heuristics and experiments show that it can identify microservice designs which support multiple microservice characteristics, such as high cohesion, low coupling, high scalability, high availability, and processing efficiency while preserving coherent features of enterprise systems. In particular, we illustrate the usefulness of this new approach by conducting a case study based on customer management systems: SugarCRM and ChurchCRM.

Keywords: Microservice discovery · System reengineering
Cloud migration

1 Introduction

Microservices have emerged as the latest style of service-based software allowing systems to be distributed through the cloud as fine-grained components, typically with individual operations, in contrast to services under a Service-Oriented Architecture (SOA) which include all logically related operations [1]. As such, microservices allow specific parts of systems and the business processes they support, down to individual tasks, to be scaled up and replicated through the cloud, and be flexibly composed in Web, mobile computing, and Internet-of-Things (IoT) applications. These benefits originally led Netflix, and now Twitter, eBay, Amazon and other Internet players, to develop novel architectures for software solutions as microservices. Nonetheless, microservices have so far not been adopted for the dominant form of software in businesses, namely enterprise systems, limiting such systems' evolution and their exploitation of the full benefits of cloud-enabled platforms such as Google Cloud, Amazon AWS and IoT [2].

Enterprise systems, such as enterprise resource planning (ERP), customer relationship management (CRM) and supply chain management are large and complex, and contain complex business processes encoded in application logic managing business objects (BOs), in typically many-to-many relationships [3]. Restructuring enterprise systems as microservices is technically cumbersome, requiring tedious search and identification of suitable parts of the system to restructure, program code rewrites, and integration of the newly developed microservices with the 'backend' enterprise systems. This is a costly and error-prone task for developers, because enterprise systems have millions of lines of code and thousands of BOs they manage, entailing a multitude of functional dependencies, in and across many software packages and modules. In addition, microservices are the most fine-grained and loosely-coupled form of software components upon which to restructure large-scale enterprise systems. This leads to major uncertainties about the best way to split enterprise systems functions as microservices, to achieve high scalability and availability and low system latencies through the cloud, while attaining high cohesion and low coupling between software components.

Automated software re-engineering techniques have been proposed to improve the efficiency of transforming legacy applications, addressing specifically cohesion and coupling of software packages and components using static analysis techniques that focus on source code and dynamic analysis techniques that focus on software execution recorded in system logs. Even though these analyses proposed to improve software search and metrics, studies show that the success rate of software re-modularisation techniques, especially for large systems, remains low [4]. The key stumbling blocks are the limited insights available from syntactic structures of software code for profiling software dependencies and not identifying the semantics available through the business object relationships [5].

Enterprise systems can provide enriched semantic insights, available through the BOs that they manage which influence the software structure and the processes they support. For instance, an order-to-cash process in SAP ERP is supported through functions of software components: multiple sales orders, deliveries shared across different customers, shared containers in transportation carriers, and multiple invoices and payments. To support this process, multiple functions are invoked asynchronously, reflecting BO relationship types and cardinalities, and are seen through cross-service interactions, correlations, and data payloads [6]. Such insights provided by BO relationships are promising for improving the feasibility of automated discovery applications. As examples, Pérez-Castillo *et* *al.* [7] used transitive closures of strong BO dependencies derived from databases to recommend software functions hierarchies, while Lu *et al.* [8] demonstrated process discovery using SAP ERP logs based on BOs.

This paper presents discovery techniques that help to identify suitable consumer-oriented parts of enterprise systems which could be re-engineered as microservices with desired characteristics such as high cohesion, low coupling, high scalability, high availability and high processing efficiency. It does so by providing an abstraction of the systems architecture, using key structural and behavioural properties common to both enterprise and microservices systems, considered essential to guide microservices discovery processes and coherently assess their potential restructuring. The structural properties address the functional composition of software, namely functions and their BO Create-Read-Update-Delete (CRUD) operations, while behavioural properties focus on system executions, at the level of operation invocations, reflecting single-entry-single-exit sequences characteristic of these systems. This, in principle, allows enterprise systems to be analysed at different units of structural and behavioural granularity, and the resulting restructure recommendations to be conveniently assessed for preservation of structural and behavioural properties. This paper addresses two fundamental areas of microservice discovery, namely function-splitting based on object subtypes (i.e., the lowest granularity of software based on structural properties) and functional splitting based on common execution fragments across software (i.e., the lowest granularity of software based on behavioural properties). This, we argue, provides a solid basis for future development of further microservices discovery heuristics.

The remainder of the paper is structured as follows. Section 2 presents structural and behavioural properties of software systems, while Section 3 exploits these properties to propose heuristics for discovering microservises in enterprise systems. Section 4 discusses an implementation and validation of the proposed heuristics. Related work is summarized in Section 5. The paper closes with a conclusion.

2 Structural and Behavioural Properties of Enterprise and Microservice Systems

This section describes the essential properties of a target system architecture that comprise an enterprise system (ES) and a microservice (MS) system, which is depicted in Fig. 1. This architecture will be used in our MS discovery approach (detailed in Sect. 3). The architecture reflects a unified software structure for both an ES and MS system, since a proper system migration from an ES to MS system is an incremental process in which the most prominent components are extracted and remodularized as MSs first [1]. Such remodularized MSs run in a cloud setting and are integrated with the 'backend' enterprise system as depicted in the Fig. 1.

The software structure of an ES (e.g., an ERP system) consists of a set of self-contained modules (e.g., software components) drawn from different subsystems (e.g., production management), deployed on a specific execution platform.

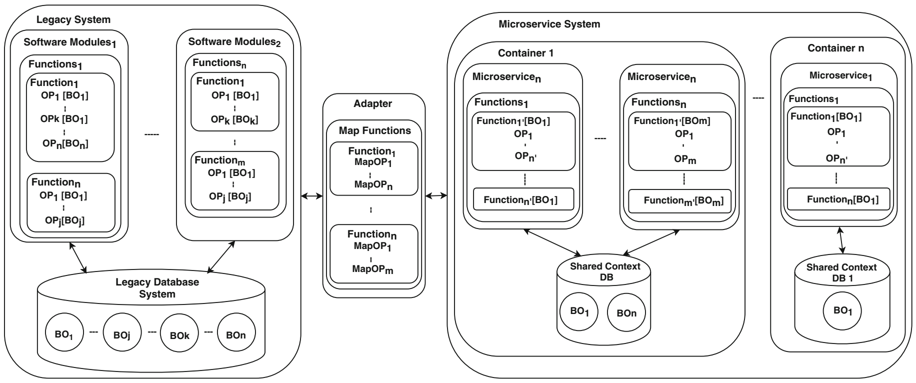


Fig. 1. Architecture of enterprise and microservices systems.

Modules consist of a set of functions (e.g., software classes) and each function consists of a number of operations (e.g., methods) aimed at manipulating BOs through CRUD operations which typically have database access logic or data processing logic applied to data stored in program variables and constants. The data stored in a centralized database associated with a deployed ES relates to BOs which process data resulting from business process executions supported through functions (such as transactions).

MSs are remodularized and potentially extended parts or functions of ESs, supporting consumer applications running in cloud applications. Since MSs are functionally isolated and loosely-coupled parts connected to each other, much like the components of a distributed system, they tend to concern individual BOs, locally managed through a database. The managed data of MSs is synchronized at discrete intervals with similar MS instances and with the backend ES. One or more MSs can run in an execution environment known as a cloud container, configured for specific execution characteristics, such as scalability or availability applying to all the MSs of the container.

Despite structural differences, the behaviour of an ES and an MS system is based on the invocation of operations, in well-defined processing sequences reflecting the relationships of BOs they manipulate. For example, the creation of a 'purchase order' will result in the invocation of functions involving the creation of 'line items' reflecting a strict containment of objects. Similarly, processing sequences between 'Shipper' and 'Shipping order' reflect weak containment while processing sequences between 'leads' and 'campaigns' reflect an association. In addition, normalization of a BO can result in additional process sequences. For example, the creation of a 'shipment' BO will result in an invocation of a function related to different shipments subtypes such as 'ground home delivery shipment' and 'intra UAESO shipment' based on the operational parameters provided at run time. These different execution sequences of operations reflect a set of single-entry-single-exit (SESE) regions [9] in an ES's executions.

Although an MS system executes in a manner similar to an ES, there are specific characteristics applicable only to an MS. Generally MSs run as distributed systems which are deployed through different containers that help the MS system to achieve high scalability and availability while executing services in an asynchronous manner and managing security through configurations of API gateways [1]. Scalability can be defined as allocation and de-allocation of the resources to containers on demand according to the configuration properties. Such configuration properties include load balancing and resource allocation policies optimizing the resources within a container allowing it to provide high scalability at lower cost. Furthermore, the configuration properties define circuit breaker threshold values for each MS which resides in a container, which assures that a request is redirected to another MS if it did not get a response from the initially accessed MS within the threshold time period resulting in high availability. Since there are multiple MSs in a single container, it can process multiple client requests in an asynchronous manner while aligning with the system requirements of the ES. All the MSs communicate through adapters which synchronize with the database system which resides in the legacy ES and this helps to achieve consistency among all the microservices which are distributed among multiple containers. Finally, each MS is developed in order to provide a specific functionality to the end user or the system, which makes them highly cohesive and loosely coupled services. This understanding leads us to the following formal characterization of the environment.

Let $\mathbb{I}$ and $\mathbb{O}$ be a universe of *input* types and *output* types, respectively. Let $\mathbb{OP}$, $\mathbb{T}$ and $\mathbb{B}$ be, respectively, a universe of *operations*, *database tables*, and *business objects*. Finally, let β be a binary relation on $\mathbb{B}$ such that β^+ is irreflexive[1]. Relation β defines a *subtype relation* on business objects, i.e., for every $(b_1, b_2) \in \beta^+$ we say that b_2 is a *subtype* of b_1. As proposed in this paper, techniques for the discovery of microservices rely on *abstractions* of ESs, as defined below.

The data related BOs in ESs are disseminated through several database tables.

Definition 2.1 (Business Object)
A *business object* b is characterized by a collection of database tables, i.e., $b \subseteq \mathbb{T}$. ⌟

The BOs in ESs have complex relationships with the operations which perform CRUD processes on them. Such operations are encapsulated in different functions of ESs and MSs.

Definition 2.2 (Operation)
An *operation* op is a triple (I, O, T), where $I \in \mathbb{I}^*$ is a sequence of *inputs*, $O \in \mathbb{O}^*$ is a sequence of *outputs*, and $T \subseteq \mathbb{T}$ is a set of *database tables*.[2] ⌟

[1] Given a binary relation α, by α^+ we denote the transitive closure of α.

[2] Given a set A, by A^* we denote the set of all finite sequences that can be generated by concatenating elements of A.

An ES can be seen as a finite automaton with operations as labels.

Definition 2.3 (Enterprise system).
An *enterprise system* is a 5-tuple $(Q, \Lambda, \delta, q_0, A)$, where:

∘ Q is a finite nonempty set of *states*,
∘ Λ is a set of *operations*, such that Q and Λ are disjoint,
∘ $\delta : Q \times (\Lambda \cup \{\tau\}) \rightarrow \mathcal{P}(Q)$ is the *transition function*, where τ a is a special *silent operation* such that $\tau \notin Q \cup \Lambda$,
∘ $q_0 \in Q$ is the *start state*, and
∘ $A \subseteq Q$ is the *set of accept states*.[3] ⌟

Let $\mathbb{C}$ and $\mathbb{M}$ be a universe of *containers* and *microservices*, respectively. Let S be an enterprise system. By $SESE(S)$, we denote the set of all (generalized) SESE fragments of S, cf. [9]; clearly, one can interpret an ES as a workflow graph with vertices defined by its states and a flow relation defined by its transition function. Each SESE fragment of an ES induces a *function*, or a *call graph*, i.e., a subgraph of ES. We abstract a function as a triple (I, O, OP), where I and O are sequences of inputs and outputs, respectively, and OP is a set of operations. For our purposes, we define a MSs system as follows.

Definition 2.4 (Microservices System). A *microservices system* of an enterprise system $S = (Q, \Lambda, \delta, q_0, A)$ is a 5-tuple (S, C, M, σ, μ), where:

∘ $C \subseteq \mathbb{C}$ is a set of *containers*,
∘ $M \subseteq \mathbb{M}$ is a set of *microservices*,
∘ $\sigma : C \rightarrow \mathcal{P}(M) \setminus \emptyset$ is a *deployment function* that maps each container $c \in C$ onto a non-empty set of microservices $\sigma(c)$ that are deployed on c, and
∘ $\mu : M \rightarrow \mathcal{P}(SESE(S)) \setminus \emptyset$ is a *microservice definition function* that maps each microservice $m \in M$ onto a non-empty set of SESE fragments, a.k.a *functions*, $\mu(m)$ of S, such that:
  – No two microservices are defined using the same function, i.e., $\forall\, m_1 \in M\, \forall\, m_2 \in M : (m_1 \neq m_2) \Rightarrow ((\mu(m_1) \cap \mu(m_2)) = \emptyset)$, and
  – Every two functions used to define the microservices in M are either disjoint, i.e., do not share an edge, or are in a subgraph relation. ⌟

Given an enterprise system S, $(S, \{c\}, \{m\}, \{(c, \{m\})\}, \{(m, \{S\})\})$, where $c \in \mathbb{C}$ and $m \in \mathbb{M}$, is its *elementary microservices system*, or the elementary enterprise and microservices architecture induced by S.

3 Automated Microservice Discovery

As described in Sect. 2, the behaviour of an ES and an MS system is based on the invocation of functions which consist of well-defined sequences of operations governed by BO relationships. Such sequences illustrate a particular execution pattern based on the structure and behaviour of an organization. Therefore, we argue that a proper analysis of these sequences of operations will help to

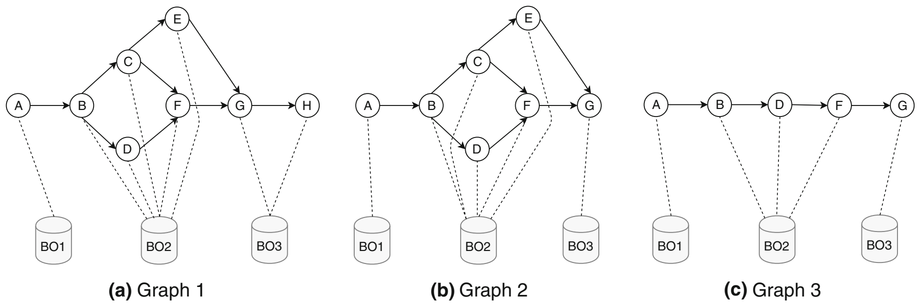


Fig. 2. Patterns of system executions and BO relationships.

derive prominent microserviceable components. This assumption leads us to two heuristics which assist in MS discovery.

As an example, assume that an ES has three hypothetical processing sequences, as depicted in Fig. 2, in which each node of the sequences represents a system state after performing a CRUD operation. These states are linked to the BOs on which different CRUD operations were performed. Figure 2(a) and (b) capture the same execution order dependencies between states 'A', 'B', 'C', 'D', 'E', 'F', and 'G'. Furthermore, the overlap between the execution patterns is high, i.e., more than 80%, which emphasizes that CRUD operations were performed on the same BOs. For instance, the campaign management module in SugarCRM describes different types of publicity campaigns, such as newsletter, email, and non-email. The execution paths and the BOs they execute upon are similar. However, the BO attributes they use in the execution processes are often different. This execution behaviour is explicit because of the structural splitting of objects at the BO level, as described by Halpin and Morgan [10]. To address this phenomenon, we define Heuristic 1.

Heuristic 1 ***(Subtype).*** *Given an enterprise system S, a subtype relation exists between a parent call graph $x = (I, O, OP) \in SESE(S)$ and a child call graph $x' = (I', O', OP') \in SESE(S)$, iff $I' \subseteq I$, $OP' \subseteq OP$, and $B' \subseteq B$, where B' and B are the BOs manipulated by OP' and OP, respectively. To ensure that the call graphs execute on the same BOs, we require that 80% of the states of the parent appear in the child.*

In addition, some execution sequences can occur often when executing a software system. As an example, the execution pattern 'A', 'B', 'D', 'F', 'G' occurs in Fig. 2(a), (b), and (c). This phenomenon depends on the functional relationships that occur during execution time. For example, 'B' precedes 'D' in every execution because, for instance, the data in 'B' is required for the execution of 'D'. In the functional structure level this can be described as a 'has a relationship' property, in which a class object of 'D' is referenced inside 'B'. Such

[3] Given a set A, by $\mathcal{P}(A)$, we denote the powerset of A.

functional structure emphasizes that the same behaviour should be preserved in all the system executions. To address this issue, we define Heuristic 2.

Heuristic 2 ***(Common Subgraph).*** *Given an enterprise system S, a common subgraph of two call graphs $x, x' \in SESE(S)$ is a call graph $x'' \in SESE(S)$, such that $x'' \subseteq x$ and $x'' \subseteq x'$.*

A common subgraph which captures frequent executions can be used as a basis for defining a microservice. This heuristic can be generalized to subgraphs common to multiple call graphs. Intuitively, choosing smaller common subgraphs produces smaller microservices which helps to achieve higher scalability. On the other hand, choosing larger subgraphs produces larger microservices which helps reduce communication overheads and improve system efficiency.

Heuristics 1 and 2 can guide the discovery of microservices that potentially support multiple microservice characteristics, such as high cohesion, low coupling, high scalability, availability, and processing efficiency, while preserving coherent features of enterprise systems. In what follows, this claim gets verified.

3.1 Discovery Process

Our microservice discovery and recommendation process based upon the above heuristics consists of two components, i.e., a *Business Object Analyser (BOA)* and a *System Dynamic Analyser (SDA)*, as depicted in Fig. 3.

Since MSs are focused around accessing and transferring states of BOs, or partitions of BOs, in the system [11], it is important to identify the BOs in a given ES. Therefore, the BOA is comprised of a System Operation Extraction Model (SOEM) and a Business Object Derivation Model (BODM). The SOEM evaluates all the SQL queries to identify the relationships between database tables, while the BODM derives the BOs based on the identified relationships and data similarities, as described by Nooijen *et al.* [12].

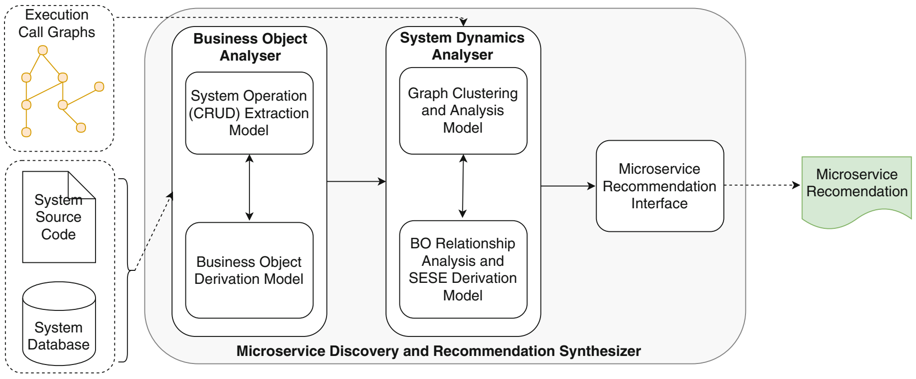

Execution Call Graphs
System Source Code
System Database
Business Object Analyser
System Operation (CRUD) Extraction Model
Business Object Derivation Model
System Dynamics Analyser
Graph Clustering and Analysis Model
BO Relationship Analysis and SESE Derivation Model
Microservice Recommendation Interface
Microservice Recomendation
Microservice Discovery and Recommendation Synthesizer

Fig. 3. An overview of our microservice discovery and recommendation process.

The BOs identified by the BOA are provided as input to the SDA along with call graphs of the ES. The graph clustering and analysis model in the SDA identifies the Frequent Execution Patterns (FEP) in the provided set of $SESE(S)$. These FEPs get evaluated against the aforementioned heuristics and classified into different categories, as described in Sect. 3.2. The categorized patterns are evaluated by BO relationship analysis and a SESE derivation model. The SESE derivation model identifies the BOs which are related to each node in the extracted graph pattern and the SESE regions related to each BO. Finally, the Microservice Recommendation Interface (MRI), provides different configuration models for MSs by evaluating the results of the system dynamic analysis model. Due to space limits, this paper only addresses the SDA and the MRI, which analyse the system execution patterns and recommend MS configuration models.

3.2 Microservice Discovery Algorithms

Given a set of call graphs of a legacy system, like the ones shown in Fig. 2, the SDA and MRI derive sets of MS recommendations based on Heuristics 1 and 2 using Algorithms 1 and 2. Algorithm 1 derives a set of subgraphs in the given set of call graphs of an ES, while Algorithm 2 analyses the subgraphs to identify functions which operate on single BOs to provide MS migration recommendations.

Algorithm 1 comprises four steps. The first step involves function $GRAPHSUMMARY$, which computes the set of adjacency matrices MT of the call graphs $SESE(S)$ (line 1). Each adjacency matrix is generated in two steps. First, the function constructs the set of all distinct states of the graphs. For example, for the three call graphs in Fig. 2, this set comprises states ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, and ‘H’. Then, for each call graph, the function creates a matrix $mt \in MT$ of size $N \times N$, where N is the number of distinct states (for the graphs in Fig. 2, the number of distinct states is eight). In a matrix $mt \in MT$, a transition between two states q and q' of the corresponding call graph is represented by ‘1’ and the absence of a transition is represented by ‘0’.

The second step of Algorithm 1 constructs two matrices, the Augmented Adjacency Matrix (AAM) mt^a and Augmented Graph Matrix (AGM) mt^g, which are of size $N \times N$ (lines 3–10). At the beginning, all the values of mt^a are initialized to ‘0’ and all the values of mt^g are initialized to the empty set. Then, the algorithm iterates over adjacency matrices $mt_k \in MT$ to compute statistics on transitions. The indices of the graphs that contain a transition are recorded in matrix mt^g and the number of graphs that contain the transition gets stored in matrix mt^a. The AAM and AGM generated for the call graphs in Fig. 2 are shown in Fig. 4. In Fig. 4(a), the value in the AAM row ‘A’, column ‘B’ is ‘3’ because all the three call graphs depicted in Fig. 2 have a transition (an edge) from node ‘A’ to node ‘B’. Similarly, in Fig. 4(b), the value in the AGM row ‘A’, column ‘B’ encodes the graphs that contain the corresponding transition. Since the transition is in all the three call graphs, the value has been set to ‘1’, ‘2’, ‘3’.

Algorithm 1. Calculate AAM and AGM

```
Require: An enterprise system $S$.
 1: $MT = \{mt_1, \ldots, mt_n\} := GRAPHSUMMARY(SESE(S))$  //Generate the summary matrix
 2: /* Iterate through each $mt_k$ in $MT$ */
 3: for each $k \in [1 .. n]$ do
 4:     for each $i \in [0 .. N-1]$, where $N$ is the number of distinct states in $SESE(S)$ do
 5:         for each $j \in [0 .. N-1]$ do
 6:             $mt^a[i][j] := mt^a[i][j] + mt_k[i][j]$
 7:             $mt^g[i][j] := mt^g[i][j] \cup \{k\}$
 8:         end for
 9:     end for
10: end for
11: $Sub = \langle sub_0, \ldots, sub_m \rangle := IDENTIFYSUBGRAPHS(mt^a, mt^g)$  //Get common subgraphs
12: for each $i \in [0 .. m]$ do
13:     $parents := \{mt \in MT \mid sub_i \text{ is a subgraph of } mt\}$
14:     /* Record the similarity value for subgraph $sub_i$ in the Sim list */
15:     $Sim_i := similarity(sub_i, parents)$
16: end for
17: return $(Sub, Sim)$
```

In the third step of Algorithm 1 the generated matrices mt^a and mt^g are passed as input to the *IDENTIFYSUBGRAPHS* function which computes the adjacency matrices of the common subgraphs *Sub* of the call graphs (line 11).

	A	B	C	D	E	F	G	H
A	0	3	0	0	0	0	0	0
B	0	0	2	3	0	0	0	0
C	0	0	0	0	2	2	0	0
D	0	0	0	0	0	3	0	0
E	0	0	0	0	0	0	2	0
F	0	0	0	0	0	0	3	0
G	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0

(a) AAM

	A	B	C	D	E	F	G	H
A	Ø	1,2,3	Ø	Ø	Ø	Ø	Ø	Ø
B	Ø	Ø	1,2	1,2,3	Ø	Ø	Ø	Ø
C	Ø	Ø	Ø	0	1,2	1,2	Ø	Ø
D	Ø	Ø	Ø	Ø	Ø	1,2,3	Ø	Ø
E	Ø	Ø	Ø	Ø	Ø	Ø	1,2	Ø
F	Ø	Ø	Ø	Ø	Ø	Ø	1,2,3	Ø
G	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
H	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

(b) AGM

Fig. 4. Intermediate matrices used by Algorithm 1 computed for the call graphs in Fig. 2.

In the fourth step, the algorithm iterates over subgraphs in *Sub* and measures the similarity Sim_i between subgraph sub_i and all its parent graphs (lines 12–16). The similarity is identified as the ratio of the number of nodes in sub_i to the number of distinct nodes in all the parent graphs. Finally, the algorithm returns the identified common subgraphs and calculated similarity values (line 17). If a similarity value is greater than 0.8 for a particular subgraph, we identify that the subgraph and its parent call graphs satisfy Heuristic 1.

The subgraphs which are common to all the call graphs in $SESE(S)$ satisfy Heuristic 2. However, further processing is required to identify functions that act upon single BOs. To accomplish this check, we present Algorithm 2.

Algorithm 2 consists of three steps. The first step involves identifying the states of the input subgraphs with no more than two incident transitions, a.k.a

Algorithm 2. Compute functions for given BOs

```
Require: A set of BOs $B$ and list of graphs $Sub = \langle sub_0, \ldots, sub_m \rangle$
1: for each $i \in [0 \mathrel{..} m]$ do
2:     $Q := \emptyset$
3:     for each node $q$ in $sub_i$ do
4:         if $q$ is incident to less than three edges then
5:             $Q := Q \cup \{q\}$
6:         end if
7:     end for
8:     $QS_i := Q$                                  //$QS$ is a list of sets of nodes
9: end for
10: $Y := GENERATEGRAPHS(QS, Sub)$
11: $Z := Y$
12: for each $y \in Y$ do
13:     /* Evaluate graph $y$ to confirm that every operation in $y$ is connected to the same $b \in B$ and
        if not remove it from $Z$ */
14:     if $y$ operates on more than one BO in $B$ then
15:         $Z := Z \setminus \{y\}$
16:     end if
17: end for
18: return $Z$                          //Each function in $Z$ operates over a single BO
```

single-entry-single-exit states (lines 1–9). It is expected that the input subgraphs are generated by Algorithm 1. The loop of lines 1–9 iterates over all the subgraphs, while the loop of lines 3–7 runs over all the nodes of a current subgraph to extract and record the "SESE" states (line 5). The constructed sets of "SESE" states get stored in list QS on line 8.

In the second step of the algorithm function *GENERATEGRAPHS* constructs connected graphs composed of the nodes in QS that are subgraphs of the graphs in Sub, and records the result in set Y (line 10).

In the third step, the algorithm evaluates each graph $y \in Y$ to verify whether all the operations captured in y are carried out on the same BO (lines 11–17). If the operations relate to more than one BO, the graph gets removed from set Z, which initially is assigned to contain all the graphs in Y. The BO mapping is achieved by evaluating each database table t associated with operations of graph y and mapping t to the BOs that are characterized by t. If an operation, or several operations, of y relates to database tables that characterize more than one BO, then y gets removed from Z. At the end of the third step of Algorithm 2, set Z is composed of all the functions that operate on a single BO, and this set is returned on line 18. Finally, the functions in set Z get recommended to the user as possible MSs.

4 Implementation and Validation

A proper MS should provide high execution efficiency with a desirable level of scalability and availability. Furthermore the packages and components in it should be highly cohesive and loosely coupled [1,16]. In order to validate our MS discovery and recommendation process provides MSs with the desirable characteristics, we developed a prototype[4] based on the algorithm presented in Sect. 3.2

[4] https://github.com/AnuruddhaDeAlwis/Subtype.git.

and experimented on SugarCRM[5] and ChurchCRM[6] which is detailed in our technical report [17].

This section only presents the details of the experiments that we conducted using the prototype on SugarCRM, which is a customer relationship management system that has a complex system structure with more than 8,000 source files, 600 attributes divided between 101 tables. We specifically focused on the campaign management module of SugarCRM to generate the execution sequences for our microservice discovery process. In order to cover all the user cases related to the campaign management module, 10 different executions[7] related to the campaign management, such as target creation, campaign creation, and template creation, were performed and their log data was generated using the SugarCRM system's log functionality. The logs were then analyzed using the process mining tool Disco[8] and 10 different call graphs were generated, all together containing around 200 unique execution nodes. The generated call graphs and the database tables were provided as the input to the prototype.

Discovered MSs: Based on the call graphs and database tables, the prototype identified three subtypes of campaigns, namely newsletter, email, and non-email, which results in functional splitting of the ES based on object subtypes (satisfying Heuristic 1). In addition, the prototype identified common sequences related to all the executions resulting in functional splitting of the ES based on execution fragments (satisfying Heuristic 2).

Validation Process: The validation process was conducted by implementing the recommended MSs in Google Cloud. Each MS was hosted in Google Cloud using a cluster of size 2 which has two virtual CPUs and a total memory of 7.5 GB. The hosted MSs were exposed through the Google Cloud kubernetes API, allowing third party computers to access them via API calls. In order to validate the sub-typing recommendations, we implemented three MSs simulating newsletter, email, and non-email campaigns, and another system to simulate the legacy campaign module which covered all the campaign sub-types. In addition, we implemented a MS with common segments, i.e., fragments with similar states, communicating with other MSs simulating the common subgraphs recommendations given by the framework. Each MS was tested against a load of 150,000 requests and 300,000 requests generated by 10 machines simultaneously, simulating the customer requests, while recording their total execution time, average memory consumption and average disk consumption. The results are shown in Tables 1 and 2.

Based on the results reported in Tables 1 and 2, we calculated the scalability, availability, and execution efficiency of different combinations and the results obtained are summarized in Tables 3 and 4. The scalability was calculated according to the resources usage over time as described by Tsai *et al.* [13].

[5] https://www.sugarcrm.com/.

[6] http://churchcrm.io/.

[7] http://support.sugarcrm.com/Documentation/Sugar_Versions/8.0/Pro/Application_Guide/.

[8] https://fluxicon.com/disco/.

In order to determine the availability, first we calculated the time taken to process 100 requests if a particular MS is not available. Then, we used the difference between the total up-time and total down-time as described by Bauer *et al.* [14]. Efficiency gain was calculated by dividing the time taken by the legacy system to process all requests by the time taken by each MS. Furthermore we calculated the structural cohesion and coupling of the packages in legacy system and the new MS systems as described by Candela *et al.* [4].

Table 1. Legacy system vs subtype MSs execution results.

Campaign type	No. of requests	Ex. time (ms)	Avg mem (GB)	Avg disk (GB)
Legacy	150,000	324,000	3.00375	2.09550
Legacy	300,000	741,600	3.04025	2.10050
Newsletter	150,000	201,600	2.95475	2.09150
Newsletter	300,000	396,000	3.00575	2.09975
Email	150,000	198,000	2.89075	2.09225
Email	300,000	446,400	2.97075	2.10125
Non-email	150,000	226,800	2.84550	2.09300
Non-email	300,000	432,000	2.92875	2.10125

Table 2. Legacy system vs common subgraphs MSs execution results.

System type	No. of requests	Ex. time (ms)	Avg mem (GB)	Avg disk (GB)
Legacy	150,000	399,600	3.0335	2.0915
Legacy	300,000	781,200	3.1665	2.1020
Common Seg.	150,000	194,400	2.9110	2.0926
Common Seg.	300,000	396,000	2.9905	2.1015

Table 3. Scalability, availability, and efficiency gains using subtyping.

Campaign type	Scalability [Mem]	Scalability [Disk]	Availability [150,000]	Availability [300,000]	Efficiency [150,000]	Efficiency [300,000]
Legacy	2.652	2.626	99.856	99.918	1.000	1.000
Newsletter	1.963	1.937	99.910	99.956	1.607	1.873
Email	2.612	2.552	99.912	99.950	1.636	1.661
Non-email	1.867	1.821	99.899	99.952	1.429	1.717

Table 4. Scalability, availability, and efficiency gains using common subgraphs.

Campaign type	Scalability [Mem]	Scalability [Disk]	Availability [150,000]	Availability [300,000]	Efficiency [150,000]	Efficiency [300,000]
Legacy	1.9947	1.9205	99.9334	99.9667	1.0000	1.0000
Common MS	2.1314	2.0839	99.9334	99.9667	2.0556	1.9727

Table 5. Comparison of lack of cohesion and structural coupling.

System type	Lack of cohesion	Structural coupling
Legacy with campaign packages	104.00	17.00
MS with campaign packages	92.00	15.89
Legacy with commonality packages	55.83	18.00
MS with commonality packages	50.67	18.50

Experimental Results: According to Tsai *et al.* [13], the lower the number the better the scalability. Thus, the newsletter and non-email MSs have better scalability than the legacy system when considering both memory and disk usage over time (refer to Table 3). In the email MS there is a scalability gain, even thought it is not as significant as that of the gain in the newsletter and non-email MSs. When considering availability we clearly observe that there is higher availability in subtype MSs than in the legacy system. As the number of requests increased from 150,000 to 300,000, subtype MSs were able to handle the request overload while providing better availability than the legacy system. Most importantly, when examining the request processing efficiency, each subtyping MS managed to process the request at at-least 1.5 times the speed of the legacy system.

Table 4 reports that there is not much of a gain in scalability and availability in the MS discovered and developed based on Heuristic 2 when compared with the legacy system. In contrast, when comparing the efficiency gain, it is evident that the common MS managed to process requests at at-least twice the accelerated speed of the legacy system. Furthermore, when comparing the coupling and cohesion values detailed in Table 5, it is evident that both campaign and common MSs attained a higher level of cohesion than the legacy system. In addition, the campaign MS managed to achieve slightly better coupling when compared with the legacy system even though there is a small increase in coupling in the common MS. Similar results were obtained for the experiments conducted on ChurchCRM's service management module [17].

Provided Solutions: The obtained results have affirmed that MSs extracted based on the recommendation of our prototype can provide the same services to the users while preserving overall system behaviour and achieving higher scalability, availability, efficiency, high cohesion, and low coupling.

5 Related Work

Microservices have emerged as the latest style of service-based software allowing systems to be distributed through the cloud as fine-grained components, typically with individual operations, in contrast to services under SOA which include all logically related operations [1]. Even though microservices can support the evolution of ERP systems by providing exploitation in cloud-enabled platforms such as the IoT [2], the research conducted in this particular area is limited. To the best of our knowledge there is no research related to the automation of MS discovery in legacy systems, apart from the manual migrations achieved by Balalaie *et al.* [15]. Balalaie *et al.* have described the complexity associated with the system reengineering process while pointing out the importance of considering BOs and their relationships in the migration system process. Martin Fowler emphasizes the importance of adapting BO relationships in microservices [16] aligning with the Domain Driven Design principles.

However, the existing software re-engineering techniques do not consider the complex relationship of BOs with their behaviours in the re-engineering process. Furthermore, studies show that the success rate of existing software remodularisation techniques, especially for large systems, remains low [4]. A key stumbling block is the limited insights available from syntactic structures of software code for profiling software dependencies and evaluating their measurements for coupling and cohesion metrics [5]. As such, to derive successful re-engineering techniques, a methodology should consider the enriched semantic insights available through the BOs and functions in an ES.

In such a process, the first challenge would be identifying the BOs which are distributed among several database tables in an ES system, and identifying the relationships between them. Nooijen *et al.* [12] and Lu *et al.* [8] proposed methodologies and heuristics to identify BOs based on the database schema and information in database tables. However, according to Lu *et al.*, the derived BOs might not be perfect and they have to be reclustered with the help of human expertise. A proper identification of BO relationships should consider the behavioural aspects of the systems as described by Hull [11]. However, there is still a gap in the area of correlating such behaviour with the underline BOs. As such, it is important to establish novel methodologies which incorporate both system behaviours and the business objects in the software re-engineering process.

6 Conclusion

This paper presented two heuristics used for functional splitting of ESs based on object subtypes and common execution fragments, while providing ground rules for MS discovery. A prototype was developed based on the proposed heuristics and validation was conducted by implementing the MSs recommended by the prototype for SugarCRM and ChurchCRM. The study has demonstrated that analysis of functions and BO CRUD operations while evaluating BO relationships helps to identify efficient solutions to migrate legacy systems into MSs with high cohesion and low coupling while achieving better scalability, availability, and execution efficiency. However, further analysis of BO relationships, such as inclusive and exclusive containment should be considered to further optimize the MS discovery process, and this will be carried out as future work.

References

1. Newman, S.: Building MSs NGINX. O'Reilly (2015)
2. Columbus L.: Internet Of Things (IoT) Intelligence Update (2017). https://www.forbes.com/sites/louiscolumbus/2017/11/12/2017-internet-of-things-iot-intelligence-update/43aa6f4c7f31
3. Magal, S.R., Word, J.: Integrated Business Processes with ERP Systems. Wiley, Hoboken (2011)
4. Candela, I., Bavota, G., Russo, B., Oliveto, R.: Using cohesion and coupling for software remodularization: is it enough? ACM Trans. Softw. Eng. Methodol. (TOSEM) **25**(3), 24 (2016)
5. Anquetil, N., Laval, J.: Legacy software restructuring: analyzing a concrete case. In: 2011 15th European Conference on Software Maintenance and Reengineering (CSMR), pp. 279–286. IEEE, March 2011
6. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71289-3_20
7. Pérez Castillo, R., García Rodríguez de Guzmán, I., Caballero, I., Piattini, M.: Software modernization by recovering web services from legacy databases. J. Softw. Evol. Process. **25**(5), 507–533 (2013)
8. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
9. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 25–41. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_2
10. Halpin, T., Morgan, T.: Information modeling and relational databases. Morgan Kaufmann, Burlington (2010)
11. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_17
12. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 316–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_36
13. Tsai, W.T., Huang, Y., Shao, Q.: Testing the scalability of SaaS applications. In: 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–4. IEEE, December 2011
14. Bauer, E., Adams, R.: Reliability and Availability of Cloud Computing. Wiley, Hoboken (2012)

15. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Migrating to cloud-native architectures using microservices: an experience report. In: Celesti, A., Leitner, P. (eds.) ESOCC Workshops 2015. CCIS, vol. 567, pp. 201–215. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_15
16. Fowler, M.: Microservices a definition of this new architectural term (2014). https://martinfowler.com/articles/microservices.html
17. De Alwis, A., Barros, A., Polyvyanyy, A., Fidge, C.: Technical report: function-splitting heuristics for discovery of microservices in enterprise systems (2018). https://eprints.qut.edu.au/119030/

Services and Processes

High Performance Userspace Networking for Containerized Microservices

Xiaohui Luo, Fengyuan Ren(✉), and Tong Zhang

NNS Group, Tsinghua University, Beijing, China
{luo-xh17,zhangt14}@mails.tsinghua.edu.cn
renfy@tsinghua.edu.cn

Abstract. Containerized microservices have become popular for building systems using simple, lightweight, loosely coupled services. Due to replacing the monolithic application with multiple microservices, inner function calls become inter-microservice communications, which increases the network pressure. However, the networking of containerized microservice built on the kernel that is inefficient. In this paper, we propose DockNet, a high-performance userspace networking solution for containerized microservices. We (1) leverage DPDK and customized LwIP as the high-performance data plane and TCP/IP stack, respectively. (2) introduce a master-slave threading model to decouple execution and management. (3) adopt namespace mechanism to control the access of microservices to data planes and employ timer-based rate limiters to achieve performance isolation. (4) construct fast channels between partner microservices to improve network performance further. In our various experiments, DockNet shows over 4.2×, 4.3×, 5.5× of higher performance compared with existing networking solutions - kernel bridge, Open vSwitch and SR-IOV, respectively.

Keywords: High-performance network · Container · Microservice

1 Introduction

Recently, containerized microservices have redefined the software development landscape. Containerized microservices have been supported by leading cloud providers, such as Amazon EC2 Container Service [4], and Microsoft Azure Container Service [6]. Companies such as Netflix [3] and Uber [16], have replaced their large monolithic applications into multiple small containerized microservices that coordinate to provide required functionalities.

However, microservices increase the network pressure. Since a monolithic application is divided into multiple containerized microservices, inner function calls become inter-microservice communications (e.g., Remote Procedure Calls, and RESTful APIs), which imposes great pressure on the network. For example, over 99% of 5-billion API calls per day in Netflix are internal (but across microservices) [14]. In other words, the microservice architecture makes a trade-off between simplifying development and increasing network pressure.

Unfortunately, network in the container is inefficient. Containers are essentially processes running in separated and isolated runtime environments. The network of a container is based on the kernel's network stack. Recent researches have shown the performance limitations in the kernel's network stack [9,12,13], which means that these limitations also affect container's network performance. Among these researches, userspace high-performance network stacks have shown significant performance improvements [9,13].

In this paper, we motivate to introduce the userspace high-performance network stack to containerized microservices. We summarize the requirements as follows: *(i) High-Performance.* Although userspace network stacks can offer high-performance, they cannot be directly used by containerized microservices. Because the container's network interface is built upon the kernel, we must provide another specialized userspace interface for the container. Meanwhile, we also need to provide a userspace TCP/IP stack for these network services. *(ii) Easy Management.* Userspace network stacks bypass the kernel, we must implement management functionalities in the userspace network stack. *(iii) Isolation.* Multiple microservices running on the same host is common. We must provide isolation mechanisms for both data plane security and performance guarantee. *(iv) Fast Intra-host Communication.* In some scenarios, partner microservices (e.g., one for requests, the other for responses) locate in the same host. Without crossing the hosts, we should provide higher performance for intra-host communications.

We propose DockNet, a userspace high-performance networking stack for containerized microservices using commodity hardware. DockNet (1) leverages DPDK [5] and customized LwIP [10] as the high-performance data plane and TCP/IP stack, respectively. (2) introduces a master-slave threading model to decouple execution and management. In the master-slave threading model, the master thread is responsible for control and management. The slave threads (i.e., the container threads) are only responsible for processing packets. (3) uses the hardware-based flow director mechanism to virtualize a physical network interface card (NIC) into multiple lightweight network interfaces, which effectively supports multiple containers. For isolation, we adopt namespace mechanism to control the access of containers to data planes. We also employ timer-based rate limiters to achieve performance isolation. (4) constructs fast channels between partner microservices to improve network performance further.

In our evaluations, DockNet achieves high throughput and low latency for both intra- and inter-host traffics, effectively supports bandwidth isolation and multiple containerized microservices, and outperforms dominant networking solutions by more than $4.2\times$ in service-oriented cases. Finally, we build a web cluster based on DockNet to scale the web performance and demonstrate its almost linear scalability.

The rest of this paper is organized as follows. In Sect. 2, we summarize the existing container networking solutions and discuss the kernel inefficiency. In Sect. 3, we present the overview of DockNet, describe its design and implementation in detail. We validate and evaluate the performance of DockNet in Sect. 4. Finally, the whole paper is concluded in Sect. 5.

2 Background

In this section, we first introduce existing networking approaches for containerized microservices. Then we discuss the performance limitations in the kernel.

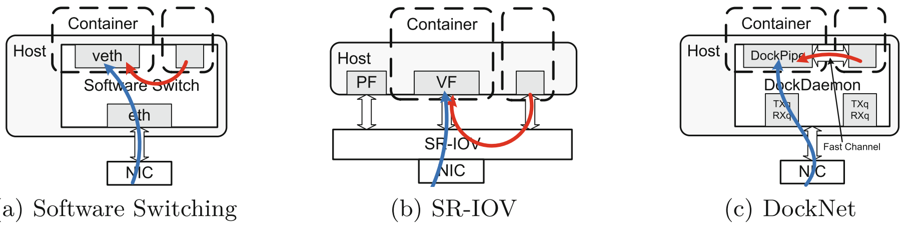


Fig. 1. Container networking architectures.

2.1 Existing Approaches

We survey the existing network approaches to build containerized microservices. Their fundamental techniques can be summarized as follows:

Software Switching: As Fig. 1(a) shows, the software switch is responsible for forwarding packets to the correct receivers. Considering that the packets have already arrived at the host, the extra forwarding operations are redundant. The extra operations waste CPU cycles and make the whole procedure heavyweight. Software switching is adopted by the mainstream container networking solutions, such as Linux bridge and Open vSwitch (OVS) [1].

SR-IOV: As shown in Fig. 1(b), with SR-IOV [2], a virtual function (VF) acts just like a real physical NIC. Packets can be directly be routed (by the hardware) to the correct receivers. SR-IOV eliminates the forwarding costs and provides a lightweight networking architecture. However, packets across different VFs needs to be switched in the NIC's embedded hardware switch. It means that in the intra-host traffic cases, packet forwarding requires both VFs and the NIC, which introduces two extra copies across the PCIe bus.

From the above, we can see that these existing networking solutions are mainly built upon the kernel's network stack that has performance limitations.

2.2 Kernel Inefficiency

Previous studies have exposed three main performance limitations of the kernel: inefficient packet processing, general resource sharing, overheads of context switching. (1) Per-packet memory (de)allocation and heavy data structures (e.g., `sk_buff`) limit the efficiency [12,13,17]. (2) Resources are shared in many folds: a multi-thread application shares only one listening sockets [12,15], sockets share file descriptors with the regular files [12]. (3) Frequent system calls result in increasing the overheads of context switching [18]. To overcome these limitations introduced by the kernel, many userspace approaches have been proposed.

High-Performance Packet I/O Frameworks: High-performance packet I/O frameworks, such as DPDK [5], PSIO [7], PF_RING [8] and netmap [17], are closely correlated with network interface cards (NICs). These frameworks share similar techniques to accelerate the packet processing: (1) Using memory pools to store packets can reduce the (de)allocating costs during processing packets. (2) Packets are sent/received in batches instead of one by one to amortize costs of system calls. (3) Polling is used for receiving packets rather than triggering interrupts. (4) Bypassing the kernel, data are transmitted between the framework and applications directly.

Userspace Networking Stack: Systems such as mTCP [13] and IX's data plane [9] run their entire networking stack in the userspace. Userspace stacks have the following benefits: (1) Without adverse effects of the complex kernel, userspace networking stacks can optimize packet processing and eliminate expensive system calls. (2) These approaches use lock-free data structures to scale on multicore systems. (3) Flow-consistent hashing of incoming traffic is used for pinning a flow to the same core. (4) Batching is extensively used to amortize system call transition overheads and improve instruction cache locality.

3 System Design and Implementation

Figure 1(c) sketches the overall architecture of DockNet. DockNet provides DockPipes for containerized microservices for high-performance packet I/O. DockDaemon is responsible for functionalities related to management and control, such as isolating DockPipes, setting flow directors to the hardware queues for directly forwarding, and constructing fast channels between specific microservices.

3.1 DockPipe

A DockPipe is essentially a lightweight networking interface for a container. Here we describe a DockPipe's data structure, attaching, lightweight data path, isolation, and fast channels.

Data Structure: Figure 2 shows that a DockPipe only consists of three types of ring buffers: a receiving ring buffer, a transmitting ring buffer, and some fast channel (FC) ring buffers. The receiving buffer and the transmitting buffer are directly pinned to a pair of hardware queues. An FC ring buffer is used to construct a fast channel. By using a fast channel, a container can directly put packets into the receiver's peer FC ring buffer.

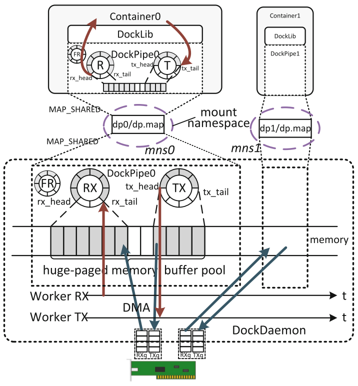


Fig. 2. Structure and isolation of a DockPipe.

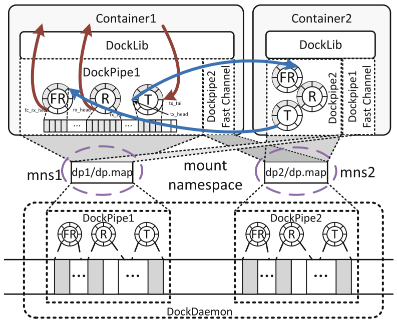


Fig. 3. Sharing of DockPipes and the structure of a fast channel.

DockPipe Attaching: The attaching of a DockPipe is controlled and managed by DockDaemon. As Fig. 2 shows, When creating a DockPipe, DockDaemon generates a mapping file which records the memory layout and offset of the DockPipe. The container attaches to the DockPipe by loading and remapping (using `mmap()`) the memory mapping file. After attaching, the container can directly send or receive packets through the DockPipe. Through a DockPipe, a containerized microservice is only allowed (1) to put packets into the buffer. (2) to read packets from the buffer. The DockDaemon is responsible for real packet moving, such as transmitting packets to the NIC and receiving packets from the NIC.

High-Performance Data Path: A DockPipe uses DPDK as the high-performance userspace packet I/O data plane. As shown in Fig. 2, a DockPipe is directly pinned to a pair of hardware queues. DockDaemon sets hardware-based filters (e.g., destination IP address) to the NIC. When packets arrive at the NIC, the pre-configured filters (e.g. destination IP address) will guide the NIC and redirect the packets to a specified hardware queue. Then, the packets are directly put into the corresponding DockPipe's receiving ring buffer through DMA.

DockPipe Isolation: When deploying multiple containerized microservices on a single host, the security concern is critical: A DockPipe should be protected and provides an isolated data plane for the granted container. Here we describe how to isolate multiple DockPipes.

Since a DockPipe is remapped to a container by `mmap()` based on memory sharing, we can conveniently isolate DockPipes by controlling their scopes of shared memory. In container-related techniques, the ***mount namespace*** offers this capability. The mount namespace of a process is essentially the set of mounted filesystems in its perspective. Conventionally, mounting or unmounting a filesystem will change all processes' scope because a global mount namespace can be seen by all processes. In the container context, we can utilize the per-process mount namespace. Then a process can have its exclusive scope of mounted filesystems. Based on the per-process mount namespace, DockNet provides a lightweight isolation for DockPipes.

Take Fig. 2 for example, DockDaemon (1) generates *DockPipe0's* memory mapping file as *dp0/dp.map*. (2) mounts file dictionary *dp0* (in mount namespace *mns0*) to *container0* by leveraging the mount namespace. In this way, *DockPipe0* can only be accessed by *container0* and is invisible to *container1*.

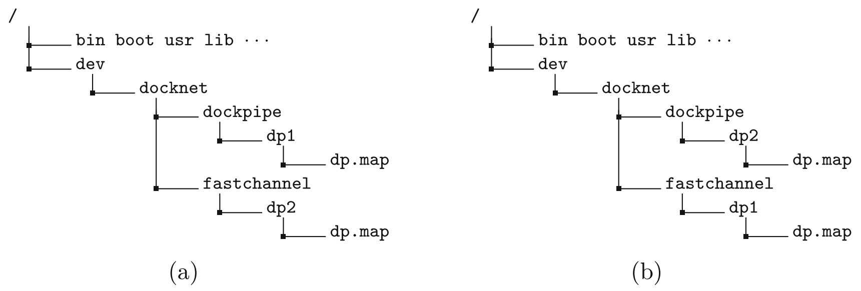


Fig. 4. Mounted filesystems (a) perspective of *container1*. (b) perspective of *container2*.

Fast Channel Between DockPipes: In some particular scenarios (such as private clouds) strict isolation is unnecessary. For example, partner containerized microservices belong to the same application and locate on the same host. In such scenarios, DockNet can offer higher throughput and lower latency for containerized microservices by constructing a fast channel.

We have explained how to use the mount namespace for controlling the scopes of shared memory. By default, a DockPipe's memory mapping file is mounted to a single container, namely a DockPipe is monopolized by a container. However, a DockPipe can also be remapped into two containers. In this way, a fast channel (FC) between two containers can be constructed.

As illustrated in Figs. 3 and 4, a DockPipe can be remapped as either a normal DockPipe or an FC, respectively. When a DockPipe is remapped as an FC, only its FC receiving ring buffer is used. When a packet is ready to be sent, the container first checks whether it can be sent through an FC. If so, the container sets the packet's pointer to the tail of the FC's receiving ring buffer. Otherwise, the container sends the packet to its DockPipe's transmitting ring buffer as usual. For instance, in Fig. 4, *DockPipe1* is remapped to *container1* as a normal DockPipe, and remapped to *container2* as an FC. When *container2* sends packets to *container1*, the packets' pointers are directly put into the tail of *DockPipe1*'s FC receiving ring buffer. Then, *container1* can directly consume the packets from the head of *DockPipe1*'s FC receiving ring buffer. For *DockPipe2*, the receiving process is the same except for exchanging *container1* and *container2* in sender/receiver locations.

3.2 DockDaemon

DockDaemon offers management and control functionalities. The management functionalities include assigning hardware queues to DockPipes, setting flow directors for directly forwarding, isolate DockPipes, and (de)constructing fast channels. Their technical essences have been described in Sect. 3.1.

In this section, we focus on the how to control DockPipes in DockDaemon.

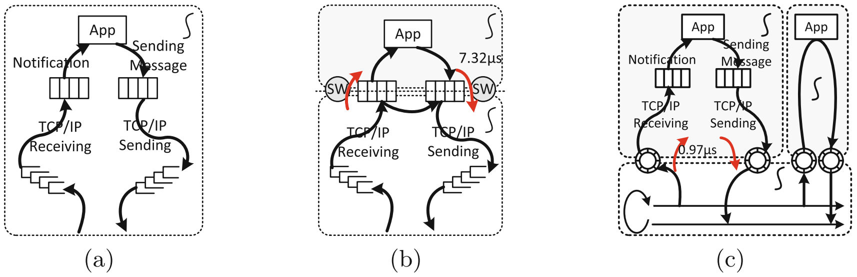


Fig. 5. (a) shows the routine of the run-to-completion execution model. (b) shows that of the multi-stage wheels execution model. (c) shows DockNet's master-slave execution model.

Threading Model: Threading model greatly affects the system performance. Here we summarize the two existing models and propose the master-slave model.

(i) *Run-to-completion model:* As shown in Fig. 5(a), this model integrates packet receiving/transmitting, protocol processing, and application logic into a single executing routine. Due to eliminating unnecessary intermediate stages (e.g. context switches), this model can achieve low latency. However, this model requires privileged access to the physical NIC, which may result in security risks and increase the managing and controlling difficulty. IX [9] adopts this threading model, and its data plane optimizes for both bandwidth and latency.

(ii) *Multi-stage wheels model:* As Fig. 5(b) shows, in this model, each pair of threads are pinned to the same CPU core. One thread is responsible for processing packets in the application. The other thread is devoted to receiving and sending packets. The decoupling can protect the physical NIC from being directly accessed by the containers. However, the decoupling introduces costs of thread context switches (the red arrow in Fig. 5(b)) and incurs higher latency. This model is adopted by mTCP [13], which is specialized for handling short messages on multicore systems. Besides that, the similar threading model is also applied to the kernel's stack.

(iii) *Master-slave model:* Inspired by the two models above, we design a master-slave model for containerized microservice networking. As shown in Fig. 5(c), in this model, there is one master thread and many slave threads.

The slave thread acts like the run-to-completion model and integrates all related processing into a single executing routine. The master thread is devoted to really moving packets, including sending packets to the physical NIC, receiving packets from the physical NIC, and copying packets between containers. The master and slave threads decouple the whole packet routine into two sub-stages. Because all the packets moving is done by the master thread, it simplifies the control and management, such as limiting the bandwidth. The decoupling also introduces extra overheads. To avoid high costs caused by thread switching, we let the master thread monopolize a core. The slave threads use the other cores located on the same CPU socket. In Fig. 5(c), the two red arrows mean the synchronization and mutual exclusion that based on atomic operations.

By using the master-slave model, a DockPipe is only allowed to put packets into or get packets from its ring buffer. DockDaemon controls all the packets sending and receiving. The master-slave model can both achieve low latency and simplify the controlling.

Timer-Based Rate Limiter: To achieve bandwidth isolation, we design a simple but efficient mechanism based on timer-based rate limiter in DockDaemon. Two timers are introduced. One is fine-grained T_{fine}, and the other is coarse-grained T_{coarse}. They are triggered at different intervals. The fine-grained timer produces tokens for each DockPipe, and the number of tokens constrains how many bytes a DockPipe can send or receive. However, since a container may keep silent for an extended period, it likely accumulates much more tokens than others. Then it may consume too much bandwidth when it becomes active. To solve this problem, the coarse-grained timer periodically clears all the tokens and sets an initial number of tokens intermittently.

3.3 APIs

DockNet provides two types of APIs: DockDaemon commands and DockLib. DockDaemon commands are received by DockDaemon through a kernel-based listening socket. These commands are used for managing the DockDaemon, such as creating or terminating a DockPipe, assigning a DockPipe to a containerized microservice, limiting a DockPipe's bandwidth. DockLib is used for containerized microservices to access the DockPipe and develop applications. DockLib contains a packet engine that interacts with DockPipe and a lightweight userspace TCP/IP stack based on LwIP [10]. Figure 7 presents the skeleton of using DockNet to write a TCP server containerized microservice.

Packet Engine: DockNet uses DPDK as the high-performance packet I/O data plane. After remapping, a DockPipe offers its transmitting and receiving queue for a container. Packet engine mainly includes two components: the sending component and the receiving component. In the sending component, when packets from the stack are sent to DockPipe, the engine first checks their destination IP address. (1) If a packet's destination IP matches a fast channel, the packet is

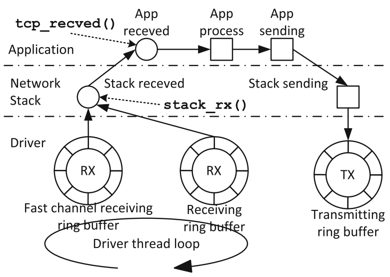


Fig. 6. Packets processing routine. Circle means a callback function that needs to be registered by the upper level code. Box means a normal function call.

```
err_t app_process(struct tcp_pcb *tcb, struct mbuf *m)
{
  // the processing details
  tcp_sent(tcb, on_sent);
  tcp_write("OK!")
}

err_t on_accecpted(struct tcp_pcb *tcb)
{
  tcp_recv(tcb, app_process);
}

int main(int argc, char **argv)
{
  struct dockpipe *dp = dockpipe_attach("/dev/docknet/dockpipe/dp0",
       ethernet_input);
  tcp_pcb tcb = tcp_new();
  tcb = tcp_listen(tcb);
  err_t err = tcp_bind(tcb, IP_ADDR_ANY, 80);
  tcp_accept(tcb, on_accecpted);
  sys_thread_new("dockpipe_driver_thread",
    dockpipe_driver_thread, NULL,
    DEFAULT_THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);
  pause();
  return 0;
}
```

Fig. 7. Skeleton of a TCP example.

directly appended to the remote peer's FC receiving ring buffer. (2) If a packet's destination IP matches no fast channels, the packet is appended to the DockPipe's transmitting ring buffer. In the receiving component, when receiving packets, the engine first tries to receive packets from its FC receiving ring buffer, then goes to receive packets from its receiving ring buffer.

As shown in Fig. 6, the packet engine exposes a function pointer `stack_rx()`, through which the received packets can be sent to the upper network stack. The packet engine thread repeatedly checks the two receiving ring buffers. After receiving a packet, the engine delivers it to the stack through the callback function.

Userspace TCP/IP Stack Library: Since DockPipe's ring buffers are all located in userspace, DockNet needs to provide a userspace TCP/IP stack for containerized microservices. In DockNet, we port the stack from LwIP. We mainly modify its data structures to achieve zero-copy and be adapted to DockNet. Here we briefly describe the key procedures of using DockNet to write a TCP server code in a container. When the server starts, it first executes initializations. Then the server registers `stack_rx()` callback function, which is a function pointer inside DockPipe. Next, the server registers `tcp_recved()` callback function, which is a function pointer in DockLib. Finally, the server launches a new packet engine thread and runs it, then blocks the main thread. Both `stack_rx()` and `tcp_recved()` will be called in the engine's thread context, which avoids introducing context switching overhead.

4 Evaluation

In this section, we evaluate DockNet with the following goals:

(i) Demonstrate that DockNet can provide high throughput and low latency in both inter-host and intra-host communications.
(ii) Validate DockNet's ability to guarantee performance isolation among microservices.
(iii) Show that DockNet can support multiple containerized microservices.
(iv) Construct a web cluster based on containerized microservices to evaluate the integrated performance of our DockNet.

Testbed: Our testbed consists of two machines, and each has two 6-core CPUs (Intel Xeon E5-2620 v3 @ 2.4 GHz), 64 GB RAM, and an Intel NIC with two 10GbE ports. The NICs of the two machines are connected directly. By default, we only use one 6-core CPU and one NIC port in the following experiments. Our DockNet runs on Linux 3.10.0. TSO is turned on in kernel-based evaluations. Each container is pinned to one single core.

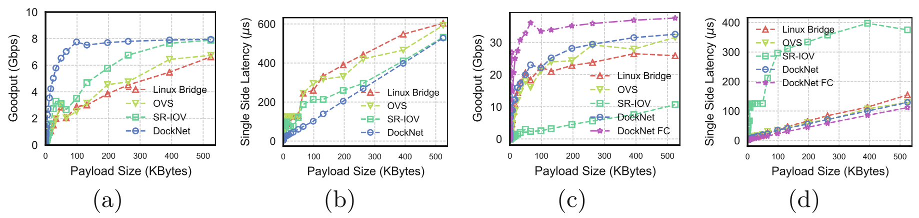


Fig. 8. The basic performance benchmark of DockNet. In (a) and (b), two microservices are deployed in two hosts. In (c) and (d), two microservices are deployed in the same host. (a) and (c) show the goodput. (b) and (d) show the singe side latency.

4.1 Throughput and Latency

High throughput and low latency are critical requirements for containerized microservices.

We construct a pair of TCP microservices, one being the server and the other being the client, and deploy each microservice in a container. The two applications directly exchange a fixed-size message back and forth. Two primary metrics – goodput and one-way latency are measured. The reference networking solutions include Linux bridge, OVS, SR-IOV, and DockNet. Figure 8 shows the two primary performance metrics of four networking solutions under different message sizes.

In Fig. 8(a) and (b), the two microservices (server and client) are deployed in two different physical hosts. In this case, DockNet shows both the highest goodput and the lowest one-way latency. Obviously, when the message size grows to

100 KB, the link utilization of DockNet reaches 8 Gbps (i.e. line rate), while the other three are still below 8 Gbps even when the message size equals 400 KB. From the perspective of latency, DockNet also has lower one-way latency than the other three solutions under all message sizes. It is because kernel bridge, OVS, and SR-IOV provide kernel-based interfaces, which inevitably introduces overheads, such as context switching and data copying between kernel and userspace. Besides, kernel bridge and OVS also introduce the software switching cost. Instead, DockNet leverages a user-level DockPipe to exchange data, thus eliminates both data copying and context switching.

In Fig. 8(c) and (d), the two microservices are deployed in the same physical host. SR-IOV shows inferior performance. Goodput is always below 8 Gbps, and one-way latency reaches up to 400 µs, which is far worse than other solutions. The reason is that SR-IOV has to switch packets through the physical NIC, and introduces the transmission cost over PCIe compared with other solutions. The other four networking solutions all achieve superior goodput and latency, and our DockNet and DockNet with FC achieve further better performance than kernel bridge and OVS. We can see that, as the message size increases to 200 KB, the goodput, of all solutions except for SR-IOV exceeds 22 Gbps, our DockNet reaches over 27 Gbps, and DockNet FC achieves more than 35 Gbps. Both Kernel bridge and OVS forward packets in the kernel, hence introduce data copying and context switching overheads. However, DockNet based on the forwarding table eliminates the context switching, and DockNet with FC removes both context switching and data copying, therefore outperforms than other networking.

In summary, SR-IOV performs poorly in one-host scenarios but performs well in two-host scenarios. Kernel bridge and OVS have good performance in the one-host condition but still perform poorly in two-host tests. DockNet shows excellent performance in both one-host and two-host tests.

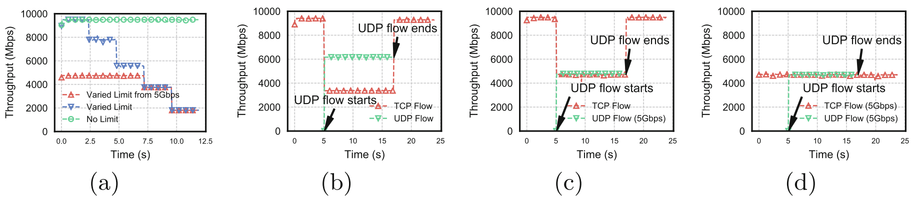


Fig. 9. Rate limiter tests. (a) shows the ability to dynamically change the rate limiter. (b), (c) and (d) shows different policies on two DockPipes.

4.2 Bandwidth Isolation

As stated above, bandwidth isolation is necessary for containerized microservices to avoid aggressive microservices consuming too much bandwidth. Next, we demonstrate DockNet's ability to limit microservices' maximal bandwidth. We construct a pair of TCP client/server microservices as well as a pair of UDP client/server microservices. The four microservices are deployed in four different containers, respectively. Two server and clients are arranged in one host, respectively. During the test, two servers keep sending packets to their corresponding receivers by a TCP flow and a UDP flow separately. Figure 9 depicts the bandwidth usage under different rate limitations.

Figure 9(a) presents the results of two tests: (1) Without any rate limiters, the throughput of the only TCP flow is evaluated as the baseline. The TCP throughput is depicted as the blue (delta dotted) line. (2) After starting containers, periodically change the rate of the limiter according to the value list {8000, 6000, 4000, 2000} (Mbps). The TCP throughput is described as the orange (square dotted) line. Clearly, with the rate limiter, the flow throughput rapidly changes in line with the rate limits, which validates the effectiveness of the rate limiter.

In the latter three subgraphs in Fig. 9, we first run the TCP flow, then start the UDP flow at the 5th second and terminate it at the 17th second. Figure 9(b) manifests the results without rate limiters, indicating that the UDP flow consumes the majority of bandwidth and nearly beats down the TCP flow's throughput. In Fig. 9(c), only the UDP flow's rate limiter is set to be 5 Gbps, in which case the TCP flow can still occupy half of the whole bandwidth. In Fig. 9(d), we set all DockPipes' rate limiters to be 5 Gbps, and the result demonstrates that the bandwidths of flows are appropriately isolated.

From above experiments, we show that DockNet can indeed limit the maximum bandwidth of a microservice and further enable the bandwidth isolation, in both single-flow and multi-flow scenarios.

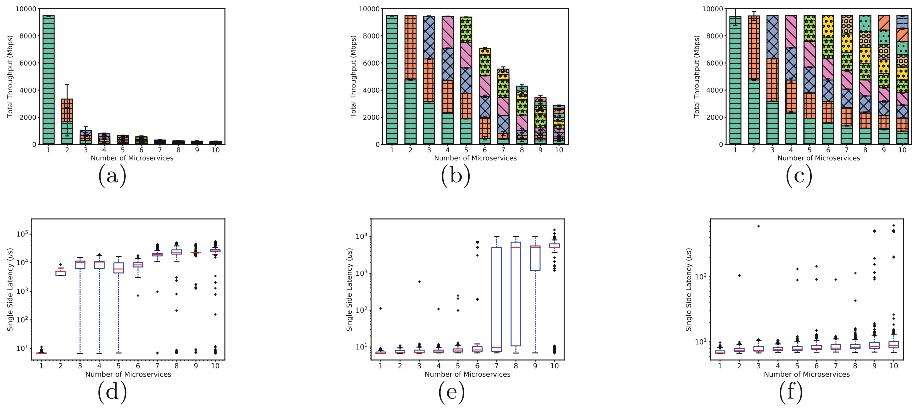


Fig. 10. Multiple microservices tests. (a) and (d) describes the total throughput and latency results when microservices runs on one slave core. (b) and (e) describes the results when microservices runs on one socket (5 slave cores). (c) and (f) describes the results when microservices runs on two sockets (10 slave cores).

4.3 Multi-container

It is common that multiple microservices are deployed on one host, in which case the underlying networking has to support multiple containers. In this subsection, we investigate DockNet's performance when supporting multiple containerized microservices.

In Fig. 10, we deploy multiple microservices in two physical hosts: one as the server side and the other as the client side. Each microservice is assigned a DockPipe, and all DockPipes are limited to the same rate by rate limiters according to the number of containers. DockDaemon dedicates one master core, while microservices are deployed on slave cores. When measuring the throughput, each client keeps sending 1024B message to a server. In one-way latency tests, each pair of server and client keeps exchanging a 1024B message back and forth. In our experiment, microservices are respectively deployed on one slave core, one socket (5 slave cores) and two sockets (10 slave cores) to observe DockNet performance with the different number of cores. In the one-core and one-socket cases, DockDaemon uses one core as the master core, while in the two-socket case, two cores are used as master cores (each NUMA-node owns a master core).

Figure 10(a) and (d) show the throughput and one-way latency when microservices are deployed on one core, respectively. Clearly, the overall throughput sharply decreases as the number running microservices grows. Similarly, the one-way latency drastically increases from 6.4 µs to 7 ms as the number of containers grows from 1 to 2. Such performance degradation is mainly caused by the container scheduling. Worse still, both the server and client will produce such scheduling costs. Because the time slice is ranging from 3 ms to 24 ms in our testbeds, such scheduling overheads are considerable and cause drastic performance loss.

In Fig. 10(b) and (e), microservices are deployed on one socket (5 slave cores). When the number of microservices is under 5, the total throughput always keeps at about 9.4 Gbps, and each microservice's latency maintains at about 6.4 µs. However, when the number of microservices is larger than 5, the scheduling cost is introduced. Take the 6-microservice case for example. The slave cores are $core_i$ (i = 1...5), two microservices are deployed on $core_1$, and the other four microservices are deployed on the rest four slave cores, respectively. From their bandwidth usage, we can see that the scheduling cost only occurs on $core_1$, because there exists scheduling between two microservices on $core_1$.

In Fig. 10(c) and (f), microservices are deployed on two sockets (10 slave cores). DockDaemon employs two master cores located in two NUMA nodes. It is shown that in the condition of no scheduling, the total throughput is always 9.4 Gbps, and all the microservices achieve low latency (about 8.7 µs).

As our benchmark microservices are just receivers and echo servers, their cost of processing a packet is at most 54.83 ns (only including consume the packet in the application logic, excluding the TCP/IP processing). Nonetheless, real workloads can spend more CPU cycles on processing received messages, which relieves the I/O pressure. In these experiments, DockNet can properly work in the actively stressful condition where containers almost do not spend any cycles on data processing. The experimental results in this subsection confirm that DockNet is lightweight and hardly becomes the bottleneck in real workloads.

4.4 Application

In this subsection, we study the performance of DockNet used in real containerized microservices. As shown in Fig. 11(a), we assume that the web server's processing power is the bottleneck, so we can scale the web cluster's processing power by simply adding more web servers. The DockNet holds five microservices. One acts as a NAT-based load balancer. The other four microservices all construct fast channels with the NAT container.

Network Address Translation (NAT): The NAT-based load balancer provides the ability that there is no need to change the client's behaviors because it can alter the destination IP into inner IPs. It introduces overheads because of modifying packets and searching lookup tables. To measure the overheads, we deploy a NAT microservice and a TCP server microservice on the server host. The client runs on another host. The server and the client behave similarly to those tests in Sect. 4.1. In the NAT-enabled test, the client sends packets with 1-Byte payload to the NAT, then the NAT transfers their destination IP addresses and forwards them to the server. When the server sends packets to the NAT, the NAT transfers the source IP addresses and sends them to the client. The results shown in Fig. 11(b) indicate that NAT microservice only add about 1.05 μs to the single side latency.

Simple Web Server: We port `httpd` to DockNet as the simple web server microservice and measure its performance. To avoid disturbance from disk access, we use `makefsdata` to transfer HTML files to C arrays. The HTML-content C codes can be directly compiled to the server. In this way, after processing HTTP request, the server can directly find the content of files located in the C arrays. One `httpd` microservice is deployed on the server host. On the client host, we use `wrk` [11] to saturate the `httpd` server. The results presented in Fig. 11(c) show that `httpd` running on DockNet can achieve the perfect performance. DockNet improves 5.2×, 6.3×, 5.5× performance gain compared with kernel bridge, OVS, and SR-IOV, respectively.

Web Cluster: To scalable the performance of processing HTTP requests, we use NAT as the load balancer to distribute requests to multiple `httpd` containers. We deploy a NAT microservice and multiple `httpd` microservices on the server host, and add some computing code to mimic processing cost (about 10 μs). `wrk` is deployed on the client host to saturate the `httpd` servers. The performance of the web cluster employing DockNet is shown in Fig. 11(d). It can be seen that the total requests per second scale when deploying more `httpd` containers. The performance scales 1.92×, 2.61×, 3.46× when deploying 2, 3, 4 `httpd` microservices comparing to a single `httpd` microservices. It validates that DockNet can be easily adopted in the real containerized microservices.

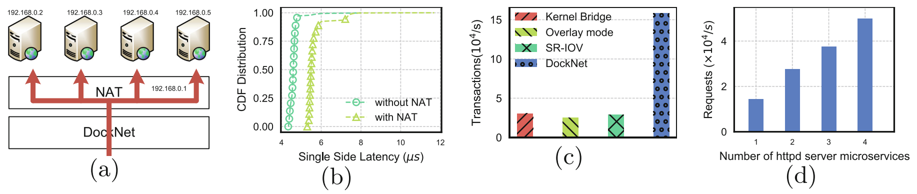


Fig. 11. Real applications tests. (a) shows the configuration of the tests. (b) shows the latency introduced by NAT. (c) shows the `httpd`'s transaction performance. (d) shows the DockNet can scale the performance of the web-cluster.

5 Conclusion

DockNet proposes a novel high-performance userspace networking solution for containerized microservices. DockNet splits the receiving/transmitting routine into two sub-stages and proposes a master-slave execution model to achieve both high performance and easy management, and provides a light-weight namespace-based access control mechanism to isolate microservices. Meanwhile, microservices mutually trusted can construct fast channels to achieve higher performance. In experiments, we verify that DockNet can provide high throughput and low latency in both intra-host and inter-host traffics, effectively support bandwidth isolation and multiple microservices. In transactions testing, DockNet shows over 4.2−52.4× of higher performance compared with existing conventional networking solutions. Finally, we build a DockNet-based web cluster and demonstrate its almost linear scalability.

Acknowledgments. The authors gratefully acknowledge the anonymous reviewers for their constructive comments. This work is supported in part by Suzhou-Tsinghua Special Project for Leading Innovation.

References

1. Open vSwitch (2009). http://openvswitch.org/. Accessed 2 June 2018
2. An Introduction to SR-IOV Technology (2011). http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html. Accessed 2 June 2018
3. Netflix: A Microscope on Microservices (2015). http://techblog.netflix.com/2015/02/a-microscope-on-microservices.html. Accessed 2 June 2018
4. Amazon EC2 Container Service (2018). https://aws.amazon.com/ecs/. Accessed 2 June 2018
5. Intel DPDK: Data Plane Development Kit (2018). http://dpdk.org/. Accessed 2 June 2018
6. Microsoft Azure Container Service (2018). https://azure.microsoft.com/en-us/services/container-service/. Accessed 2 June 2018
7. Packet i/o engine (2018). http://shader.kaist.edu/packetshader/io_engine. Accessed 2 June 2018

8. PF_RING (2018). http://www.ntop.org/products/packet-capture/pf_ring/. Accessed 2 June 2018
9. Belay, A., Prekas, G., Klimovic, A., Grossman, S., Kozyrakis, C., Bugnion, E.: Ix: a protected dataplane operating system for high throughput and low latency. In: OSDI 2014, No. EPFL-CONF-201671 (2014)
10. Dunkels, A.: lwIP - A Lightweight TCP/IP stack (2002). https://github.com/vadimsu/ipaugenblick. Accessed 2 June 2018
11. Glozer, W.: wrk- A Modern HTTP Benchmarking Tool (2012). https://github.com/wg/wrk. Accessed 2 June 2018
12. Han, S., Marshall, S., Chun, B.G., Ratnasamy, S.: Megapipe: a new programming interface for scalable network i/o. In: Presented as Part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), pp. 135–148. Hollywood, CA (2012)
13. Jeong, E., et al.: mTCP: a highly scalable user-level TCP stack for multicore systems. In: NSDI 2014, pp. 489–502 (2014)
14. Musser, J.: KPIs for APIs (2014). http://www.slideshare.net/jmusser/kpis-for-apis. Accessed 2 June 2018
15. Pesterev, A., Strauss, J., Zeldovich, N., Morris, R.T.: Improving network connection locality on multicore systems. In: Proceedings of the 7th ACM European Conference on Computer Systems. EuroSys 2012, pp. 337–350 (2012)
16. Reinhold, E.: Rewriting Uber engineering: the opportunities microservices provide (2016). https://eng.uber.com/building-tincup/. Accessed 2 June 2018
17. Rizzo, L.: Netmap: a novel framework for fast packet i/o. In: 2012 USENIX Annual Technical Conference (USENIX ATC 12), pp. 101–112. Boston, MA, June 2012
18. Soares, L., Stumm, M.: Flexsc: Flexible system call scheduling with exception-less system calls. In: Proceedings of the 9th USENIX Conference On Operating Systems Design and Implementation, pp. 33–46. USENIX Association (2010)

Guiding Architectural Decision Making on Quality Aspects in Microservice APIs

Uwe Zdun[1](✉), Mirko Stocker[2], Olaf Zimmermann[2], Cesare Pautasso[3], and Daniel Lübke[4]

[1] Faculty of Computer Science, Research Group Software Architecture, University of Vienna, Vienna, Austria
uwe.zdun@univie.ac.at

[2] University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland
{mirko.stocker,olaf.zimmermann}@hsr.ch

[3] Software Institute, Faculty of Informatics, USI, Lugano, Switzerland
cesare.pautasso@usi.ch

[4] innoQ Schweiz GmbH, Cham, Switzerland
ich@daniel-luebke.de

Abstract. Microservice APIs represent the client perspective on microservice-based software architecture design and related practices. Major issues in API design concern the quality aspects of the API. However, it is not well understood today what the established practices related to those quality aspects are, how these practices are related, and what the major decision drivers are. This leads to great uncertainty in the design process. In this paper, we report on a qualitative, in-depth study of 31 widely used APIs plus 24 API specifications, standards, and technologies. In our study we identified six recurring architectural design decisions in two API design contexts with a total of 40 decision options and a total of 47 decision drivers. We modelled our findings in a formal, reusable architectural decision model. We measured the uncertainty in the resulting design space with and without use of our model, and found that a substantial uncertainty reduction can be potentially achieved by applying our model.

1 Introduction

Many approaches have been proposed for designing service-based architectures (see e.g. [15,17,23]). A recent approach which evolved from established practices in service-oriented architectures are *microservices* [14,22]. The microservices approach emphasizes business capability- and domain-driven design, service development in independent teams, cloud-native technologies/architectures, polyglot persistence, lightweight containers, and a continuous DevOps approach to service delivery (see [11,14,22]). When realizing microservices architectures a core task is to design the service contracts or Application Programming Interfaces (APIs). In this context, we focus on the problem to design for, realize, enforce and maintain *quality aspects of the microservice API* – which are of

© Springer Nature Switzerland AG 2018
C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 73–89, 2018.
https://doi.org/10.1007/978-3-030-03596-9_5

key importance as the API is usually the only visible aspect of the microservice from the client's perspective. An API provider has to perform the balancing act of providing a high-quality service in a cost-effective way. Quality of an API has many dimensions, starting with the functionality, but also including many other qualities such as reliability, performance, security, and scalability – sometimes the latter are referred to as Quality of Service (QoS) guarantees. They are usually conflicting with each other, but almost always need to be balanced with economic qualities such as costs and time to market. Many quality measures related to service QoS exist, but only a few of them are directly related to APIs [13].

The main challenge for microservice API designers is to determine the appropriate quality trade-off during the design of the microservice API. Numerous practices exist and have complex relations among each other. Many decision drivers have to be understood and might be conflicting among each other. Therefore, architects orienting themselves and navigating in the microservice API design space usually face a high uncertainty in their decision making, related to finding and assessing the knowledge needed for making an informed decision. Once all required knowledge has been gathered, a high uncertainty on how to combine which practices remains; the impact of these practices and their combinations on the many potentially relevant quality trade-offs is not clear either. This paper aims to study the following resulting research questions:

- **RQ1 (a)** What are established practices to design for, realize, communicate and maintain the quality of a microservice API? **(b)** What are the relations among those practices? **(c)** What are decision drivers of those decisions? **(d)** Which impact do the practices and their combinations impose on the decision drivers?
- **RQ2 (a)** How high is the decision making uncertainty in this design space? **(b)** Can this decision making uncertainty be reduced? If so, how?

This paper makes three major contributions. First, we gather knowledge about established practices, their relations, and their decision drivers in the form of a **microservice APIs design space based on a qualitative study** of 55 knowledge sources (including 31 widely used APIs). Our second contribution is the codification of this knowledge in form of a **reusable Architectural Design Decision (ADD) model** which we formally modelled based on a UML2 metamodel. We also described newly documented patterns using pattern templates (which can be found in another publication along with technical details [19], whereas this paper focuses only on decision modelling aspects). In total we documented six decisions in two contexts with 40 decision options and 47 decision drivers. Please note that we limited our scope to message representations in the interface contracts, and excluded e.g. the architectural decisions required in service implementations (which were addressed in our earlier works [23]). Finally, we **estimate the decision making uncertainty** in this design space, calculate the uncertainty left after applying the guidance of our ADD model, and compare the two. Our model shows a potential to substantially reduce the uncertainty not only by documenting established practices, but also by organizing the knowledge in a model.

The remainder of this paper is organized as follows: In Sect. 2 we compare to the related work. Section 3 explains the research methods we have applied in our study. Then Sect. 4 describes our reusable ADD model. Section 5 provides the uncertainty reduction estimation. The findings are discussed in Sect. 6, and Sect. 7 concludes.

2 Related Work

Quite a number of studies on services focus on QoS aspects (see e.g. [13,18,21]). In microservice-specific studies related to quality, topics like the increased operations qualities when combined with DevOps (see e.g. [1]), qualities in service decomposition [7], or specific qualities like trade-offs in self-adaptive architectures [8] are studied. The specific quality aspects of the API, which is of high practical relevance as the API is the only part of the microservices visible to the client, is not yet a major focus of study.

A number of approaches study microservice patterns and best practices: The microservice patterns by Richardson [17] address microservice design and architecture practices. Another set of patterns on microservice architecture structures has been published by Gupta [6], microservice best practices are discussed in [11], and similar approaches are summarized in a recent mapping study [15]. So far, none of those approaches has been combined with a formal model and API quality is not a major focus.

Decision documentation models (examples are those covering service-oriented solutions [23], service-based platform integration [12], REST vs. SOAP [16], and big data repositories [5]) promise to improve the situation, but the focus on this kind of research is not yet on API design. The model developed in our study can be classified as a reusable ADD model [23]. Other authors have combined decision models with formal view models [9]. We apply those techniques in our work, but also extend them with a modelling approach and a detailed uncertainty reduction estimation. Exploiting uncertainties has been used in software architecture traceability research before [20].

3 Research Method

This paper aims to systematically study the established practices in the field of microservice API quality aspects. A number of methods have been suggested to study established practices. A classical method is pattern mining (see e.g. [3]) which starts with the authors' own experiences, searches systematically for other known uses in real-life systems, and then applies a series of feedback loops to improve the pattern. A number of techniques have been suggested for improving this research method. Hentrich et al. [10] define a pattern mining method as a form of qualitative research resembling methods like Grounded Theory (GT) [4]. Like GT, we studied each knowledge source in depth. We followed a similar coding process, as well as a constant comparison procedure to derive our model. In contrast to classical GT, our research began with initial research questions, as in Charmaz's constructivist GT [2]. Whereas GT typically uses textual analysis, we used textual codes only initially and then transferred them into formal UML models and text in pattern templates.

Our knowledge mining happened in many iterations. That is, we searched for one or a few new knowledge sources, applied open and axial coding [4] to identify candidate categories, and compared with the so-far-designed model continuously. We improved this model incrementally. A crucial question in GT is when to stop this process; here, theoretical saturation [4] has attained widespread acceptance in qualitative research: We stopped our analysis when 5 to 7 additional knowledge sources did not add anything new to our understanding of the research topic. As a result of this very conservative operationalization of theoretical saturation, we studied a rather large number of knowledge sources in depth (55 in total, summarized in Table 1, see [19] for more details on the sources), whereas most qualitative research often saturates with a much lower number of knowledge sources. In addition to 31 APIs and their documentations, our search led us to 24 additional knowledge sources (specifications, standards, and technologies) directly related to the design of a number of APIs. Our search was based on our own experience, APIs we have access to and worked with. We also used on major search engines (e.g., Google, Bing) and online API directories and topic portals (e.g., ProgrammableWeb, InfoQ) to find known uses and validate intermediate results. We included knowledge sources, if they were about widely used APIs (i.e., many more users than just the original authors), use modern service technologies, and follow at least some of the microservice tenets summarized in Sect. 1. Note that not always those APIs are labelled as microservice APIs, but as microservice tenets have been in use long before the microservice term was coined, and as some RESTful HTTP APIs share the same technological underpinnings, we have also considered them in our study.

Table 1. Knowledge sources included in the study

APIs studied	31	Amazon EC2 API, Amazon S3, AWS Lambda, Cloud Convert API, Confluence REST API, Facebook Graph API, File Transfer Service API, Finance Industry Web Service API, GitHub API v3, GitHub API v4, Google Calendar API, Google Compute Engine, JIRA Cloud API, LinkedIn API, Microsoft Azure, Microsoft Dynamics CRM, Microsoft Graph API, Open Weather Map, Optimizely, PayPal API, Quandl API, Salesforce API, Singlewire, Stripe API, SWIFT, Swiss Bank API, Swiss Federal Administration registry of companies web service API, Swiss Insurance API, TMForum REST API, Twitter API, YouTube Data API
API-related specifications, standards, technologies studied	24	Adidas API Spec, Amazon API Gateway, apistylebook.com, Basic Authentication, CHAP, EAP, EC SLA Guidelines, HTTP/1.1: Conditional Requests, JSON API Spec, Kerberos, LDAP, MuleSoft API Manager, OAuth, OpenID Connect 1.0, OWASP REST Security, Play2 Guard, REST API design book, RESTful Web Services Cookbook, RFC 7519, SAML, SLA Best Practices, SLA Whitepaper, Suggested REST Practices, TM Forum Applications Framework 3.0

4 Reusable ADD Model for API Quality

In this section, we report on the reusable ADD model that resulted from our study. Figure 1 shows an overview of the reusable decisions and their relationships, as well as their major decision contexts. Our model contains all kinds of *decision contexts in API design* in a separate domain model of which we only use a small part in this paper: Most decisions on API quality have to be made for *combinations of API clients and the API* those clients access. Many such decisions can be made for large groups of those combinations, such as all clients with freemium access or all clients accessing a specific API. One decision needs to be made at the level of *operations* in the API. Note that in Fig. 1 the decisions inherit those contexts from their (sub-)categories. Below we describe our findings for each of the decisions in detail. Note that all elements of our reusable ADD model are instances of a meta-model (with meta-classes such as *Decision*, *Category*, *Pattern*, *AND Combined Group*, and so on), which we introduce in this paper implicitly in the text with the uses of the meta-classes in our model. Please note that in this paper we focus on the decision models and their decision drivers. For space reasons, it is not possible to provide all technical details. Detailed patterns explaining the decision options used in this paper, including examples, known uses, and detailed discussions of decision drivers can be found in another publication describing the patterns using pattern templates [19]. The patterns are part of a larger pattern language effort started and summarized in [24].

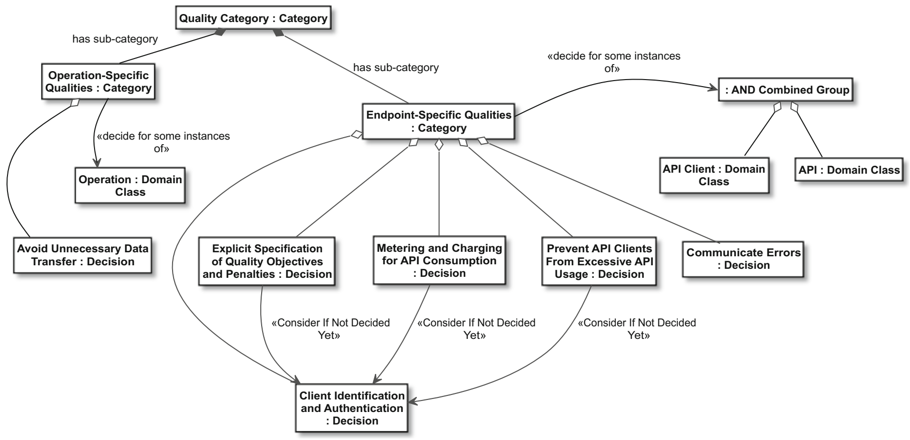

Quality Category : Category
has sub-category
has sub-category
Operation-Specific Qualities : Category
«decide for some instances of»
Operation : Domain Class
Avoid Unnecessary Data Transfer : Decision
Endpoint-Specific Qualities : Category
«decide for some instances of»
: AND Combined Group
API Client : Domain Class
API : Domain Class
Explicit Specification of Quality Objectives and Penalties : Decision
Metering and Charging for API Consumption : Decision
Prevent API Clients From Excessive API Usage : Decision
Communicate Errors : Decision
«Consider If Not Decided Yet»
«Consider If Not Decided Yet»
«Consider If Not Decided Yet»
Client Identification and Authentication : Decision

Fig. 1. API quality: reusable ADD model – overview of required decisions and categories

Reusable Decision: Identification and Authentication of the API Client. Identification and authentication are important for API providers that are paid for or use freemium models: the API provider can grant authorization based on the API client's proven identity. This is key to ensure security, but also impacts many other qualities; e.g., if unknown clients can access the API without control or known clients can make excessive use of the API, the performance of the system can degrade, reliability can be in danger, or costs (e.g., for used cloud resources) can rise. The typical decision to be made here is shown in Fig. 2. The simplest option is to chose *no secure identification and authentication needed*, which is suitable only if the a number of clients is limited and if the risks with respect to abuse or excessive use are low. The obvious alternative is to introduce an *Authentication* mechanism for the API (which includes identification). An *API Key* that assigns each client a unique token that the client can present to the API endpoint for identification is a minimalistic solution. If security is an issue, *API Keys* are not enough. In conjunction with an *additional secret key* that is not transmitted, *API Keys* can be used to securely authenticate a client. Another secure alternative are *authentication* or *authorization protocols* such as OAuth, SAML, Kerberos, or LDAP.

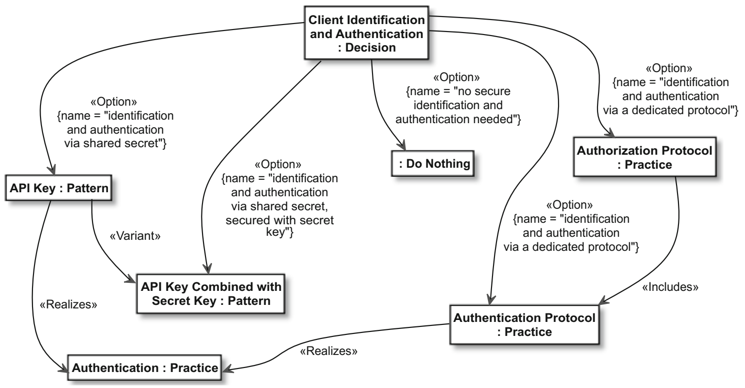

Client Identification and Authentication : Decision
«Option» {name = "identification and authentication via shared secret"}
«Option» {name = "identification and authentication via shared secret, secured with secret key"}
«Option» {name = "no secure identification and authentication needed"}
«Option» {name = "identification and authentication via a dedicated protocol"}
«Option» {name = "identification and authentication via a dedicated protocol"}
API Key : Pattern
: Do Nothing
Authorization Protocol : Practice
«Variant»
API Key Combined with Secret Key : Pattern
«Realizes»
«Includes»
Authentication Protocol : Practice
«Realizes»
Authentication : Practice

Fig. 2. Client identification and authentication decision

There are a number of decision criteria that need to be considered in this decision. First of all, the *level of required security*, as outlined above. In addition, *API Keys* are only a slight degradation in terms of *ease of use for clients* compared to doing nothing; the other options are less easy to use as they require dealing with more complex protocol APIs and setting up the required infrastructure. In addition, the *management of user account credentials* required in *authentication and authorization protocols* can be tedious both on client and provider side; this is avoided in all options using *API Keys*. With regard to the performance of the solution, doing nothing has no overhead. *API Key* options have a slight overhead for processing the key(s). *Authentication and authorization protocols* tend to have more overhead as they also offer additional features. The *API Key* options also *decouple the client making an API call from the client's organization*, as using the customer's account credentials would needlessly give system administrators and developers full account access.

Please that in our models we use the term practice as a superset of patterns and other established practices; only those practices that have been described in pattern form (e.g., in our related publications [19]) are denoted with the stereotype *Pattern*, all other existing practices are denoted as *Practice*.

Reusable Decision: Communicate Errors. A common quality concern for APIs is how to communicate errors as this has direct impacts on qualities like avoiding and fixing defects, costs of defect fixing, robustness and reliability problems due to unfixed defects, and so on. Of course, one option is not to handle the error at all, but this is usually not advisable – at least, for production APIs. A common solution, if only one protocol stack is used (e.g., HTTP over TCP/IP), is to use *Protocol-level Error Codes*, e.g., status codes in HTTP, but this does not work if error reporting needs to work across multiple protocols, formats, and platforms. In such cases the *Error Reporting* pattern should be used so that replies include in addition to machine-readable error codes, also a textual description of the error is provided for the API developer. Also such error messages can carry parameters and constants in order to allow internationalization of error messages when reporting the to the user of the client application.

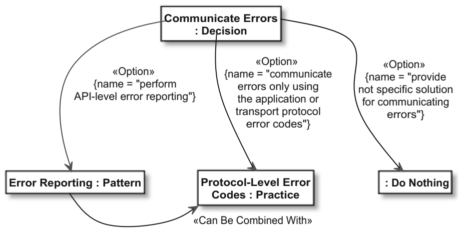

Communicate Errors : Decision
«Option» {name = "perform API-level error reporting"}
«Option» {name = "communicate errors only using the application or transport protocol error codes"}
«Option» {name = "provide not specific solution for communicating errors"}
Error Reporting : Pattern
Protocol-Level Error Codes : Practice
: Do Nothing
«Can Be Combined With»

Fig. 3. Communicate errors decision

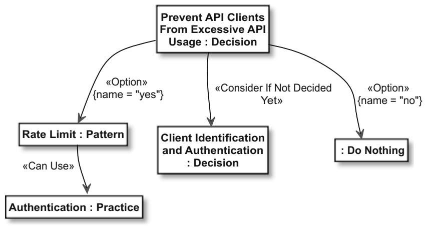

Prevent API Clients From Excessive API Usage : Decision
«Option» {name = "yes"}
«Consider If Not Decided Yet»
«Option» {name = "no"}
Rate Limit : Pattern
Client Identification and Authentication : Decision
: Do Nothing
«Can Use»
Authentication : Practice

Fig. 4. Prevent API clients from excessive API usage decision

The main decision drivers (see Fig. 3) to introduce any kind of error reporting are *help in fixing defects* and increased *robustness* and *reliability*. Error reporting leads to better *maintainability* and *evolvability*, and the more it explains errors and thus reduces the effort in the task of finding the cause of a defect, the more effective it is; thus the *Error Reporting* pattern performs better in this regard than simple error codes. *Error Reporting* is also better performing with regard to *interoperability* and *portability* as it better enables supporting protocol, format, and platform autonomy. However, the more elaborate error messages can reveal information that is problematic with regard to *security*, as revealing more information about system internals opens up attack vectors. *Error Reporting* requires more work, if *internationalization* is required, as the more detailed information needs to be translated.

Reusable Decision: Preventing API Clients from Excessive API Usage. Excessive use by a few clients can significantly limit the availability of the service for other clients. Thus preventing excessive API usage by clients is needed.

Assuming API clients can be identified as previously discussed, their individual usage of the API can be monitored for billing purposes. If offsetting the expense of operating the microservice to its clients is not enough to limit their traffic (e.g., using a *Rate Plan*, see next decision), an explicit *Rate Limit* can be introduced to safeguard against API clients that overuse the API. The limit can be expressed in number of requests per period of time. If this limit is exceeded, further requests can either be declined, be processed later or with lower priority.

The decision is shown in Fig. 4. The major decision criteria to be considered in this decision are: A certain level of *scalability and performance* needs to be maintained by the provider, but could be in danger if clients abuse the API. Means for supporting *client awareness* of *Rate Limits* are required so that clients can find out know how much of their limits they have already used up. Establishing *Rate Limits* helps the provider to support qualities such as *resilience, reliability, and fault tolerance* as they make it hard for clients to abuse the API in a way that puts those qualities at risk. All these potential benefits must be contrasted to the *impact and severity of risks of API abuse* and *economic aspects*. Introducing *Rate Limits* produces costs and can be seen critically by clients as well as additional complexity if clients are allowed to negotiate their limits.

Reusable Decision: Metering and Charging for API Consumption. If the API is a commercial offering, the API provider might want to charge for its usage. Thus a means for identifying and authenticating clients is required (see decision above). Then the provider can monitor clients and assign a *Rate Plan* which measures API usage e.g. on a per-call level and is used to bill API clients, advertisers, or other stakeholders accordingly. As shown in Fig. 5, we can alternatively not meter and charge the client. In the context of a *Rate Plan* sometimes a *Rate Limit* is used to ensure fair use. Figure 5 also illustrates possible variants of the *Rate Plan* pattern: Pricing can be based on actual usage, on

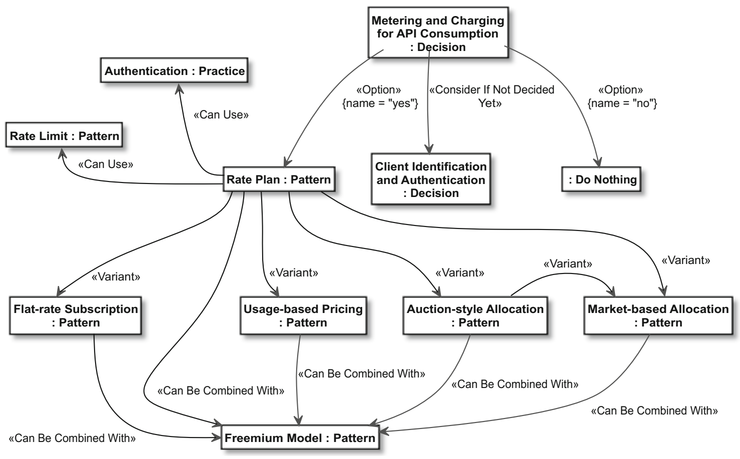


Fig. 5. Metering and charging for API consumption decision

market-based allocation (or with its sub-variant based on auctions), or on flat-rate subscriptions. All those variants can be combined with a freemium model.

The major drivers for this decision are usually *economic aspects*, such as pricing models and selecting a variant of the pattern that suits the provider or the consumer business model best. The benefits of applying the pattern need to be contrasted to the efforts and costs required to meter and charge customers. *Accuracy* is central as API clients expect to be billed only for the services they actually have consumed. Accurate metering requires an adequate *meter granularity* to be defined. As information about metering and charging contains sensitive information about clients, e.g. indications of how well they do in their markets, it needs extra protection with regard to *security*.

Reusable Decision: Explicit Specification of Quality Objectives and Penalties. Quality objectives are kept implicit and vague for many APIs. If the client requires (or even pays for) stronger guarantees or the provider wants to make explicit guarantees (e.g., to differentiate from competitors), an explicit specification of quality objectives and penalties can be considered. This can be done by introducing a *S*ervice Level Agreement (SLA) which is an extension of the API description detailing measurable *Service Level Objectives* (SLOs) and penalties in case of violation. Any *Rate Plan* and *Rate Limit* should refer to the SLA if these patterns are used (and vice versa). SLAs require means for identifying and authenticating clients; usually authentication practices have to be used. There are a number of typical variants of the pattern: SLAs only used for internal use, SLAs with formally specified SLOs, and those with only informally specified SLOs, e.g., with natural language.

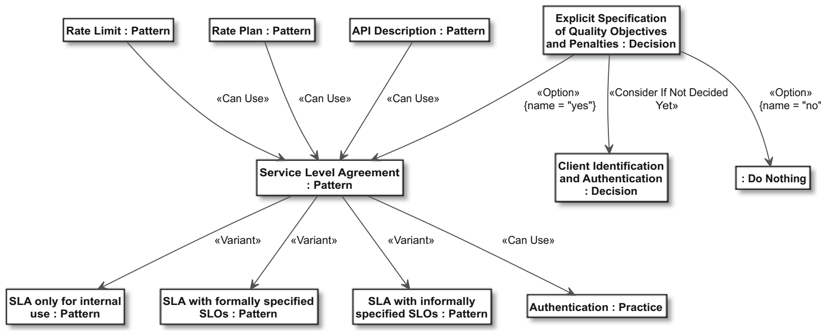

Rate Limit : Pattern
Rate Plan : Pattern
API Description : Pattern
Explicit Specification of Quality Objectives and Penalties : Decision
«Can Use»
«Can Use»
«Can Use»
«Option» {name = "yes"}
«Consider If Not Decided Yet»
«Option» {name = "no"}
Service Level Agreement : Pattern
Client Identification and Authentication : Decision
: Do Nothing
«Variant»
«Variant»
«Variant»
«Can Use»
SLA only for internal use : Pattern
SLA with formally specified SLOs : Pattern
SLA with informally specified SLOs : Pattern
Authentication : Practice

Fig. 6. Explicit specification of quality objectives and penalties decision

As shown in Fig. 6, the main decision drivers are: *Attractiveness from consumer point of view* can be higher if guarantees about qualities can be made. However, this must be contrasted to possible issues related to *cost-efficiency and business risks from a provider point of view*. Some guarantees are required by *government regulations and legal obligations* like those related to personal data protection such as the EU General Data Protection Regulation (GDPR). If a provider intends to make any guarantees about the quality of its service (typical candidates concern the microservice *availability*, *performance and scalability*, or *security and privacy*), then such qualities become decision drivers for this decision. Finally, the decision relates to *business agility and vitality* as the business model of a client might rely on the above named qualities of a service.

Reusable Decision: Avoid Unnecessary Data Transfer. The decision described in this section contains four patterns addressing different situations in which unnecessary data is transferred by the operations of an API. Note that in contrast to the prior decisions, this one needs to be made per operation as only a detailed analysis of the individual needs of the clients can indicate whether the data transfer can be reduced or not.

It can be hard for API providers to design operations that provide the required data exactly: the needs of clients might not be predictable and/or differ from each other. One solution is to let the API client provide a *Wish List* in the request enumerating all desired data elements so that the API provider can deliver only the desired elements in the response. A *Wish List* is not always easy to specify, e.g., if only certain fractions of nested or repetitive parameter structures are required. An alternative that works better for complex parameters is to let the client send a more expressive *Wish Template* that mirrors the structure of the desired responses (but contains dummy data) in its request.

If multiple clients repeatedly request the same data, which seldom changes, unnecessary data transfer can be avoided through a *Conditional Request*. To make requests conditional, they contain additional metadata parameters so that the provider may only process the request if a condition is met; e.g., in RESTful HTTP APIs, the provider could provide a *fingerprint*, which the client can then include in subsequent requests to indicate the latest known version of the resource that the client already has retrieved.

Another scenario is when one client makes multiple related requests that form logical batches. If the provider receives and replies to all requests individually, *performance* and *scalability* may suffer. This can be avoided by defining a *Request Bundle* as a container message that assembles multiple individual requests and is accompanied by metadata such as number of and identifiers of individual requests. Exchanging a single large message is usually more efficient than transferring multiple short messages. This comes at a price of increased effort for request processing on the provider side.

Sometimes no data transfer reduction is possible or wanted for the target operation(s); no action has to be taken in that case. Alternatively, unnecessary data transfer can be avoided through the patterns explained above. A combination of *Conditional Request* with either *Wish List* or *Wish Template* can be useful to indicate which subset of changed data is requested. *Request Bundle* can be combined with any of the prior alternatives, but combining multiple of the patterns increases the *complexity of the API*.

The main decision driver for this decision as illustrated in Fig. 7 is the *individual information needs of a client* which need to be analyzed to find out which of

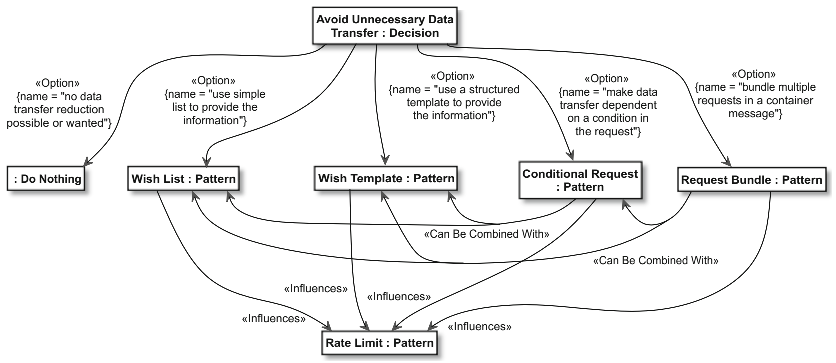

Avoid Unnecessary Data Transfer : Decision
«Option» {name = "no data transfer reduction possible or wanted"}
«Option» {name = "use simple list to provide the information"}
«Option» {name = "use a structured template to provide the information"}
«Option» {name = "make data transfer dependent on a condition in the request"}
«Option» {name = "bundle multiple requests in a container message"}
: Do Nothing
Wish List : Pattern
Wish Template : Pattern
Conditional Request : Pattern
Request Bundle : Pattern
«Can Be Combined With»
«Can Be Combined With»
«Influences»
«Influences»
«Influences»
«Influences»
Rate Limit : Pattern

Fig. 7. Avoid unnecessary data transfer decision

the patterns combinations can provide benefits. Consider situations where data transfer over the network is perceived as a potential bottleneck: *Data parsimony* can further drive the decision as the patterns can help to reduce bandwidth consumption during data transmission. Avoiding unnecessary data transfers can improve *performance*, as transferring all data elements to all clients all the time would harm response time, throughput, processing time, cost, etc. *Security* can be a driver to apply or not to apply the patterns *Wish List* and *Wish Template*: enabling clients to provide options on which data to receive may unwittingly expose sensitive data or open up additional attack vectors. On the other hand, data that is not transferred cannot be stolen and cannot be tampered with. Finally, going from an API with a fixed data representation to an API where clients can dynamically determine what content will be retrieved – which all four patterns do – increases the *complexity of API design and programming.* This is e.g. evidenced in GraphQL which can be seen as an extreme form of *Wish Template.* In addition, the special cases introduced by the patterns cause more *testing* and *maintenance* efforts.

5 Preliminary Estimation of Uncertainty Reduction

Architectural decision making is always tied to the given context; e.g., our ADD model documents decisions in two different contexts $CON = \{API\ \&\ API\ Client,\ API\ Operation\}$. In each context the architect needs to make a set of decisions DEC. For each, $d \in DEC$ there are a number of decision options OPT_d possible to choose for decision d. Finally, there is a set of criteria CRI_d that need to be considered when making a decision d. There are many different kinds of uncertainties involved in making ADDs in a field in which the architect's experience is limited. The obvious contribution of our ADD model is that it helps to reduce the uncertainty whether all relevant, necessary and sufficient elements for making a correct decision have been found (for each of the sets named above CON, DEC, OPT, and CRI). Another kind of uncertainty reduction is the uncertainty reduction our ADD model provides compared to using the same knowledge, but in a completely unorganized fashion. We want to estimate this kind of uncertainty reduction here. Exploiting uncertainties has been used in a similar way in software architecture traceability research before [20]. Our model provides organization to the design knowledge at a number of levels:

- It groups and interrelates decisions; e.g., *Metering and Charging for API Consumption* requires consideration of *Client Identification and Authentication* (see Fig. 5).
- It groups decision options in decisions and interrelates them; e.g., *Perform Error Handling* has three options $OPT_{Perform\ Error\ Handling} = \{$*Error Reporting, Protocol- Level Error Codes, Do Nothing*$\}$. Two of those are related further: *Error Reporting* can be combined with *Protocol-Level Error Codes* (see Fig. 3).
- It associates decision criteria to decisions; e.g., *Metering and Charging for API Consumption* has 4 criteria $CRI_{Metering\ and\ Charging\ for\ API\ Consumption} = \{$*Economic Aspects, Accuracy, Meter Granularity, Security*$\}$ (see criteria explanations for Fig. 5).
- It pre-selects many of those criteria for the options; e.g., in *Metering and Charging for API Consumption*, the criterion *Security* is pre-selected: Using a *Rate Plan* has negative impacts on it; the option *Do Nothing* is preferable as it has no security impact. In contrast, the criterion *Economic Aspects* needs to be investigated further for both decision options in the concrete context and thus cannot be pre-decided.

Here, we estimate the uncertainty reduction both for each individual decision and possible decision combinations in each context (uncertainty reduction estimations are reported as rows in Table 2). We calculate each number both for using our ADD model (denoted with $^\oplus$ below) and not using our model (denoted with $^\ominus$ below):

- *Number of decisions nodes (ndec):* Our ADD model represents each decision separately. So the number of decision nodes for a single decision d is always $ndec_d^\oplus = 1$. Without our ADD model, each design solution (i.e., decision option in the design space) that is not *Do Nothing* is a possible decision node, and it can either be selected or not: $ndec_d^\ominus = |OPT_d \setminus \{Do\ Nothing\}|$. Please note that, if a design solution has variants, OPT_d contains the base variant plus each possible variant.
- *Number of required criteria assessments in a decision (ncri):* Our ADD model includes explicit decision criteria per decision. Some of those are pre-decided, others not. Let the functions *decided*() and *undecided*() select them, respectively. If all criteria are decided in a decision, we only require one criteria assessment (assessing the whole vector of decided criteria). If all criteria are undecided, we need to make $|CRI_d|$ assessments of criteria. Often some criteria are decided, others not, so the number of criteria to be decided is in general:

$$ncri_d^\oplus = \begin{cases} 1 + |undecided(CRI_d)|, & for\ decided(CRI_d) > 0 \\ |CRI_d|, & for\ decided(CRI_d) = 0 \end{cases}$$

Without our ADD model, we need to assess each criterion for each decision node (as we have no pre-decided choices): $ncri_d^{\ominus} = |CRI_d| \times |ndec_d^{\ominus}|$.

– *Number of possible decision outcomes (ndo):* Our ADD model already models each decision option separately in $|OPT_d|$ including *Do Nothing*, so $ndo_d^{\oplus}$ usually equals $|OPT_d|$ unless the design space allows explicit combinations of solutions as additional outcomes. For instance, in the decision *Metering and Charging for API Consumption* the variant *Freemium Model* can be combined with the base variant and all four other variants, leading to an additional five outcomes. Let the function *solComb*() return the set of possible solution combinations in the options of a decision; then $ndo_d^{\oplus} = |OPT_d| + |solComb(OPT_d)|$.
  The same is true in principle for the decisions made without our ADD model, but as the decision d is here split into multiple separate decision nodes $ndec_d^{\ominus}$ and without the ADD model no information on which combinations are possible is present, we need to consider any possible combination in $ndec_d^{\ominus}$, i.e., the size of the powerset of the decision nodes: $ndo_d^{\ominus} = |\mathcal{P}(ndec_d^{\ominus})| = 2^{|ndec_d^{\ominus}|}$.

For the context *API Operation*, there is only a single decision (i.e., avoid unnecessary data transfer), but in the context *API & API Client* there are five decisions. It is thus also important to calculate the total uncertainty reduction in this context, where any number of those five decisions can be taken. The combinations of *ndec* and *ncri* in a context $c \in CON$ is with or without our ADD model simply their sum for the decisions d in the context c; let *inCon()* be a function selecting all decisions in a context:

$ndec_c = \sum_{d \in \{dec \in DEC | \, dec \in inCon(c)\}} |ndec_d|$

$ncri_c = \sum_{d \in \{dec \in DEC | \, dec \in inCon(c)\}} |ncri_d|$

If multiple decisions need to be made, the combinations for *ndo* require us to consider all possible combinations of decision outcomes of each and every decision:

$ndo_c = |\mathcal{P}(\bigcup_{d \in \{dec \in DEC | \, dec \in inCon(c)\}} ndo_d)|$.

Table 2 shows the results of the uncertainty reduction estimation. As can be seen without our ADD model in general more decision nodes *ndec* need to be considered, ranging from 0% to 83,33% for individual decisions; and totally 70,59% in the API client/API context and 75.00% in the operation context. For the necessary criteria assessments *ncri* improvements are even higher, ranging from 50% to 97,73% for individual decisions; and totally 90,16% in the API client/API context and 93,75% in the operation context. Here this high improvement is mainly due to the pre-selected criteria, which lead to criteria assessments in whole sets of criteria rather than evaluating each criterion separately. Finally, for the number of possible decision outcomes *ndo*, the improvement in uncertainty reduction for individual decisions ranges from 0% to 81,25%. The large spread is due to the fact that without our ADD model, the number of options rises exponentially: For decisions with larger numbers of decision options the improvement is greater than for those with only a few options. In total we see a 25% improvement in the operation context, as this is just a single decision.

Table 2. Uncertainty reduction estimation

Decision		# Decision nodes *ndec*	# Criteria assessments *ncri*	# Possible decision outcomes *ndo*
Client Identification and Authentication Decision	With design space	1	1	5
	Without design space	4	44	16
	Uncertainty reduction	75,00%	97,73%	68,75%
Perform Error Handling	With design space	1	1	4
	Without design space	2	18	4
	Uncertainty reduction	50,00%	94,44%	0,00%
Preventing API Clients from Excessive API Usage	With design space	1	4	2
	Without design space	1	8	2
	Uncertainty reduction	0,00%	50,00%	0,00%
Metering and Charging for API Consumption	With design space	1	4	12
	Without design space	6	24	64
	Uncertainty reduction	83,33%	83,33%	81,25%
Explicit Specification of Quality Objectives and Penalties	With design space	1	2	5
	Without design space	4	28	16
	Uncertainty reduction	75,00%	92,86%	68,75%
Total in Context API Client / API	With design space	5	12	268435456
	Without design space	17	122	482754917909
	Uncertainty reduction	70,59%	90,16%	99,94%
Avoid Unnecessary Data Transfer = Total in Context Operation	With design space	1	2	12
	Without design space	4	32	16
	Uncertainty reduction	75,00%	93,75%	25,00%

The total for the API client/API context shows a 99,94% improvement; here we use for both cases the same exponential function for calculation, but as individual decisions were performing much better with our model, the resulting total number is much lower than without it.

Please note that the numbers are rough estimates only, not a formal evaluation. They indicate that substantial uncertainty reduction is possible. To harden them, further such estimations in other design spaces are required, which could be the basis for developing a theory. Such a theory could then be validated in empirical studies in realistic cases.

6 Discussion and Threats to Validity

We have studied knowledge on established practices on API quality aspects, relations among those practices, and decision drivers to answer **RQ1** with multiple iterations of open coding, axial coding, and constant comparison to first codify the knowledge in informal codes and then in a reusable ADD model. Some of our decision options were design patterns (documented in [19] and designated as such in our models, see Figs. 2, 3, 4, 5, 6 and 7). Precise impacts on decision drivers of design solutions and their combinations were documented as well; for space reasons we only summarized those in the text and did not show them in the UML models (see [19] for technical details).

The contributions to RQ1 in part already answer **RQ2**, in so far as each of the pieces of knowledge is systematically derived from established knowledge, which helps to reduce uncertainty regarding finding knowledge at all and finding it correctly. In addition, we estimated the uncertainty reduction achieved through the organization of knowledge in our ADD model in Sect. 5. We may conclude that our ADD model (and similar models) have the potential to lead to substantial improvements in uncertainty reduction in all evaluation variables due to the additional organization it provides and pre-selections it makes. For individual decisions, mastering and keeping in short term memory the necessary knowledge for design decision making seems infeasible without the ADD model (e.g., four decision nodes with 44 criteria assessments and 16 possible outcomes for the first decision in Table 2), but quite feasible with our ADD model. Our model also helps to maintain an overview of the decisions $ndec^{\oplus}$ and criteria assessments $ncri^{\oplus}$ in the combined API client/API context. Only the number of possible decision outcomes for the combination of multiple decisions seem challenging to handle, both in the $ndo^{\oplus}$ and $ndo^{\ominus}$ case. That is, despite all benefits of our approach, the uncertainty estimations also show that a limitation of the approach is that when multiple decisions need to be combined in a context, maintaining an overview of possible outcomes and their impacts remains a challenge – even when a substantial uncertainty reduction and guidance is provided as in our ADD model. Further research and tool support is needed to address this challenge. As our numbers are only rough estimates, further research is needed to harden them and confirm them in empirical studies, possibly based on a theory developed based on such preliminary estimations.

While generalizability beyond the 55 knowledge sources we studied is possible to a large extent, our results are limited to those sources and to a lesser extent to very similar APIs. Most of the 55 source were public, Internet-wide APIs; we have studied a few in-house APIs as well. This mix might have introduced bias or left to the omission of important in-house practices in commercial enterprises. We could only study the API quality aspects addressed in those sources. Thus, we do not claim any form of completeness. Our results are only valid in our set scope. In the various coding process and review stages of our research method, each finding was checked in at least five iterations by different members of our author team. However, possible misinterpretations or biases of individual researchers or the whole author team cannot be fully excluded and might have influenced our results. As the authors have many years of experience in the field (gained both in industrial projects and in education of students and practitioners), we are optimistic that this threat to validity is mitigated in our study to a large extent.

7 Conclusions

We have performed a qualitative study in which we have studied microservice API quality aspects in 55 unique sources. Our study led to the identification of six architectural design decisions with in total 40 decision options and in total 47 decision drivers modelled in a formal ADD model. In our uncertainty reduction estimations we were able to indicate that the knowledge organization in our ADD model can lead to a significant reduction of uncertainty where multiple decisions need to be combined. In our future work, we plan to combine our ADD model with other aspects of API design, apply the results in case studies e.g. in different verticals or industries, and build and empirically validate a theory based on the preliminary uncertainty reduction estimations.

Acknowledgements. This work was partially supported by Austrian Science Fund (FWF) project ADDCompliance and FFG project DECO (no. 864707).

References

1. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Softw. **33**(3), 42–52 (2016)
2. Charmaz, K.: Constructing Grounded Theory. Sage, Thousand Oaks (2014)
3. Coplien, J.: Software Patterns: Management Briefings. SIGS, New York (1996)
4. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine de Gruyter, New York (1967)
5. Gorton, I., Klein, J., Nurgaliev, A.: Architecture knowledge for evaluating scalable databases. In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA 2015), pp. 95–104 (2015)
6. Gupta, A.: Microservice design patterns (2017). http://blog.arungupta.me/microservice-design-patterns/
7. Gysel, M., Kölbener, L., Giersche, W., Zimmermann, O.: Service cutter: a systematic approach to service decomposition. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol. 9846, pp. 185–200. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_12
8. Hassan, S., Bahsoon, R.: Microservices and their design trade-offs: a self-adaptive roadmap. In: Proceedings of the SCC, pp. 813–818 (2016)
9. van Heesch, U., Avgeriou, P., Hilliard, R.: A documentation framework for architecture decisions. J. Syst. Softw. **85**(4), 795–820 (2012)
10. Hentrich, C., Zdun, U., Hlupic, V., Dotsika, F.: An approach for pattern mining through grounded theory techniques and its applications to process-driven SOA patterns. In: Proceedings of the 18th European Conference on Pattern Languages of Program, pp. 9:1–9:16 (2015)
11. Lewis, J., Fowler, M.: Microservices: a definition of this new architectural term, March 2004. http://martinfowler.com/articles/microservices.html
12. Lytra, I., Sobernig, S., Zdun, U.: Architectural decision making for service-based platform integration: a qualitative multi-method study. In: Proceedings of WICSA/ECSA (2012)
13. Menascé, D.A.: QoS issues in web services. IEEE Internet Comput. **6**(6), 72–75 (2002)

14. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly, Sebastopol (2015)
15. Pahl, C., Jamshidi, P.: Microservices: a systematic mapping study. In: 6th International Conference on Cloud Computing and Services Science, pp. 137–146 (2016)
16. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful web services vs. big web services: making the right architectural decision. In: Proceedings of the 17th World Wide Web Conference (WWW), pp. 805–814, April 2008
17. Richardson, C.: A pattern language for microservices (2017). http://microservices.io/patterns/index.html
18. Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., Dustdar, S.: An end-to-end approach for QoS-aware service composition. In: IEEE International Conference on Enterprise Distributed Object Computing Conference (EDOC 2009), pp. 151–160. IEEE (2009)
19. Stocker, M., Zimmermann, O., Lübke, D., Zdun, U., Pautasso, C.: Interface quality patterns - crafting and consuming message-based remote APIs. In: Proceedings of the 23nd European Conference on Pattern Languages of Programs, EuroPLoP 2018 (2018)
20. Trubiani, C., Ghabi, A., Egyed, A.: Exploiting traceability uncertainty between software architectural models and performance analysis results. In: Weyns, D., Mirandola, R., Crnkovic, I. (eds.) ECSA 2015. LNCS, vol. 9278, pp. 305–321. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23727-5_26
21. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the 12th International Conference on World Wide Web, pp. 411–421. ACM (2003)
22. Zimmermann, O.: Microservices tenets. Comput. Sci. - Res. Dev. **32**(3), 301–310 (2017)
23. Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. J. Syst. Softw. **82**(8), 1249–1267 (2009)
24. Zimmermann, O., Stocker, M., Lübke, D., Zdun, U.: Interface representation patterns: crafting and consuming message-based remote APIs. In: Proceedings of the 22nd European Conference on Pattern Languages of Programs, EuroPLoP 2017 (2017)

Adaptive Temporal Verification and Violation Handling for Time-Constrained Business Cloud Workflows

Haoyu Luo[1,2], Xiao Liu[3], Jin Liu[1](✉), Bo Han[1], and Yun Yang[4]

[1] School of Computer Science, Wuhan University, Wuhan, China
{luohy, jinliu, bhan}@whu.edu.cn
[2] School of Computer Science, South China Normal University, Guangzhou, China
[3] School of Information Technology, Deakin University, Geelong, Australia
xiao.liu@deakin.edu.au
[4] School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia
yyang@swin.edu.au

Abstract. To achieve on-time completion of time-constrained business cloud workflows, a large number of parallel cloud workflow instances need to be constantly monitored so that temporal violations (namely intermediate runtime delays) can be detected and handled timely. Over the last few years, many strategies have been proposed but they are not adaptive enough to capture the dynamic behaviors of business cloud workflows. In this paper, we introduce the idea of "adaptiveness" into our strategy design. Specifically, we first present an adaptive temporal checkpoint selection strategy where the time intervals between checkpoints are adaptively determined at runtime, and then propose a matching temporal violation handling strategy which can determine the required lifecycle of cloud services. The evaluation results demonstrate that our adaptive strategy can achieve both higher efficiency and better cost effectiveness compared with conventional strategies.

Keywords: Temporal verification · Violation handling · Business workflow Adaptiveness · Cloud computing

1 Introduction

Business workflow can provide partial or even full automation of business processes in the domains of such as e-business and e-government. A notable feature of business workflow is that there is usually a large number of workflow instances running in a parallel fashion triggered by a large amount of concurrent user requests. To ensure the scalability in processing parallel workflow instances, a rapidly increasing number of business workflow applications are being deployed into the Cloud.

In business scenarios, a time-constrained workflow application needs to provide timely response to business requests [1]. Failing to deliver requested results in time may lead to the deterioration of user satisfaction, even huge financial loss. Therefore, the correctness of a business workflow application depends not only on its logical correctness, but also on its temporal correctness [2]. However, due to the dynamic nature and uncertainties that exist during the running of workflows in the cloud, temporal violations often occur which may have a serious impact on on-time completion of workflow instances. “temporal violation” means an intermediate violation of time constraints during workflow execution that can be fixed locally to achieve overall timely completion.

To ensure temporal correctness, workflow temporal behaviors need to be constantly monitored at runtime so that temporal violations can be timely detected and handled [3, 4]. When dealing with the monitoring and verification of temporal behaviors for a large number of parallel business workflow instances, throughput has been proved to be a competent measurement for the requirements of efficiency and scalability [5]. The basic idea is to select a set of time points along execution timeline as checkpoints, at which temporal behavior (represented by workflow throughput) is verified to determine whether a temporal violation occurs or not. If a temporal violation is detected, the current temporal behavior needs to be adjusted by temporal violation handling strategies.

To handle temporal violations, a temporal verification strategy and a temporal violation handling strategy are required to address the problem of “Where” and “How” respectively. In recent years, many related approaches have been proposed. However, a common problem is that they are not adaptive enough to capture the dynamic behaviors of business cloud workflows. To detect temporal violations, existing strategies verify temporal behavior at a set of temporal checkpoints which are selected from predefined system time points before workflow execution. These time points are usually equally distributed and the fixed time intervals between them are empirically set. This is inefficient as temporal behaviors constantly fluctuate at workflow runtime. In the meantime, most approaches handle the detected temporal violations by adding new resources. But the current work simply adds new resources with fixed lifecycles once a temporal violation is detected, in regardless of workflow runtime temporal behavior. Such kind of static approach may cause unnecessary cost of resource consumption.

For such an issue, we introduce the idea of “adaptiveness” into our strategy design. Two adaptive strategies are proposed to answer the questions of “Where” and “How” to handle temporal violations respectively. Specifically, we first present an adaptive temporal checkpoint selection strategy for the question of “Where”. Instead of using fixed time intervals, time interval between adjacent checkpoints is adaptive to workflow execution states. This strategy is more sensitive to the variation of temporal behaviors, which can improve the efficiency of temporal verification and decrease unnecessary resource consumption. Then we present a matching temporal violation handling strategy to answer the question of “How”. This strategy is designed to reduce or eliminate time delays by accelerating the execution of workflow activities with extra resources, e.g. service instances, provisioned to the cloud service nodes where temporal violations are detected. Specifically, it addresses three major issues for the question of “How”: (1) where to add extra resources; (2) how many resources are needed; (3) how long their lifecycles should be. Experimental results show that our adaptive strategy can achieve the target on-time completion rate with higher verification efficiency and at least 19.4% less resource consumption compared with conventional strategies.

The rest of the paper is organized as follows. Section 2 presents preliminary definitions. Section 3 presents the adaptive temporal verification and violation handling strategies. Section 4 demonstrate the experimental results. Section 5 concludes this paper.

2 Preliminary

(1) Workflow throughput

A business workflow is made up of a set of activities in partial order. We denote the i th activity of a business workflow as a_i. The mean, expected and runtime completion duration of a_i is denoted as $M(a_i)$, $E(a_i)$ and $R(a_i)$ respectively. Accordingly, WF_i is a workflow with its mean, expected and runtime completion duration denoted as $M(WF_i)$, $E(WF_i)$ and $R(WF_i)$ respectively.

Definition 1 (Workflow Throughput). Given a batch of q business workflow instances $WF\{WF_1, WF_2, \ldots, WF_q\}$ which starts at system time S_0, the completion of workflow activity a_{ij} contributes to the completion of the entire collection of workflows with a value of $M(a_{ij})/T$ where $\mathrm{T} = \sum_{i=1}^{q} M(WF_i)$. Here, we assume that at the current observation time point S_t, the set of new completed activities from the preceding nearest observation time point S_{t-1} is denoted as $a\{\}|_{S_{t-1}}^{S_t}$, then the system throughput is defined as $TH|_{S_{t-1}}^{S_t} = M(a\{\}|_{S_{t-1}}^{S_t})/T$.

Workflow throughput constraints is the expected accumulated workflow throughput (namely the percentage of completion) that should be achieved by a specific system time point. The value is decided by the throughput deadline assignment strategy [6].

(2) Queueing model for cloud services

At workflow runtime, a large number of instances are initialized in a short time. Since the number of parallel workflow instances is much more than the dedicated cloud services, workflow activities have to queue up on limited services. In this paper, queueing model is employed to depict the queueing and execution process of parallel workflow activities. We employ M/G/m/m+r model to formulate the behavior of the first queueing system and G/G/m/m+r model for the rest $k - 1$ queueing systems [7]. The discussion for the rationale of the model design can be found at our previous work [8].

(3) Throughput-based temporal consistency model

Temporal verification requires a temporal consistency model which defines the relationship between the current workflow execution state and target deadline.

Definition 2 (Throughput Consistency Model). Given the same batch of workflows in Definition 1 and its final deadline $F(WF)$, at throughput checkpoint S_p, it is said to be of $\alpha\%$ consistency if:

$$F(\lambda_\alpha) = TH|_{S_0}^{S_p} + Exp\left(TH|_{S_p}^{F(WF)}\right) \quad (1)$$

where λ_α is defined as the $\alpha\%$ confidence percentile with the cumulative standard normal distribution function of $F(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{\mu_i+\lambda\sigma_i} e^{-(x-\mu_i)^2/2}\sigma_i^2 dx = \alpha\%$. $TH|_{S_0}^{S_p}$ is the current runtime throughput until S_p, $Exp\left(TH|_{S_p}^{F(WF)}\right)$ is the expected workflow throughputs during the time between the checkpoint and final deadline. $\alpha\%$ consistency is a probability confidence for on-time completion.

3 Adaptive Temporal Verification and Violation Handling

3.1 Adaptive Temporal Checkpoint Selection Strategy

Definition 3 (Candidate Throughput Checkpoints). Given the same batch of workflow instances as in Definition 1, a system time point S_t along the workflow execution timeline is a candidate throughput checkpoint if $S_t - S_{t-1} = k * bt$ (k = 1, 2, 3, …).

Algorithm 1. Throughput consistency verification strategy.

```
Require:   Target on-time completion rate $\theta\%$
           A collection of candidate checkpoints $S_p$ ($0 \le p \le n$)
Ensure:    A set of temporal checkpoints
           Throughput consistency states at checkpoints
Step1:     Temporal checkpoint selection
1:         Deficit=0; Redundancy=0; k=1; p=1;
2:         While p ≤ n
3:             If $TH|_{S_{p-k}}^{S_p} < THCons|_{S_{p-k}}^{S_p}$
4:                 Deficit= $THCons|_{S_{p-k}}^{S_p} - TH|_{S_{p-k}}^{S_p}$;
5:                 If Deficit < Redundancy
6:                     If k>1
7:                         k=⌊k/2⌋;
8:                     End if
9:                 Else
10:                    $S_p$ is selected as a checkpoint, continue to Step 2;
11:                    If $\alpha\% \ge \theta\%$
12:                        Break;
13:                    Else
14:                        k=1; Redundancy=0; Report a temporal violation;
15:                    End if
16:                End if
17:            Else
18:                k=k+1; Redundancy= $TH|_{S_{p-k}}^{S_p} - THCons|_{S_{p-k}}^{S_p}$;
19:            End if
20:            p=p+k;
21:        End while
Step 2:    Temporal verification
           Calculate workflow throughput:
             (1) the completed workflow throughput until $S_p$.
             (2) the expected remaining throughput from $S_p$ to $F(WF)$.
           Given the throughput consistency model depicted in Definition 2, calculate
           throughput consistency state $\alpha\%$ at checkpoint $S_p$
```

$k * bt$ is the time interval between adjacent candidate throughput checkpoints.bt is the minimum time interval for system monitoring. k is a variable which is decided at workflow runtime. Checkpoint selection strategy needs to figure out candidate throughput checkpoints first, then determines whether they should be selected as a checkpoint.

Algorithm 1 depicts the adaptive checkpoint selection strategy, which selects checkpoints one by one at workflow runtime. Whether a candidate checkpoint is selected as checkpoint or not depends on both the temporal behavior during the latest monitoring window $k * bt$ and the temporal behaviors of several prepositive checkpoints. Temporal consistency state of the newly selected checkpoint influences the decision of the next checkpoint. To reflect such a feedback and adjustment mechanism, we integrate the proposed checkpoint selection strategy (Step 1) with throughput-based temporal consistency model (Step 2) as an overall throughput consistency verification approach.

3.2 Temporal Violation Handling Strategy

Once a temporal violation is detected at a checkpoint, temporal violation handling strategy will be triggered to deal with the recovery of violation by accelerating the workflow execution after the current checkpoint. In the scenario of business cloud workflows, if a throughput violation is reported, temporal violation handling strategy needs to increase the system throughput in a short period of time.

We design to handle temporal violations by recruiting extra resources (namely adding more cloud service instances in queueing systems). The extra resources for violation handling are recruited only for temporary use. Specifically, our proposed temporal violation handling strategy needs to answer three major questions:

(1) **Where to add resources?**

Queueing system for a cloud service is regarded as a basic unit for violation handling. Therefore, new resources are added into the queueing systems where local throughput constraints are violated and these queueing systems are called violation handling points.

Algorithm 2 (Step 1) explains the violation handling point selection strategy. When a throughput violation is detected at checkpoint S_p (line 1), we need to calculate the average response time (ART) of workflow activities in each queueing system during S_{p-1} and S_p (lines 2–3). If the ART of a queueing system exceeds the response time constraint (RTC), the local throughput constraint will be violated inevitably, and this queueing system is selected as a handling point (lines 4–5).

(2) **How many resources are needed?**

Adding extra service instances into a queueing system can increase the throughput of cloud services and reduce the average response time of workflow activities. Here, an essential question aroused is "to timely and completely handle temporal violations, does adding more extra resources mean better effectiveness?" Our answer is "not always".

Since it is impossible to obtain a closed formula to represent the probability distribution of general distribution “G” in M/G/m/m+r and G/G/m/m+r models, the numerical relationship between the number of service instances and average response time is not clear. Therefore, we conduct testing experiments and use the results as reference.

Algorithm 2. Temporal violation handling strategy

```
Require:   – Target on-time completion rate $\theta\%$.
           – Throughput consistency state $\alpha\%$ at checkpoint $S_p$.
           – Response time constraint (RTC) of each kind of activity.
           – Time interval $k * bt$ bewteen checkpoint $S_p$ and candidate checkpoint $S_{p+1}$,
             which is achieved by the adaptive checkpoint selection strategy.
Ensure:    – Add new resources with proper lifecycles into selected handling points.
Step 1:    Temporal violation handling point selection
1:         If $\alpha\% \leq \theta\%$
2:             For i=1:n
3:                 Calculate average response time (ART) of activities in each QS;
4:                 If $ART_i > RTC_i$
5:                     $QS_i$ is selected as a handling point;
6:                     Continue to Step 2;
7:                 End if
8:                 i=i+1;
9:             End for
10:        End if
Step 2:    Violation handling at handling points
11:        If newResource=false
12:            Add one extra resource with basic lifecycle L into $QS_i$
13:        Else
14:            Check residual life (RL) of the new resource;
15:            If RL < $k * bt$
16:                RL=RL+L;          //append a basic lifecycle
17:            End if
18:        End if
```

We conduct two rounds of experiments to figure out how response time changes with the number of cloud service instances in queueing systems with the above two different queueing models. In the two queueing systems, the average execution time of workflow activities is 6.1729 s and 13.5686 s. The minimum numbers of service instances based on queueing rule are 6 and 12 (at the worst level of service). When we add the first service instance into the two queueing systems, the average response time of both systems declines dramatically. Afterwards, the average response time is gradually getting close to the average execution time but has no evident decrease despite that more service instances are being recruited. The reason is that adding new service instances can only decrease the waiting time of workflow activities in the queue.

Therefore, we only add one extra service instance into the queueing system for violation handling to achieve the best cost-effectiveness. Even if in some cases adding one service instance may not be sufficient to compensate all throughput deficit, the handling strategy will be called again timely to add another service instance as temporal behaviors are still constantly monitored by our temporal verification strategy.

(3) **How long the lifecycle of these resources should be?**

Algorithm 2 (Step 2) depicts how to determine the lifecycle of new resources for temporal violation handling. If there is currently no recruited resource in the queueing system, then we add a new resource with a basic lifecycle L (lines 11–12). The real lifecycle $T = m{*}L$ $(m = 1, 2, 3, \ldots)$. If there is already a recruited resource in the queueing system (this resource is added in the queueing system at previous checkpoints for violation handling and has not expired yet), then we check the residual time of this resource. If $RL < k * bt$ (namely the resource will expire before the next candidate checkpoint S_{p+1}), we extend its lifecycle by an extra basic lifecycle (lines 14–17).

By adaptively extending the resource's lifecycle, our strategy can make sure only one extra resource is required in the queueing system for temporal violation handling.

4 Evaluation

4.1 Experimental Settings

The simulation experiments are conducted in our cloud workflow system SwinFlow-Cloud [9]. First, we simulate a continuous running of a large number of parallel workflow instances. Basic experimental settings are similar to the settings our previous work [10]. Arrival time and execution time of activities follow general distribution which are simulated by *Simulink*[1]. Basic time unit *bt* is set as an equal interval of 30 s. Basic lifecycle of recruited resources L is set as 1 min.

We compare our approach TV_{adap} with the following two representative strategies:

- TV_{fixed}: It selects checkpoints from a collection of time points along system timeline with fixed interval. This strategy handles temporal violations by adding one extra resource with fixed lifecycle into each selected queueing system [10].
- TV_{acti}: It selects every workflow activity as candidate checkpoint [11]. If a temporal violation is detected, one resource with a fixed lifecycle is added into each selected queueing system (if there is no extra resource in the queueing system).

To get the baseline results for comparison purpose, we record the on-time completion rates of workflow instances under natural situation, i.e., without any temporal verification or violation handling strategies (denoted as **NIL**).

In business scenario, a best strategy should be the one that can reach the target on-time completion rate with high cost-effectiveness. Cost-effectiveness is measured by the average resource consumption for every 1% increment from the baseline. The formula is as follows:

$$\frac{H}{\gamma'\% - \gamma\%} \tag{2}$$

where H denotes the total number of basic lifecycle of recruited resources needed by each strategy, $\gamma\%$ is the baseline on-time completion rate and $\gamma'\%$ is the on-time completion rate achieved by each strategy.

[1] Simulink: https://www.mathworks.com/products/simulink.html.

4.2 Experimental Results

First, we compare TV_{adap} with the other two strategies under different batches of workflows. The workflow size is 25. In Fig. 1, each strategy can significantly improve on-time completion rate when compared with the baseline **NIL**, and all of them can reach the target on-time completion rate of 90%. However, in Table 1 the number of checkpoints and resources consumption are obviously different. Since TV_{acti} is working at each workflow activity, the number of checkpoints is much more than the other two strategies, and it can detect more temporal violations. But its resource consumption for violation handling is several times higher than the other two strategies. TV_{fixed} achieves a slightly higher on-time completion rate than TV_{adap}, but it consumes more resources. In contrast, TV_{adap} can reach the target on-time completion rate with the lowest resource cost. Also, it achieves the highest verification efficiency with the minimum number of checkpoints.

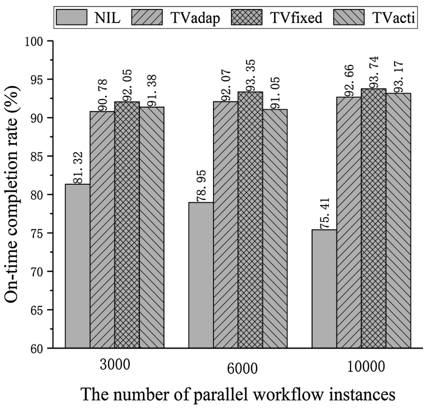


Fig. 1. On-time completion rates with different number of instances

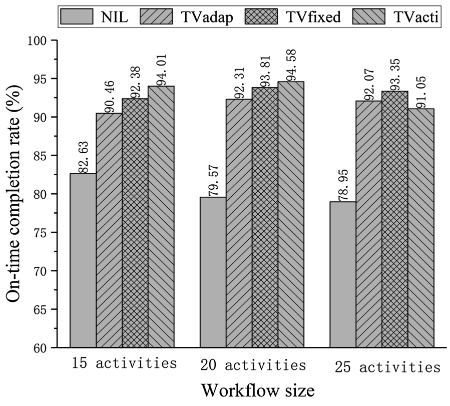


Fig. 2. On-time completion rates with different workflow sizes

Figure 2 and Table 2 show the experimental results under different workflow sizes. The number of parallel workflow instances is 6000. Similar to the above experiment results, all the three strategies can reach 90% on-time completion. Compared with TV_{fixed} and TV_{acti}, TV_{adap} is the most cost-effective, which can achieve target on-time completion with the least resource consumption.

Table 1. Experimental results with different number of workflow instances

Strategies	Checkpoints			Resource lifecycle			Average resource consumption for every 1% increment from the baseline		
	3000	6000	10000	3000	6000	10000	3000	6000	10000
TVadap	**86**	**102**	**121**	**118**	**141**	**252**	**12.47**	**10.75**	**14.61**
TVfixed	132	145	159	172	169	336	16.03	11.74	18.33
TVacti	832	1471	2984	576	902	1685	57.26	74.54	93.36

Table 2. Experimental results with different workflow sizes

Strategies	Checkpoints			Resource lifecycle			Average resource consumption for every 1% increment from the baseline		
	15	20	25	15	20	25	15	20	25
TVadap	**94**	**101**	**102**	**113**	**116**	**141**	**14.43**	**9.11**	**10.75**
TVfixed	152	159	145	141	148	169	14.46	10.39	11.74
TVacti	957	1098	1471	623	816	902	54.75	67.44	74.54

5 Conclusion and Future Work

To achieve on-time completion of time-constrained business cloud workflows, temporal violations occurred at workflow runtime need to be timely detected and handled. In this paper, we present a temporal verification strategy and a temporal violation handling strategy to tackle the problem of "Where" and "How" respectively for handling temporal violations. Considering the fluctuation of workflow temporal behaviors, the idea of "adaptiveness" is introduce into our strategy design. Compared with conventional non-adaptive strategies, our strategies can achieve both higher efficiency and better cost effectiveness.

In the future, we plan to extend the proposed strategies to a more complex environment where instances of different business workflow are mixed in the batch of parallel workflow instances.

Acknowledgement. The authors would like to acknowledge the support provided by the grants of the National Natural Science Foundation of China (61572374, 61300042, U163620068, U1531122, U1135005), the Academic Team Building Plan from Wuhan University (WHU2016012), and the Australian Research Council Discovery Project (DP180100212).

References

1. Kumar, A., Sabbella, S.R., Barton, R. R.: Managing controlled violation of temporal process constraints. In: 13th International Conference on Business Process Management, pp. 280–296 (2015)
2. Wegener, J., Grochtmann, M.: Verifying timing constraints of real-time systems by means of evolutionary testing. Real-Time Syst. **15**(3), 275–298 (1998)
3. Falcone, Y., Havelung, K., Reger, G.: A tutorial on runtime verification. J. Eng. Dependable Softw. Syst. **34**, 141–157 (2013)
4. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Exception handling patterns in process-aware information systems. BPM Center Report BPM-06-04, BPMcenter.org (2006)
5. Liu, X., Wang, D., Yuan, D., Wang, F., Yang, Y.: Workflow temporal verification for monitoring parallel business processes. J. Softw.: Evol. Process. **28**(4), 286–302 (2016)

6. Liu, X., Wang, D., Yuan, D., Yang, Y.: A novel deadline assignment strategy for a large batch of parallel tasks with soft deadlines in the cloud. In: Proceedings of 15th IEEE International Conference on High Performance Computing and Communications & 10th IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), pp. 51–58 (2013)
7. Khazaei, H., Mišić, J., Mišić, V.B.: Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. IEEE Trans. Parallel Distrib. Syst. **5**, 936–943 (2011)
8. Luo, H., Liu, J., Liu, X., Yang, Y.: Predicting temporal violations for parallel business cloud workflows. Softw.: Pract. Exp. **48**(4), 775–795 (2018)
9. Cao, D., Liu, X., Yang, Y.: Novel client-cloud architecture for scalable. In: Proceedings of the 14th International Conference on Web Information Systems Engineering (WISE), pp. 270–284 (2013)
10. Luo, H., Liu, X., Liu, J., Yang, Y.: Propagation-aware temporal verification for parallel business cloud workflows. In: Proceedings of IEEE International Conference on Web Services, pp. 106–113 (2017)
11. Wang, F., Liu, X., Yang, Y.: Necessary and sufficient checkpoint selection for temporal verification of high-confidence cloud workflow systems. Sci. China Inf. Sci. **58**(5), 1–16 (2015)

Towards Creating Business Process Models from Images

Neelamadhav Gantayat, Giriprasad Sridhara(✉), Anush Sankaran, Sampath Dechu, Senthil Mani, and Gargi B. Dasgupta

IBM Research AI, Bengaluru, India
{neelamadhav,girisrid,anussank,sampath.dechu,sentmani,gaargidasgupta}@in.ibm.com

Abstract. Business process modeling is an integral task needed for efficient running of business operations. Often process models remain buried in unstructured documents as images or screenshots. Such embedded process model images may become quickly obsolete as the underlying business process evolves. Thus, there is value in digitizing the unstructured images. We propose a novel automated solution to transform a process model image into the standard Business Process Model and Notation (BPMN) format. Our deep-learning based approach performs well in practice achieving good precision and recall.

Keywords: Business process models · Image processing
Deep learning

1 Introduction

Business Process Management (BPM) remains a central, foundational element of running organizations today [1]. Companies use business process models for documenting various business operations such as the process for hiring a new employee, clearing a travel reimbursement claim and so on. Often process models remain buried in unstructured documents as images or screenshots. They could be drawn using standard drawing tools like PowerPoint or specialized modelling tools such as Visio. In either case, such embedded process model images may become quickly obsolete as the underlying business process evolves. This makes transitioning of tasks and maintenance of process know-how adhoc and manual.

Thus, there is a lot of value in digitizing the unstructured images using image processing technologies. The digitized flows could be used to understand deviations in actual process operations or be updated to reflect evolution of the process itself.

Standard off-the-shelf image recognition tools (such as Vision API (Google), Watson Visual Recognition (IBM)) cannot recognize the various different shapes and connectors in business process model images drawn according to the Business Process Model and Notation (BPMN) standard [2].

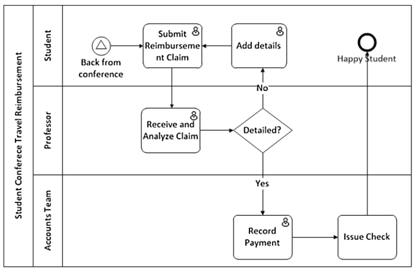
C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 100–108, 2018.
https://doi.org/10.1007/978-3-030-03596-9_7

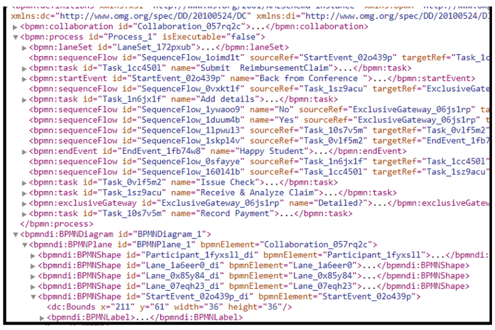
In this paper, we provide a technique to automatically identify images describing business processes and converting them to the standard BPMN format (as shown in the left and right parts of Fig. 1, respectively). To the best of our knowledge we are the first to provide such a system.

Our specific contributions in this paper are as follows:

1. Given any image, automatically identify if the image represents a business process (using a Convolutional Neural Network (CNN)).
2. Given a business process image, automatically identify 64 different kinds of process model shapes (using another CNN).
3. Using optical character recognition techniques, extract text from the different shapes of a process model image.
4. Identify the flow of activities in a business process image using computer vision techniques.
5. Generate the output BPMN XML representing the input image.

The remainder of this paper is structured as follows: Sect. 2 describes our deep learning based approach to transform a given image into a business process model. Section 3 shows our evaluation while Sect. 4 puts our work in the context of related works. We conclude in Sect. 5.





```
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI
▶<bpmn:collaboration id="Collaboration_057rq2c">...</bpmn:collaboration>
▼<bpmn:process id="Process_1" isExecutable="false">
  ▶<bpmn:laneSet id="LaneSet_172pxub">...</bpmn:laneSet>
   <bpmn:sequenceFlow id="SequenceFlow_1oimd1t" sourceRef="StartEvent_02o439p" targetRef="Task_1c
  ▶<bpmn:task id="Task_1cc450l" name="Submit  ReimbursementClaim">...</bpmn:task>
  ▶<bpmn:startEvent id="StartEvent_02o439p" name="Back from Conference ">...</bpmn:startEvent>
   <bpmn:sequenceFlow id="SequenceFlow_0vxkt1f" sourceRef="Task_1sz9acu" targetRef="ExclusiveGate
  ▶<bpmn:task id="Task_1n6jx1f" name="Add details">...</bpmn:task>
   <bpmn:sequenceFlow id="SequenceFlow_1ywaoo9" name="No" sourceRef="ExclusiveGateway_06js1rp" ta
   <bpmn:sequenceFlow id="SequenceFlow_1duum4b" name="Yes" sourceRef="ExclusiveGateway_06js1rp" t
   <bpmn:sequenceFlow id="SequenceFlow_1lpwu13" sourceRef="Task_10s7v5m" targetRef="Task_0vlf5m2"
   <bpmn:sequenceFlow id="SequenceFlow_1skpl4v" sourceRef="Task_0vlf5m2" targetRef="EndEvent_1fb7
  ▶<bpmn:endEvent id="EndEvent_1fb74w8" name="Happy Student">...</bpmn:endEvent>
   <bpmn:sequenceFlow id="SequenceFlow_0sfayye" sourceRef="Task_1n6jx1f" targetRef="Task_1cc450l"
   <bpmn:sequenceFlow id="SequenceFlow_160l41b" sourceRef="Task_1cc450l" targetRef="Task_1sz9acu"
  ▶<bpmn:task id="Task_0vlf5m2" name="Issue Check">...</bpmn:task>
  ▶<bpmn:task id="Task_1sz9acu" name="Receive & Analyze Claim">...</bpmn:task>
  ▶<bpmn:exclusiveGateway id="ExclusiveGateway_06js1rp" name="Detailed?">...</bpmn:exclusiveGatew
  ▶<bpmn:task id="Task_10s7v5m" name="Record Payment">...</bpmn:task>
 </bpmn:process>
▼<bpmndi:BPMNDiagram id="BPMNDiagram_1">
  ▼<bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Collaboration_057rq2c">
    ▶<bpmndi:BPMNShape id="Participant_1fyxsll_di" bpmnElement="Participant_1fyxsll">...</bpmndi:
    ▶<bpmndi:BPMNShape id="Lane_1a6eer0_di" bpmnElement="Lane_1a6eer0">...</bpmndi:BPMNShape>
    ▶<bpmndi:BPMNShape id="Lane_0x85y84_di" bpmnElement="Lane_0x85y84">...</bpmndi:BPMNShape>
    ▶<bpmndi:BPMNShape id="Lane_07eqh23_di" bpmnElement="Lane_07eqh23">...</bpmndi:BPMNShape>
    ▼<bpmndi:BPMNShape id="StartEvent_02o439p_di" bpmnElement="StartEvent_02o439p">
       <dc:Bounds x="211" y="61" width="36" height="36"/>
     ▶<bpmndi:BPMNLabel>...</bpmndi:BPMNLabel>
```

Fig. 1. Sample business process model image (left) and its BPMN XML (right).

2 Approach (System Architecture and Implementation)

In this section, we describe our approach. The input to our system is an image (in standard formats such as JPEG, PNG) and the output is an XML in the BPMN format (as shown in the left and right parts of Fig. 1, respectively).

A high-level architecture diagram of our approach is shown in Fig. 2. Each step in our approach is described in a separate sub-section below:

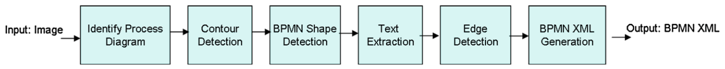


Fig. 2. System architecture diagram.

2.1 Identifying Business Process Model Images

We used a binary classifier to distinguish between business process model images and non-business process model images. To train the binary classifier, we need training data. Essentially, we need a large number of BPMN images as shown in the left part of Fig. 1. For the model to learn effectively, there needs to be variety in training data. Figure 1 has eight different *activities* spread across three *swim-lanes*. It has five tasks (rectangles), one decision or gateway (rhombus/diamond) and two events (circles). There are other basic BPMN shapes as well which are not used in Fig. 1. Thus, we need to create variants of Fig. 1 with different number of tasks, gateways, events, swimlanes and other BPMN shapes. using the Activiti Java library. We programatically created 35,000 BPMN XML files with varying number of swim-lanes and varying number of BPMN entities such as tasks, events, gateways. We then imported the BPMN XML files into an open source modeling tool for BPMN, http://bpmn.io/ to render the process models. Finally, we automatically downloaded the resulting model images using Selenium, the browser automation tool. Thus, we gathered 35,000 business process model images.

To handle variations in rendering across modeling tools, we also used Microsoft Visio to generate another 20,000 business process model images. We used the Visual Basic for Applications (VBA) to programatically create 20,000 process models in Visio and exported them as PNG images. Using *Activiti*, an open source java BPM library we created another 10,000 images. Finally, we created 5000 BPMN images from another modeling tool, IBM Blueworks Live. We thus generated 70,000 business process model images overall.

We also created an equal number of non-process model images. To create this set of images, we could have downloaded random images from the web. However, such a set would *not help distinguish* between a process model (as in Fig. 1) and a similar looking diagram such as a flow chart or a system architecture diagram (as in Fig. 2). We thus automatically created charts and other diagrams that look *superficially similar* to process models.

We set aside 20% of the above set of 140,000 images (i.e., 28,000 images) for evaluation (described in Sect. 3). With the remaining 112,000 images we trained different binary classifiers [3] as shown below:

1. Neural Network (Multi Layer Perceptron [MLP])
2. Support Vector Machines
3. Random Forests
4. Decision Tree
5. Naive Bayes
6. Logistic Regression.

During training, each image was first resized to a resolution of 224 × 224 pixels and passed through a popular Convolutional Neural Network (CNN) called *VGG19*. We extracted 4096 features from its last convolutional layer, which were then used for classification with each of the above classifiers. The evaluation of this model is described in Sects. 3.1 and 3.2.

2.2 Contour Detection

We now describe our approach to automatically extracting contours. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. The contours are a useful tool for shape analysis, object detection and recognition.

The goal here is that given a business process model image input as in Fig. 1, we want to identify contours such as the contour denoting the rectangle around the shape with the text "Submit Reimbursement Claim". A shape may have sub-contours (especially if there is an icon inside it as in Fig. 3), while the entire Fig. 1 can be considered a contour. Thus, we need to find contours at the appropriate level (i.e., the level of shapes). We use the OpenCV library [4] to find contours as described below:

We first find all the contours recognized by OpenCV in the input image. We then check if the contour is the outer-most contour (which represents the entire figure) or a child contour i.e., a contour within another contour. We ignore inner-most child contours and the outer-most contour. The remaining contours are at the level of the swim-lanes, shapes and edges, which is the appropriate level for our shape detection described next.

2.3 Identifying Different BPMN Shapes

Here we followed an approach similar to identifying whether an image represents a business process model as described in Sect. 2.1. We used the same set of six classifiers, with the difference being that these classifiers were *not binary* but given a *contour*, were trained to classify the *contour* as being one of the 64 different BPMN shapes, a subset of which are shown Fig. 3.

As before, we need to create training data for these shapes. We will describe the procedure for one shape, viz., the *start shape* denoted by a *circle* (the very first shape in the first row of Fig. 3): We drew this shape using the tool *bpmn.io*. We then downloaded the resulting BPMN XML and programmatically created size variants of the same shape by varying its properties (For example, altering the radius for a circle, changing width and height for a rectangle, varying text in the shape and so on). With each variation, we used Selenium to import the varied XML into the *bpmn.io* tool and then downloaded the rendered image. We thus created 3,000 images for one shape. We repeated the process for all shapes. These 192,000 images were used these to train *another* model whose task is to distinguish between the 64 different shapes.

Here too we scaled the images to a resolution of 224 by 224 pixels and passed them through the *VGG19* Convolutional Neural Network (CNN) and used the

4096 features to classify a shape as belonging to one of 64 types. The evaluation of this model is described in Sects. 3.1 and 3.2.

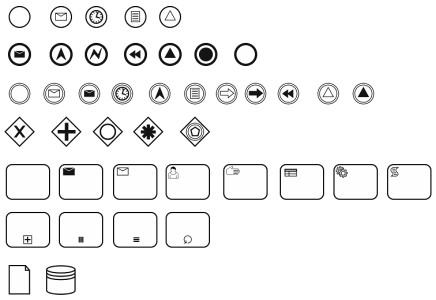


Fig. 3. A subset of the BPMN shapes used in training

2.4 Text Extraction from Shapes

We now describe our text extraction technique. Most of the shapes have the text inside, but some shapes have the text below as shown in Fig. 1. Typically, *tasks* have text inside the shape, whereas shapes like *start*, *end* and *certain kinds of gateways* contain the text below it. We used the well-known optical character recognition tool, *Tesseract* [5] to extract text. As Tesseract is a well-proven tool we did not evaluate text extraction.

2.5 Sequence Flow Detection (Edge Detection)

Here we describe the flow (edge) detection. The challenging part in edge detection is how in detecting the edge *direction*. The arrow heads which denote the edge direction are actually small lines and may not be detected by existing line detection algorithms. Thus, we need other approaches to detecting the direction. We partition the contour (i.e., a small rectangle around the edge) into two equal halves. The half with the arrow head will have more pixels and intensity than the half without the arrow head. Thus, we can find the direction. Evaluation of edge detection is described in Sect. 3.2.

2.6 Generating Output

Algorithm 1 shows the outline of how we generate the XML (as in the right half of Fig. 1) which represents the output of our technique. We used the *Activiti* java library to aid in the XML generation. As the output generation is primarily an engineering task consisting of putting together identified shapes and edges, we do not evaluate it.

Algorithm 1. Output Generation in BPMN

```
1 foreach identified shape do
2     create a corresponding shape object in BPMN
3     create bounds for the object (i.e., location, width,height) using dimensions from the
      original figure
4 end
5 foreach identified edge do
6     create a corresponding flow between the shapes connected by edge
7     create a waypoint i.e, line length, origin and endpoints using dimensions from the
      original figure
8 end
```

3 Evaluation

In this section, we describe our evaluation. Our experiments were geared towards answering the following research questions (RQ):

– **RQ1:** With what accuracy can we distinguish between process model images and non-process model images?
– **RQ2:** How accurately can we distinguish between the 64 BPMN shapes?
– **RQ3:** What is the accuracy of our edge detection (i.e., sequence flow)?

To answer the above research questions, we used two types of data:

1. Automatically generated data
2. Data obtained from our clients.

In the following sub-sections, we describe our evaluation with the above two sets of data. Across both sets, the standard evaluation metrics of *precision*, *recall* and *F1-Score* are used.

3.1 Evaluation with Generated Data

As described in Sect. 2, we had accumulated a set of 140,000 images consisting of process models and non-process models. Of this, we had set aside 20% (i.e., 28000) of the images for testing. We used these images to answer our research question, *RQ1*. While we performed the experiments with the different classifiers described in Sect. 2, we show the results for the MLP (Multi Layer Perceptron) classifier [3] as it had the best results. The first row of Table 1 shows the results. We achieve a very high precision and recall.

For answering *RQ2* i.e., precision and recall of shape identification, we gathered 20% of the 192,000 shapes we had generated for shape detection training. These 38,400 shapes were used to find the precision and recall of shape detection. The second row of Table 1 shows the results. We again achieve a very high precision and recall.

For BPMN image identification and shape detection, we used deep learning and hence generated training data. Thus, we evaluated on a *held-out* set of data. However, for *edge detection* i.e., *RQ3* we did not use a learning approach, hence we do not evaluate it here, its evaluation is described in the next sub-section.

Table 1. Evaluation on generated data (MLP classifier)

RQ	Precision (%)	Recall (%)	F1-Score
RQ1: BPMN image identification	1.0	0.98	0.99
RQ2: BPMN Shape detection	1.0	0.96	0.98

3.2 Evaluation with Client Data

We gathered twenty (20) random documents (Powerpoint or Word) from our clients, which had a mixture of business process model images and other images such as charts, app screen-shots and so on. Overall, there were 96 images with 24 of them being business process model images. In these 24 business process models, we had overall 200 shapes and 210 edges.

Note that unlike the generated data set, here we do not have a readily available oracle or gold-set i.e., here we have to manually inspect the documents and the images to find the total number of business process model images, shapes and edges. Further, we have to manually compare the output from our technique with the actual image to compute precision and recall for images, shapes and edges. As this is a time consuming task, we have lesser data points in this evaluation as compared with the evaluation on generated data.

Table 2 shows the results. Similar to the evaluation on generated data, here too the Multi-Layer Perceptron (MLP) performed the best and its results are depicted. As can be seen from the table, our technique does achieve good precision and recall in BPMN image, shape and edge identification on client data.

Table 2. Evaluation on client data (MLP classifier)

RQ	Precision (%)	Recall (%)	F1-Score
RQ1: BPMN image identification	1.00	0.92	0.95
RQ2: BPMN Shape detection	0.8	0.7	0.75
RQ3: Sequence Flow detection	0.75	0.6	0.67

3.3 Threats to Validity

Our approach is able to identify 64 different types of BPMN shapes, many of which are similar to each other. The overall number of allowed shapes in the official BPMN standard [2] is about 360. Thus, it is possible that our approach may not work well with all variants. However, we believe that with additional training data depicting the shapes not covered so far, our approach can identify all the different BPMN shapes.

4 Related Work

To the best of our knowledge, we are the first to automatically identify images that depict process models and convert a business process model image into the standard BPMN XML format.

Sethi et al. [6] propose an approach to extract and understand deep learning design flow diagrams available in a research paper and convert it into execution ready source code. In their approach, there is no need to detect different kinds of shapes (as the diagram only contains rectangles). The vocabulary is also limited, making text recognition easier and the sequence flow is linear (top-down). Thus, their approach cannot be used *as is* for our problem.

Other broad category of work deals with generating synopsis from images such as tables and figures in papers [7], extracting information from line graphs [8], charts [9]. *None* of the these work can be used partly or entirely in our approach.

5 Conclusion

We described a novel approach using deep learning to automatically identify process model images. We then proposed an automated approach to transform a process model image into the standard BPMN XML format. This conversion involved identifying different BPMN shapes using convolutional neural networks, detecting edges between the shapes via image processing, extracting text from the shapes utilizing optical character recognition and finally generating the BPMN XML. We demonstrated empirically that our approach works well with good precision and recall.

References

1. Hull, R., Motahari Nezhad, H.R.: Rethinking BPM in a cognitive world: transforming how we learn and perform business processes. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 3–19. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_1
2. OMG: About the Business Process Model and Notation Specification Version 2.0 (2011). https://www.omg.org/spec/BPMN/2.0/About-BPMN/
3. Mitchell, T.M.: Machine Learning. McGraw-Hill, Inc., New York City (1997)
4. ITSEEZ: The Open Source Computer Vision Library Reference Manual, 2.4.9.0 edn. April 2014
5. Smith, R.: An overview of the tesseract OCR engine. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, vol. 2 (2007)
6. Sethi, A., Sankaran, A., Panwar, N., Khare, S., Mani, S.: DLPaper2Code: autogeneration of code from deep learning research papers. CoRR abs/1711.03543 (2017). http://arxiv.org/abs/1711.03543
7. Bhatia, S., Mitra, P.: Summarizing figures, tables, and algorithms in scientific publications to augment search results. ACM Trans. Inf. Syst. **30**(1), 3:1–3:24 (2012)

8. Ray Choudhury, S., Giles, C.L.: An architecture for information extraction from figures in digital libraries. In: Proceedings of the 24th International Conference on World Wide Web (2015)
9. Böschen, F., Scherp, A.: Formalization and preliminary evaluation of a pipeline for text extraction from infographics (2015)

Service Trust and Security

Empowering Business-Level Blockchain Users with a Rules Framework for Smart Contracts

Tara Astigarraga[1], Xiaoyan Chen[2], Yaoliang Chen[3], Jingxiao Gu[2], Richard Hull[1](✉), Limei Jiao[2], Yuliang Li[4], and Petr Novotny[1]

[1] IBM T.J. Watson Research Center, Yorktown Heights, USA
{asti,hull,p.novotny}@us.ibm.com
[2] IBM China Research Laboratory, Beijing, China
{xiaoyanc,gjxgu,jiaolm}@cn.ibm.com
[3] Fudan University, Shanghai, China
yaoliangchen15@fudan.edu.cn
[4] Megagon Labs, UC San Diego, San Diego, USA
yul206@eng.ucsd.edu

Abstract. The importance and adoption of Blockchain to support secure and trusted collaborations between businesses continues to grow. In today's practice, most Blockchain smart contracts (which capture the business processing logic) are written primarily by software developers. To enable widespread adoption of Blockchain, business analysts and subject matter experts will need to have direct access to the smart contract logic, including the abilities to understand, modify, and create substantial portions of that logic. This paper describes a fully functioning framework and system for specifying and executing smart contracts in which the core logic is specified by a controlled English, business-level rules language. The framework includes a browser-based smart editor for rules; a parser generator that enables substantial variation in the rules syntax; code generation that maps to a RETE based rules engine; and execution of the rules in either on-chain (using Hyperledger Fabric) or off-chain modes. The paper describes the rules framework and possible extensions, and identifies key aspects of Blockchain that impact the implementation.

1 Introduction

The shared ledger and Blockchain paradigms hold the promise of transforming the ways that businesses collaborate by enabling a single source of truth, and increased transparency through shared agreement about how business workflows will be conducted. In particular, "smart contracts", i.e., the programs that guide the execution of transactions on Blockchain, are visible to and agreed upon by the participants in a Blockchain-enabled collaboration. As such, Blockchain for business collaboration opens new research challenges and industrial opportunities in service-oriented computing, in particular in the areas of new styles of business process management, distributed computing, and secure services. Today's smart contracts are primarily created by software developers, using standard programming languages such as Golang, C++ or Java, and/or using domain-specific languages and frameworks such as Ethereum's Solidity [1] and Hyperledger's Composer [2]. However, because of the anticipated volume of business collaborations on Blockchain in coming years, and thus the volume of smart contracts to be created, it is paramount that business-level users be empowered to understand, create and modify smart contracts, or at least large portions of them. This paper describes a Business Collaboration Rules Language (BCRL) framework and implemented system, that enables business-level users to specify and maintain intricate business logic in Blockchain-enabled solutions. In particular, the framework enables the use of the same rules language that can be executed both as smart contracts on the Hyperledger Fabric [3], and in an off-chain rules engine, thereby enabling a more seamless experience for managing overall business collaboration solutions. This paper describes the rules framework, including illustrations of the current rules language, overview of the system architecture and extensibility, and discussion of the implications on the system related to implementation on top of the Hyperledger Fabric.

The importance of empowering business-level users in connection with smart contracts is highlighted in [4]. Recent articles have focused primarily on the business process level of smart contracts, e.g., by providing an implementation of BPMN on top of Ethereum [5,6] or describing how the business artifact approach can leverage the data-centric nature of Blockchain [7]. This paper complements that work by providing a framework for specifying intricate business logic through the use of business rules expressed in a controlled English. The integration of the BCRL framework with workflow-based smart contract frameworks is left as future work.

This paper illustrates the framework with a particular version of the rules language, called Business Collaboration Rules Language version 1.0 ("BCRL 1.0" or "BCRL" for short). This language is inspired largely from the BERL language of IBM's Operational Decision Manager (ODM) product [8]; this was a tactical design decision based in part on enabling more rapid creation of the first implementation. But the framework itself can support substantial extensions and variations to that language. For example, it could be adapted to follow the styles of other ODM languages, SBVR [9], the Oracle [10] and FlexRule [11] business rule languages, etc. One enabler for this is the use of the Business Domain Specific Language (BDSL) parser-generator, which is used to generate both the language parser and a browser-based smart editor; BDSL is an internal component [12] of the ODM product. BDSL includes multiple features specific to the creation of domain-specific languages (DSLs) that are based on controlled natural language, in particular around handling phrasing that would be ambiguous for a traditional LALR parser. A second enabler is the use of a lightweight, prototype, JavaScript-based rules engine called "nanoRETE", which supports the RETE algorithm for rules execution.

The rules engine framework has been implemented on the Hyperledger Fabric in two ways. One modality relies on the fact that Hyperledger Fabric v1.1

provides native support for JavaScript-based smart contracts. In this modality the nanoRETE rules engine and some integration modules are loaded directly into Fabric. The other modality takes advantage of the Hyperledger Composer [2] and runs on Hyperledger Fabric v1.0.

As shown by the examples in this paper, the BCRL framework enables business-level users to express and "own" large portions of the business logic underlying smart contracts. The language itself follows the spirit of other business-level rules languages and can be modified to fit popular syntactic styles, thereby enabling more rapid adoption. The primary contributions of the paper are the development of a framework that enables seamless use of such rules language both on- and off-chain, and the description of how the framework was brought into a fully implemented prototype system.

Section 2 provides an overview of the framework, illustrates BCRL 1.0, and discusses possible extensions. Section 3 overviews the system architecture and describes key components. Section 4 describes the main challenges that arise when embedding a rules framework onto Hyperledger Fabric and how the BCRL framework addresses them. Section 5 describes related languages and research, and Sect. 6 offers brief conclusions.

2 Framework Overview and Illustrations

This section provides an overview of the rules framework, then illustrates it with an example taken from the domain of billing for Technical Service Support (TSS), and concludes with a discussion of some additional rule constructs that can be incorporated into the framework. More details about the architecture, language, and implementation are provided in subsequent sections.

As mentioned in the Introduction, the framework is focused on enabling business-level users to specify and execute possibly intricate rules logic in a Blockchain-enabled solution. The framework enables the use of the same rules engine in two different ways – one on-chain and the other off-chain – to provide a more seamless experience for business-level users who are working on a comprehensive Blockchain-enabled solution.

Figure 1 shows the high-level architecture of the rules framework. As shown in the upper left, the framework includes a template-based editor for specifying the domain model for a given solution. (In some contexts, the domain model might be defined elsewhere, in which case it can be imported as JSON into the solution.) The Smart Editor for Rules is shown in the upper right of the figure.

The primary function of the Code Generation component is to perform code generation of executable rules based on the business-level rules and domain model, and then to deploy them in rules engines both on-chain and off-chain. The on-chain rules are triggered by transaction invocations, and typically result in updates to the ledger and to the Worldstate (which in our architecture is maintained in CouchDB), and may yield notifications about the transaction outcome. For off-chain rules, rules can be loaded into the off-chain Rules Engine Container for execution. This engine acts primarily as a Policy Decision Point, but can also read and update an off-chain database (currently, Cloudant).

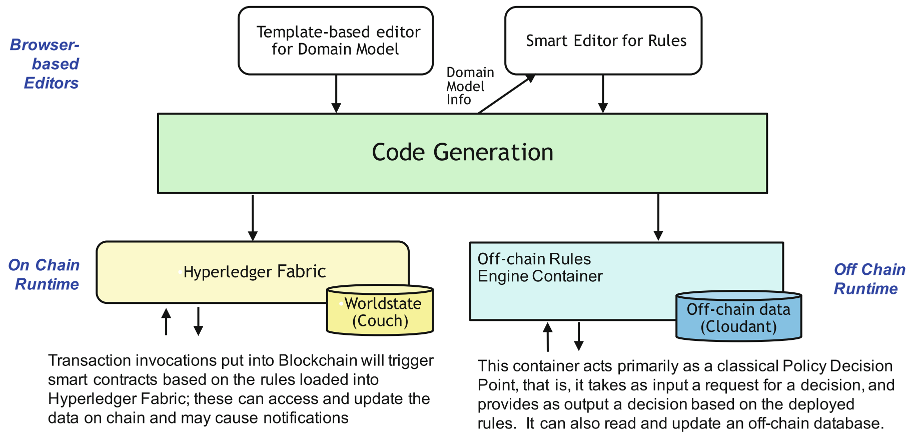

Browser-based Editors
Template-based editor for Domain Model
Smart Editor for Rules
Domain Model Info
Code Generation
On Chain Runtime
Hyperledger Fabric
Worldstate (Couch)
Off-chain Rules Engine Container
Off-chain data (Cloudant)
Off Chain Runtime
Transaction invocations put into Blockchain will trigger smart contracts based on the rules loaded into Hyperledger Fabric; these can access and update the data on chain and may cause notifications
This container acts primarily as a classical Policy Decision Point, that is, it takes as input a request for a decision, and provides as output a decision based on the deployed rules. It can also read and update an off-chain database.

Fig. 1. High-level architecture of rules framework

2.1 Illustration of Rules in Billing Use Case

We now present a case study of the rules framework in action. The example was chosen in part to illustrate several different elements of the rules language currently supported by the framework. This is based on a real-world use case involving the generation of bills for a specific kind of Technical Support Services (TSS). In this example, IBM is providing maintenance for a client, called here ServerFarm, that operates numerous computer servers in data-centers spread across numerous countries. The payment for services may vary by country, and also by the level of service (in this case, either “Next Business Day” or “within 4 Hours”). The monthly bill for a given country and level of service is based on the number of machines being maintained. That number is in turn determined by examining the number of machines being maintained each week, and then taking the maximum weekly quantity for the month.

Figure 2 shows the two main process flows used to manage the Billing processing; these can be thought of as lifecycle models for the two primary entity types arising in the application, namely Weekly Usage Records and Monthly Billing Items. These lifecycle flows might be managed by a BPMN-based engine or other processing engine. We use here an informal, direct representation for these flows that includes constructs germane to Blockchain enabled solutions. This includes responding to events of a given type (black diamond), tasks performed off-chain (dashed line rounded box), and tasks performed on-chain (solid line rounded box). In our example some off-chain tasks are performed manually, and others performed automatically based on rule execution.

The key data sets are shown in Fig. 3. These are depicted more-or-less as tables, but in fact are collections of JSON documents that are stored in noSQL databases. The Install_Base table, which resides off-chain in a Cloudant database, holds data corresponding to contractual agreements between ServerFarm and its customers. Figure 4 shows a representative document, showing a hypothetical

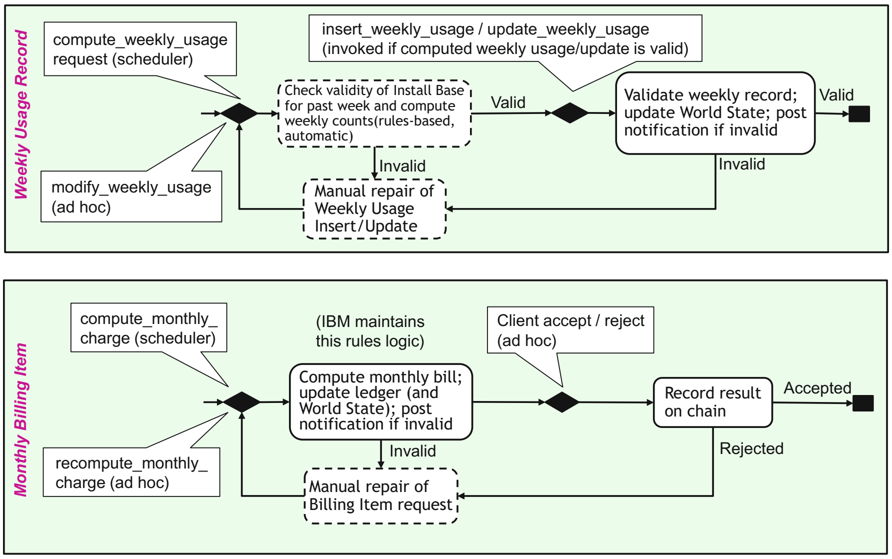

Weekly Usage Record
compute_weekly_usage request (scheduler)
modify_weekly_usage (ad hoc)
Check validity of Install Base for past week and compute weekly counts(rules-based, automatic)
insert_weekly_usage / update_weekly_usage (invoked if computed weekly usage/update is valid)
Valid
Validate weekly record; update World State; post notification if invalid
Valid
Invalid
Invalid
Manual repair of Weekly Usage Insert/Update
Monthly Billing Item
compute_monthly_charge (scheduler)
recompute_monthly_charge (ad hoc)
(IBM maintains this rules logic)
Compute monthly bill; update ledger (and World State); post notification if invalid
Client accept / reject (ad hoc)
Record result on chain
Accepted
Rejected
Invalid
Manual repair of Billing Item request

Fig. 2. Two high-level process flows that guide the billing example

Install_Base	Rate_Table	Weekly_Usage_Record	Monthly_Billing_Item
customer_name	country	week_start_date	month_start_date
country	offering_category	week_end_date	month_end_date
serial_number	offering_service_rate	country	offering_category
asset_sla		offering_category	country
contract_start_date		quantity	max_weekly_count
contract_end_date			offering_service_rate
			monthly_charge
			weekly_usage_records[]

Fig. 3. Data schema used in billing example

customer in China, and one of their machines that ServerFarm is maintaining offering category (called "asset_sla" in that table) of "NBD", between the specified contract start and end dates.

We now describe the three data sets maintained on the Hyperledger Fabric. Following Hyperledger terminology the objects in these data sets are called "assets"; they are identified by a unique key and have values that are JSON documents that may change over time. Rate_Table assets hold the monthly charges paid by ServerHost to IBM for maintaining one server in a given country and offering category. Weekly_Usage_Record assets hold weekly counts of servers being supported. Monthly_Billing_Item assets hold the monthly charge for the maintenance service for each country and offering category.

We now return to the processing flow for Weekly Usage Records shown in Fig. 2. A scheduler is used to invoke the computation of these records on a weekly basis. A first computation is performed by rules running off-chain in response to a compute_weekly_usage event – an example of the payload of this kind of event is shown in Fig. 5. This typically produces a record holding a weekly usage count for a given country and offering category. That data is used as input for a transaction request of type insert_weekly_usage sent to the Hyperledger Fabric, and results in the weekly usage data being loaded onto the Blockchain. Figure 6 shows a representative Hyperledger asset that would be written into the ledger as the outcome of that sequence of events.

Figure 9 illustrates the smart editor for rules and three of the rules used in response to a compute_weekly_usage event. The rules are in a controlled English, and the editor provides color coding for different syntactic elements.

```
{
  "customer_name": "GSG INC",
  "serial_number": "P3029287XA59288",
  "asset_sla": "NBD",
  "country_name": "China",
  "contract_start_date": "2016-11-22T17:00:00.000Z",
  "contract_end_date": "2017-11-21T07:00:00.000Z",
  "docType": "com.ibm.monthly_charge.Install_Base"
}
```

Fig. 4. Representative JSON document from Install_Base, which identifies one of the machines that ServerHost is maintaining for one of its customers. (Customer name is hypothetical.)

```
{
  "event_type": "compute_weekly_usage",
  "country_name": "China",
  "week_start_date": "2017-10-01T17:00:00.000Z",
  "week_end_date": "2017-10-08T17:00:00.000Z"
}
```

Fig. 5. Representative event, of type compute_weekly_usage, which would trigger an off-chain computation followed by an on-chain computation.

Rule 1.1 illustrates the overall structure of rules, which includes a "when" clause that refers to the type of event being processed, an "if" clause that includes conditions, and a "then" clause that holds one or more actions. In Rule 1.1 the "if" clause is testing whether the offering_category field of the incoming event is outside of the permitted values "NBD" or "4HR".

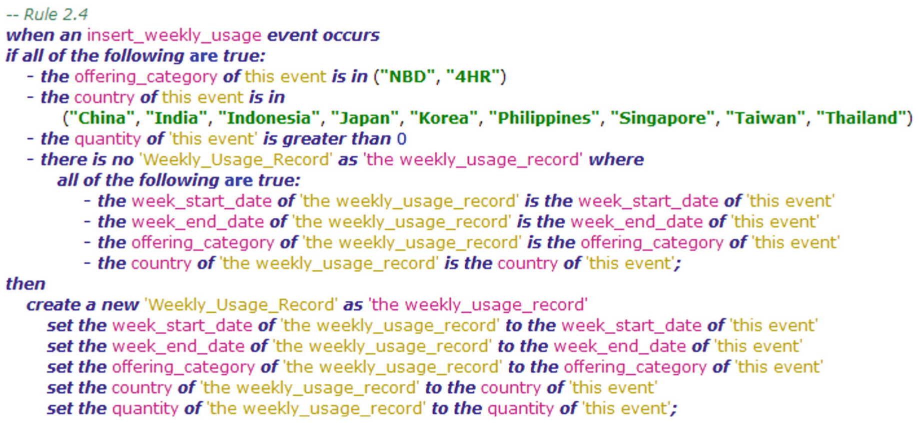
Rule 1.2 provides a simple illustration of syntax checking by the smart editor. Here the keyword 'is' is missing from the "if" clause; this is indicated at the bottom of the screen, and also in a pop-up box if the user mouses over the erroneous text.

Rule 1.3 illustrates several features supported in BCRL 1.0. This includes a fourth building block for the rules, called "definitions", that allows to define sets of records (shown in this figure) and to select individual records (see Fig. 8).

```
{
  "country": "China",
  "docType": "com.ibm.monthly_charge.Weekly_Usage_Record",
  "offering_category": "NBD",
  "quantity": 184,
  "week_start_date": "2017-10-01T17:00:00.000Z",
  "week_end_date": "2017-10-08T17:00:00.000Z",
  "weekly_usage_record_ID": "3b1f0eb0-2ad2-11e8-bb2b-2bd0e4e667c6"
}
```

Fig. 6. Representative value of Hyperledger asset (key-value pair) written onto the ledger as result of off-chain compute_weekly_usage event followed by on-chain insert_weekly_usage transaction

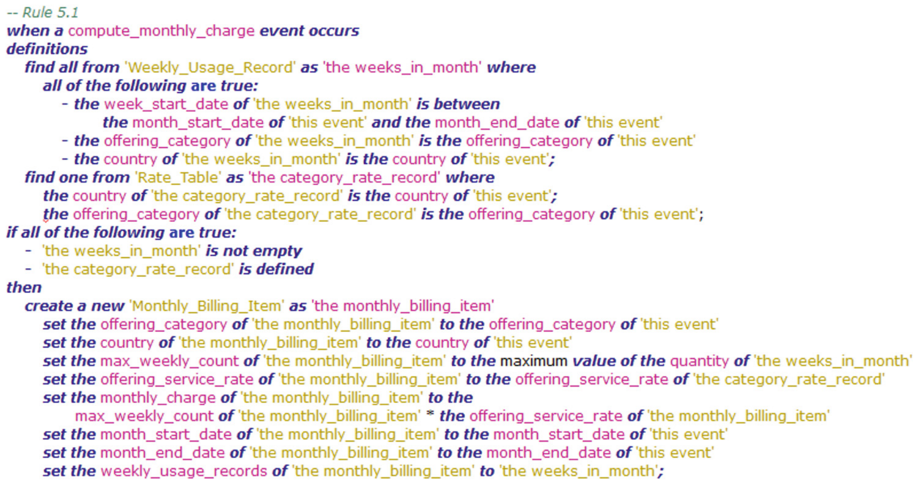
In Rule 1.3 a set designated with variable name "the usage_records" is built from the Install_Base database. A representative document from Install_Base is shown in Fig. 4. The rule builds "the usage_records" using a combination of 'and' ("all of the following are true") and 'or' ("at least one of the following are true") constructs. This rule also illustrates an action of creating a new record and writing it into a database. The built-in function "sum" is used to take a sum of values from the quantity field of the records in 'the usage_records'.



```
-- Rule 2.4
when an insert_weekly_usage event occurs
if all of the following are true:
   - the offering_category of this event is in ("NBD", "4HR")
   - the country of this event is in
        ("China", "India", "Indonesia", "Japan", "Korea", "Philippines", "Singapore", "Taiwan", "Thailand")
   - the quantity of 'this event' is greater than 0
   - there is no 'Weekly_Usage_Record' as 'the weekly_usage_record' where
        all of the following are true:
            - the week_start_date of 'the weekly_usage_record' is the week_start_date of 'this event'
            - the week_end_date of 'the weekly_usage_record' is the week_end_date of 'this event'
            - the offering_category of 'the weekly_usage_record' is the offering_category of 'this event'
            - the country of 'the weekly_usage_record' is the country of 'this event';
then
   create a new 'Weekly_Usage_Record' as 'the weekly_usage_record'
      set the week_start_date of 'the weekly_usage_record' to the week_start_date of 'this event'
      set the week_end_date of 'the weekly_usage_record' to the week_end_date of 'this event'
      set the offering_category of 'the weekly_usage_record' to the offering_category of 'this event'
      set the country of 'the weekly_usage_record' to the country of 'this event'
      set the quantity of 'the weekly_usage_record' to the quantity of 'this event';
```

Fig. 7. Rule for inserting a weekly usage record into blockchain

The outputs of firings of Rule 1.3 will be used as the payload for insert_weekly_usage transactions on the Hyperledger Fabric. Such transactions may result in the firing of Rule 2.4 shown in Fig. 7, (The first digit in the rule numbering scheme correspond to the different types of events/transactions that can lead to rule firing.) This rule shows another capability of BCRL, specifically the ability to perform a "not exists", or said differently, to check that there are no records satisfying a certain property. In this case we check that there is no Weekly_Usage_Record asset already on-chain that corresponds to the same country, offering category, and week. (Updates to an existing Weekly_Usage_Record



```
-- Rule 5.1
when a compute_monthly_charge event occurs
definitions
   find all from 'Weekly_Usage_Record' as 'the weeks_in_month' where
      all of the following are true:
         - the week_start_date of 'the weeks_in_month' is between
                the month_start_date of 'this event' and the month_end_date of 'this event'
         - the offering_category of 'the weeks_in_month' is the offering_category of 'this event'
         - the country of 'the weeks_in_month' is the country of 'this event';
   find one from 'Rate_Table' as 'the category_rate_record' where
      the country of 'the category_rate_record' is the country of 'this event';
      the offering_category of 'the category_rate_record' is the offering_category of 'this event';
if all of the following are true:
   -  'the weeks_in_month' is not empty
   -  'the category_rate_record' is defined
then
   create a new 'Monthly_Billing_Item' as 'the monthly_billing_item'
      set the offering_category of 'the monthly_billing_item' to the offering_category of 'this event'
      set the country of 'the monthly_billing_item' to the country of 'this event'
      set the max_weekly_count of 'the monthly_billing_item' to the maximum value of the quantity of 'the weeks_in_month'
      set the offering_service_rate of 'the monthly_billing_item' to the offering_service_rate of 'the category_rate_record'
      set the monthly_charge of 'the monthly_billing_item' to the
            max_weekly_count of 'the monthly_billing_item' * the offering_service_rate of 'the monthly_billing_item'
      set the month_start_date of 'the monthly_billing_item' to the month_start_date of 'this event'
      set the month_end_date of 'the monthly_billing_item' to the month_end_date of 'this event'
      set the weekly_usage_records of 'the monthly_billing_item' to 'the weeks_in_month';
```

Fig. 8. Rule used to compute monthly charge on blockchain

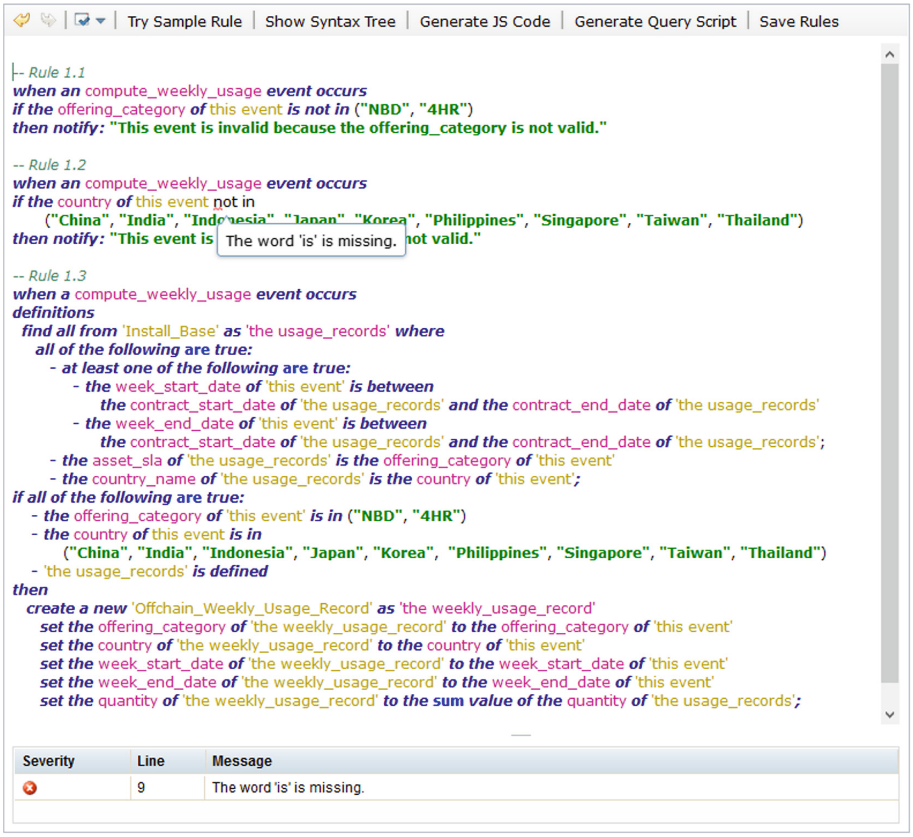
asset can be made using an update_weekly_usage transaction request.) As noted above, Fig. 6 shows a representative Hyperledger asset that will be written onto the ledger after Rules 1.3 and 2.4 have fired.

We pause to comment on the "when" clause in BCRL rules. This is used primarily to provide a clustering and modularity for the overall set of rules. In essence, if an event of a given type is pushed into the rules engine, then all rules with "when"clause referring to that type are eligible for firing, including as the result of rule chaining.

Finally, we describe Rule 5.1 in Fig. 8, which computes Monthly_Billing_Item assets. The rule focuses on all weeks which start within the month. This rule illustrates the construct for selecting a single element from the Rate_Table, using the "find one from" construct and also checking that the resulting record is "defined", which in this case includes a check that only one record was found. The "then" clause illustrates the use of built-in arithmetic functions and the aggregate operator "max". Note also that the definition of one field value of 'the monthly_billing_item', specifically the "monthly_charge" value, can refer to previously defined field values of 'the monthly_billing_item'. A representative asset produced by this rule is shown in Fig. 10.

2.2 Discussion

As illustrated in the examples above, BCRL 1.0 provides business-level users with the ability to express a broad variety of data manipulations, mainly in the area of accessing and manipulating documents (off-chain) and assets (on-chain), and also lists of such objects. Although not highlighted in the above examples, it is important to note that the code generation maps the rules into the nanoRETE engine, which can support rich chaining of the rules.



```
-- Rule 1.1
when an compute_weekly_usage event occurs
if the offering_category of this event is not in ("NBD", "4HR")
then notify: "This event is invalid because the offering_category is not valid."

-- Rule 1.2
when an compute_weekly_usage event occurs
if the country of this event not in
      ("China", "India", "Indonesia", "Japan", "Korea", "Philippines", "Singapore", "Taiwan", "Thailand")
then notify: "This event is  The word 'is' is missing.  not valid."

-- Rule 1.3
when a compute_weekly_usage event occurs
definitions
  find all from 'Install_Base' as 'the usage_records' where
     all of the following are true:
        - at least one of the following are true:
             - the week_start_date of 'this event' is between
                  the contract_start_date of 'the usage_records' and the contract_end_date of 'the usage_records'
             - the week_end_date of 'this event' is between
                  the contract_start_date of 'the usage_records' and the contract_end_date of 'the usage_records';
        - the asset_sla of 'the usage_records' is the offering_category of 'this event'
        - the country_name of 'the usage_records' is the country of 'this event';
if all of the following are true:
    - the offering_category of 'this event' is in ("NBD", "4HR")
    - the country of this event is in
          ("China", "India", "Indonesia", "Japan", "Korea",  "Philippines", "Singapore", "Taiwan", "Thailand")
    - 'the usage_records' is defined
then
   create a new 'Offchain_Weekly_Usage_Record' as 'the weekly_usage_record'
     set the offering_category of 'the weekly_usage_record' to the offering_category of 'this event'
     set the country of 'the weekly_usage_record' to the country of 'this event'
     set the week_start_date of 'the weekly_usage_record' to the week_start_date of 'this event'
     set the week_end_date of 'the weekly_usage_record' to the week_end_date of 'this event'
     set the quantity of 'the weekly_usage_record' to the sum value of the quantity of 'the usage_records';
```

Severity	Line	Message
⊗	9	The word 'is' is missing.

Fig. 9. Illustration of smart rules editor

While BCRL can be used as-is to provide decision support for a variety of applications, there are multiple simplifications and extensions that would be beneficial; we mention some of these here.

BCRL 1.0 is quite verbose. This was an appropriate first step because it provides a completely explicit way for expressing the constructs, a feature that will be useful in contexts where more abbreviated variants may be confusing for some users.

An important streamlining currently underway is to enable the parser and code generation to take advantage of the meta-data about the data sets and the event signatures. For example, in Rule 1.3 (Fig. 9) this would allow for replacing the phrase “the week_start_date of ‘this event’” by “the week_start_date”, since the only relevant object with that attribute is ‘this event’.

In terms of extensions, we see considerable value in enabling a construct of form “then for each <variable> in <defined list>”. Inside that would be a full “definitions-if-then” block, which is to be executed for each element of the defined list.

Another extension would be to permit richer modularity in the specification of rule sets. For example, we could imitate a paradigm found in ODM, which is to allow the specification of several sets of rules that are connected into a flowchart.

3 Implementation of Rules Framework

This section describes the primary components of the BCRL framework, including the smart editor, the parsing, the code-generation, and the deployment both on-chain and off-chain. Some technical considerations specific to operating on Hyperledger Fabric are deferred until the next section.

The rules framework architecture separates execution code of business rules from the other application components, such as rule scheduling, access control and business data storage. This separation helps reduce the costs of application maintenance by allowing the business users to modify the rules as necessary without the need for other code changes. It also allows for maximum re-use of components across on-chain and off-chain rules specification, deployment and execution.

```
{
  "country": "China",
  "docType": "com.ibm.monthly_charge.Monthly_Billing_Item",
  "max_weekly_count": 187,
  "month_end_date": "2017-10-31T17:00:00.000Z",
  "month_start_date": "2017-10-01T17:00:00.000Z",
  "monthly_billing_item_ID": "7cf06c30-2ad2-11e8-bb2b-2bd0e4e667c6",
  "monthly_charge": << omitted -- proprietary >>
  "offering_category": "NBD",
  "offering_service_rate": << omitted -- proprietary >>
  "weekly_usage_records": [
   {
    "country": "China",
    "docType": "com.ibm.monthly_charge.Weekly_Usage_Record",
    "offering_category": "NBD",
    "quantity": 184,
    "week_end_date": "2017-10-08T17:00:00.000Z",
    "week_start_date": "2017-10-01T17:00:00.000Z",
    "weekly_usage_record_ID": "3b1f0eb0-2ad2-11e8-bb2b-2bd0e4e667c6"
   },
   {
    "country": "China",
    "docType": "com.ibm.monthly_charge.Weekly_Usage_Record",
    "offering_category": "NBD",
    "quantity": 187,
    "week_end_date": "2017-10-15T17:00:00.000Z",
    "week_start_date": "2017-10-08T17:00:00.000Z",
    "weekly_usage_record_ID": "51cb5920-2ad2-11e8-bb2b-2bd0e4e667c6"
   },
   ...
}
```

Fig. 10. Part of representative asset written onto the ledger as a result of a compute_monthly_charge transaction. (The rate and monthly charge are proprietary so omitted.)

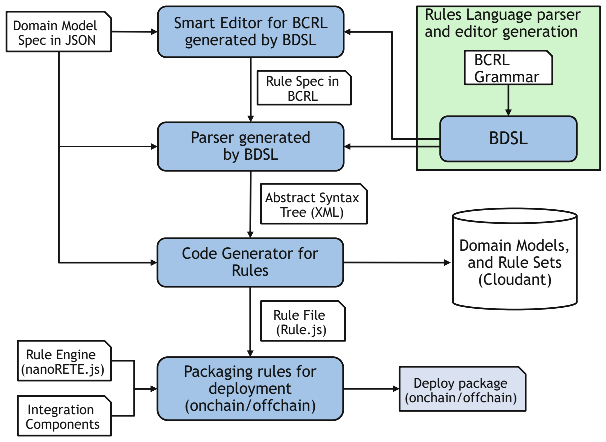

Domain Model Spec in JSON
Smart Editor for BCRL generated by BDSL
Rules Language parser and editor generation
BCRL Grammar
Rule Spec in BCRL
Parser generated by BDSL
BDSL
Abstract Syntax Tree (XML)
Code Generator for Rules
Domain Models, and Rule Sets (Cloudant)
Rule File (Rule.js)
Rule Engine (nanoRETE.js)
Integration Components
Packaging rules for deployment (onchain/offchain)
Deploy package (onchain/offchain)

Fig. 11. Components that support the rules language

An overview of the rule system generation architecture is shown in Fig. 11. The four main components (forming a vertical column in the center of the figure) act as a pipeline that maps BCRL rules into executable code.

The browser-based smart editor (top-most component) and the parser (just below that) were created using the BDSL plugin [12] from IBM's ODM product [8]. BDSL can support grammars that capture highly flexible domain-specific languages based on controlled English (or other natural languages). The smart editor was illustrated in Fig. 9 above. The Smart Editor also incorporates information about the domain model and types of events.

The parser produces an Abstract Syntax Tree (AST) based on BCRL. This AST, along with the domain model information, serves as the input for the code generator. This is the key component in this architecture, which generates runnable JavaScript rule objects from the AST. In order to make the generated rule objects that can be run on both on-chain and off-chain environments, a utility interface is abstracted to isolate the difference between on-chain and off-chain. The main difference is related to database operations. The database operations for on-chain are using Fabric APIs, such as getState, putState, getQueryResult, to access the Hyperledger Fabric Worldstate, which in our case is maintained by a CouchDB instance. (See also Sect. 4.) In contrast, the database operations for off-chain are using Cloudant APIs, such as insert and find, to access the Cloudant database. Therefore, we use a variable named handler to represent the utility interface in generated JavaScript rule objects. Both on-chain and off-chain provide their own utility implementation in runtime.

The code generation process includes the following steps: (1) Traverse the input AST to construct the rule set using the internal rule structure. (2) Generate JavaScript rule objects from the rule sets using templates. (3) Store the generated JavaScript rule objects into a file named rules.js for deployment into the rules engine. (4) Store the generated JavaScript rule objects into the Cloudant if required.

For the rule engine itself we use nanoRETE, a lightweight prototype JavaScript engine that supports the RETE algorithm. (This can run natively on Hyperledger Fabric v1.1, and can also run on Hyperledger Fabric v1.0 with Composer and some integration modules.) The rule engine runs in a data-driven approach. The context of an event changes, which may result in one or more rules being concurrently eligible and scheduled for execution. The rule execution may include querying business data, making computations and finally arriving at some conclusions, possibly including notifications and/or database updates.

The bottom component is in charge of generating deployable packages. The deployable packages include not only the generated rule file, but also supported components, such as the rule engine and utility for database operations. According to on-chain and off-chain requirements, the deployment packaging will choose the right components for packaging. The output packages can be deployed to on-chain or off-chain environments.

The framework also provides a family of straightforward APIs to interact with the deployed on-chain and off-chain rule engines. This includes APIs to deploy rules and also to invoke rules processing. In the on-chain runtime, the APIs communicate with Fabric Client SDK by gRPC and invoke smart contracts on Hyperledger Fabric Blockchain. In the off-chain runtime, the APIs invoke the business rules code directly by gRPC.

4 Implications of Execution on Hyperledger

In this section, we describe three aspects of the Hyperledger Fabric that must be considered, when embedding a rule engine into smart contracts. We first provide a high-level overview of the Fabric architecture and of the transaction processing flow within which the Smart contract is executed. We then highlight the implications and challenges of these to the design of the BCRL framework.

4.1 Fabric Architecture Overview and Transaction Processing Flow

Hyperledger Fabric provides the combination of the immutable shared ledger, high performance of transaction processing, and security and privacy features of private Blockchain. It is architected as a highly distributed system network consisting of several specialized types of components. Note that this architecture differs from standard Blockchain networks such as Ethereum, where all the functionality is integrated into the peer. The center element is the ledger consisting of a chain of blocks, which store the history of transactions along with the identity of the transactions' submitters and endorsers.

The submission of a transaction into the Fabric follows a precise sequence of steps. Initially, a client node submits a "transaction proposal" containing the arguments and name of the function to invoke in the smart contract, to the endorsing peers (i.e., peers which aside from the ledger, host the smart contract). Upon reception, a peer invokes the smart contract in context of the peer's "Worldstate". The Worldstate (similarly as the "state trie" in Ethereum) provides fast and efficient access to the data recorded in the ledger. Next, the client node collects the response along with an endorsement (i.e., the digital signature of the result) from each of the endorsing peers. Note that no modifications of the ledger or Worldstate were made at this point. The client node then validates that the responses are matching and that the collected endorsements adhere to the endorsement (i.e., consensus) policy of the Blockchain network, and submits them to the "ordering service". The ordering service is a centralized component, which receives endorsed transaction proposals, orders them into blocks, and submits the blocks to all peers of the Blockchain network. Upon reception of a block, the peer validates the transactions of the block based on the endorsement policy, commits the block into its ledger and projects the content of the block's transactions onto its Worldstate. Only at this point, the transaction and the state modifications it introduces are added to the ledger.

In the context of the transaction processing flow, a smart contract of our Rule-based framework is hosted on the endorsement peers of the network and executed in response to the transaction proposals submitted by the client nodes. The distributed nature of the network and the sequence of the transaction processing steps has implications on the implementation of the of the smart contract functions and the availability of the data during the smart contract execution. In the following, we outline the key points.

4.2 Eliminating Non-determinism from Smart Contracts

The consensus in the Fabric network is centered on the comparison of the results of execution of the smart contract on multiple peers. Only if the results of the execution match among the required peers, the resultant transaction will be added to the ledger. Thus, to produce a matching outcome, the execution of the smart contact must lead to a deterministic result (i.e., given a set of arguments and a state of the ledger) the smart contract hosted on any of the network peers, must produce the same result.

In Hyperledger Fabric, the smart contract programming infrastructure of golang and JavaScript languages does not prevent the execution of any functions including those, which may produce a non-deterministic output (e.g., `new Date()`, `process.hrtime()`, `Math.random()`, etc.). Therefore, unaware or mistaken use of these functions may lead to a faulty behavior of the smart contract. It is thus the responsibility of the software engineer to ensure that the use of these functions is avoided.

Our Rule-based Framework for Smart contracts prevents producing non-deterministic results similarly as the Solidity language of Ethereum; by eliminating the non-deterministic functions from the set of features available in the grammar. Furthermore, the execution instructions produced by the Codegen are based only on deterministic functions.

This implementation leads to minor restrictions on the smart contract capability. When an output of a non-deterministic function is needed to be included as part of a transaction; the function is executed once, prior to execution of the smart contract. The non-deterministic function execution can be implemented either in the middleware components prior to submitting a transaction proposal to the peers or in the client node invoking the middleware. In either case, the output of the non-deterministic function becomes an argument of the execution of the smart contract, and the same value is submitted to all peers of the network.

4.3 Blockchain Phantom Reads

The invocation of a smart contract triggers a "simulation" of transaction execution. During the processing, invocations of the functions modifying the state of the ledger are recorded into the WriteSet of the resultant transaction and do not impact the state of the ledger and the Worldstate. In effect, during execution of a single transaction, after the modification of an asset (i.e., key and value pair), the subsequent request for the value of the asset will return the unmodified asset value (i.e., phantom read).

The Rule-based framework is designed for smart contracts consisting of long sequences of rules repeatedly modifying and accessing the same assets, incrementally working towards the final result of the transaction. Thus, due to the phantom read behavior, using the native functions returning unmodified values would lead to faulty results. To support the needed functionality we are now encapsulating the Fabric API such that the Worldstate is integrated with an additional caching mechanism. This will allow for accessing the latest modifications necessary for incremental result building. At the same time, this mechanism allows obtaining the currently valid, unmodified values stored in the Worldstate with additional functions available in smart contract processing when needed.

4.4 Worldstate Indexes

In many smart contract contexts the queries against the Worldstate are focused on requesting data based on a key of an asset. However, it is typical of many rules to include associative queries against the Worldstate, that retrieve assets based on properties and ranges of different attributes. The performance of these queries is bound to the number of assets stored in the Worldstate and declines with increasing size of the ledger.

To improve the speed of the asset retrieval, Fabric provides the ability to define custom indexes along with the smart contract. However, it depends on the skill of the smart contract designer, whether and which indexes will be defined as well as how well these will support the execution of the smart contract queries.

The Rule-based grammar provides the opportunity for automated static code analysis of the smart contract, necessary to automatically determine the indexes needed for efficient execution of the associative queries in the rules. This is because the Rule-based grammar leads to a clear expression of the sequences of data access operations as well as it explicitly includes the names of the keys (either the unique keys of the assets or the paths within the assets when the assets contain JSON) used to search for the data.

The index generating algorithm would have two main steps:

1. Identification of candidate keys - in this step the data access operations are analyzed to determine which keys are good candidates for indexes.
2. Generation of indexes - in this step the candidate keys transformed into definitions of the database indexes.

The generation of the indexes can be executed as part of the Codegen and will allow for transparent optimization of the overall performance of the rule-based smart contract.

5 Related Work

In the early days, all smart contracts were programmed using full-purpose programming languages such as Golang, C++ or Javascript.

Solidity [1] is more specifically targeted for the creation of smart contracts on Ethereum. It is essentially a Turing-complete language based on C++ and JavaScript-like syntax. Therefore, the language targets professional developers rather than business users. In particular, expressing rules-style logic in Solidity involves the use of 'if' and 'else' statements along with a set of braces containing the rule specific code. Moreover, the chaining of rules has to be coded explicitly. In contrast, the BCRL framework supports a business level DSL based on controlled English, and the rules chaining is supported implicitly with a general-purpose RETE engine.

Some of the recent and emerging DSLs and environments for smart contracts are intended for the software developer community. This includes the Hyperledger Composer [2], which includes abstractions for "assets"(business-relevant entities that whose representations are manipulated onchain), "participants" and "transactions" (which manipulate the assets). As noted in [7], it would be natural to extend the Composer notion of assets with lifecycle models, in the sense of Business Artifacts and Case Management. The Obsidean language [13] focuses on abstractions for linear types and state-machine-based object lifecycles, in part to reduce errors in smart contracts and to facilitate verification.

The R3 Corda initiative is developing an approach designed to support the creation of smart contracts for Financial Services [14].

Citation [5] focuses on empowering business-level users to create and understand smart contracts, by showing how the BPMN standard can be implemented on top of Ethereum. This has led to the open-source Caterpillar system [6].

There is a long tradition of extending traditional business process management systems with rules engines, to provide more flexibility to the business-level users who maintain the processing logic. In a recent development the IBM ODM rules engine has been integrated with Hyperledger, enabling on-chain smart contracts to use REST APIs to invoke an ODM engine for decision support [15].

While the ODM engines are not running directly within the smart contract, they are resident on the peer nodes that are executing the smart contracts, and can thus be tightly integrated with the immutability, privacy and consensus-based features of the Hyperledger Fabric.

The BCRL language presented here is inspired from the BERL rules language, one of several supported by the ODM product. Another Controlled-English rules language is found within the Semantics of Business Vocabulary and Rules (SBVR) model [9], an Object Management Group (OMG) standard that provides the vocabulary and syntax for documenting the semantics of business vocabularies, business facts, and business rules. As noted in the introduction, BCRL could be adapted to follow the style of SBVR and/or commercially available business rule languages [8,10,11]. The semantics of SBVR is a version of higher-order logic that can be machine-processed and automatically analyzed [16]. The expressive powers of BCRL and SBVR are not comparable. SBVR supports deontic logic operators for expressing business rules in the style of obligations and permissions; these are not supported by BCRL. On the other hand, SBVR does not have an operational semantics (e.g. to express updates to the business facts) so it is not directly executable on Blockchain as allowed by BCRL. We believe that the BCRL framework could be extended to support most or all SBVR's expressive power.

6 Conclusions

This paper presents the BCRL framework that enables business-level users to specify and deploy business rules as smart contracts on the Hyperledger Blockchain fabric. The framework considers the use of Blockchain in the larger context of business collaborations, and enables use of the same rules language to be executed on-chain in smart contracts, and executed off-chain in business processes that are hosted by individual collaboration stakeholders. As such, the paper provides a key building block for empowering business-level users to program and manage intricate business logic for business collaborations, that are supported in a secure, distributed, service-oriented manner. Section 4 discussed key aspects of Blockchain that our rules implementation needs to address.

The BCRL framework as presented here is focused largely on decision support and associated updates to persistent data (both on-chain and off-chain). It will be valuable to explore approaches for integrating BCRL with BPMN-oriented and business artifact-oriented smart contract frameworks, to provide them with rich decision support.

Verification of smart contracts has become a major desired feature in Blockchain platforms like Ethereum [17,18] due to the high impact of vulnerabilities in smart contracts (e.g. \$50 million USD caused by the DAO hack [19]). Specifying smart contracts in a rule language like BCRL not only enables smart contract programming by business users but also introduces new opportunities for formal verification. This is because unlike common smart contract languages which are Turing-complete in general (so verification is undecidable), the expressive power of BCRL is close to first-order logic (FOL), making formal verification more feasible. In fact, the research on automatic verification of data-centric business processes (see [20] for a survey) has followed this trend and shown decidability results on various expressive specification models based on FOL. Implementation of a verifier for data-centric business processes was recently shown successful [21]. These results indicate that BCRL can be a good starting point for future research on smart contract verification.

Acknolwedgements. The authors are grateful to Stephane Mery, Philippe Bonnard and Jean Michel Bernelas from the ODM product group at IBM for their inspiration around rules languages and Blockchain, and also for assistance with the nanoRETE rules engine and the BDSL parser-generator. The authors are grateful to Jerome Simeon for his numerous insights, especially in connection with the spectrum of design and engineering issues that arise in Domain-Specific Languages. The authors are also grateful to the team at IBM working on Blockchain for Technical Support Services, including Saurabh Sinha, Nerla JeanLouis, and Shu Tao, for providing an environment and grounded use cases for the exploration of business rules for smart contracts.

References

1. Ethereum: Solidity (2018). https://solidity.readthedocs.io/en/v0.4.21/. Accessed 17 Mar 2018
2. IBM: Hyperledger Composer Home Page. https://www.hyperledger.org/projects/composer. Accessed 17 Mar 2018
3. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, p. 30. ACM (2018)
4. Mendling, J., et al.: Blockchains for business process management-challenges and opportunities. ACM Trans. Manag. Inf. Syst. (TMIS) **9**(1), 4 (2018)
5. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 329–347. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_19
6. López-Pintado, O., et al.: Caterpillar: a blockchain-based business process management system. In: Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling, BPM 2017, Barcelona, Spain, 13 September 2017
7. Hull, R., Heath III, F.F.T., Vianu, V., Batra, V.S., Chen, Y.M., Deutsch, A.: Towards a shared ledger business collaboration language based on data-aware processes. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 18–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_2
8. IBM: IBM Operational Decision Manager. https://www.ibm.com/automation/software/business-rules-management. Accessed 17 Mar 2018
9. OMG: Semantics of business vocabulary and business rules (SBVR), version 1.4 (2017). http://www.omg.org/spec/SBVR/1.4/PDF. Accessed 17 Mar 2018
10. Oracle: Fusion middleware designing business rules with Oracle business process management (2016). https://docs.oracle.com/middleware/12212/bpm/rules-user/toc.htm. Accessed 4 Aug 2018

11. Aghlara, A.: Business rule language (of FlexRule) (July 2015). http://www.flexrule.com/archives/business-rule-language/. Accessed 4 Aug 2018
12. IBM: IBM Operational Decision Manager Version 8.6.0: Business Rules Embedded developer guide (2014). http://www-01.ibm.com/support/docview.wss?uid=swg21660485&aid=1. Accessed 17 Mar 2018
13. Coblenz, M.: Obsidian: a safer blockchain programming language. In: Companion Proceedings of the 39th International Conference Software Engineering, ICSE Companion 2017 (2017)
14. Brown, R.G., et al.: Corda: an introduction. R3 CEV, August 2016
15. Mery, S., et al.: Make your blockchain smart contracts smarter with business rules. http://ibm.biz/odm-blockchain. Accessed 17 Mar 2018
16. Marinos, A., Krause, P.: An SBVR framework for RESTful web applications. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 144–158. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04985-9_15
17. Bhargavan, K., et al.: Formal verification of smart contracts: short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, pp. 91–96. ACM (2016)
18. Amani, S., et al.: Towards verifying ethereum smart contract bytecode in Isabelle/HOL. In: CPP. ACM (2018, To appear)
19. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8
20. Deutsch, A., et al.: Automatic verification of database-centric systems. ACM SIGLOG News **5**(2), 37–56 (2018)
21. Deutsch, A., et al.: Verifas: a practical verifier for artifact systems. In: VLDB (2017)

Context-Aware Trustworthy Service Evaluation in Social Internet of Things

Maryam Khani(✉), Yan Wang(✉), Mehmet A. Orgun, and Feng Zhu

Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
{maryam.khani,feng.zhu3}@hdr.mq.edu.au
{yan.wang,mehmet.orgun}@mq.edu.au

Abstract. In Social Internet of Things (SIoT) environments, a large number of users and Internet of Things (IoT) based devices are connected to each other, so that they can share SIoT-based services. IoT-based devices establish social relations with each other according to the social relations of their owners in Online Social Networks (OSNs). In such an environment, a big challenge is how to provide trustworthy service evaluation. Currently, the prevalent trust management mechanisms consider QoS-based trust and social-relation based trust mechanisms in evaluating the trustworthiness of service providers. However, the existing trust management mechanisms in SIoT environments do not consider the different contexts of trust. Therefore, dishonest SIoT devices, based on their owners' social relations, can succeed in advertising low-quality services or exploiting maliciously provided services. In this paper, we first propose three contexts of trust in SIoT environments including the status and environment (time and location) of devices, and the types of tasks. Then, we propose a novel Mutual Context-aware Trustworthy Service Evaluation (MCTSE) model. The experiments demonstrate that our proposed contextual trust evaluation model can effectively differentiate honest and dishonest devices and provide a high success rate in selecting the most trustworthy services and providing high resilience against different attacks from dishonest devices.

Keywords: Social Internet of Things · Contextual trust
Trustworthy service evaluation

1 Introduction

In recent years, a combination of the Internet of Things (IoT) and Online Social Networks (OSNs) has led to the Social Internet of Things (SIoT) to facilitate the discovery, selection, and composition of services provided by distributed IoT based things [1,2,24]. Those *things* include personal devices (*e.g.*, smartphones, tablets), devices fitted with tags (*e.g.*, RFIDs) in our environment, sensors and actuators [24]. In SIoT environments, a device with a specific owner requests services from or provides services to other devices, and establishes social relations with other devices based on social rules determined by their owners in an

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 129–145, 2018.
https://doi.org/10.1007/978-3-030-03596-9_9

autonomous manner by considering their owners' social networks [1,2]. Then, the devices can exchange their friend lists with each other [1,2]. Moreover, devices may establish different types of *social relations* with each other including *ownership* (devices belonging to the same user), *co-work* (devices collaborating to provide common services), *co-location* (devices that are always used in the same place), *parental* (devices belonging to the same manufacturers) and *social device relations* (devices coming into contact continuously) [1,2].

Recently, a broad range of Social Internet of Things (SIoT) based applications have emerged [1], such as smart traffic management [23], smart airport [26], and smart home [25]. To find the right source of information in such an SIoT environment, a user's devices can connect with other devices which are identified by means of co-location relations. However, devices can be either *honest*, providing good quality services, or *dishonest*, providing poor-quality services. Dishonest devices may perform malicious trust-related attacks, such as *Bad-Mouthing Attacks* (BMA), *Ballot-Stuffing Attacks* (BSA), *Self-Promoting Attacks* (SPA), and *On-Off Attacks* (OOA) [3–5,9]. In order to mitigate against such attacks, the issue of trust evaluation in SIoT environments arises and becomes prominent. Firstly, when a service-consuming device looks for its needed service, some service-providing devices may behave dishonestly and provide low-quality services for their own benefit. Secondly, the resources of a service-providing device could be maliciously exploited by some dishonest service-consuming devices [14]. Thirdly, dishonest devices may perform trust-related attacks to ruin the reputation of other devices by reputation attacks (BMA and BSA) or to boost their importance by self-interest attacks (SPA and OOA). Therefore, a reliable SIoT environment needs to be built based on an effective trust management mechanism for selecting trustworthy service-providing devices and trustworthy service-consuming devices.

1.1 Background and Problem

A variety of context-aware trust evaluation approaches have been proposed in Online Social Networks (OSNs) [12,22]. These approaches are mostly concerned with the trust evaluation of social participants by considering the social contexts between them. However, they do not consider social relations among devices and the features of Internet of Things (IoT) service computing environments. Furthermore, the existing trust management approaches in IoT [16,17,24,28] only consider QoS (Quality of Service) trust metrics, without considering the social relations between devices, which are very important characteristics of SIoT environments.

To select trustworthy service-providing devices or service-consuming devices, a variety of trust service evaluation approaches have been proposed in SIoT environments [3–5,7,14,15,17,23]. To date, SIoT trust evaluation systems use direct evidence, such as QoS-based trust, and indirect experiences, such as social relation based trust, to evaluate the trust level of service-providing devices or service-consuming devices. Though such trust evaluation mechanisms are applied for indicating a device's trustworthiness in many studies, they do not consider the different contexts of devices (*e.g.,* the status and environment of devices) and the types of tasks. Therefore, they cannot ultimately select the most trustworthy service-providing devices or trustworthy service-consuming devices. A motivating example is given below.

Example 1: There are different SIoT-based communities and IoT social networks, and users can register their IoT-based devices to these communities and networks to use different SIoT-based services [1,2]. Users want to share the provided services by their devices specially when a device cannot provide requested services from its user. Suppose that users A, B and C register their IoT-based devices (*e.g.*, smartphones, tablets, *etc.*) in the same SIoT-based communities. Then, suppose that the smartphone of user A, has low battery, and thus automatically searches the nearest devices to delegate the task of recording an on-line video from an important event. Suppose, user B is on the way to leave the place where user A is. B has a smartphone, with a low battery. User C is on the way to reach the place where user A is, and user C has a tablet with full battery. While the devices of users B and C provide the same services and have the same social relations with those of user A, the tablet of user C is more trustworthy when the status and environment (*i.e.*, time and location) of devices are taken into account. However, the existing trust evaluation mechanisms cannot differentiate user B's device and user C's device in such a context because they do not consider devices' trustworthiness in different contexts, such as the status, and the environment of the devices, and the types of tasks [9]. In the literature, the existing studies on trust evaluation only consider a service-providing device's single context, such as a service context, but a multi-contextual model will be more accurate in evaluating the trustworthiness of devices and thus in demand.

1.2 Contributions

To overcome the above-mentioned drawbacks, this paper proposes a novel Mutual Context-aware Trustworthy Service Evaluation (MCTSE) model in SIoT environments for trust enhanced service evaluation. The characteristics and contributions of our proposed model are summarised as follows:

1. We first propose a classification of contexts of trust in SIoT environments including the status of devices, environment (time and location) of devices, and the types of tasks. Based on the context of trust in SIoT environments, we propose a Contextual SIoT Trust Model consisting of independent and dependent metrics.
2. Then, we propose two new concepts Context-aware QoS Similarity based Trust (CQSST) and Context-aware Social Similarity based Trust (CSST), and propose novel models for evaluating them. Then, we apply these new concepts in the MCTSE model to evaluate the trustworthy of service-consuming and service-providing devices.
3. We conduct simulations with 600 randomly generated service-consuming devices and service-providing devices to evaluate the effectiveness of our model. The experimental results show that our model outperforms three

state-of-the-art models effectively in evaluating the trustworthiness of service-providing devices and service-consuming devices. It can also differentiate honest and dishonest devices with a high accuracy which perform tasks without attacks or with different types of attacks, respectively. Therefore, our model can select the most trustworthy services with high quality and with high resilience against different malicious attacks of dishonest devices.

2 Literature Review

In this section, we first review the most relevant the contextual trust evaluation techniques applied in OSNs. We then review the trust management techniques proposed in IoT, and SIoT studies that are related to our work. We categorize the proposed techniques into single-context (one or two simple contexts are applied to trust evaluation) and multi-context (more complicated contexts are applied to trust evaluation).

2.1 Trust Models in Online Social Networks (OSNs)

In the studies of trust evaluation in OSNs, some qualitative approaches have been proposed. As an example of a single-context trust evaluation, Kuter *et al.* [12] consider the confidence calculated by a person toward another in *FilmTrust*, a movie recommendation system, but it is unclear how they calculate this context factor. As an example of multi-context trust evaluation, Liu *et al.* [8] proposed a complex online social network structure with a new concept called "Quality of Trust" to introduce the evaluation of the trustworthiness of a service provider along with a certain social trust path from the service consumer to the service provider. They considered social information including social position, social relation, and preferences of participants to select trustworthy trust paths. Zhan *et al.* [22], in online multimedia social networks, used credible feedback of digital contents, a feedback weighting factor, and user share similarity to evaluate a direct trust between users.

Though context-aware trust evaluation and trust recommendation approaches have been proved to be effective in OSNs, they are not directly applicable in SIoT environments.

2.2 Trust Models in Internet of Things (IoT)

In IoT environments, there have been a few studies on trust management models. The categorising of trust remains unclear due to the lack of classification of the listed research activities in an obvious sorting logic. Razzaque *et al.* [24] proposed different architectures of the IoT, and identified the relevant research challenges in communication problems and information gathering problems. However, they did not propose any solution for security and privacy problems. Zheng *et al.* [17] indicated that trust contains more meanings than security. Trust in IoT is built based on not only security, but also many other important factors such as honesty, goodness, competence, reliability, and ability. Sfar *et al.* [16] reported that trust management systems could be defined as deterministic trust (including policy-based mechanism and certificates systems) and non-deterministic trust (including recommendation-based, reputation-based systems, prediction-based, and social network based systems). Recently, Chen *et al.* [28] proposed a trust computation model based on fuzzy reputation in IoT systems. For trust composition, QoS trust parameters such as end-to-end packet forwarding ratio, energy consumption, and packet delivery ratio are considered. However, contextual information in both trust evaluation and trust recommendation has not been considered yet.

Although IoT trust management systems share common features with SIoT environments to provide services with different devices, the existing studies on trust management in IoT systems do not consider the social aspects of the owners of IoT devices.

2.3 Trust Models in Social Internet of Things (SIoT)

In SIoT environments, the existing trust management systems can be broadly categorised into non-contextual and single contextual methods.

In a non-context-aware trust management model proposed in [6], Bao *et al.* consider social relations in trust management for IoT. For trust composition, they consider both QoS trust properties including honesty, cooperativeness, and social trust such as community interest. However, the proposed factors for computing cooperativeness based on the percentage of common friends are very simple. Chen *et al.* [7] proposed an access service recommendation scheme for effective service composition as well as resistance against malicious attacks. For trust composition, they consider QoS trust metrics such as quality reputation and energy status. Also, social trust is considered based on certain social similarities. However, Chen *et al.* did not consider some trust properties such as contextual and dynamic characteristics of trust. Chen *et al.* [5] proposed an adaptive and scalable trustworthy service composition in SOA-based IoT systems. They only apply a single QoS trust to rate a service provider. However, the social relations between devices are not considered.

As a single-context trust management model, Nitti *et al.* [15] proposed a trust computation method which considers both direct and indirect trust. For trust composition, QoS based trust (including transaction service quality and computational capability) and social relation based trust (including centrality, relation factor) are applied. In this model, trust is context-dependent but only factors such as the number of transactions in a QoS based trust is considered as a context. Therefore, their model is a single-context trust. Furthermore, Lin *et al.* [14] proposed a contextual trust management model in which a context consists of two components, task type and environment. They considered different types of environments, for example, a hostile environment means that the external condition is unsuitable for the current task. For trust composition, QoS based trust (*e.g.*, bandwidth, packet loss, *etc.*) and social based trust (social relationships, such as friendship) is applied. However, they only consider the type of task and the situation of the environment as contexts, and they do not consider other contexts such as time, location, and the features of a device, to be multi-context.

To sum up, the existing trust management systems in SIoT environments have not investigated context-aware (*i.e.*, multi-contextual) trust evaluation and recommendation yet. Moreover, context-aware trust models in OSNs cannot be directly applied in SIoT environments because the specific characteristics of trust in SIoT systems include direct (*e.g.*, QoS-based trust), dynamic, *etc*, which should be considered. In addition, existing trust models in IoT environments do not consider the social relation among devices in SIoT environments.

3 Problem Statement and Metrics of Contextual Trust

3.1 Problem Statement

In our SIoT model, there are M devices which are denoted by $D = \{d_1, ..., d_M\}$ and there are N users which are denoted by $U = \{u_1, ..., u_N\}$. Let the social network between users be represented by an undirected graph $G = \{U, E\}$, where $E \subseteq U \times U$, and $< u, v > \in E$ means there is a social relation between u and v. Moreover, there are I service-consuming devices and J service-providing devices by considering the social relations o their owner which are represented by $SC = \{SC_1, ..., SC_I\}$ and $SP = \{SP_1, ..., SP_J\}$ respectively. In addition, each of SC_i or SP_j is represented by a vector in a three dimensional space of the contexts in SIoT including status (C_S), environment (C_E), and task type (C_T), which are represented by $C = \{C_S, C_E, C_T\}$. Each of C_S, C_E, C_T has different values presented by $C_S = \{C_{S_1}, ..., C_{S_h}\}$, $C_E = \{C_{E_1}, ..., C_{E_{\acute{h}}}\}$, and $C_T = \{C_{T_1}, ..., C_{T_{\acute{\acute{h}}}}\}$, respectively. The vectors of $\overrightarrow{SC}_i$ and $\overrightarrow{SP}_j$ are represented by Eqs. (1) and (2), respectively. Each of SC_i and SP_j has a list of the owner's friends which is denoted by $UFre_{SC_i}$ and $UFre_{SP_j}$, respectively, and a list of owner's community of interests which is denoted by $UCom_{SC_i}$ and $UCom_{SP_j}$, respectively. Also, let S $= \{s_1, ..., s_l\}$ denote the set of services which are provided or consumed by devices in different time $\tau = \{t_1, ..., t_p\}$, and locations $L = \{l_1, ..., l_q\}$. Moreover, each of SC_i and SP_j has a user satisfaction level or ground truth [19] which is shown by GT_{SC_i} and GT_{SP_j}, respectively. The aim of this paper is to provide a list of trustworthy SPs and SCs for each of SP_i and SC_j in each transaction.

$$\overrightarrow{SC}_i = \begin{bmatrix} C_{S_i} \\ C_{E_i} \\ C_{T_i} \end{bmatrix} \tag{1}$$

$$\overrightarrow{SP}_j = \begin{bmatrix} C_{S_j} \\ C_{E_j} \\ C_{T_j} \end{bmatrix} \tag{2}$$

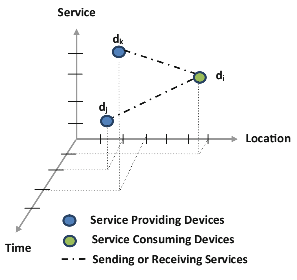

Service
d_k
d_i
d_j
Location
Time
● Service Providing Devices
● Service Consuming Devices
- · - Sending or Receiving Services

Fig. 1. Contexts of Trust in IoT environments

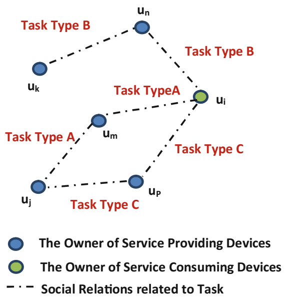

u_n
Task Type B
Task Type B
u_k
Task TypeA
u_i
Task Type A
u_m
Task Type C
u_j
u_p
Task Type C
● The Owner of Service Providing Devices
● The Owner of Service Consuming Devices
- · - Social Relations related to Task

Fig. 2. Contexts of Trust in OSN

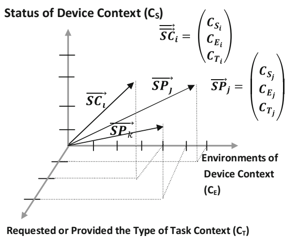

Status of Device Context (C_S)
$\overrightarrow{SC}_i = \begin{pmatrix} C_{S_i} \\ C_{E_i} \\ C_{T_i} \end{pmatrix}$
$\overrightarrow{SC_i}$
$\overrightarrow{SP_j}$
$\overrightarrow{SP}_j = \begin{pmatrix} C_{S_j} \\ C_{E_j} \\ C_{T_j} \end{pmatrix}$
$\overrightarrow{SP_k}$
Environments of Device Context (C_E)
Requested or Provided the Type of Task Context (C_T)

Fig. 3. Contexts of Trust in SIoT environments

3.2 The Contexts of Trust in SIoT Environments

In general, devices in IoT environments may trust each other based on different contextual factors, including different statuses of devices, such as energy, and capability of computing, which provide or request different services at different time and locations. In addition, the owners of devices in a contextual OSN [27] may trust each other based on common social relations for different types of tasks. For example, suppose that there are two devices d_j and d_k, as service-providing devices, advertising the services requested by device d_i, as the service-consuming device, in an SIoT environment. In this scenario, the QoS based trust value evaluated by d_i for d_j and d_k varies at different time, locations and different statuses of d_j and d_k. These contexts are considered as the contexts of trust in IoT environments, as depicted in Fig. 1. Moreover, the social relation based trust values evaluated by d_i by considering the common social relations between its owner (u_i) and the owner u_i of d_j and the owner u_k of d_k for different types of tasks. Therefore, the task type context is considered as the context of trust in OSNs which is shown in Fig. 2. By considering different contextual aspects between devices in IoT environments and their owners in OSNs, we classify the contexts of trust in SIoT environments in three categories including the status of devices, environment (time and location) of devices, and the types of tasks. Figure 3 depicts the space of the contexts of trust in SIoT environments. In such a space, each device is considered as a service-providing device or a service-consuming device which is shown with a vector in the three-dimensional space of contexts including the status of devices, environment (time and location) of devices, and the types of tasks. The contexts of trust in SIoT environments are described as follows.

- **The Status of a device (C_S):** The features of devices such as energy, and the capability of computing.
- **The Environment of a device (C_E):** Service-consuming devices and service-providing devices may be located in different locations and may be

available in different time (*e.g.*, next 1 hour, next 2 hours, next 3 hours, and *etc.*).

- **Task type (C_T):** For example, a service-consuming device could trust a service-providing device for task type A but not for task type B. A task type context which is requested by a service-consuming device could be made by a combination of some services. Here, only two services are considered. For example, the task type of A is a combination of services including S_1 and S_2.

3.3 The Metrics of Contextual Trust Evaluation

Based on the classified contexts of trust in SIoT environments, we propose the following metrics of contextual trust with significant effects on trust evaluation.

3.3.1 Independent Metrics

Independent metrics of a service-consuming device and a service-providing device in SIoT environments refer to the individual preferences of the service-consuming device and individual capabilities of the service-providing device that has direct influence on contextual QoS based trust evaluation. Moreover, QoS refers to a level of service that is satisfactory to some user requirements including performance, cost, availability, *etc.* The independent metrics include expected QoS and advertised QoS. Each of these parameters is shown in a vector in the two-dimensional space of the status and environment contexts of trust.

- Let $\boldsymbol{\overrightarrow{ExQoS}_{SCi}^{C_S,C_E}}$ denote the *Expected Quality of Service (ExQoS)* that is requested by a service-consuming device i (SC_i) at a specific status and environment contexts (C_S, S_E)
- Let $\boldsymbol{\overrightarrow{AdQoS}_{SP_j}^{C_S,C_E}}$ denote the *Advertised Quality of Service (AdQoS)* that is provided by service-providing device j (SP_j) at a specific status and environment contexts (C_S, S_E). These parameters are depicted by Eqs. (3) and (4) respectively.

$$\overrightarrow{ExQoS}_{SCi}^{C_S,C_E} = \begin{bmatrix} C_{S_j} \\ C_{E_j} \end{bmatrix} \tag{3}$$

$$\overrightarrow{AdQoS}_{SP_j}^{C_S,C_E} = \begin{bmatrix} C_{S_i} \\ C_{E_i} \end{bmatrix} \tag{4}$$

3.3.2 Dependent Metrics

The dependent metrics illustrate the contextual social based trust value between a service-providing device and a service-consuming device, which include social similarity friendship, social similarity community, social similarity relations, and contextual feedback of trust in the context of task type. We consider the fact that the idea of friends has an important effect on the decision of someone. Therefore, the more interests one has with another in a specific task type context the more likely they trust each other in that task type context.

- Let $\boldsymbol{SSimFre_{SC_i,SP_j}^{C_T}}$ denote the *Social Similarity Friendship (SSimFre)* that captures the degree of the common social friends between the owner of a service-consuming device i and the owner of a service-providing device j respectively which are evaluated by the service-consuming device i based on its direct observations at the task type context. After two service-providing and service-consuming devices exchange the friend lists of their owners [2], *i.e.*, $UFre_{SC_i}$ and $UFre_{SP_j}$, they can compute two binary lists including $LFre_{SC_i}^{C_T}$ and $LFre_{SP_j}^{C_T}$ where the size of each list is equal to $S_{Fre} = |UFre_{SC_i} \cup UFre_{SP_j}|$. Each element in these lists will be 1 if the corresponding owner is in $UFre_{SC_i}$ or $(UFre_{SP_j})$ and has a relationship in the specific task type context C_T with SC_i or (SP_j), otherwise 0. The metric of $SSimFre_{SC_i,SP_j}^{C_T}$ is calculated by Eq. (5).

$$SSimFre_{SC_i,SP_j}^{C_T} = \frac{LFre_{SC_i}^{C_T}.LFre_{SP_j}^{C_T}}{S_{Fre}} = \frac{\sum_{\acute{h}=1}^{h} LFre_{SC_i}^{C_T}[\acute{h}].LFre_{SP_j}^{C_T}[\acute{h}]}{S_{Fre}} \tag{5}$$

- Let $\boldsymbol{SSimCom_{SC_i,SP_j}^{C_T}}$ denote the *Social Similarity Community (SSimCom)* that captures the degree of the common communities between the owner of a service-consuming device i and the owner of a service-providing device j respectively which are evaluated by the service-consuming device i based on its direct observations at the task type context. Moreover, the two service-providing and service-consuming devices exchange the lists of community interest of their owners [2], $UCom_{SC_i}$ and $UCom_{SP_j}$. Then, they compute two binary lists including $LCom_{SC_i}^{C_T}$ and $LCom_{SP_j}^{C_T}$ where the size of each list is equal to $S_{Com} = |UCom_{SC_i} \cup UCom_{SP_j}|$. Each element in these lists will be 1 if the corresponding community interest is in $UCome_{SC_i}$ or $(UCome_{SP_j})$ and is related to the specific task type context C_T, otherwise 0. The metric of $SSimFre_{SC_i,SP_j}^{C_T}$ is calculated by Eq. (6).

$$SSimCom_{SC_i,SP_j}^{C_T} = \frac{LCom_{SC_i}^{C_T}.LCom_{SP_j}^{C_T}}{S_{Com}} = \frac{\sum_{\acute{q}=1}^{q} LCom_{SC_i}^{C_T}[\acute{q}].LCom_{SP_j}^{C_T}[\acute{q}]}{S_{Com}} \tag{6}$$

- Let $\boldsymbol{SSimR_{SC_i,SP_j}^{C_T}}$ denote the *Social Similarity Relation (SSimR)* that captures the degree of common social relations (*e.g.* ownership, co-work, co-location, parental) [1,2] between a service-providing device j with a service-consuming device i at the task type context. We consider different weighted values for each device social relations with other devices which are listed in Table 1. For example, if two devices have the same owner while they provide or request the same type of tasks, the weighted value is equal to 1. If they have the same owner but they provide or request different types of tasks, the weighted value is equal to 0.9. Moreover, if there are different social relations between two devices, only the highest weight is considered.

Table 1. Social Similarity Relations (SSimR)

Relationship	Value with C_T	Value without C_T	Description
Ownership	1	0.9	Between devices that belong to the same owner
Co-work	0.8	0.7	Between devices that collaborative to provide common service
Co-location	0.6	0.5	Between devices that are in the same area
Social	0.4	0.3	Between devices that continuously interact with each other
Parental	0.2	0.1	Between devices that belong to the same production batch

- Let $\boldsymbol{CFT^{C_S,C_E,C_T}_{SP_j \to SC_i}(n-1)}$ and $\boldsymbol{CFT^{C_S,C_E,C_T}_{SC_i \to SP_j}(n-1)}$ denote the *Contextual Feedback of Trust (CFT)* in the view of SC_i and in the view of SP_j respectively, where n denotes the number of transactions between SC_i and SP_j in the status and environment contexts of devices and the task type context. $CFT^{C_S,C_E,C_T}_{SP_j \to SC_i}(n-1)$ denotes the previous direct feedback of a service-providing device j toward a service-consuming device i at status and environment contexts of devices and the task type context and $CFT^{C_S,C_E,C_T}_{SC_i \to SP_j}(n-1)$ denotes the previous direct feedback of the service-consuming device i toward service-providing device j in the status and environment contexts of device and the task type context, if there is any direct feedback. Moreover, let $\boldsymbol{Variance^{C_S,C_E,C_T}_{SC_i \to SP_j}(K)}$ denote the *Variance* of $CFT^{C_S,C_E,C_T}_{SC_i \to SP_j}(n-1)$ in its K latest transactions and let $\boldsymbol{Variance^{C_S,C_E,C_T}_{SP_j \to SC_i}(K)}$ denote the *Variance* of $CFT^{C_S,C_E,C_T}_{SP_j \to SC_i}(n-1)$ in its K latest transactions. The metrics of $Variance^{C_S,C_E,C_T}_{SC_i \to SP_j}(K)$ and $Variance^{C_S,C_E,C_T}_{SP_j \to SC_i}(K)$ are calculated by Eqs. (7), (8), (9) and (10) respectively. Then, the metrics of $e^{Variance^{C_S,C_E,C_T}_{SC_i \to SP_j}(K)}$ and $e^{Variance^{C_S,C_E,C_T}_{SP_j \to SC_i}(K)}$ have been considered as coefficients applied to the previous direct feedback of service-providing device in our MCTSE model. Therefore, if there is more variance in K latest transactions of a device, it means that it was a dishonest device. Therefore, its dishonest behaviour is memorized and it decreases the importance of its previous direct feedback. We apply the e^{-x} function where x is equal to the $Variance$ because the more variance in the previous feedbacks, the less the trust value between them. Moreover, the e^{-x} function keeps the value of $Variance$ between 0 and 1.

$$Variance^{C_S,C_E,C_T}_{SC_i \to SP_j}(K) = \frac{\sum_{x=n-k}^{n} (CFT^{C_S,C_E,C_T}_{SC_i \to SP_j}(x) - \overline{\text{CFT}}^{C_S,C_E,C_T}_{SC_i \to SP_j}(K))^2}{k-1} \tag{7}$$

$$Variance^{C_S,C_E,C_T}_{SP_j \to SC_i}(K) = \frac{\sum_{x=n-k}^{n} (CFT^{C_S,C_E,C_T}_{SP_j \to SC_i}(x) - \overline{\text{CFT}}^{C_S,C_E,C_T}_{SP_j \to SC_i}(K))^2}{k-1} \tag{8}$$

$$\overline{\text{CFT}}^{C_S,C_E,C_T}_{SC_i \to SP_j}(K) = \frac{\sum_{x=n-k}^{n} CFT^{C_S,C_E,C_T}_{SC_i \to SP_j}(x)}{K} \tag{9}$$

$$\overline{\text{CFT}}^{C_S,C_E,C_T}_{SP_j \to SC_i}(K) = \frac{\sum_{x=n-k}^{n} CFT^{C_S,C_E,C_T}_{SP_j \to SC_i}(x)}{K} \tag{10}$$

4 Mutual Context-Aware Trustworthy Service Evaluation (MCTSE) Model

4.1 Overview of the MCTSE Model

In our proposed MCTSE Model, we consider two concepts, including *Context-aware QoS Similarity based Trust*, *Context-aware Social Similarity based Trust*, in the computation of MCTSE, which are described below.

- **Context-aware QoS Similarity based Trust (CQoSSTrust):** Let $CQoSSTrust_{SC_i,SP_j}^{C_S,C_E}$ denote the *Context-aware QoS similarity based Trust* that captures the degree of similarity between the expected Quality of Service which is requested by a service-consuming device i and the advertised quality of service which is provided by a service-providing device j at status and environment context of the device. We apply the cosine similarity function to calculate the similarity between two vectors $\overrightarrow{ExQoS}_{SC_i}^{C_S,C_E}$ and $\overrightarrow{AdQoS}_{SP_j}^{C_S,C_E}$. Therefore, $CQoSSTrust_{SC_i,SP_j}^{C_S,C_E}$ is calculated by Eg. (11). As the maximum QoS similarity based trust, $CQoSSTrust_{SC_i,SP_j}^{C_S,C_E}= 1$ captures that the SP_j can provide the maximum expected QoSs of SC_i while $CQoSSTrust_{SC_i,SP_j}^{C_S,C_E}=$ 0 indicates that there is no similarity between the expected QoSs of SC_i and the advertised QoSs of SP_j. If $\overrightarrow{ExQoS}_{SC_i}^{C_S,C_E}$ = A and $\overrightarrow{AdQoS}_{SP_j}^{C_S,C_E} = B$ then:

$$CQoSSTrust_{SC_i,SP_j}^{C_S,C_E} = \cos(\theta) = |\overrightarrow{A} \times \overrightarrow{B}| = \frac{A.B}{\parallel A \parallel_2 \parallel B \parallel_2} = \frac{\sum_{\acute{h}=1}^{h} A_{\acute{h}}.B_{\acute{h}}}{\sqrt{\sum_{\acute{h}=1}^{h} A_{\acute{h}}^2}\sqrt{\sum_{\acute{h}=1}^{h} B_{\acute{h}}^2}} \quad (11)$$

- **Context-aware Social Similarity based Trust (CSSTrust):** Let $CSSTrust_{SC_i,SP_j}^{C_T}$ denote the *Context-aware Social Similarity based Trust* that indicates the overall degree of social similarity between service consumer SC_i and service provider SP_j at the task type context. Equations (12), (13), and (14) are applied to compute $CSSTrust_{SC_i,SP_j}^{C_T}$. We apply the e^{-x} function in Eq. (12) where x is equal to $SDissimilarity^{C_T}$ (it denotes *Social Dissimilarity* between SC_i and SP_j in the task type context) because the more the dissimilarity between a service-consuming device and a service-providing device, the less the trust value between them. Moreover, the e^{-x} function keeps the value of $CSSTrust_{SC_i,SP_j}^{C_T}$ between 0 and 1. $CSSTrust_{SC_i,SP_j}^{C_T}$ is applied as a weight for computing direct trust. If there is no social similarity between the owners of two devices in SIoT environments, $CSSTrust_{SC_i,SP_j}^{C_T} = e^{-SDissimilarity^{C_T}}$ means that there is a less trust value between the owners of devices. The social factors including social similarity friendship, social similarity community, social similarity relations may have different importance. Therefore, weight parameters w_i are applied to adjust the importance of these three social similarity factors.

$$CSSTrust_{SC_i,SP_j}^{C_T} = e^{-SDissimilarity^{C_T}} \quad (12)$$

$$SDissimilarity^{C_T} = 1 - SSimilarity^{C_T} \tag{13}$$

$$SSimilarity^{C_T} = w_1 \times SSimFre^{C_T}_{SC_i,SP_j} + w_2 \times SSimCom^{C_T}_{SC_i,SP_j} + w_3 \times SSimR^{C_T}_{SC_i,SP_j} \tag{14}$$

4.2 Assessing Trust in SIoT Environments by MCTSE Model

Mutual Context-aware Trustworthy Service Evaluation (MCTSE) indicates the trust evaluation between a service-providing device and a service-consuming device while both of them evaluate each other and consider the contextual information. Below, we describe two parts of the mutual context-aware trustworthy service evaluation including: (1) *Trustworthy Service Evaluation from Service-Consuming Device i to Service-Providing Device j* ($MCTSE^{C_S,C_E,C_T}_{SC_i \rightarrow SP_j}$). It is calculated by Eq.(15). It denotes the direct trust value from service-consuming device j to service-providing device i. (2) *Trustworthy Service Evaluation from Service Providing Device j to Service-Consuming Device i* ($MCTSE^{C_S,C_E,C_T}_{SP_j \rightarrow SC_i}$). It is calculated by Eq.(16). It denotes the direct trust value from service-providing device j to service-consuming device i. Moreover, the variance is applied to consider the trend of a service-providing device in its K previous transactions. In the following equations, we apply δ as a weight ($0 \leq \delta \leq 1$) to balance the importance of $CQoSSTrust^{C_S,C_E}_{SC_i,SP_j}$, $CSSTrust^{C_T}_{SC_i,SP_j}$, $CFT^{C_S,C_E,C_T}_{SC_i \rightarrow SP_j}$ and $CFT^{C_S,C_E,C_T}_{SP_j \rightarrow SC_i}$.

$$\begin{aligned} MCTSE^{C_S,C_E,C_T}_{SC_i \rightarrow SP_j} &= \delta \times CQoSSTrust^{C_S,C_E}_{SC_i,SP_j} \times CSSTrust^{C_T}_{SC_i,SP_j} \\ &+ (1-\delta) \times e^{Variance^{C_S,C_E,C_T}_{SC_i \rightarrow SP_j}(K)} \times CFT^{C_S,C_E,C_T}_{SC_i \rightarrow SP_j}(n-1). \end{aligned} \tag{15}$$

$$\begin{aligned} MCTSE^{C_S,C_T}_{SP_j \rightarrow SC_i} &= \delta \times CQoSSTrust^{C_S}_{SC_i,SP_j} \\ + (1-\delta) \times e^{Variance^{C_S,C_E,C_T}_{SP_j \rightarrow SC_i}(K)} &\times CFT^{C_S,C_E,C_T}_{SP_j \rightarrow SC_i}(n-1). \end{aligned} \tag{16}$$

5 Experiments

In this section, we introduce two experiments of our proposed MCTSE model in a simulation where 300 service-consuming devices need to select the most trustworthy service-providing devices from 300 service-providing devices.

5.1 Simulation Settings and Performance Comparison in SIoT Environments

To simulate an SIoT environment, because there is a lack of a real dataset in the literature, we create a synthetic dataset with 600 randomly generated devices with different statuses, in which there are 300 service-providing devices and 300 service-consuming devices. These devices are randomly assigned to 200 users who are selected from the synthetic dataset of the online social network Facebook obtained from the synthetic Stanford Large Network Dataset Collection [13]. We assume that each user owns two devices on average. Each device has a role as either a service provider or a service consumer. In addition, we assume that after a direct interaction between the devices of two users, they exchange their friend lists and profiles.

In our simulation, we classify the devices into two groups of honest and dishonest devices who provide high-quality services and poor-quality services respectively. The percentage of dishonest devices is set to 0% and 50% respectively. The dishonest devices perform trust related attacks including BMA, BSA, SPA, and OOA. To assess the performance of our proposed trust model, the user satisfaction levels of service selections (or real service qualities of devices) are considered as the *"ground truth"*. We compute the trust values of all honest or dishonest devices using our proposed model and compare with the *"ground truth"* to assess the accuracy of our model. For each honest device, a random value in the range of [0.80, 0.85] is assigned to its ground truth (it shows that an honest device provides high-quality service), and for each a dishonest device a random value in the range of [0.55, 0.60] is assigned to its ground truth (it shows that dishonest device provides poor-quality services). Moreover, we consider the optimal parameters in our models obtained by trial and test: σ=0.8, δ=0.5, w_1=0.33, w_2=0.33, and w_3=0.33. In this paper, we select three state-of-the-art trust management models in this field as the baseline models. They are (1) SOA [5], as a non-context trust management model, (2) SubM [15] and (3) ObjM [15], as two single-context trust management models, which are subjective and objective models respectively. Each of these models is implemented using C# programming. Then, trust-related attacks are modeled by applying their descriptions [3–5,9].

5.2 Experiment 1: Effectiveness of Trustworthy Service Evaluation

Results & Analysis: Figure 4 shows the success rates of the MCTSE, SOA, SubM, and ObjM models when there are different percentages of dishonest devices (0% and 50%). When there are 50 % dishonest devices, we consider three cases of different attacks, *i.e*, with BMA-BSA, with SPA, and with OOA, respectively. From Fig. 4, we can see that MCTSE always has the best success rate in all the cases. On average, MCTSE is 2%, 13.8%, 7.4%, 10.6%, and 10.2% higher in success rate than the average of the three baseline models when there are 0% and 50% dishonest devices who provide or consume services "without attacks" and "with attacks" including BMA-MSA, SPA, and OOA respectively.

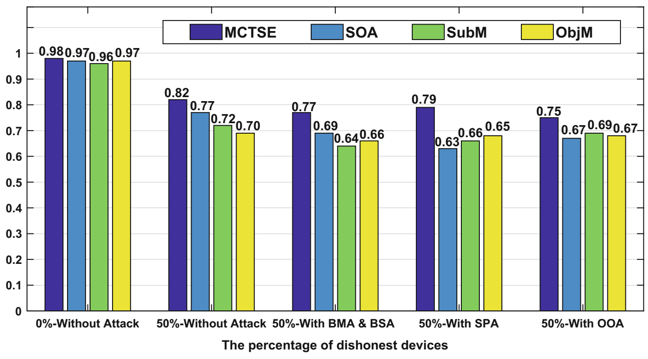
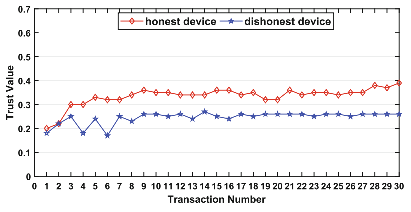


Fig. 4. Comparison of the success rate of an honest device (iterations = 20) by increasing the percentage of dishonest devices from 0% to 50%, which are also categorized into the cases of "without attack" and "with different types of attacks"

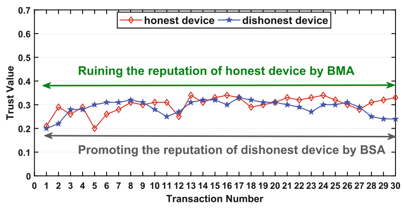
The experimental results illustrate that the MCTSE model can select the most trustworthy devices with the best quality service when compared with the other three baseline models. This is because the MCTSE considers multiple contexts of trust and thus is able to distinguish dishonest devices more accurately.

5.3 Experiment 2: Effectiveness in Resiliency Against Attacks

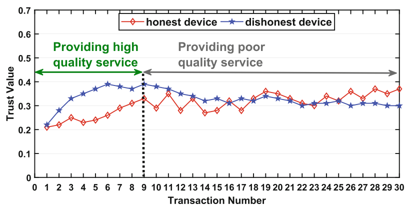
Results & Analysis: Figure 5a to d depict the trust results of a service-consuming device toward the honest and the dishonest devices, who provide or consume services without attacks, and with attacks including BMA-MSA, SPA, and OOA. From Fig. 5b, we can see that, although the trust value of the dishonest device has been promoted by good recommendation of other dishonest devices, its trust value decreases quickly after it provides poor-quality services. Moreover, although the trust value of the honest device was ruined by wrong recommendations, its trust value increases after providing good service. From Fig. 5c, we can see that the dishonest device boosts its importance when the transaction number changes from 1 to 9, to be selected as a service provider, but then from transaction 10 onwards it starts to provide poor-quality services. Our model decreases the trust value of the dishonest device when it starts to provide poor-quality services by applying the variance of feedback. From Fig. 5d, we can see that when dishonest devices perform OOA, they behave alternatively well and badly. The MCTSE model with the consideration of the contextual feedback of trust and its variance can detect this attack. The experimental results illustrate that: (1) when an honest device provides high-quality services and acts cooperatively, MCTSE increases its trust value; and (2) when a dishonest device provides poor-quality services and acts maliciously, performing different types of attack, MCTSE decreases the trust value of the dishonest device.



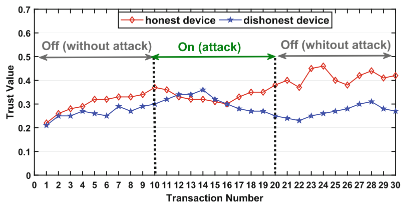
(a) Without attack



(b) Bad-Mouthing and Ballot-Stuffing Attacks



(c) Self-Promoting Attack



(d) On-Off Attack

Fig. 5. The effect of feedback and context on the trust value of a dishonest device and an honest device

6 Conclusion

In SIoT environments, trust evaluation has been taken as an important task [3–5,7,14,15,17,23]. In this paper, we have proposed three contexts of trust, including the status, the environment (time and location) of devices and the task type. Then, we have proposed a Mutual Context-aware Trustworthy Service Evaluation (MCTSE) model. The experimental results on a synthetic dataset have demonstrated that the MCTE model can effectively identify honest and dishonest devices. In our future work, we plan to propose a Mutual Context-aware Trustworthy Service Recommendation model (MCTSR) and validate our model on larger datasets.

References

1. Atzori, L., Iera, A., Morabito, G., Nitti, M.: The Social Internet of Things (SIoT)-when social networks meet the internet of things: concept, architecture and network characterization. Comput. Netw. **56**(16), 3594–3608 (2012)
2. Atzori, L., Iera, A., Morabito, G.: SIoT: giving a social structure to the internet of things. IEEE Commun. Lett. **15**(11), 1193–1195 (2011)
3. Saied, Y.B., Olivereau, A., Zeghlache, D., Laurent, M.: Trust management system design for the internet of things: a context-aware and multi-service approach. Comput. Secur. **39**, 351–365 (2013)
4. Chen, I.R., Bao, F., Guo, J.: Trust-based service management for social internet of things systems, **13**(6), 684–696 (2016)

5. Chen, I.R., Guo, J., Bao, F.: Trust management for SOA-Based IoT and its application to service composition, **9**(3), 482–495 (2016)
6. Bao, F., Chen, R.: Trust management for the internet of things and its application to service composition, Mobile and Multimedia Networks (WoWMoM). In: IEEE International Symposium on a World of Wireless, pp. 1–6 (2012)
7. Chen, Z., Ling, R., Huang, C.M., Zhu, X.: A scheme of access service recommendation for the social internet of things. Int. J. Commun. Syst. **29**(4), 694–706 (2015)
8. Liu, G.: Trust Management in Online Social Networks. PhD thesis, Macquarie University (2013)
9. Guo, J., Chen, I.R.: A classification of trust computation models for service-oriented internet of things systems. In: IEEE International Conference on Services Computing, pp. 324–331 (2015)
10. Zhang, H: Context-Aware Transaction Trust Computation in E-Commerce Environments. PhD thesis, Macquarie University (2014)
11. Lei Li: Trust Evaluation in Service-Oriented Environments. PhD thesis, Macquarie University (2011)
12. Kuter, U., Golbeck, J.: Using probabilistic confidence models for trust inference in web-based social networks. ACM Trans. Internet Technol. **10**(2), 1–23 (2010)
13. Leskovec, J.: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/
14. Lin, Z., Dong, L.: Clarifying trust in social internet of things. IEEE Trans. Knowl. Data Eng. **30**(2), 234–248 (2018)
15. Nitti, M., Girau, R., Atzori, L.: Trustworthiness management in the social internet of things. IEEE Trans. Knowl. Data Eng. **26**(5), 1253–1266 (2014)
16. Sfar, A.R., Natalizio, E., Challal, Y., Chtourou, Z.: A roadmap for security challenges in the internet of things. Digit. Commun. Netw. **4**, 118–137 (2017)
17. Yan, Zheng, Zhang, Peng, Vasilakos, Athanasios V.: A survey on trust management for internet of things. J. Netw. Comput. Appl. **42**, 120–134 (2014)
18. Yang, G.: A health-IOT platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. IEEE Trans. Industr. Inf. **10**(4), 2180–2191 (2014)
19. Zheng, Z., Zhang, Y., Lyu, M.R.: Investigating QoS of real-world web services. IEEE Trans. Serv. Comput. **7**(1), 32–39 (2014)
20. Zou, J., Wang, Y., Orgun, M.A.: A dispute arbitration protocol based on a peer-to-peer service contract management scheme. In: 2016 IEEE International Conference on Web Services (ICWS) (2016)
21. Zheng, Y., Mobasher, B., Burke, R.: Deviation-based contextual SLIM recommenders. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 271-280 (2014)
22. Zhang, Z., Wang, K: A trust model for multimedia social networks. Soc. Netw. Anal. Min. **3**(4), 969–979 (2013)
23. Truong, N.B., Lee, H., Askwith, B., Lee, G.M.: Toward a trust evaluation mechanism in the social internet of thing. Sensors **17**(6), 1346 (2017)
24. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S.: Middleware for internet of things: a survey. IEEE Internet Things J. (2016)
25. Kim, J.E., Fan, X., Mosse, D.: Empowering end users for social internet of things. In: Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, pp. 71-82 (2017)
26. Hussein, D., Han, S.N., Lee, G.M., Crespi, N., Bertin, E.: Towards a dynamic discovery of smart services in the social internet of things. Comput. Electr. Eng. **58**, 429–443 (2017)

27. Nitti, M., Girau, R., Atzori, L., Iera, A., Morabito, G.: A subjective model for trustworthiness evaluation in the social Internet of Things. In: IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC), pp. 18–23 (2012)
28. Chen, D., Chang, G., Sun, D., Li, J., Jia, J., Wang, X.: TRM-IoT: a trust management model based on fuzzy reputation for internet of things. ComSIS **8**(4), 1207–1228 (2011)

Cloudchain: A Blockchain-Based Coopetition Differential Game Model for Cloud Computing

Mona Taghavi[1](✉), Jamal Bentahar[1], Hadi Otrok[1,2], and Kaveh Bakhtiyari[3,4]

[1] Concordia Institute for Information System Engineering, Concordia University, Montreal, Canada
m_tag@encs.concordia.ca, bentahar@ciise.concordia.ca, hadi.otrok@kustar.ac.ae

[2] Department of ECE, Khalifa University, Abu Dhabi, UAE

[3] Interactive Systems, University of Duisburg-Essen, Duisburg, Germany
kaveh.bakhtiyari@uni-due.de

[4] Department of Electrical Engineering, The National University of Malaysia, Bangi, Malaysia

Abstract. In this paper, we introduce, design and develop *Cloudchain*, a blockchain-based cloud federation, to enable cloud service providers to trade their computing resources through smart contracts. Traditional cloud federations have strict challenges that might hinder the members' motivation to participate in, such as forming stable coalitions with long-term commitments, participants' trustworthiness, shared revenue, and security of the managed data and services. Cloudchain provides a fully distributed structure over the public Ethereum network to overcome these issues. Three types of contracts are defined where cloud providers can register themselves, create a profile and list of their transactions, and initiate a request for a service. We further design a dynamic differential game among the Cloudchain members, with roles of cloud service requesters and suppliers, to maximize their profit. Within this paradigm, providers engage in coopetitions (i.e., cooperative competitions) with each other while their service demand is dynamically changing based on two variables of gas price and reputation value. We implemented Cloudchain and simulated the differential game using Solidity and Web3.js for five cloud providers during 100 days. The results showed that cloud providers who request services achieve higher profitability through Cloudchain compared to those providers that supply these requests. Meanwhile, spending high gas price is not economically appealing for cloud requesters with a high number of requests, and fairly cheaper prices might cause some delays in their transactions during the network peak times. The best strategy for cloud suppliers was found to be gradually increasing their reputation, especially when the requesters' demand is not significantly impacted by the reputation value.

Keywords: Cloud service federation · Smart contract · Blockchain Ethereum · Differential game

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 146–161, 2018.
https://doi.org/10.1007/978-3-030-03596-9_10

1 Introduction

To mitigate the issue of underutilized and over provisioned computing resources, cloud providers scaled their pool of resources by forming cloud federations to maximize their profit and provide guaranteed Quality of Services (QoS) [4,6,11]. In spite of the prominent federation advantages, cloud providers are reluctant to participate in due to some strict challenges, mainly: 1- The stability of a federation is a key factor for the cloud providers to ensure their profitability [6]. Such a stability requires long-term commitments from the providers, which is very hard to obtain. 2- A federation needs to address the complications of a fair revenue sharing model to warrant that each cloud provider will gain a revenue according to the amount of computational resources contributed to the federation. 3- The presence of unknown and untrusted participants in a federation can degrade the QoS of the federated services [16]. The trust issue limits conventional federations to enroll only trusted providers and disregard the new ones. 4- Having a large pool of computing resources in a grand coalition might increase the opportunity of botnet attacks. Meanwhile, forming a small federation might hinder the revenue maximization of its participants [2]. 5- There are some security and privacy concerns regarding the managed data and services as well as the creation and management of the cloud federation itself. All the necessary information to manage a federation is usually maintained in a centralized trusted third party. This implies that a federation must maintain roles concerning the authorization to manage the participants' information that yields, which makes not only a single point of failure, but also raises trustfulness concerns [11].

Contributions: This research overcomes the traditional cloud federation issues by contributing a novel architecture and an innovative strategic game model:

1. To provide a practical cooperative solution that any cloud provider can embrace regardless of their market position and trustworthiness, we advocate a fully distributed architecture with a democratic governance structure, called *Cloudchain*. To effectively enforce such a structure, Cloudchain proposes an innovative exploitation of blockchain to prompt and support interoperability and coopetition among the cloud providers over the public Ethereum network. Within Cloudchain, cloud providers endeavor to overcome the resource limitation in their local infrastructure by outsourcing their customers' requests to other members of the Cloudchain. Moreover, it allows providers to access underutilized resources and lease them at cheaper prices. By leveraging blockchain-enabled smart contracts [17], we eliminate the need for trust in the federation and reduce barriers of entry [9].
2. To incentivize the cloud providers and help them make wise decisions about the utilization of Cloudchain, a dynamic differential game is designed, solved and simulated. This game aims to maximize the profit of the Cloudchain members who cooperatively compete while their service demand is dynamically changing. Two variables are considered to impact the cloud provider's revenue, the demand variability and the quality of the provided service: gas cost and reputation value. Gas is a proportional amount that Ethereum pays

to motivate the miners to participate in the mining process and to supply a fair compensation for their computation effort [12]. Reputation value is defined to assign a credibility proportional to the quality that a Cloudchain member provides.

We implement the Cloudchain prototype using Solidity and Web3.js which is available open source in Github[1]. We further simulated the differential game using the Gratner's rating dataset[2] where five real-world providers trade their services. Despite being costlier to transact for cloud-service providers who request a service rather than supply, the obtained results proved it is economically justified to adopt Cloudchain.

2 Related Work

The literature about cloud-providers cooperation focuses on federation formation as coalitional games where capacity and revenue are shared [15]. Coronado et al. had an intensive investigation on federation-formation variables among cloud providers, including revenue sharing mechanisms, capacity and cost disparity, and the presence of a big competitor [5]. They defined revenue sharing mechanisms as the most important factor. Among these mechanisms, shapely value and outsourcing models had the least and best performance, respectively. They indicated that collaborating cloud providers can implement a mechanism in which a provider outsources some of its business and gets a percentage of the revenue. The outsourcing model allows the provider to keep some of the revenue of its secured business, even though it is not able to fulfill that business alone. The authors had an insight through the demand peaks and concluded that cloud providers tend to stay in outsourcing collaboration when the demand is high. However, interoperability, trust among cloud providers and service quality or SLA are not considered in their study. The findings from this study confirm the superiority of outsourcing in terms of maximizing the profit of cloud providers, which is what we are proposing in this paper in addition of having the advantage of coopetition among cloud providers. The fact that providers tend to collaborate when they face a hike in their demand, reinforces the consideration of a dynamic and long/short-term federation like Cloudchain. The challenges of interoperability and trust issues among cloud providers are also addressed by the blockchain platform we propose in this paper. Another cloud outsourcing model has been performed by Chen et al. [4] who analyzed the interrelated workload factoring and coalition formation game among private clouds. The authors integrated two types of federations: (1) vertical (outsource workload to public clouds), ad (2) horizontal (share resources with other private clouds. Their experiments found this approach to be promising to improve the cloud's service quality and decrease the delay by 11%. However, their research was limited to service quality and economic aspects of stable cooperation patterns without

[1] https://github.com/kavehbc/Cloudchain.
[2] https://www.gartner.com/reviews/market/public-cloud-iaas.

considering other challenges of a traditional federation explained in the previous section.

Very few efforts have been made to study the potential of blockchain in real-world applications despite its great potential for businesses to share data and collaborate in a secure and customized manner [13]. According to Tractica, a market research firm, the annual revenue for enterprise applications of blockchain is estimated to increase to $19.9 billion by 2025 [8]. The majority of studies about blockchain's application have focused on finance [19], energy [14] and IoT applications [21]. In cloud computing and service industry, to the best of our knowledge, there has been only one academic initiative that proposed a cloud marketplace based on the blockchain technology. Klems et al. designed Desmaa, a conceptual framework for trustless intermediation in service marketplaces using blockchain [9]. This conceptual framework modeled the interactions between a service provider and a service consumer and tried to overcome problems of conventional marketplace systems, such as barriers of entry and transaction costs. Yet, the outsourcing model with collaboration and competition among cloud providers themselves are not considered in their research. Moreover, the providers' profit and the best strategies for utilizing this marketplace is not elaborated nor modeled. Even though the authors developed a prototype, no evaluation and validation against real-world's scenarios were provided.

3 Cloudchain Architecture

Cloudchain incorporates three types of smart contracts including a set of executable functions and state variables. Similar contracts are proposed in [1] in the context of medical data management. *Contract 1* (*C1*) or Cloudchain Registery (CCR) is a global contract that maps cloud providers identification values (including *Name*, *Reputation Value*, *Computing Capacity* and *Storage Capacity*) to their Ethereum address identities (equivalent to public keys). The reputation values can be computed from the customers' ratings given to each provider through online rating platforms. Policies coded into the contract can regulate registering new providers or changing the mapping of the existing ones. The cloud provider registration can be restricted only to certified providers. CCR also maps identities to the Cloudchain Contract (CCC) address on the blockchain, where a special contract regarding each provider profile and list of services is recorded.

Contract 2 (*C2*) denotes Cloudchain Profile (CCP). It holds a list of references to CCC, representing all the participants' previous and current engagements with other nodes in the system. CCP also implements a functionality to enable provider notifications. Providers should register their requests in this contract. Each transaction list stores a status variable. This indicates whether the transaction is newly established, awaiting pending updates and has or has not been completed. This contract is important as it stores the address of all new CCC contracts, without which Cloudchain can simply lose the track of all the contracts.

Contract 3 (*C3*) represents the Cloudchain Contract (CCC). It is issued between two nodes in the system when one node accepts and provides the requested service for the other. The beneficiaries can also complete, or cancel the contract. Once the contract is completed or canceled, the contract balance would be transferred to the supplier-, or requester address respectively, and the contract status would also be updated. There are two approaches to reduce the size of the data as well as the cost of transactions over Cloudchain. The first approach is a common practice for data storage in smart contracts and consists of storing raw data off-chain, and meta-data, small critical data, and hashes of the raw data on-chain [20]. However, the selection of off-chain data storage has some concerns regarding the interaction between the blockchain and the off-chain data storage. The other approach is to provide a common glossary among cloud providers to define the generic terms and policies to be referred to in the contract.

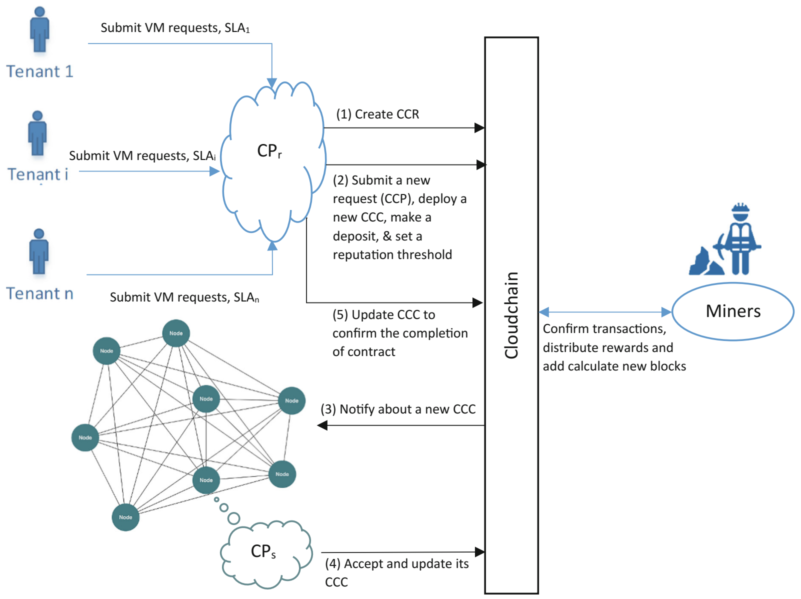

Tenant 1
Submit VM requests, SLA₁
Tenant i
Submit VM requests, SLAᵢ
Tenant n
Submit VM requests, SLAₙ
CPᵣ
(1) Create CCR
(2) Submit a new request (CCP), deploy a new CCC, make a deposit, & set a reputation threshold
(5) Update CCC to confirm the completion of contract
(3) Notify about a new CCC
(4) Accept and update its CCC
CPₛ
Node
Cloudchain
Miners
Confirm transactions, distribute rewards and add calculate new blocks

Fig. 1. Cloudchain interactions

Figure 1 provides the steps taken by the Cloudchain members to register and establish their requirements by interacting via Cloudchain. In step 1, a provider registers in CCR. Each registered user is assigned with a public key pair. Guaranteed SLA tenants require performance consistency and scale predictability. When a member faces a computing-resources deficiency to meet its end users' demand with guaranteed SLA, it can submit a request for a service using CCP to deploy a CCC to the blockchain in step 2. Requesters are required to pay a deposit in advance and it is stored in the contract. Meanwhile, a rule for providers is set by the requesters to ensure that qualified providers could ultimately receive the task, e.g. reputation value threshold. Function calls on contracts are transactions, and those which update the contract storage need to be validated by miners. Once a new block is mined with the newly linked CCC, it would be broadcasted to other nodes in step 3. Through step 4, the first node that accepts the request should update the respective CCC contract. Each provider who accepts a task should deposit some coins or its reputation value to guarantee the quality of the task. The contract termination and delivery of the requested service have to be confirmed by the service requester in step 5. The requester is required to rate the supplier based on the received service quality.

4 Cloudchain Members' Revenue Optimization

A true blockchain-led federation will not happen unless cloud providers are widely engaged and able to manage the costs and properly play their role. The use of a differential game [3] is motivated by the need to model the time constrained and dynamic strategies of selfish cloud providers willing to maximize their own revenue. Let us consider two typical cloud providers (CP) over Cloudchain, CP_r as the provider who is facing a peak time and is going to request some VMs from other Cloudchain members, and CP_s as the Cloudchain member that has some idle servers and is willing to rent them out with the price offered by CP_r. For simplicity and without losing generality, we will focus on a single VM type, with ϕ denoting the capacity and the process rate of VM instances that can be hosted by a typical CP_j that can be CP_r or CP_s.

To make a request and create a contract, CP_r has to define the price of gas G_r for the created transaction (e.g. 5 gwei). If the price is high enough, the transaction will be executed sooner, since miners will execute transactions with the highest gas price first. If the price is set too low, CP_r may end up waiting longer for execution of its transaction and distribution of its request. This waiting time may degrade the service quality for its users and hinder its profit. On the other hand, setting a high gas price for every single transaction and update incurs higher costs. So, the gas price is a decisive factor in profit optimization and we define it as the control path at time t for CP_r, denoted by $G_r(t)$. In this game, VM price is assumed to be given by CP_r. To be qualified to supply a cloud service, the provider CP_s has to maintain a good reputation R_s which is given based on the quality of service for end users and the quality of collaboration (e.g. speedy communication) with the cloud provider that requested the service, CP_r. Even though the reputation value is given by CP_r and not CP_s itself, yet it has a control over this value through the service quality and gas price of its own transactions. Therefore, $R_s(t)$ is considered as the control path of CP_s to coopete with CP_r within Cloudchain to gain higher profit. Table 1 provides a summary of the notations used in our model.

In order to capture the demand elasticities and variations specific for each user, we define the user demand using the Cobb-Douglas function that models well these elasticity aspects in terms of price and reputation, adopted from [18]. It is assumed that the user will have the opportunity to check the cloud provider

Table 1. Notations used in Cloudchain

u	End user index
$r, s \in \{1, 2, \ldots, j, \ldots, k\}$	Requester and supplier in the set of k cloud providers in Cloudchain
G	Cost of gas
M	The amount of the cumulated gas for each block
$M^{'}$	The amount of required gas for each transaction
X	Number of the transactions occurring over Cloudchain
ω	Rate of transactions arrival for a CP in $[0 - T]$
Γ	Rate of the block generation for miners in Cloudchain
$\phi/\phi^{'}/\phi^{''}$	Provider's active/idle/mining capacity
$p/p^{'}$	Price per VM for the end user/for the members of Cloudchain
R	Reputation value of cloud provider in Cloudchain
Rw	Reward value of mining
τ	Block propagation time
η	Rate of the impact of M over τ
$\check{\theta}$	Rate of CP_r demands rise due to the higher quality services of CP_s
$\hat{\theta}$	Rate of $D_r^{'}$ demands increase due to higher reputation of CP_s
ψ	Rate of CP_r demands increase due to higher gas and higher quality
α_u/β_u	CP price/rating variation for user u
δ	Demand decay rate
μ	The amount of VMs
$C/C^{'}$	Cost of the primary/outsourced capacity

rating that represents the actual user satisfaction level and reputation value defined through Cloudchain. The user demand function is defined as follows:

$$D_u = \mu \, p^{-\alpha_u} \; R^{\beta_u} \tag{1}$$

In the mining race, miners have to compete to solve proof of work and propagate the block to reach consensus. The new blocks' generation follows a Poisson process with a constant rate $\frac{1}{\Gamma}$ throughout the whole Cloudchain network [10]. Before the race, miners collect their selected pending transactions into their blocks with a total gas amount of $\sum_{j=1}^{k} M_j$. When miner j propagates its block to Cloudchain for consensus, the time for verifying each transaction is affected by the size of transactions M_j. The first miner j who successfully has its block achieves consensus will be rewarded based on the amount of the assigned capacity $\phi_j^{''}$. Thus, miner j's expected reward $Rw_j(\phi_j^{''})$ is:

$$Rw_j(\phi_j^{''}) = Rw_j \mathbb{P}_j(\phi_j^{''}, M_j) \tag{2}$$

where $\mathbb{P}_j(\phi_j^{''}, M_j)$ is the probability that miner j receives the reward by contributing a block. To win the reward, provider must perform a successful mining and instant propagation. The miner may fail to obtain the reward if its new block does not achieve consensus as the first. This kind of mined block that cannot be added to the blockchain is called orphaned block. The block containing a larger size of transactions has a higher chance of becoming orphaned since a larger block requires more propagation time, thus, causing a higher delay for consensus. As the arrival of new blocks follows a poisson distribution, miner j's orphaning probability, $\mathbb{P}_j^0$, can be approximated as:

$$\mathbb{P}_j^0 = 1 - exp(-\frac{1}{\Gamma})\tau_j \tag{3}$$

Here, we assume miner j's block propagation time τ_j is linear with the size of transactions in its block, $\tau_j = M_j \eta_j$, where η_j is a constant that reflects the impact of M_j over τ_j. Therefore, we obtain the reward probability as follows:

$$\mathbb{P}_j(\phi_j^{''}, M_j) = 1 - \mathbb{P}_j^0 = \phi_j^{''} e^{-\frac{1}{\Gamma} M_j \eta_j} \tag{4}$$

Substituting Eq. 4 into Eq. 2 provides an estimation of total revenues that CP_j may obtain by attending the mining tournament. To model the transactions' distribution, we use the compound Poisson process, which is a generalization of the Poisson process where each arrival is weighted according to a distribution. The compound Poisson process represents better the transactions dynamics. In this case, the assumption is that transactions sent to Cloudchain follow a Poisson process, but the amount of gas they require follows a compound Poisson process. The reason is that the difference between the amount of gas is based on the complexity of the transaction, for example, the creation of a contract requires a much higher amount of gas than updating the contract. Therefore, the probability of the required gas by X_j transactions occurring in $[0-T]$ follows an exponential distribution based on the compound Poisson process as follows:

$$\mathbb{P}_j(X) = \frac{e^{-\omega T}(\omega T)^{X_j}}{X_j!} \tag{5}$$

4.1 Cloud Provider as a Requester

Here we explain the scenario from the perspective of CP_r that has to optimize its profit CPP_r while requesting VMs as follows:

$$\begin{aligned} CPP_r(G_r(t), D_r^{'}(t), t) &= (p_r - \ \phi_r \ C_r) \ D_r + (p_r - p_s^{'}\phi_s^{'})D_r^{'}(t) \\ &- \frac{e^{-\omega T}(\omega T)^{X_r}}{X_r!} M_r^{'} G_r(t) + Rw_r \phi_r^{''} e^{-\frac{1}{\Gamma} M\eta} \end{aligned} \tag{6}$$

$M_r^{'}$ represents the amount of required gas that depends on the complexity of the transaction a provider wants to initiate. The transaction fees go to the miner that mines the block, so if a provider attends a mining process, it will be rewarded according to Eqs. 2 and 4. $D_r^{'}(t)$ is the demand that CP_r intends to outsource to obtain the idle capacity of $\phi_s^{'}$ for a secondary price of $p^{'}$. Considering the time-dependent profit functions of CP_r in Eq. 6, the objective function is the total discounted cloud provider's payoff over the planning horizon $[0-T]$:

$$\begin{aligned}
\text{maximize} \quad & \int_0^T e^{\rho t}\{CPP_r(G_r(t), D_r^{'}(t), t)\}dt \\
\text{subject to} \quad & \dot{D}_r^{'}(t) = G_r^{\beta_u}(t)\psi_r + \check{\theta} R_s^{\beta_u}(t) - \delta_r D_r^{'}(t) \\
& D_r^{'}(0) = D_{0r}^{'}
\end{aligned} \tag{7}$$

The users' demands evolution over time is represented as $\dot{D}_r^{'}(t)$ for CP_r that increases when the service quality rises. The service quality is aggregated through two factors of gas price that CP_r pays and the reputation of CP_s. The demand decays at a certain rate of δ_r. It is important to note that Eq. 7 formulates an optimal control problem with the gas price as a control variable and the cumulative demand of CP_r as a state variable. The analysis of differential games relies profoundly on the concepts and techniques of optimal control theory [7]. To study the dynamics of the payoff function and the path of control variable, we leverage the Hamiltonian systems. Equilibrium strategies in the open-loop structures can be found by solving a two-point boundary value problem for ordinary differential equations derived from the Pontryagin maximum principle in Hamiltonian functions. The Pontryagin maximum principle gives the necessary condition for a control path to be optimal open-loop control. To acquire the optimal control, we first formulate the Hamiltonian system of the cloud provider's payoffs:

$$\begin{aligned}
H_r(G_r(t), D_r^{'}(t), \lambda_r(t), t) = (p_r - \ \phi_r \ C_r) \ D_r(t) + (p_r - p_s^{'}\phi_s^{'})D_r^{'}(t) \\
- \frac{e^{-\omega T}(\omega T)^{X_r}}{X_r!} M_r^{'} G_r(t) + R w_r \phi_r^{''} e^{-\dfrac{1}{\Gamma} M\eta} \\
+ \lambda_r(t)(G_r^{\beta_m}(t)\psi_r + \check{\theta} R_s^{\beta_m}(t) - \delta_r D_r^{'}(t))
\end{aligned} \tag{8}$$

According to the control theory, the optimal control strategy of the original problem must also maximize the corresponding Hamiltonian function. Thus, based on the Pontryagin maximum principle, the candidate optimal strategy has to satisfy the following necessary conditions:

$$\frac{\partial H_r(t)}{\partial G_r(t)} = -\frac{e^{-\omega T}(\omega T)^{X_r}}{X_r!} M_r^{'} + \lambda_r(t)\beta_m G_r^{\beta_m - 1}(t)\psi_r = 0 \tag{9}$$

$$\dot{\lambda}_r(t) = \rho\lambda_r(t) - \frac{\partial H_r(t)}{\partial D_r^{'}(t)} = (\rho + \delta_r)\lambda_r(t) - p_r + p_s^{'}\phi_s^{'}, \lambda_r(T) = 0 \tag{10}$$

When only one boundary condition is specified as $D_r^{'}(0) = D_{0r}^{'}$, the free-end condition is used as $\lambda_r = 0$ at $t = T$. The formulated differential equation Eq. 10 can lead us to the adjoint variable:

$$\lambda_r(t) = \frac{p_r - p_s^{'}\phi_s^{'}}{\rho + \delta_r}(1 - e^{(\rho+\delta_r)(t-T)}) \tag{11}$$

Replacing Eq. 11 in Eq. 9 gives us the optimal gas price control path as follows:

$$G_r^*(t) = \left(\frac{M_r^{'} e^{-\omega T} (\omega T)^{X_r} (\rho + \delta_r)}{X_r!(p_r - p_s^{'}\phi_s^{'})(1 - e^{(\rho+\delta_r)(t-T)})\beta_m \psi_r}\right)^{\frac{1}{\beta_m - 1}} \tag{12}$$

4.2 Cloud Provider as a Supplier

Cloud provider as a supplier has a different scenario. CP_s observes the total demand of its own users, D_s, and the capacity preserved for the mining process to determine the remaining capacity $\phi_s^{'}$, to optimize its profit as follows:

$$\begin{aligned} CPP_s(R_s(t), D_r^{'}(t), t) = (p_s - \ \phi_s\ C_s)\ D_s + (p_s^{'} - \ \phi_s^{'}\ C_s^{'}) D_r^{'}(t) \\ - \frac{e^{-\omega T} (\omega T)^{X_s}}{X_s!} M_s^{'} G_s(R_s(t)) + R w_s \phi_s^{''} e^{-\dfrac{1}{\Gamma} M\eta} \end{aligned} \tag{13}$$

$G(R_s(t))$ denotes the gas cost that the suppliers pay to earn higher reputation for having prompt communication. Considering the time-dependent profit functions of CP_s in Eq. 13, the objective function is the total discounted cloud provider's payoff over the planning horizon $[0 - T]$:

$$\begin{aligned} \text{maximize} \quad & \int_0^T e^{\rho t} \{CPP_s(R_s(t), D_r^{'}(t), t)\} dt \\ \text{subject to} \quad & \dot{D}_r^{'}(t) = \hat{\theta}_r R_s(t)^{\beta_n} - \delta_s D_r^{'}(t) \\ & D_r^{'}(0) = D_{0r}^{'} \end{aligned} \tag{14}$$

The demand dynamics of CP_s is defined based on the demand that it receives from CP_r that evolves with its own reputation and decays at a rate δ_s. By solving a corresponding Hamiltonian system of Eq. 14, similar to Eq. 8, the optimal reputation control path is obtained as follows:

$$R_s^*(t) = \left(\frac{M_s^{'} e^{-\omega T} (\omega T)^{X_s} G_s (\rho + \delta_s)}{X_s!(p_s^{'} - \phi_s^{'}\ C_s^{'})(1 - e^{(\rho+\delta_s)(t-T)})\beta_n \hat{\theta}_r}\right)^{\frac{1}{\beta_n - 1}} \tag{15}$$

5 Implementation, Simulation and Discussion

We implemented the coopetitive Cloudchain prototype on Ethereum using Solidity (version 0.4.24), the script language on Ethereum, to test our proposed framework and the effect of gas price and reputation values on cloud providers revenues. This program is available open source in Github (See footnote 1). The program was written with the main concern of the minimum consumption of gas per each transaction and was tested using remix[3], an online IDE for Solidity.

[3] http://remix.ethereum.org/.

The gas price unit is in gwei, which is 1×10^{-9} ether. Ethereum stores arbitrary data in smart contracts in two ways. The first option is to store the data as a variable in a smart contract. The cost of storing data in the contract storage is based on the number of SSTORE operations on the contract variable. The second option is to store arbitrary data as a log event. There are also memory variables such as contract arguments and defined memory variables, which are not stored permanently inside the contracts. Memory variables are disposed after the function execution is complete. In our implemented prototype, we used solidity structures and variables to store provider's data and requests inside the contracts. Meanwhile, each transaction is logged with a summary using an event to make it easily accessible for the other providers (blockchain nodes) to track new transactions. Once a new transaction with a specific event (e.g. New Request) is created, other providers can call the contract to get more information and/or change contract stored data (e.g. to accept a new request). Calling a contract and retrieving data are expensive transactions, the stored data as events can provide enough information without any retrieval cost. The events are retrieved and filtered using the Web3.js platform to notify the providers on important changes (e.g. New registration, updates, deactivations, new requests, etc.) in Cloudchain. CCR and CCP contracts are deployed once, but CCC would be deployed every time a new request is registered.

Table 2. Provider's estimated transactions and costs on Cloudchain based on the proposed scenarios

	Amazon EC2	Microsoft Azure	Rackspace	Century Link	Alibaba cloud
Reputation value	88	82	84	60	82
Price per hour (p)	0.0058	0.005	0.084	0.025	0.0125
Price per hour (p')	0.003	0.0025	n/a	n/a	n/a
Requests[a]	0	0	8	15	17
Supplies[a]	23	17	0	0	0
Cancellations [a]	0	0	0	3	2
Total gas	1,290,668	953,972	15,292,736	34,310,286	36,254,668
Gas price (gwei)[b]	15	15	15	12	11
Gas cost (gwei)[c]	19,360,020	14,309,580	229,391,040	411,723,432	398,801,348
Gas cost (USD)[c]	$12.06	$8.91	$142.91	$256.50	$248.45
Transaction delay (s)[d]	27-66	27-66	27-66	27-4000	27-5459

[a]Quantity [b]Total Gas×Gas Price [c]Average [d]Time range of each transaction in seconds

For the sake of representation, we assumed a small number of 5 cloud providers (Amazon, Microsoft, Rackspace, Alibaba cloud, and Century Link) using Cloudchain for a duration of 100 days to investigate their economic gain through the differential game. The scalability of our system for higher number of cloud providers is not questioned since the Ethereum platform is proven to be scalable. We simulated Rackspace, Alibaba and Century Link as cloud requesters who make 8, 17 and 15 requests of service, respectively. Meanwhile, Amazon accepts Rackspace and Century Links requests with a reputation threshold of 75, and Microsoft takes Alibaba's orders, which were set for a minimum reputation of 85. Due to the limitation of Solidity in defining float numbers, we scaled the reputation values collected from Gartner to [0–100]. The on-demand cloud services' prices are borrowed from the providers' websites with an assumption of the secondary price of Amazon and Microsoft to be two times less for the Cloudchain members. The collected real-world data (e.g. reputation and price), simulated number of requests and supplies, as well as the simulated results of total gas consumption, gas price and transactions delays are shown in Table 2. Since there is no time-dependent profit maximization model similar to our proposal, not even in traditional centralized federations or related experiments to be compared to, only the results of our model are reported.

In our simulated scenario, three cloud providers of Amazon, Microsoft and Rackspace are supposed to be miners and collect their rewards. To make the simulation more realistic, we followed up all the contract transactions from registering in the Cloudchain up to confirmation of the contract completion, depositing the payment and assigning a reputation. Century Link and Alibaba are assumed to cancel their requests for few times after making the contract before acceptation. As Table 2 depicts, the obtained gas consumptions of cloud service requesters are much higher than those that answer these requests and supply these services. This is why Alibaba has the most and Microsoft the least gas consumption.

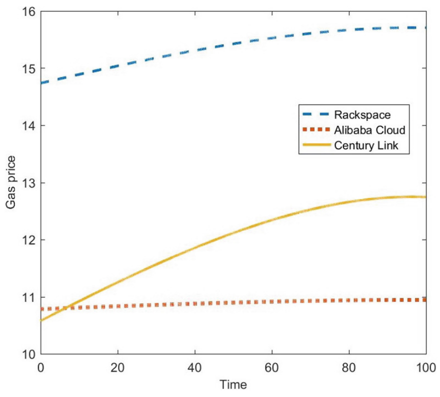


Fig. 2. Gas prices of the three cloud service requesters

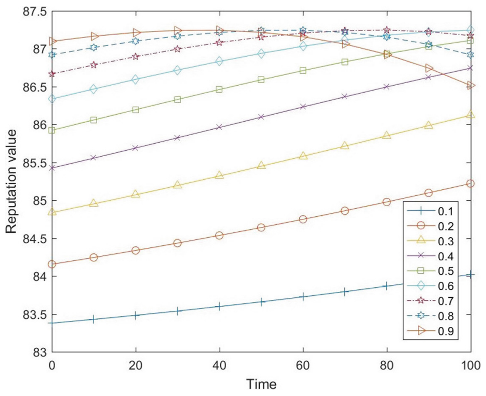


Fig. 3. Microsoft's optimal reputation with different values of $(0.1 \leq \hat{\theta}_r \leq 0.9)$

The gas price of Amazon and Microsoft are considered as constant inputs and they are set to 15. This price guarantees a fast execution of transactions to avoid tarnishing their reputation and will not impose them huge cost due to their minimal gas required as the role of suppliers. To estimate the time delay for each transaction, we tested different prices in different time slots to obtain an

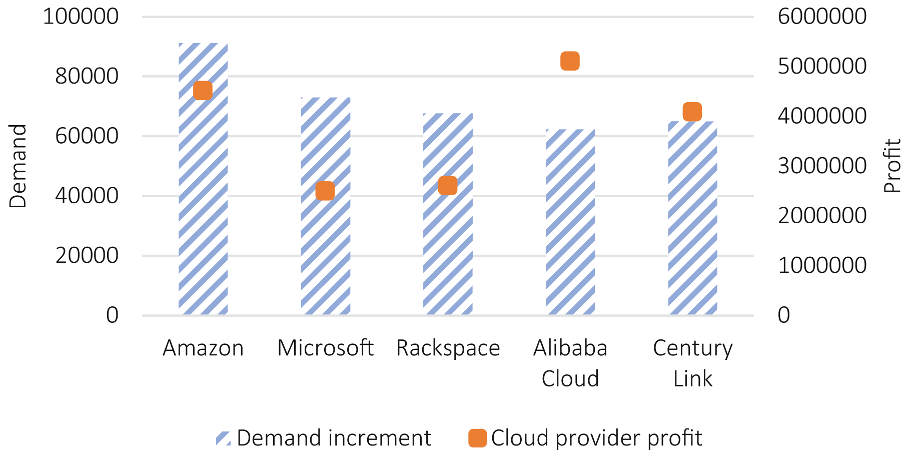


Fig. 4. Average of cloud providers' profit and demands' evolution in Nash Equilibrium

approximate range of delay depending on the traffic of the Ethereum network. The obtained optimal gas prices for the three cloud requesters are shown in Fig. 2. Alibaba has to pay the minimal price, which is almost 11 gwei for the whole period of time. This cloud provider has the highest number of requests, so it is not profitable if it invests more money over gas. With this price, Alibaba has to pay almost $248.45, at the time of writing this paper. However, because of the cheap gas price, Alibaba has a delay of 27 to 5459 s for each transaction (refer to Table 2). Even though high traffic happens not very often, yet, it would be advisable to predict its demand in advance to avoid the delays that can cause user dissatisfaction. Century Link also has to pay cheap gas price, but not as cheap as Alibaba. It is reasonable since this cloud provider has less gas consumption, higher end-users' prices and lower reputation. To win the users' satisfaction proportional to its service's price, Century Link has to increase the gas price sharply, to speed up the communication and avoid major delays. Based on the results, Rackspace has to pay the highest price for the gas among the cloud requesters. The main reason can be the highest end users' price, the low amount of transactions and gas consumption. The participation in the mining process could also add up to its wealth to afford higher price and higher quality with minimum delays. It worth to note that even though the gas is costly for all cloud providers, it is a one-time cost for a permanent storage.

Figure 3 depicts Microsoft's optimal reputation value during these 100 days as obtained in our experiment. It is worth mentioning that Amazon showed a similar pattern. To investigate the behavior of cloud requesters' demand over these reputation values, we considered the demand rate $\hat{\theta}_r$ varying from 0.1 to 0.9. As the effectiveness of reputation over demands' rate raises, the provider has to aim for a higher reputation at the beginning to earn the eligibility for more demands. However, these optimums do not follow the same trend. In the case of lower effectiveness, the provider has to increase the service quality and gas price leading to a higher reputation over time, but as effectiveness gets intense, the reputation starts to decline. This is where the provider has established its credibility at first and made the major profit halfway through the period, and the increase of reputation is not profitable anymore. This confirms that keeping a high reputation is costly and not always economically justified.

Figure 4 presents a comparative analysis of the average of profit and demands' evolution for the Cloudchain members. The demands' evolution $\dot{D}_r^{'}(t)$ for cloud service suppliers have noticed a higher spike. Yet, interestingly, cloud service requesters have received a higher profit from Cloudchain due to fulfilling their initial demand and selling to their own end-users. The cloud service requesters could obtain cheaper prices from the suppliers and sell at their own prices. However, it should be noted that they can face the risk of not fulfilling their commitments to the end-users if none of the suppliers have the required preserved capacity to rent out. Although it seems that Cloudchain benefits more the cloud requesters, yet it is not true. The main profit of cloud suppliers is from their own market and users, and they only rent the partial idle computing resources, which are not being used. As the number of cloud service requesters elevates, their share of profit from the outsourced demand and the mining rewards increases.

6 Conclusion

In this paper, we introduced a new distributed blockchain-based framework for cloud providers federation to overcome the limitations of conventional centralized federations. Due to the coopetitive environment of Cloudchain, and high expense of public smart contracts, we further designed and solved a differential game. This game modeled the best strategies of cloud providers to make a request with an optimal transaction cost and time, as well as, to optimize their reputation value to receive the requests from other providers. Cloudchain was implemented using Solidity over the Ethereum network and the differential game was simulated for a sample of five cloud providers during 100 days. The findings can be summarized from two perspectives of the cloud service requesters and suppliers. For cloud requesters with a high number of requests, spending high gas price is not economically appealing. With cheaper gas prices, they might face some delays in peak times, which needs to be predicted in advance. Although requesters incurred higher costs from Cloudchain, yet they gained a significantly high income by outsourcing some parts of their customers' demands that could not be fulfilled by their own. The results showed that cloud suppliers have minimal gas consumption, which makes it more affordable for them to pay higher prices and enhance their communication and reputation. Though increasing the reputation was not always the best strategy for highly reputed cloud providers, a gradual increase is recommended when the requesters' demand is not significantly impacted. The end-user's service price is found to be a very decisive factor in deciding the level of quality and gas/reputation values for both of the cloud service requesters and suppliers.

References

1. Azaria, A., Ekblaw, A., Vieira, T., Lippman, A.: MedRec: using blockchain for medical data access and permission management. In: International Conference on Open and Big Data, pp. 25–30 (2016)
2. Bairagi, A.K., Alam, M.G.R., Talukder, A., Nguyen, T.H., Hong, C.S., et al.: An overlapping coalition formation approach to maximize payoffs in cloud computing environment. In: 2016 International Conference on Information Networking, pp. 324–329 (2016)
3. Basar, T., Olsder, G.: Dynamic Noncooperative Game Theory, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (1998)
4. Chen, H., An, B., Niyato, D., Soh, Y.C., Miao, C.: Workload factoring and resource sharing via joint vertical and horizontal cloud federation networks. IEEE J. Sel. Areas Commun. **35**(3), 557–570 (2017)
5. Romero Coronado, J.P., Altmann, J.: Model for incentivizing cloud service federation. In: Pham, C., Altmann, J., Bañares, J.Á. (eds.) GECON 2017. LNCS, vol. 10537, pp. 233–246. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68066-8_18
6. Hassan, M.M., Alelaiwi, A., Alamri, A.: A dynamic and efficient coalition formation game in cloud federation for multimedia applications. In: Proceedings of the International Conference on Grid Computing and Applications, p. 71 (2015)
7. Hocking, L.M.: Optimal Control: An Introduction to the Theory with Applications. Oxford University Press, Oxford (1991)
8. Jiao, Y., Wang, P., Niyato, D., Xiong, Z.: Social welfare maximization auction in edge computing resource allocation for mobile blockchain. arXiv preprint arXiv:1710.10595 (2017)
9. Klems, M., Eberhardt, J., Tai, S., Härtlein, S., Buchholz, S., Tidjani, A.: Trustless intermediation in blockchain-based decentralized service marketplaces. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 731–739. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_53
10. Kraft, D.: Difficulty control for blockchain-based consensus systems. Peer-To-Peer Netw. Appl. **9**(2), 397–413 (2016)
11. Lee, C.A.: Cloud federation management and beyond: requirements, relevant standards, and gaps. IEEE Cloud Comput. **3**(1), 42–49 (2016)
12. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 254–269 (2016)
13. Mendling, J., et al.: Blockchains for business process management-challenges and opportunities. ACM Trans. Manag. Inf. Syst. **9**(1), 4 (2018)
14. Münsing, E., Mather, J., Moura, S.: Blockchains for decentralized optimization of energy resources in microgrid networks. In: Conference on Control Technology and Applications, pp. 2164–2171 (2017)
15. Niyato, D., Vasilakos, A.V., Kun, Z.: Resource and revenue sharing with coalition formation of cloud providers: game theoretic approach. In: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 215–224 (2011)
16. Ray, B., Saha, A., Khatua, S., Roy, S.: Quality and profit assured trusted cloud federation formation: game theory based approach. IEEE Trans. Serv. Comput. (2018). https://doi.org/10.1109/TSC.2018.2833854

17. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997). https://doi.org/10.5210/fm.v2i9.548
18. Taghavi, M., Bentahar, J., Otrok, H., Wahab, O.A., Mourad, A.: On the effects of user ratings on the profitability of cloud services. In: International Conference on Web Services (ICWS), pp. 1–8 (2017)
19. Underwood, S.: Blockchain beyond bitcoin. Commun. ACM **59**(11), 15–17 (2016)
20. Xu, X., et al.: A taxonomy of blockchain-based systems for architecture design. In: International Conference on Software Architecture, pp. 243–252 (2017)
21. Zhang, Y., Wen, J.: The IoT electric business model: using blockchain technology for the internet of things. Peer-To-Peer Netw. Appl. **10**(4), 983–994 (2017)

Business Services and Processes

Prediction of Invoice Payment Status in Account Payable Business Process

Tarun Tater(✉), Sampath Dechu(✉), Senthil Mani(✉), and Chandresh Maurya(✉)

IBM Research AI, Bengaluru, India
{ttater03,sampath.dechu,sentmani,cmaurya1}@in.ibm.com

Abstract. Account payables are amount owed to vendors for goods and services delivered to a company. Vendors raise invoices which go through several processing steps before they are paid by a company. Companies have contractual obligations with vendors for paying the invoices within a stipulated time. Invoices that exceed this time attract penalty and affect vendor satisfaction to work with the company. It is very critical for large firms dealing with thousands of vendors for their day to day operations to meet the service level agreements with vendors to avoid penalties. Any assistance for practitioners, warning them of potential invoices that can breach the service level agreements, can help them in minimizing the penalties. In this research, we model the problem of identifying delayed invoices as a supervised classification task. There are three characteristics of this problem which are challenging from a classification perspective: (i) the status of an invoice is affected by other invoices that are simultaneously being processed, as there are limited resources to process the huge volume of invoices, (ii) feature engineering to capture the temporal aspect of the invoice and having the optimal representation of the multiple data entries created per invoice, and (iii) the number of paid late invoices are much smaller in percentage compared to paid on time invoices in the training data set, hence the classes are imbalanced. The results obtained by training an ensemble of classifiers show that penalties can be avoided on more than 82% of the invoices which are currently being penalized.

Keywords: Account payables · Vendor invoice management
Business process optimization · Predictive monitoring

1 Introduction

Procure to Pay (P2P)[1] is a business process that integrates functionalities of purchasing and Accounts Payables (AP) departments in large enterprises. Invoice processing is a cumbersome process involving both manual and automated steps. *Vendor* raises an invoice once he/she supplies goods/services to a *company*. It is

[1] https://wiki.scn.sap.com/wiki/pages/viewpage.action?pageId=14508.

common for large enterprises to deal with thousands of vendors and pay millions of invoices per year. Companies have contractual agreements with the vendors to pay invoices in a stipulated time. *Paid Late* invoices attract penalties and strain the relationship between the company and the vendor. In an independent market research [1] across 500 accounts payable (AP) departments, conducted in UK in 2015, the top concern expressed by participants was mitigating the *paid late* invoices.

An invoice goes through several stages of scrutiny before it gets paid by the account payables department. Large enterprises have dedicated teams for invoice processing and payments. These teams are equipped with enterprise work flow tools such as SAP VIM[2] and similar ERP applications to ease the orchestration of invoice processing, involving multiple levels of sanctity checks to validate the invoice for payment. Most of the process work flow tools log the actions performed on the invoice, the actor who performed the actions, start and end time for every action taken, as change logs or event logs such as those generated by SAP VIM. Companies have organizational roles such as *Functional process owner*, *Regional process owner*, *Global process owner* who use these logs to check on the health of the process, identify bottlenecks and take actions to improve process efficiency and efficacy. There could be multiple reasons for an invoice to get delayed for payment:

1. *Information Mismatch* between Vendor's provided details & Invoice details.
2. *Limited Resources* available for processing the invoice leading to high priority invoices delaying the processing of low priority invoices.
3. *Invoice Expiry*, which is the number of days remaining for the invoice to be processed also affects the priority and assignment of resources to process the invoice.

Identifying invoices which may be delayed is a laborious task due to the large volume of invoices and their attributes. Some of the mentioned problems such as *information mismatch* can be addressed by building software capabilities like data validation in the invoice management system. However, such validation which might require re-work by vendors, does not impact the due payment date of the invoice. Given such constraints, we have approached the problem of predicting invoices' status ("Paid late" or "Paid on time") as a supervised binary classification task. We consider the invoice details and the logs of actions taken on an invoice to predict if the invoice is likely to be delayed and flag it for moderation by concerned people at early stages of processing. This would help in allocating resources appropriately to minimize the penalties being incurred due to delayed payments.

Predicting invoice payment status will enable the *process owners* to take remedial actions such as prioritizing the invoice for processing or re-negotiating contracts with vendors. In our scenario, the problem translates to placing greater emphasis on the *paid late* invoices. In our dataset, approximately 10.3% of the

[2] https://www.opentext.com/what-we-do/products/opentext-suite-for-sap/opentext-vendor-invoice-management-for-sap-solutions.

invoices are *paid late*. We focus on predicting the *paid late* class (minority class) with high precision as identifying the wrong invoices for moderation, will affect the processing of other invoices, resulting in a vicious cycle. Similarly, we need to have a high recall for the *paid late* class to avoid penalties on as many invoices as possible. The metric used for evaluation in related work [16] which is *accuracy*, can be misleading [5] in our scenario. For example, if we simply predict all invoices to be paid on time, then we will report a 90% accuracy, but we would not have addressed the problem of identifying *paid late* invoices. So, we use metrics like precision and recall for the *paid late class*, and F1-score for the classifier.

We use an ensemble of classifiers and achieve a precision of 89.3% and recall of 82.7% on the *paid late* invoices.

The key contributions of our research are:

1. Using machine learning to predict the payment status of invoices to minimize the penalties incurred due to the invoices being delayed. Hence, our predictions can enable the process owners to pro-actively work on flagged invoices rather than rely on teams monitoring invoice processes or conduct time consuming analysis on each invoice.
2. Modeling categorical features in a domain which has historical (temporal) information as numerical features. This reduces the feature space considerably (from $\sim$1900 to 88) in a way that all the unique values in a category are replaced by few extra columns for each row. Otherwise, if one hot-encoding or indexing is done on categorical features, it would result in $\sim$1885 features.
3. We propose and evaluate an ensemble approach for invoice late payment prediction, encompassing supervised learning algorithms like Random forest and Boosted Trees which are better suited for categorical features and SVM and Logistic Classification better suited for numerical data.

We discuss prior art in Sect. 2, methodology in Sect. 3, empirical evaluation in Sect. 4 and a better way to represent the categorical type features instead of one-hot encoding for historical analysis of invoices in Sect. 5.

2 Related Work

In this section, we present some related work that has been done in the invoice prediction domain. Zeng et al. [16] tackle the problem of invoice outcome prediction in *Accounts Receivable* (AR) case. They formulate the prediction problem as supervised classification problem and apply the existing classifiers (C4.5, Naive Bayes etc.) to it. Our work differs from theirs in following ways: (1) we tackle the problem in the Account Payables (AP) case while they tackle it in the AR case. Secondly, they report *Accuracy* as a metric for class-imbalance classification problem which is not a suitable metric of choice in this setting. Instead, we report the metrics which are better suited for class-imbalance setting.

Smirnov et al. [12] models the invoice late payment time by survival analysis and an ensemble of random survival forest on real data and show that random survival forest performs better when combined with historical data of repeated debtors. Hu et al. [6] discuss prediction of invoice payment and improvement in the process of collection. They use various supervised algorithms such as decision tree, random forest, logistic regression, SVM, and cost-sensitive learning for prediction and conclude that random forest outperform the other methods. Additionally, their data set has paid late invoices as the majority class while in our dataset, paid late is the minority class. Along similar lines, Hu et al. [7] use supervised learning algorithms for invoice payment prediction.

An important point to note is that all the work discussed in this section solve the problem of invoice prediction in the AR case. From our literature survey, we find one work that solves the invoice prediction in the AP case. Younes et al. [15] attempt to address the problem of invoice processing time, understanding the delinquent invoices and the impact of delay in the invoice processing. They use integrated lean-manufacturing and discrete event simulation as the first approach and Markov chain modeling as the second approach for minimizing the overdue invoices in the AP case. Lean manufacturing borrows the idea from assembly line scheduling for managing the invoices and show encouraging results via simulation. The second approach that uses Markov process theory assumes each service station as node in the Markov graph and compute the transition probability from one node to another node as the service time of the invoice. From a monitoring business processes perspective, Meroni et al. [9] discuss an artifact based approach. Cabanillas et al. [3] also discuss predictive task monitoring to signal and control possible misbehaviors at runtime in business processes.

3 Methodology

In this section, we present our approach as illustrated in Fig. 1 to predict the invoice payment status. We predict the invoice status at 2 logical time steps in the complete process. One, when the invoice comes in the system, and there is only invoice specific data present and no process data on that invoice. From a business process perspective, this would help in flagging an invoice from the start if its likely to be delayed. This prediction would primarily be based on invoice specific features like amount, number of days allocated for invoice payment and vendor details. Another prediction is done three days (variable hyper-parameter) before the invoice is due. This would flag the invoice if its likely to be delayed giving enough time to the processing team to expedite the invoice processing.

3.1 Data Preprocessing

We obtained the invoice process data from one of our large clients, which process around 200 thousand invoices across multiple vendors per month. We had access to two sets of data - basic invoice details, when it was raised in the system for processing and the associated change log consisting of all the process actions taken on the invoice. In total, we had 523147 invoices and associated 3.5M log data. The invoice value ranged from less than a dollar to ∼236M$. We observed

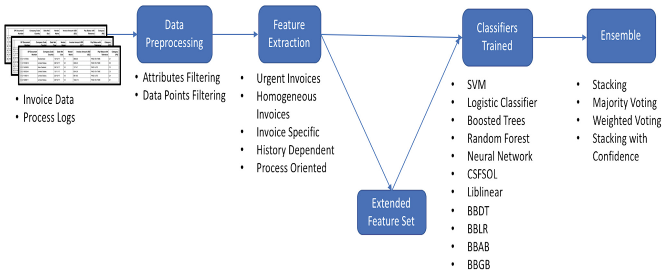


Fig. 1. Approach to predict late payment of invoices

around 7% of invoices as rated urgent, and around 9% of paid late invoices. These paid late invoices were occurring for less than 50% of the vendors. The summary stats of the data set is presented in Table 1, column 2. As illustrated in Fig. 2(a), there was always a steady flow of urgent invoices across the year, and the invoices that were paid late were also observed consistently across the time period (Fig. 2(b)). We now discuss the various data preprocessing techniques applied:

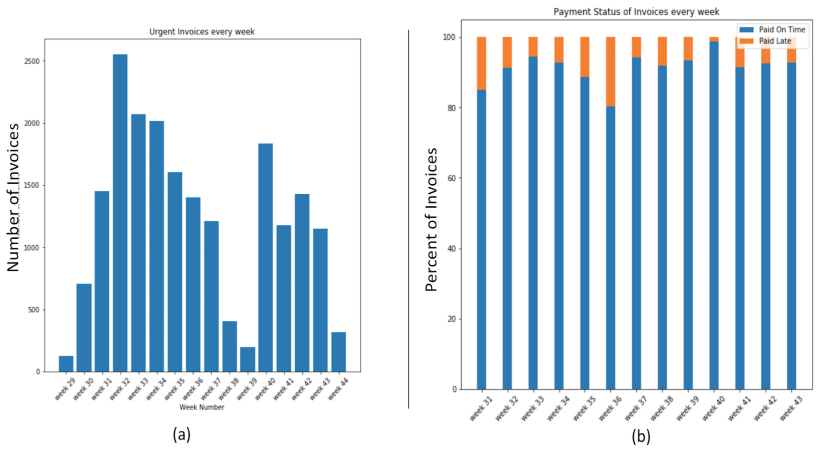


Fig. 2. Urgent invoices and invoice payment statistics

1. **Attributes Filtering:** There are 47 attributes describing each invoice with information regarding vendors, their demographics, raw materials, amount payable and the deadline for the payment. We removed attributes which uniquely identify an invoice such as “Document Id”. Also, attributes containing information populated after the invoice was created, which if considered, could bias the prediction of labels i.e., “Paid Late” or “Paid on Time”

Table 1. Summary statistics of our data set

Attribute	Before preprocessing	After preprocessing
Number of invoices	523147	282622
Number of attributes	47	16
Number of paid late invoices	49835 (9.52%)	29221 (10.33%)
Number of paid on time invoices	473312 (90.5%)	253401 (89.6%)
Number of urgent invoices	7.04%	7.67%
Number of vendors	11532	8527
Number of countries	47	47
Categories of raw materials	10	10
Maximum amount of invoice (in $)	236 million	236 million
Mean amount of invoices (in $)	4352	4180

were filtered. Finally, we removed attributes which did not show any variation across invoices. This retained 16 out of the initial 47 attributes.

2. **Data Points Filtering:** We removed data points which had incorrect, misleading or missing values for attributes like "vendor name", "invoice amount", "company code", "due date" and "posting date". Also, some invoices had amount less than 0 which meant that the invoice was a "credit" invoice i.e. the payment was done before the raw materials were procured. Some invoices had due date prior to the posting date of the invoice making such invoices delayed even before the processing had started. Few invoices did not have matching data in the process logs and hence the information related to the processing was missing. After removing all such erroneous data points, we had 282622 invoices. The summary stats of the processed data set is listed in Table 1, column 3.

3.2 Feature Extraction

In this section we describe the feature engineering approach and the resulting features extracted to train the classifiers.

As discussed in the introduction, there are several reasons for an invoice to be delayed. We concentrate on extracting features which tackle all these challenges. The processing time of an invoice depends on invoice specific attributes as well as on other invoices which are being processed concurrently, as they are competing for the same scarce resource (humans for manual tasks). The physical analogy can be that of vehicular traffic - more cars on the road will strongly correlate to most of the cars being delayed. Similarly, if there are high number of invoices in the system at a given time and each requiring multiple steps, this may create bottleneck at few places in the process leading to delays. On the other hand, lesser invoices may speed up the processing time. Inspired by Senderovich et al. [11], we categorize the features broadly based on inter-case and intra-case invoices:

Table 2. Features & their types used for predicting invoice status

Feature class	No.	Feature & type	Description
Urgent invoices	1	urgent_invoices_due (int)	Number of urgent invoices due on the day of prediction
	2	urgent_overall (int)	Number of urgent invoices which were due over the entire duration of an invoice from "posting date" to "due date"
Homogeneous invoices	3	load_invoices (float)	Number of invoices which were posted between the days of *date posting* and *due date*
	4	actions_load_invoices (float)	Number of actions (process steps) taken on all invoices between the days of *date posting* and *due date*
	5	num_invoice_due_previous_day (int)	Number of invoices that were due the previous day before an invoice was due
Invoice specific	6	number_of_days (int)	Number of days between *date posting* and *due date*
	7	number_of_working_days (float)	Number of weekdays between *date posting* and *due date*
	8	number_of_invoices_due (float)	Number of invoices which are due when an invoice is posted
	9	number_of_outstanding_invoices (float)	Number of invoices which are due and already delayed
	10	ratio_paid_late_outstanding (float)	Ratio of 8 and 9
	11	invoices_amount (in dollars) (float)	The amount due for this invoice
	12	vendor_name (category)	Vendor name who has submitted the invoices
	13	company_code (category)	Division of the multi-national company which requested the material
	14	company_code_country (category)	Country of the division of the multi-national company which requested the material
	15	document_type (category)	Type of submitted invoice
	16	raw_material (category)	Category type of the raw material
	17	commercial_area (category)	Commercial area of raw material
	18	payment_terms (category)	Terms dictating invoice payment
	19	invoice_amount_bucket (category)	Invoice amount bucketed in bins
History dependent	20	paid_late_invoices (float)	Number of invoices which were paid late before the *date posting* of an invoice
	21	paid_on_time_invoices (float)	Number of invoices which were paid on time before the *date posting* of an invoice
	22	ratio_paid_late_paid_on_time (float)	Ratio of 20 and 21
	23	percent_paid_late_vendor (float)	Percentage of invoices which were delayed to a vendor till the present date
Process oriented	24	action (category)	The action code taken on the invoice
	25	number_of_actions (int)	The total number of actions taken on the invoice
	26	number_of_days_before_first_action (int)	The number of days between the invoice is raised and first action is taken on the invoice

Inter-case Invoices

- **Case #1 Urgent Invoices:** Invoices which need to be paid in a day or two get preference over other invoices. If there are a bulk of such invoices coming constantly, it will adversely affect the other invoices which were to be processed and paid. So, we engineer the following two features:
  - Number of invoices which were due in 1–2 days(n), 3 days prior to particular invoice(p_i), since these n invoices may delay the processing for p_i invoice. (#urgent_invoices_due)
  - Number of such invoices which were supposed to be paid in 1-2 days(m) over the complete duration of a particular invoice. (#urgent_overall)

- **Case #2 Homogeneous Invoices:** For invoices other than urgent invoices, we consider that there is no priority between them. So, at a particular time, the processing time of an invoice would depend on the number of other invoices(#load_invoices) and the number of actions(#actions_load_invoices) being taken on them. It is a representative of the amount of load in the system when a particular invoice was being processed.

Intra-case Invoices

- **Case #3 Invoice Specific:** Some invoices may go through a longer process then others depending on multiple reasons such as the demographics, amount(#invoices_amount(in $)), information provided or missing, etc. The processing speed of the invoice will also depend on the number of days(#no_of_working_days) between the invoice posting date and the due date. This would mean that different invoices may be treated differently depending on these criteria. For example: between an invoice which needs to go through 5 stages and is due in 10 days and another invoice which is due only after 45 days, the earlier invoice would take precedence to ensure both the invoices are being paid on time.
- **Case #4 History Dependent:** We only consider invoices from vendors who have at-least 10 invoice payment transaction. We had this threshold since, it represents the importance of the relationship with the vendor based on the transaction history, and to have enough data points to consider history dependent features like #percent_paid_late_vendor. We also consider features which signify historical payment status of all invoices(#paid_late_invoices and #paid_on_time_invoices).
- **Case #5 Process Oriented:** Once the invoice is posted, it goes through multiple checks and steps before the invoice is paid. As discussed earlier, we predict the invoice payment status 3 days before it is due. So, we take into account the type of action(#action) and the total number of actions (#number_of_actions) performed on the invoice at the time of prediction.

To summarize, we have 26 features with 17 numerical and 9 categorical types across the five categories as listed in Table 2.

Table 3. Comprehensive list of various classifiers evaluated in our approach

Classifier	Description
SVM	A non-probabilistic binary classifier
Logistic classifier	Models the probability of the classes using the logit function
Boosted Tree	An ensemble of decision trees using gradient boosting
Random forest	Ensemble of decision trees combining ideas of random selection of features and bagging
Neural network	Trained a neural network with 3–8 layers with binary cross-entropy loss
Liblinear [4]	A classifier for solving large-scale regularized linear classification problems
Cost-sensitive first-order sparse online learning (CSFSOL) [13]	Online learning based algorithm for cost-sensitive learning on large-scale sparse data
BBDT	Balanced Bagging Decision Trees
BBLR	Balanced Bagging Logistic Regression
BBAB	Balanced Bagging Adaboost
BBGB	Balanced Bagging Gradient Boosting

3.3 Classifiers Used

We used a supervised learning approach to train the classifiers listed in Table 3. We evaluated each of these classifiers and used an ensemble on these classifiers to improve our results as different models will be better suited for different subsets of data [10]. We discuss the different approaches we evaluated for the ensemble:

1. Stacking [14]: Different classifiers such as Boosted Trees, Logistic classification, SVM were trained over the predictions of different classifiers giving each classifier an equal weight.
2. Plurality Voting (Most voted): The final prediction is the most predicted value amongst all the classifiers.
3. Weighted Voting: The predictions of each model are weighted according to the number of correct predictions made by them. So, the weight of each model is the accuracy the individual model has. This was tried both for overall accuracy as well as *paid late* accuracy.
4. Stacking with Confidence: Ensemble was trained on predictions from different models. Along with the predictions, the confidence scores from each classifier (wherever possible) are also considered as features.

4 Empirical Evaluation

In this section, we define the metrics and demonstrate the empirical evaluation of different machine learning models on invoice late payment prediction.

4.1 Metrics

Owing to the data imbalance in our case and contrary to the evaluation metrics used in some of the literature for invoice late payment prediction [16] (mostly *accuracy*), we aim to achieve high precision and reasonably high recall on paid late invoices (minority class) because no action is needed for paid on time invoices. High precision would mean that most of the invoices our approach labels as "paid late" are indeed "paid late". High recall here implies that our approach is able to detect majority of the invoices which are going to be "paid late". We report precision-recall (PR) curve rather than Receiver operating characteristic (ROC) curve because PR curve does not account for true negatives (TN) (as TN is not a component of both precision and recall) and would not be affected by the relative imbalance. The metric used for our evaluation are Precision, Recall, F1-score, Average precision (AP) score and Area under PR-curve (AUPRC).

4.2 Training

For evaluating our approach, we consider only those invoices which have a minimum of 10 days for payment as majority of the invoices (93%) are due only after 15 or more days from the date of posting. Also, for evaluating, we consider invoices only from vendors which have more than 10 transactions. This serves couple of purposes. First, this helps in concentrating on only vendors which are dependable which in turn implies the importance of the relationship with that vendor which may be because of the raw materials, cost or other demographics. Therefore, invoices from such vendors should be given attention to maintain this relationship. Secondly, it helps us identify additional features that capture the vendor behavior, e.g. the number of times payment is delayed to a vendor. We had a approximately 60:20:20 data split across train, test and validation. Since, our task is time dependent, we don't split according to approaches such as cross-validation. We split the data based on time. Invoices which are cleared before a date are considered for training and after that date in test data. The data split is shown in Table 4.

Table 4. Data split for evaluating our classifiers

	Training	Validation	Test
Paid on time	165721	41589	46091
Paid late	21112	5109	3200

4.3 Results

As discussed, we make predictions at 2 time steps, once when the invoice is raised and once 3 days before the due date of the invoice. The precision (P), recall

(R) and F1-score of the classifiers evaluated are listed in Table 5. In summary, "Boosted Trees" and "Random Forest" performed the best. Although, we did an extensive parameter search for these models, the recall of "paid late" class was poor across classifiers. We had 26 derived features out of which 9 were categorical type. Since, we had ∼1885 categorical values for these 9 categorical features, the best results were observed for decision tree based models namely random forest and boosted trees as the decisions at each node are based on values of the features. Inspired by Avati et al. [2], we tried to address this problem through Deep Learning. But the results were not satisfactory. Upon further analysis of results, we figured one of the major reasons could be the explosion of features while converting categorical features to numerical features. To avoid having an implicit ordering between the categorical features when converted to numerical features, the conversion was done using one-hot encoding instead of indexing the values. This meant that there were ∼1885 binary features out of ∼1900 features.

Table 5. Precision, Recall and F1-score of classifiers evaluated.

Prediction time	Method	F1-score (%)	Paid late (R) (%)	Paid late (P) (%)
Prediction when invoice raised	Boosted Trees	**95.84**	**39.93**	**91.10**
	Random forest	95.21	30.25	88.24
	Logistic classification	94.69	35.4	67.32
	SVM	95.48	36.18	86.41
	Neural network	94.38	35.34	61.80
Prediction 3 days before invoice due	Boosted Trees	**96.24**	**43.87**	**96.36**
	Random forest	95.63	37.25	89.08
	SVM	95.74	41.71	85.19
	Logistic classification	95.16	33.18	72.58
	Neural network	95.61	39.78	84.41

5 Extended Feature Set

To tackle the above mentioned feature explosion problem, we devised a better representation of these categorical features into numerical features based on the historical information present without resulting in the explosion in feature space. The representation is based on the intuition about, how these features would affect or factor into an individual's analysis while processing these invoices. And also how the historical data flow about each categorical type feature serves a meaningful purpose for analysis. Our data comprised of categorical features like vendor details, country, and other such demographics. So, for each of the 9 categorical type features and their values, we derived following features which is a representation of their influence on payment status. For each invoice, until the date (d_1) of prediction (i.e. 3 days before the due date):

1. n_1 - Number of total invoices for particular value of that categorical feature prior to d_1
2. Percentage and number of invoices paid late out of n_1
3. Percentage and number of invoices paid on time out of n_1.

Further, for vendors, which had the most number of unique values (1459) among other categorical features, we considered a moving window to accommodate *seasonality* and change in the vendor's recent history of payment status. For example: if a vendor has 25 invoices, out of which 20 are paid late and 5 are paid on time, it might be that the last 5 invoices were paid on time. So, this seasonality was taken into account. We derived more vendor specific features which accounted for the payment status of vendor's previous 3 and previous 5 invoices. So, vendor names(#vendor_name) were represented as following features:

– Total number of invoices processed for the vendor prior to a date.
– Percentage and Number of times invoices were paid late and paid on time for the particular vendor prior to a date. This is to understand the previous record with a particular vendor.
– Percentage and Number of times invoices paid late and paid on time for the particular vendor during the last "n" $(3, 5)$ times. (seasonality)

Table 6. Precision (P), Average Precision (AP), Recall (R), and F1-score of classifiers evaluated with extended feature set.

Method	F1-score (%)	AP	Paid late (R) (%)	Paid late (P) (%)
Liblinear	97.58	0.65	64.41	97.59
CSFSOL	**98.40**	**0.77**	**78.24**	**96.57**
Boosted trees	96.74	0.54	51.96	96.18
Random forest	96.73	0.52	53.46	93.34
Logistic classification	97.56	0.65	68.80	91.50
SVM	97.95	0.70	77.8	89.31
BBDT	96.24	0.49	59.19	77.55
BBLR	**97.09**	0.64	**84.83**	74.15
BBAB	96.40	0.54	70.95	72.93
BBGB	98.01	0.72	81.86	86.77
Neural network	96.15	0.50	60.23	78.20

5.1 Experimental Testbed and Settings

In this section, we show the parameters and setting used in all the experiments. For Liblinear, command line options were (-s 5 -B 1 -e 0.001 -c .1 -w-1 4 -w1 1). That means, we train L_1-regularized l_2-loss SVM with a bias of 1 added to the training examples, parameter C is set to 0.1, weights given to negative and positive examples are 4 and 1 respectively. We train the SVM until error tolerance falls below 0.001. For training CSFSOL, we vary the parameter λ and η in the range $\{0.003, 0.09, 0.3, 1, 2, 4, 8, 16, 32, 64\}$ and $\{0.0312, 0.0625, 0.125, 0.25, .5, 1, 2, 4, 8, 16, 32\}$ respectively while weights are set to $(0.1, 0.9)$ for the negative and positive examples respectively. For training ensemble of classifiers based on Balanced Bagging approach default settings of Imblearn python package [8] is used.

For the Ensemble with confidence (SVM) used on all different models, we performed a grid search on validation dataset, the parameters which worked best were *penalty* (SVM) of 0.001 (penalty term on the misclassification loss of the model), *max iterations* as 300 and *class weights* were inversely proportional to the number of examples in the training data for each class. One of the intuitive reason behind using the low penalty value was that: for a model, higher the penalty, the model tries to maximize the margin for correctly classified examples. But, the aim was to correctly classify as many invoices as possible (Fig. 3).

5.2 Results

In this section, we discuss the result of our empirical evaluation of the methods used to predict paid late invoices with extended feature set (Table 6). *CSFSOL* gave the best result when an individual classifier is considered. The Ensemble classifier which performed the best was a SVM trained on predictions of all classifiers along with the confidence score wherever available (Table 7). We would like to emphasize the fact that F1-score is an overall performance score on both the classes.

Table 7. Results obtained on different metrics using the various ensemble models. AP = Average Precision P = Precision, and R = Recall

Method	F1-score (%)	AP	Paid late R (%)	Paid late P (%)
Majority voting	**96.30**	0.46	43.96	**98.04**
Weighted voting	98.04	0.72	79.28	89.45
Stacking	98.07	0.73	83.46	86.38
Stacking with confidence	**98.23**	**0.75**	**82.75**	**89.33**

The precision-recall curve shows the relationship between precision and recall for different thresholds. Our PR curve demonstrates that the Ensemble classifier with confidence (0.75) and CSFSOL (0.77) are better suited for our task as they

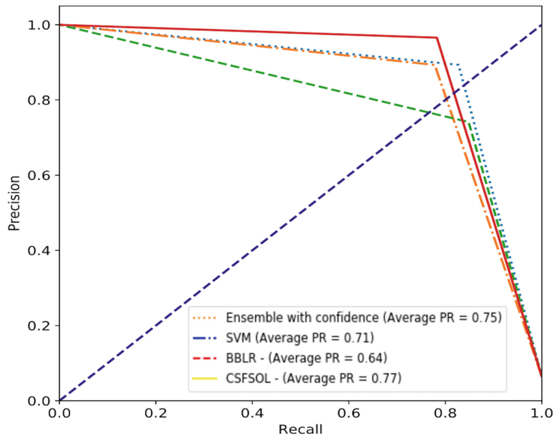


Fig. 3. PR curves for best results

have a greater area under the curve. And with 0.77 and 0.75 AP score, both precision and recall are reasonably good without affecting the other.

6 Discussion and Conclusion

From the results, we can observe that Ensemble with confidence and CSFSOL outperform other methods in terms of the metrics evaluated.

Table 8. Top 5 Influential Features across classifiers

Feature category	No.	Feature & type	Description
History dependent	1	paidLate_vendor_3 (int)	Number of times the payment was made late to a particular vendor out of the previous 3 times
History dependent	2	paid_on_time_vendor_5 (int)	Number of times the payment was made on time to a particular vendor out of the previous 5 times
Inter-case invoices	3	num_invoice_due_previous_day (int)	Number of invoices that were due the previous day before an invoice was due
History dependent	4	category_paid_late_3Days (int)	Number of times the payment was made late in a particular category out of the previous 3 times
Process oriented	5	process_id_4 (int)	Number of times the step 4 (step id - one of the steps invoices may go through) was done on the invoice

Influential Features: The top 5 influential features as shown in Table 8 prove that historical dependence and seasonality play a major role in deciding whether an invoice will be "paid on time" or "delayed". Further, how far the invoice has been in the process, along with number of concurrent invoices being processed, also affects the invoice payment status.

Generalizability: Based on our experience, most of the inter-case and intra-case features used for prediction are expected to be available for any account payables process, and collecting this data is very much feasible. The analysis, preprocessing, features or models have nothing specific to the dataset we have evaluated on. Hence, we can safely argue that most of our features and models are generalizable for other data sets obtained from client accounts. We are in the process of evaluating the same with few more client accounts data.

Future Work: Possible future work in this research includes, predicting the number of days by which an invoice would be delayed, suggesting advancing of processing of invoices in favor of other invoices such that the penalty (if any) on late payments is minimized. Also, if the amount of invoice is high, they should be treated separately since the penalty incurred on the delay would be higher. Finally, these is scope for identifying which process steps are most time consuming and provide suggestions on human resource management.

Implementation: We implemented the preprocessing of data using python2.7 and pandas library. Different libraries were used for different classifiers namely PyTorch for implementing neural networks, liblinear [4] for liblinear, imbalance-learn [8] for BBDT, BBLR, BBAB & BBGB. We implemented the CSFSOL algorithm in C++. We used scikit-learn libraries for SVM, logistic classification and boosted trees. The service to predict the payment status for a new invoice was hosted on a server using flask which is a python based framework.

References

1. http://www.cimaglobal.com/Documents/Thought_leadership_docs/white-paper-hub/2016-04-12-Invoice-Benchmark-Report.pdf. Concur (2016)
2. Avati, A., Jung, K., Harman, S., Downing, L., Ng, A., Shah, N.H.: Improving palliative care with deep learning. arXiv preprint arXiv:1711.06402 (2017)
3. Cabanillas, C., Di Ciccio, C., Mendling, J., Baumgrass, A.: Predictive task monitoring for business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 424–432. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_31
4. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: a library for large linear classification. J. Mach. Learn. Res. **9**(Aug), 1871–1874 (2008)
5. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
6. Hu, P.: Predicting and improving invoice-to-cash collection through machine learning. Ph.D. thesis, Massachusetts Institute of Technology (2015)
7. Hu, W.: Overdue invoice forecasting and data mining. Ph.D. thesis, Massachusetts Institute of Technology (2016)
8. Lemaître, G., Nogueira, F., Aridas, C.K.: Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning. J. Mach. Learn. Res. **18**(17), 1–5 (2017). http://jmlr.org/papers/v18/16-365
9. Meroni, G., Di Ciccio, C., Mendling, J.: An artifact-driven approach to monitor business processes through real-world objects. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 297–313. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_21

10. Rokach, L.: Ensemble-based classifiers. Artif. Intell. Rev. **33**(1–2), 1–39 (2010)
11. Senderovich, A., Di Francescomarino, C., Ghidini, C., Jorbina, K., Maggi, F.M.: Intra and inter-case features in predictive process monitoring: a tale of two dimensions. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 306–323. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_18
12. Smirnov, J., et al.: Modelling late invoice payment times using survival analysis and random forests techniques. Ph.D. thesis (2016)
13. Wang, D., Wu, P., Zhao, P., Hoi, S.C.: A framework of sparse online learning and its applications. arXiv preprint arXiv:1507.07146 (2015)
14. Wolpert, D.H.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992)
15. Younes, B.: A framework for invoice management in construction. University of Alberta, Canada (2013)
16. Zeng, S., Melville, P., Lang, C.A., Boier-Martin, I., Murphy, C.: Using predictive analysis to improve invoice-to-cash collection. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1043–1050. ACM (2008)

Explaining Non-compliance of Business Process Models Through Automated Planning

Fabrizio Maria Maggi[1], Andrea Marrella[2](✉), Giuseppe Capezzuto[2], and Abel Armas Cervantes[3]

[1] University of Tartu, Tartu, Estonia
f.m.maggi@ut.ee
[2] Sapienza Università di Roma, Rome, Italy
{marrella,capezzuto}@diag.uniroma1.it
[3] The University of Melbourne, Melbourne, Australia
abel.armascervantes@unimelb.edu.au

Abstract. Modern companies execute business processes to deliver products and services, whose enactment requires to adhere to laws and regulations. Compliance checking is the task of identifying potential violations of such requirements prior to process execution. Traditional approaches to compliance checking employ formal verification techniques (e.g., model checking) to identify which process paths in a process model may lead to violations. However, this diagnostics is, in most of the cases, not rich enough for the user to understand how the process model should be changed to solve the violations. In this paper, we present an approach based on finite-state automata manipulation to identify the specific process activities that are responsible to cause violations and, in some cases, suggest reparative actions to be applied to the process model to solve the violations. We show that our approach can be expressed as a planning problem in Artificial Intelligence, which can be efficiently solved by state-of-the-art planners. We report experimental results using synthetic case studies of increasing complexity to show the scalability of our approach.

1 Introduction

This paper falls within the scope of Business Process Management (BPM), an active area of research that is based on the observation that each product and/or service that a company provides to the market is the outcome of a number of activities [13]. Business processes (BPs) are the key instruments for organizing such activities and understanding their interrelationships.

BPs are described using *process models*, which capture the ways in which BP activities are carried out to accomplish a business objective, often with the help of an explicit control flow expressed through a suitable graphical notation, such as the ISO/IEC 19510:2013 standard BPMN. The BPM philosophy is built on the idea that there always exists a BP model that can be used to automate the

BP execution. For this reason, process modeling is recognized as one of the most important steps in the BPM lifecycle [13], and the modeling task is nowadays supported by advanced techniques and tools that assist (human) process designers in the definition of the BP model. Nonetheless, despite all efforts, design flaws in BP models may still occur, and their impact may range from syntactically incorrect models that cannot be properly executed to catastrophic faults that yield legal aftermaths [20]. Consequently, a large branch of research in BPM has focused on devising techniques for BP *verification*, with the aim of identifying and fixing errors prior to BP execution.

The bulk of the research on BP verification has been devoted to check domain-independent correctness criteria that depend exclusively on the structure of BP models, such as proper termination, absence of deadlocks, etc. (e.g., see [1,7,17]). However, the design of a BP requires also that its model adheres to several existing domain-dependent *compliance requirements* set by managers, laws, national and international regulations and standards. For instance, the Italian Ministry of Economy and Finance enforces rules to control financial processes in the public sector. The task of identifying potential violations of compliance requirements in a BP model is known as *compliance checking* [28].

Existing approaches to compliance checking employ formal verification techniques (e.g., model checking) to explain violations through a path in the BP from the initial to the error state [6] or by detecting which specific activity [2] or decision [20] has triggered a violation. However, such techniques do not identify *all the activities* of a BP path that can cause violations of compliance requirements. On the other hand, performing this task in a manual way can be time-consuming and error-prone since the amount of compliance requirements to be checked, as well as the number of activities that violate them, can be large.

In this paper, we tackle the above issues by presenting a compliance checking approach to *identify* all those activities of a BP model violating compliance requirements and *explaining* the causes that lead to the violations. The starting points of our approach are a BP model represented in BPMN and a list of compliance requirements expressed as temporal declarative rules with the well-established DECLARE [25] language, which enjoys formal semantics grounded in Linear Temporal Logic with finite execution semantics (LTL$_f$) [26]. Then, our approach leverages the fact that: *(i)* BPMN models can be converted into Petri nets [12], which can be automatically unfolded to derive execution paths of the BP model; and *(ii)* any LTL$_f$ formula ϕ can be translated into a non-deterministic finite-state automaton (NFA) that accepts all paths satisfied by ϕ [10]. Based on this, we provide a technique, based on NFA manipulations, to detect all activities in any path of the BP model that violate a formula. We also show that such a technique can be expressed as a planning problem in Artificial Intelligence (AI), which can be efficiently solved by state-of-the-art planners.

We have implemented a plug-in of the Apromore BP analytics platform (cf. http://apromore.org/) that realizes our approach by employing the FAST-DOWNWARD planner (cf. http://www.fast-downward.org/) to solve the compliance checking problem. To motivate the employment of planning techniques, we

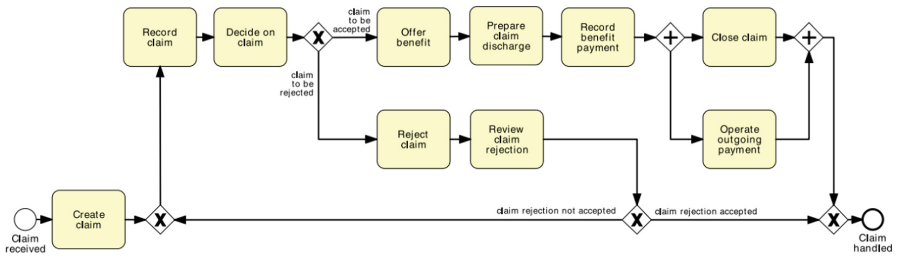


Fig. 1. BPMN model of the running example (original figure in [13])

performed several experiments that show how the complexity of the problem is large and that a technique is necessary that can scale up adequately.

The rest of the paper is organized as follows. Section 2 introduces a running example that will be used to explain our approach. In Sect. 3, we provide the relevant background necessary to understand the paper. Section 4 presents an overview of our compliance checking approach, while Sect. 5 discusses how to reduce the compliance checking problem to a planning problem in AI. Then, in Sect. 6, we report on experiment results performed on synthetic case studies of growing complexity that show the *scalability* of our approach, while, in Sect. 7, we discuss related work. Finally, in Sect. 8, we conclude the paper.

2 Running Example

Figure 1 shows the BPMN model of a running example taken from [13], which will be used in the rest of the paper. It describes a BP for claim handling. A claim is initially created and recorded. Then, a decision is taken on the claim. In case of acceptance, an offer of benefit is made, the claim discharge is prepared, and the benefit payment is recorded. Finally, in parallel, the claim is closed and the outgoing payment is made. In case of rejection, the claim is marked as "rejected". After this, it is possible to review the claim rejection, and in case the claim rejection is not accepted, the claim can be recorded again.

Supposing that this BP is executed in an insurance agency, we can realistically assume that, due to internal regulations or new governmental rules: *(i)* the review of the claim rejection is not allowed anymore; *(ii)* due to the previous rule, the rejection of the claim cannot be followed by a new recording; *(iii)* if the outgoing payment has not been executed, then the claim cannot be closed. Given these compliance requirements, we want to identify the parts of the BPMN model that are not compliant, explain the violations and, in some cases, suggest reparative actions to be applied to the BPMN model to make it compliant with the requirements. Note that techniques based on model checking produce a diagnostics based on counterexamples specifying, for example, that the path ⟨Create claim, Record claim, Decide on claim, Reject claim, Review claim rejection, Record claim, Decide on claim, Offer benefit, Prepare claim discharge, Record benefit payment, Close claim, Operate ongoing payment⟩ is not compliant with the requirements. However, starting from this information, it is hard to pinpoint what is the reason why each requirement is violated and what to do to solve such violations.

3 Background

3.1 BPMN and Petri Nets

Many notations have been introduced to represent BPs, such as BPMN, EPC, YAWL or UML Activity Diagrams [13]. These languages allow process designers to specify aspects linked to different perspectives, ranging from expressing the ordering with which activities need to be executed and the mutual exclusions among activities (control-flow perspective) to modeling the objects manipulated by activities and the resources allowed to execute them. In this paper, we focus on the *core sub-set* of BPMN,[1] which is considered the de-facto standard for modeling BPs. Specifically, to make our approach work, we impose the following syntactic restrictions: *(i)* the admissible flow objects for a BPMN model are activities, start and end events, intermediate events (if they can be translated into activities), exclusive and parallel gateways, and a flow relation is used to connect them; *(ii)* the model provides a finite number of start events and end events; *(iii)* any admissible flow object is on a path from a start to an end event. The use of core BPMN is not a significant limitation, since it realistically allows us to cover the majority of modeling needs [13].

If, on the one hand, BPMN provides an intuitive way for BPM users to model BPs, on the other hand, it is characterized by an ambiguous semantics. Therefore, in order to explain the technical aspects of our approach, we needed a simple language with clear semantics. For this reason, we opted for Petri nets (PNs) [24], which provide the formal foundations of the core sub-set of BPMN [12] and have proven to be adequate for modeling BPs [1]. This is especially true when the focus is only on the control-flow perspective, which is the case in this paper. A PN is a directed bipartite graph with two node types: *places* (graphically represented by circles) and *transitions* (graphically represented by squares) connected via directed arcs. Technically, a PN is a triple (P, T, F) where P and T are the set of *places* and *transitions*, respectively, such that $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.

At any time, places in a PN may contain a discrete number of marks called *tokens*, drawn as black dots. Any distribution of tokens over the places, formally represented by a total mapping $M : P \mapsto \mathbb{N}$, represents a configuration of the net called a *marking*. When PNs are used to represent BPs, transitions are associated with BP activities, and more specifically to activity labels, and markings indicate the BP state [1]. Since concrete executions of BPs have a start and an end, PNs need to be associated with an initial (respectively final) marking, characterized by the presence of one token in at least one of the starting (respectively ending) places of the PN and no tokens in any other place. The semantics of a PN defines

[1] Notice that our approach can easily be transferred to other BP modeling languages.

how transitions route tokens through the net so that they correspond to a BP execution. Due to page limit, we refer to [1,24] for the semantics of PNs. In this paper, we focus on 1-bounded PNs (a.k.a. *safe* PNs), which impose that the number of tokens in all places is at most 1 in all reachable markings, including the initial one. Safe PNs are the basis for best practices in BP modeling, and the behavior allowed by most of real-world BPs can be represented as safe PNs [18].

3.2 Declarative Temporal Rules and Finite State Automata

In order to provide automated support for compliance checking, compliance requirements need to be expressed in a formal language. This, in turn, allows for leveraging mature AI techniques. In this work, we focus on rules that can be expressed using Linear Temporal Logic with finite execution semantics (LTL$_f$). We refer to [10] for the complete syntax and semantics of LTL$_f$.

Since the direct use of temporal logics is inappropriate for most process analysts that conduct BP modeling, we decided to use DECLARE [25] for the specification of compliance requirements. DECLARE is a declarative modeling language that allows us to describe a set of (temporally extended) rules that must be satisfied throughout the BP execution. Unlike procedural models, where all allowed executions must be explicitly represented, in DECLARE, the orderings of activities are implicitly specified by rules and anything that does not violate them is possible during execution. The semantics of DECLARE is grounded on LTL$_f$. A DECLARE model $\mathcal{D} = (\mathcal{Z}, \pi_{\mathcal{D}})$ consists of a set of activities $\mathcal{Z}$ involved in a BP and a collection of temporal rules $\pi_{\mathcal{D}}$ defined over such activities.

Among all possible LTL$_f$ rules, some specific *patterns* have been singled out as particularly meaningful for expressing DECLARE models. For instance, if we indicate as Rev activity Review claim rejection, *absence*(Rev) means that activity Rev cannot ever be performed; if we indicate as Rej activity Reject claim and as Rec activity Record claim, *not succession*(Rej, Rec) means that if Rej is performed, Rec cannot eventually be performed; finally, if we indicate as Pay activity Operate outgoing payment and as Close activity Close claim, *precedence*(Pay, Close) imposes that activity Pay must precede Close.

Given a BPMN model, the problem we want to address in this paper is to identify and explain *all potential violations* of DECLARE rules in the model. To this end, we exploit the well-known equivalence between (regular) languages and automata: any LTL$_f$ formula ϕ can be associated with a non-deterministic finite-state automaton (NFA) $\mathcal{A}$ that accepts exactly all paths satisfying ϕ [10]. Formally, such NFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where: *(i)* Σ is the *input alphabet*; *(ii)* Q is the finite set of *automaton states*; *(iii)* $q_0 \in Q$ is the *initial state*; *(iv)* $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*; and *(v)* $F \subseteq Q$ is the set of *final states*. Let $t = e_1 \cdots e_n$ be a path such that $e_i \in \Sigma$ (with $1 \leq i \leq n$) and $\mathcal{A}$ the NFA associated with an LTL$_f$ formula ϕ. A *computation* of $\mathcal{A}$ on t is a sequence $\sigma = q_0 \xrightarrow{e_1} q_1 \cdots q_{n-1} \xrightarrow{e_n} q_n$ such that, for $i = 0, \ldots, n-1$, there exists a transition $q_i \xrightarrow{e_i} q_{i+1} \in \delta$. Since $\mathcal{A}$ is non-deterministic, there exist, in general, many computations of $\mathcal{A}$ on the path t. We say that $\mathcal{A}$ *accepts* t if there exists a computation σ on t such that the last state is final, i.e., belongs to F.

3.3 Automated Planning

Planning systems are problem-solving algorithms that operate on explicit representations of states and actions [16]. PDDL [15] is the standard Planning Domain Definition Language; it allows us to formulate a *planning problem* $\mathcal{P} = \langle I, G, \mathcal{P}_{\mathcal{D}} \rangle$, where I is the description of the initial state of the world, G is the desired goal state, and $\mathcal{P}_{\mathcal{D}}$ is the planning domain. A planning domain $\mathcal{P}_{\mathcal{D}}$ is built from a set of *propositions* describing the state of the world (i.e., the set of propositions that are true) and a set of *operators* Ω (i.e., *actions*) that can be executed. An *action schema* $a \in \Omega$ is of the form $a = \langle Par_a, Pre_a, E\!f\!f_a \rangle$, where Par_a is the list of *input parameters* for a, Pre_a defines the *preconditions* under which a can be executed, and $E\!f\!f_a$ specifies the *effects* of a on the state of the world. Both Pre_a and $E\!f\!f_a$ are stated in terms of *propositions* in $\mathcal{P}_{\mathcal{D}}$, represented as boolean predicates and numeric fluents.

In recent years, the planning community has developed a plethora of planners that embed very effective (i.e., scaling up to large problems) search heuristics, which have been employed to solve collections of challenging problems from several Computer Science domains [22]. There exist several forms of planning in the AI literature. In this paper, we focus on planning techniques characterized by *fully observable* and *static* domains, i.e., we rely on the *classical* planning assumption of a "perfect world description" [16]. A solution for a planning problem is a sequence of operators—a *plan*—whose execution transforms the initial state I into a state satisfying the goal G. To find a plan, we represent planning domains and problems making use of the STRIPS fragment of PDDL 2.1 [15] enhanced with the numeric features provided by the same language for keeping track of the costs of planning actions and synthesize plans satisfying pre-specified metrics.

4 The Approach

Our approach to compliance checking relies on 4 main steps to be performed in sequence. First of all, in order to be properly enacted, the approach requires that a process analyst provides as inputs: *(i)* a BPMN model to be checked for compliance; *(ii)* a list of compliance requirements expressed as temporal rules in DECLARE [25]; *(iii)* a severity function that assigns non-negative costs to the detected violations (see also Sect. 5).

With a BPMN model and a set of DECLARE rules as inputs, we rely on well-established transformation algorithms to convert these representations into their corresponding formal counterparts. In particular, we first translate the BPMN model into a PN by leveraging the technique described in [12]. In Fig. 2, we show the PN derived from the BPMN model of Fig. 1. The black-colored transitions are *invisible* transitions, i.e., they do not represent actual pieces of work, but their introduction is sometimes necessary to properly represent the process behavior. Then, we generate all the paths allowed by the PN, i.e., all the complete executions of the PN from the initial to the final marking(s). To this aim, we compute the *complete prefix unfolding* [23] of the PN, which provides a

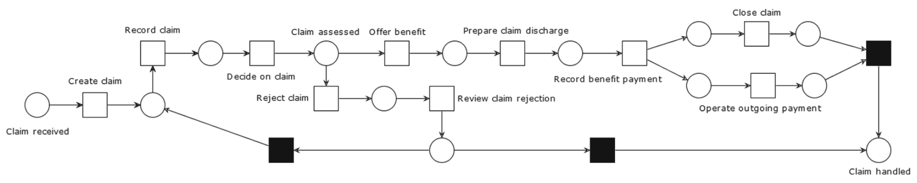


Fig. 2. Petri net derived from the BPMN model in Fig. 1

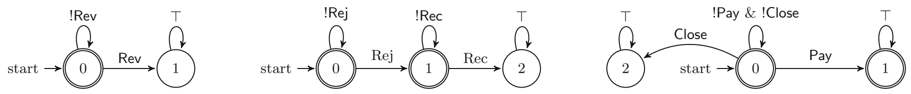


Fig. 3. Automata derived from the DECLARE rules introduced in Sect. 2: (a) *absence*(Rev), (b) *not succession*(Rej,Rec), (c) *precedence*(Pay, Close)

finite behavioral representation of the model. Since PNs can contain cycles, i.e., infinite paths (possibly of infinite length), we adopt the technique defined in [3] for computing the unfolding of the PN. This technique truncates the unfolding once all possible markings of the PN have been observed so that only a finite number of paths of finite length is generated.

We also translate the LTL$_f$ formula ϕ, associated to each input DECLARE rule, into a NFA that accepts exactly all paths satisfied by ϕ, using the technique developed in [10]. For example, the DECLARE rules defined in Sect. 2 can be represented with the NFAs shown in Fig. 3.

At this point, we invoke an external planner that is in charge of identifying the violations of the compliance requirements in any path extracted from the PN. In particular, the violations are identified both wrt. each *individual requirement* (by using the *local* automata represented in Fig. 3) and wrt. their *conjunction* (by using a *global* automaton given by the product of all the local automata).

We address this problem by resorting to *cost-optimal planning*, a form of classical planning where actions have costs, and where a successful plan of minimal cost (defined as the sum of the costs of the component actions) has to be found. The intuition behind our solution is that actions capture wrong/missing activities (having non-zero costs defined by the severity function) in the execution path under observation, and the goal is to make the path compliant with the behavior expressed in the automata at a minimal cost. To this aim, the planner implements a technique based on NFA manipulations, which is extensively presented in Sect. 5 together with its encoding in PDDL.

Finally, we visually present to the process analyst the violations of the compliance requirements on the input BPMN model. Figure 4 shows the resulting visualization of the running example. The figure is a screenshot of a plug-in implementing the proposed technique available in the Apromore advanced BP

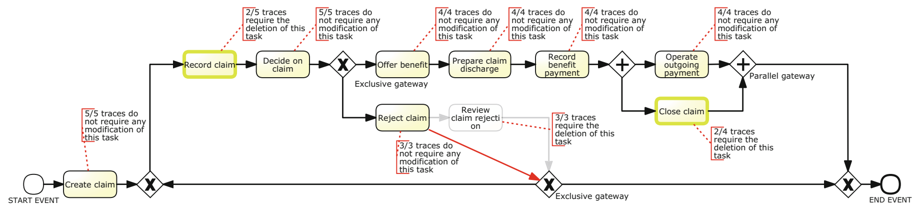

2/5 traces require the deletion of this task
5/5 traces do not require any modification of this task
4/4 traces do not require any modification of this task
4/4 traces do not require any modification of this task
4/4 traces do not require any modification of this task
4/4 traces do not require any modification of this task
Record claim
Decide on claim
Offer benefit
Prepare claim discharge
Record benefit payment
Operate outgoing payment
Parallel gateway
Exclusive gateway
5/5 traces do not require any modification of this task
Reject claim
Review claim rejection
3/3 traces require the deletion of this task
Close claim
2/4 traces require the deletion of this task
3/3 traces do not require any modification of this task
START EVENT
Create claim
Exclusive gateway
END EVENT

Fig. 4. Visualization in Apromore

analytics platform.[2] The visualization uses a color coding to suggest the amount of paths that support a given change. If 100% of paths support the removal of an activity, then the activity is greyed out; if from 66% to 99% of paths support this removal, then the activity is highlighted in orange; if from 33% to 66% of paths support the removal, then the activity is highlighted in yellow. Similarly, the plug-in shows the percentage of paths supporting the addition of an activity and, in case all paths agree with the addition, the plug-in suggests to add the activity to the model. Note that the plug-in provides a way to automatically repair the model. In particular, the grayed out elements can be removed and the suggested additions can be confirmed. In the running example, the tool suggests to remove Review claim rejection, since all paths support this operation, whereas this is not the case for the violations related to Close claim and Record claim, since these operations are not supported by all paths.

A screencast showing how the plug-in developed in Apromore works is publicly available at: https://youtu.be/pVcv5DSSt5A.

5 Compliance Checking as Planning

In this section, we first demonstrate that the problem of identifying violations of compliance requirements in a BP path can be solved with a technique based on NFA manipulations (Sect. 5.1), and then we show how this technique can be encoded as a planning problem in PDDL (Sect. 5.2).

5.1 A NFA Manipulations Technique for Compliance Checking

Let us consider a BP path $t = \langle e_1, ..., e_{k-1}, e_k, e_{k+1}, ..., e_n \rangle$ and an LTL_f formula ϕ. We are interested in "transforming" t into a new path $\hat{t}$ that is compliant with ϕ. To realize this transformation, we consider two kinds of violations, which can be caused by *wrong* or *missing* activities, respectively. For example, suppose that $e_k \in t$ violates ϕ: e_k is said to be *wrong* wrt. ϕ, and its deletion from t results in a new path $\hat{t} = \langle e_1, ..., e_{k-1}, e_{k+1}, ..., e_n \rangle$ that is compliant with ϕ. Similarly, a *missing* activity p can be added to a non-compliant path t at position k (with

[2] Apromore is an open source platform, and the code as well as the links to the cloud versions of Apromore can be found at http://apromore.org/.

$1 \leq k \leq n+1$) to make it compliant wrt. ϕ. After the addition, the resulting compliant path is $\hat{t} = \langle e_1, ..., e_{k-1}, p, e_k, e_{k+1}, ..., e_n \rangle$.

The *addition* and *deletion* actions allow us to understand if the reason of a non-compliance path is due to the absence/presence of missing/wrong activities in t. Furthermore, these two actions are characterized by two values for the cost quantifying the severity of the violation found. The final cost of the transformation will be the sum of the number of deletion multiplied by the deletion cost plus the number of addition multiplied by the addition cost. Given what explained above, we can define the compliance checking problem as follows:

Definition 1 (Compliance Checking). *Given a BP path t and an* LTL$_f$ *formula ϕ such that t violates ϕ, find a path $\hat{t}$ that satisfies ϕ and such that the transformation cost is minimal.*

Compliance checking can be addressed by resorting to NFA. To see this, let $t = \langle e_1, ..., e_n \rangle$ be a BP path, ϕ the LTL$_f$ rule to check t against, and $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$ the corresponding NFA, which we call the *constraint automaton*. From t, we define a further automaton, called the *path automaton*, $\mathcal{T} = \langle \Sigma_t, Q_t, q_0^t, \delta_t, F_t \rangle$, where: *(i)* $\Sigma_t = \{e_1, ..., e_n\}$; *(ii)* $Q_t = \{q_0^t, \ldots, q_n^t\}$ is a set of $n+1$ states; *(iii)* $\delta_t = \bigcup_{i=0,\ldots,n-1} \langle q_i^t, e_{i+1}, q_{i+1}^t \rangle$; *(iv)* $F^t = \{q_n^t\}$. By construction, $\mathcal{T}$ is deterministic and accepts only t.

Next, we augment $\mathcal{T}$ and $\mathcal{A}$ to make them suitable to our definition of compliance checking, i.e., by adding transitions related to *addition* and *deletion* of activities. From $\mathcal{T}$, we generate the automaton $\mathcal{T}^+ = \langle \Sigma_t^+, Q_t, q_0^t, \delta_t^+, F_t \rangle$, where:

– Σ_t^+ contains all the activities in Σ_t, plus: one new activity *del_p*, for all activities $p \in \Sigma_t$; and one new activity *add_p*, for all activities $p \in \Sigma \cup \Sigma_t$;
– δ_t^+ contains all the transitions in δ_t, plus: a new transition $\langle q, del\_p, q' \rangle$, for all transitions $\langle q, p, q' \rangle \in \delta_t$; and, for all activities $p \in \Sigma \cup \Sigma_t$ and states $q \in Q_t$, a new transition $\langle q, add\_p, q \rangle$.

For example, if we indicate as Crt activity Create claim, and as Dec activity Decide on claim, the augmented path automaton $\mathcal{T}^+$ associated to path $t_{ex} = \langle$Crt, Rec, Dec, Rej, Rev$\rangle$, derived from the BP model of our running example, is shown in Fig. 5.

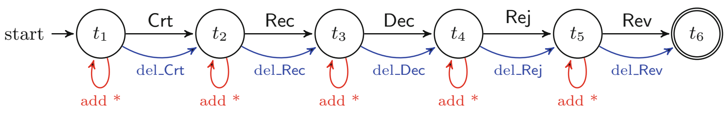


Fig. 5. Example of augmented path automaton

Similarly, from $\mathcal{A}$, we obtain an automaton $\mathcal{A}^+ = \langle \Sigma^+, Q, q_0, \delta^+, F \rangle$, such that: *(i)* Σ^+ contains all the activities in Σ, plus: one new activity *add_p*, for all activities $p \in \Sigma$; and one new activity *del_p*, for all activities $p \in \Sigma \cup \Sigma_t$; and *(ii)* δ^+ contains all the transitions in δ, plus: one new transition $\langle q, del\_p, q\rangle$ for all $q \in Q$ and $p \in \Sigma \cup \Sigma_t$; and one new transition $\langle q, add\_p, q'\rangle$ for all transitions $\langle q, p, q'\rangle \in \delta$. For instance, the DECLARE rules defined in our running example can be represented with the three augmented constraint automata shown in Fig. 6.

Intuitively, $\mathcal{A}^+$ accepts all paths $\hat{t}$ that satisfy ϕ and have been obtained by adding/removing missing/wrong activities to/from t, with the additions/deletions explicitly marked. For instance, if we consider path t_{ex} and its augmented path automaton in Fig. 5, neither the constraint automata in Fig. 3 nor their augmented versions in Fig. 6 accept t_{ex}. However, if we "repair" t_{ex} by removing Rev at the end, and we explicitly mark the repair with *del*_Rev, then all the augmented automata accept the new path $\hat{t}_{ex} = \langle \mathsf{Crt}, \mathsf{Rec}, \mathsf{Dec}, \mathsf{Rej}, del\_\mathsf{Rev}\rangle$.

Thus, given a BP path t and many LTL$_f$ rules $\phi_1, \ldots, \phi_n$, compliance checking is equivalent to searching for a "repaired path" $\hat{t}$ accepted by both $\mathcal{T}^+$ and $\mathcal{A}_1^+, \ldots, \mathcal{A}_n^+$ (i.e., the augmented constraint automata for all LTL$_f$ rules), with a minimal number of add/delete activities. We next show how to take advantage of the planning technology to efficiently search for the desired repaired path.

5.2 Encoding in PDDL

In this section, we show how, given a set of augmented constraint automata $\mathcal{A}_1^+, .., \mathcal{A}_n^+$ obtained from n LTL$_f$ formulas $\phi_1, .., \phi_n$, and an augmented path automaton $\mathcal{T}^+$ obtained from a path t, we build a cost-optimal planning domain $\mathcal{P_D}$ and a problem instance $\mathcal{P}$ in PDDL. $\mathcal{P_D}$ and $\mathcal{P}$ can be used to feed any state-of-the-art planners accepting PDDL 2.1 specification, as discussed in Sect. 3.3. A solution plan for $\mathcal{P}$ amounts to the set of interventions of minimal cost to repair the path wrt. the LTL$_f$ formulas.

Planning Domain. In $\mathcal{P_D}$, we provide two abstract types: `activity` and `state`. The first captures the activities involved in a transition between two different states of a constraint/path automaton. The second is used to uniquely identify the states of any constraint automaton (through the sub-type `automaton_state`) and of the path automaton (through the sub-type `path_state`). To capture the structure of the automata and to monitor their evolution, we defined four *domain propositions* as boolean predicates in $\mathcal{P_D}$:

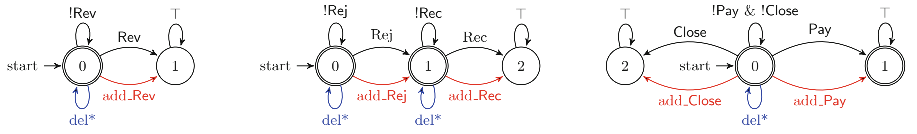


Fig. 6. Augmented constraint automata for DECLARE rules: (a) *absence*(Rev), (b) *not succession*(Rej,Rec), (c) *precedence*(Pay, Close)

- `(path ?t1 - path_state ?e - activity ?t2 - path_state)` holds if there exists a transition in the path automaton between two states `t1` and `t2`, being `e` the activity involved in the transition.
- `(automaton ?s1 - automaton_state ?e - activity ?s2 - automaton_state)` holds if there exists a transition between two states `s1` to `s2` of a constraint automaton, being `e` the activity involved in the transition.
- `(cur_state ?s - state)` holds if `s` is the current state of a constraint/path automaton.
- `(final_state ?s - state)` holds if `s` is a final state of a constraint/path automaton.

Furthermore, we define a *numeric fluent* `total-cost` to keep track of the cost of the violations. Notice that: *(i)* in PDDL, parameters are written with a question mark character '?' in front, and the dash character '-' is used to assign types to parameters; and *(ii)* we remain consistent with the PDDL terminology, which allows for both the values of predicates and fluents to change as a result of the execution of an action.

Planning actions are used to express the *repairs* on the original path t. Each action is characterized by its *preconditions* and *effects*, stated in terms of the domain propositions. In our encoding, we have defined three actions to perform *synchronous moves* in the path automaton and in the constraint automata, or to add/remove activities to/from the path automaton. In the following example, we suppose that both actions `add` and `del` have cost equal to 1. We notice that their cost can be customized to properly define the severity of a violation.

```
(:action sync
 :parameters (?t1 - path_state ?e - activity ?t2 - path_state)
 :precondition (and (cur_state ?t1) (path ?t1 ?e ?t2))
 :effect(and (not (cur_state ?t1)) (cur_state ?t2)
            (forall (?s1 ?s2 - automaton_state)
              (when (and (cur_state ?s1)
                         (automaton ?s1 ?e ?s2))
                    (and (not (cur_state ?s1))(cur_state ?s2))))))
```

```
(:action add
 :parameters (?e - activity)
 :effect (and (increase (total-cost) 1)
          (forall (?s1 ?s2 - automaton_state)
            (when (and (cur_state ?s1)
                       (automaton ?s1 ?e ?s2))
                  (and (not (cur_state ?s1))
                       (cur_state ?s2))))))
```

```
(:action del
 :parameters (?t1 - path_state
              ?e - activity
              ?t2 - path_state)
 :precondition (and (cur_state ?t1)
                    (path ?t1 ?e ?t2))
 :effect(and (increase (total-cost) 1)
   (not (cur_state ?t1))(cur_state ?t2)))
```

We modeled `sync` and `del` in such a way that they can be applied only if there exists a transition from the current state `t1` of the path automaton to a subsequent state `t2`, being `e` the activity involved in the transition. Notice that, while `del` yields a *single* move in the path automaton, `sync` yields, in addition, one move per constraint automaton (all to be performed synchronously). In particular, a synchronous move is performed in each constraint automaton for which there exists a transition involving activity `e` connecting `s1` – the current state of the automaton – to a state `s2`. Finally, `add` is performed only for transitions involving activity `e` connecting two states of any constraint automaton, with the

Table 1. Experimental results: the time (in milliseconds) is the average per path

Path Length	Search Time (isolated rules)	Search Time (entire model)	Search Time (isolated rules)	Search Time (entire model)	Search Time (isolated rules)	Search Time (entire model)
0 const. violated	5 constraints		10 constraints		15 constraints	
10	615	146	1,189	282	1,763	1,764
25	725	164	1,454	295	2,181	1,792
35	732	165	1,508	300	2,256	1,799
50	774	172	1,542	311	2,292	1,801
1 const. violated	5 constraints		10 constraints		15 constraints	
10	599	147	1,173	414	1,769	3,716
25	646	156	1,304	421	1,971	3,665
35	719	166	1,450	439	2,181	3,889
50	747	172	1,481	458	2,250	4,091
3 const. violated	5 constraints		10 constraints		15 constraints	
10	596	158	1,164	558	1,727	5,700
25	725	187	1,422	586	2,160	6,186
35	752	196	1,518	618	2,286	6,432
50	771	201	1,535	631	2,319	6,710
5 const. violated	5 constraints		10 constraints		15 constraints	
10	614	162	1,214	625	1,838	6,113
25	760	207	1,552	626	2,310	6,362
35	767	211	1,569	628	2,364	6,401
50	801	219	1,651	641	2,506	6,612

current state of the path automaton that remains the same after the execution of the action.

Planning Problem. In $\mathcal{P}$, we first define a finite set of constants required to properly ground all the domain propositions defined in $\mathcal{P}_{\mathcal{D}}$. In our case, constants correspond to the state and activity instances involved in the path automaton and in any constraint automaton. Secondly, we define the *initial state* of $\mathcal{P}$ to capture the exact structure of the path automaton and any constraint automaton. This includes the specification of all the existing transitions that connect two states of the automata. The current state and the final states of any path/constraint automaton are identified as well. Thirdly, to encode the goal condition, we first pre-process each constraint automaton by: *(i)* adding a new dummy state with no outgoing transitions; *(ii)* adding a new special action, executable only in the final states of the original automaton, which makes the automaton move to the dummy state; and *(iii)* including in the set of final states only the dummy state. Then, we define the goal condition as the conjunction of the final states of the path automaton and of all the constraint automata. In this way, we avoid using disjunctions in goal formulas, which are not supported by all planners. Finally, as our purpose is to minimize the total cost of the plan, $\mathcal{P}$ contains the following specification: `(:metric minimize (total-cost))`.

6 Evaluation

In order to investigate the *level of scalability* of our planning-based approach, we performed (with our tool) several synthetic experiments employing BP models and compliance requirements of increasing complexity. First, we created 4 BPMN models with an increasing number of parallel branches.[3] Then, we converted the BPMN models into Petri nets and unfolded them thus obtaining 1121, 800, 961 and 1025 execution paths of average length 10, 25, 35 and 50, respectively.

Secondly, we defined 3 DECLARE models (having the same alphabet of activities) containing 5, 10 and 15 rules that are known to be compliant with the tested BPMN models. Then, to create DECLARE models non-compliant with the BPMN models, we changed some of the rules in the original DECLARE models by replacing 1, 3 and 5 rules in each model with their negative counterparts. In this way, we were able to understand how performance scales up with longer paths and more DECLARE rules, considering a growing amount of violations. We point out that real execution paths involve most often less than 50 activities, and compliance requirements with 15 DECLARE rules are considered to be large. So, one should consider our test not only as a practical one based on realistic settings, but also as one that is challenging. We used a standard cost function with cost 1 for any step that adds/removes activities to/from the input path, and cost 0 for synchronous moves. We tested our approach on the grounded version of the problem presented in Sect. 5.2. The experiments were performed with a machine consisting of a 2,7 GHz Intel Core i5 CPU and 8GB RAM. We configured the FAST-DOWNWARD planner, which is integrated in our tool, to employ the A* searching algorithm to guarantee the optimality of the solution.

The results of our experiments can be seen in Table 1 and Fig. 7. Concerning the table, for each combination "path length - size of DECLARE model", the table reports the search time (averaged over all paths) required by the planner for checking the compliance of all the rules included in the DECLARE model: *(i)* when tested in isolation (column "Search Time (isolated rules)"); and *(ii)* when tested as a conjunction (column "Search Time (entire model)"). Notice also that the rows of the table are split into clusters according to the number of DECLARE rules violated by any of the tested paths.

By analyzing Table 1 and Fig. 7, some conclusions can be drawn. Regarding the experiments performed to check the compliance of a path against the individual rules included in a DECLARE model, it is evident that the amount of rule violations has a small influence on the performance, differently from the size of the DECLARE models and the length of the paths. The latter two factors have a major influence on the search time, though not in a dramatic way to preclude from practical applicability. Conversely, the tests performed against the entire DECLARE models suggest that the presence of a large number of violations becomes the key factor that influences the search time. This is more evident with the largest DECLARE model consisting of 15 rules. However, the results show that the approach is feasible for offline reasoning and scales up well when DECLARE models and paths grow in size.

[3] To guarantee the repeatability of our experiments, at https://goo.gl/rwNRw7, we provide: the instructions to configure and run the tool; the BPMN and DECLARE models used for the experiments.

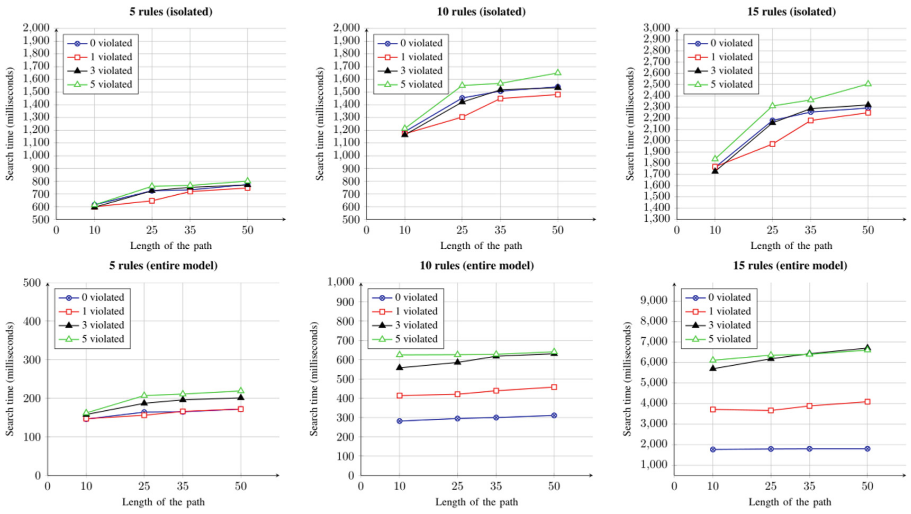


Fig. 7. Performance of repairing violations through planning

7 Related Work

In the context of process mining, some techniques have approached the problem of BP model repair based on event logs [4,14,21,27]. These techniques aim at keeping the repaired model as similar as possible to the original model, while capturing all the possible behaviors from the event log. Some research works also exist that use planning techniques to deal with trace alignment in the context of conformance checking [8,9,19]. Readers should observe that the aforementioned approaches use event logs, while our approach is able to compare BP models and temporal rules.

Approaches addressing a problem that is closer to the one we face in this paper are presented in [6,20]. These approaches provide insights about violations of temporal rules in BP models through model checking. In [6], the authors use BPMN-Q to express compliance requirements visually and computational tree logic (CTL) expressions to formally represent their semantics. Based on these requirements a counterexample is provided to explain possible discrepancies wrt. an input BPMN model. As already mentioned, model checkers provide counterexamples to explain non-compliance, but the diagnostics can be difficult to interpret. In [20], the authors aim at reducing the error paths produced by model checkers to make their diagnostics more user-friendly. This approach provides the user with diagnostics to understand the root cause of the violations, but it does not provide detailed feedback to the user in terms of where each compliance requirement is violated and why, and what to do to solve the non-compliance. Instead, we try to provide a richer feedback that can be used to adjust the input BP model in a semi-automatic way.

In [5], the authors introduce a compliance checking approach that identifies 4 pre-defined violation types and, for each of them, generates a resolution strategy using automated planning. While we adopt a similar technique to the synthesis of repair strategies, our approach is able to detect violations of any rule that can be represented as an NFA.

In [11], the authors still deal with compliance checking of a BP model wrt. temporal rules. However, this work is more focused on finding reparative actions to be applied to the model to solve the violations. This problem is addressed by making some assumptions on the input BP model and compliance requirements. In particular, the input model needs to be a block-structured Workflow net without loops. In addition, the compliance requirements can be expressed with a sub-set of DECLARE and the intersection of the set of the behaviors of the BP model and the set of the behaviors of the DECLARE rules needs to be non-empty. These assumptions are needed to solve the repair problem, which is, in general, extremely challenging. In our contribution, we find a different trade-off in addressing this problem by providing general insights on where and why a violation occurs and providing only in some specific cases suggestions about reparative actions. This allows us to relax the assumptions made in [11]. Our approach can indeed be applied to any safe PN and any compliance requirement that can be expressed in LTL$_f$.

For a general introduction to the topic of regulatory compliance checking, the reader is referred to [29].

8 Concluding Remarks

Existing approaches to compliance checking employ verification techniques to explain violations through counterexamples, which can be extremely unintuitive in the presence of a large number of violations. Consequently, explaining violations using counterexamples could not be the most suitable solution to understand how to change BP models to solve non-compliance. To tackle this issue, in this paper, we have shown how automated planning can be used to efficiently solve the problem of checking compliance requirements expressed in terms of LTL$_f$ rules, by pointing at the BP activities where compliance is breached.

It is worth noticing that the objective of the evaluation was to stress the scalability of planning techniques for checking the compliance of *single* BP paths and DECLARE models of *growing size*. However, when BP models include a large number of parallel gateways and cycles, the number of unfolded BP paths quickly explodes. As a consequence, part of the complexity moves from the size of BP paths and DECLARE models to the total amount of BP paths to be checked for compliance. Therefore, as future work, we aim at improving the presented technique to make it able to suggest reparative actions in larger sets of paths wrt. the ones considered in this paper. In addition, we plan to detect non-compliance that relates to time, resource and data aspects, such as activities that are not performed by authorized actors or within given deadlines. This problem is far from being trivial since it is, in general, undecidable. Thus, we need to explore what kind of expressiveness limitations are required to ensure decidability.

Acknowledgments. The work of Fabrizio M. Maggi was supported by the Estonian Research Council Grant IUT20-55. The work of Andrea Marrella was partially supported by the projects METRICS (Sapienza grant), Social Museum & Smart Tourism (Italian CTN01 00034 23154), DTC Lazio and Resilience of Metropolitan Areas (ROMA) (Italian SCN 00064).

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circuits Syst. Comput. **08**, 21–66 (1998)
2. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process mining and verification of properties: an approach based on temporal logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005). https://doi.org/10.1007/11575771_11
3. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Diagnosing behavioral differences between business process models: an approach based on event structures. Inf. Sys. **56**, 304–325 (2016)
4. Armas-Cervantes, A., van Beest, N.R.T.P., La Rosa, M., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: CoopIS (2017)
5. Awad, A., Smirnov, S., Weske, M.: Resolution of compliance violation in business process models: a planning-based approach. In: CoopIS (2009)
6. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. Vis. Lang. Comput. **22**(1), 30–55 (2011)
7. Clempner, J.: Verifying soundness of business processes: a decision process Petri nets approach. Expert. Syst. Appl. **41**(11), 5030–5040 (2014)
8. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the disruptive effectiveness of automated planning for LTLf-based trace alignment. In: AAAI (2017)
9. De Giacomo, G., Maggi, F.M., Marrella, A., Sardiña, S.: Computing trace alignment against declarative process models through planning. In: ICAPS (2016)
10. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. IJCAI **15**, 1558–1564 (2015)
11. De Masellis, R., Di Francescomarino, C., Ghidini, C., Lapõnin, A., Maggi, F.M.: Rule propagation: adapting procedural process models to declarative business rules. In: EDOC (2017)
12. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
13. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33143-5
14. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
15. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. (JAIR) **20**, 61–124 (2003)
16. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: Theory and Practice. MK, Burlington (2004)
17. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Inf. Sci. **385**, 39–54 (2017)

18. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Inform. **39**(3), 143–209 (2003)
19. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Expert. Syst. Appl. **82**, 162–183 (2017)
20. Lohmann, N., Fahland, D.: Where Did I Go Wrong? Explaining errors in business explaining errors in business. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 283–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_18
21. Maggi, F.M., Corapi, D., Russo, A., Lupu, E., Visaggio, G.: Revising process models through inductive learning. In: zur Muehlen, M., Su, J. (eds.) BPM 2010. LNBIP, vol. 66, pp. 182–193. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20511-8_16
22. Marrella, A.: What automated planning can do for business process management. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 7–19. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_1
23. McMillan, K.L., Probst, D.K.: A technique of state space search based on unfolding. Form. Methods Syst. Des. **6**(1), 45–65 (1995)
24. Murata, T.: Petri nets: properties, analysis and applications. In: IEEE (1989)
25. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC, pp. 287–300 (2007)
26. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science (1977)
27. Polyvyanyy, A., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wynn, M.T.: Impact-driven process model repair. ACM Trans. Softw. Eng. Methodol. **25**(4), 1–60 (2016)
28. Sadiq, S., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_12
29. Sadiq, S., Governatori, G.: Managing regulatory compliance in business processes. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 2. IHIS, pp. 265–288. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-642-45103-4_11

A Genetic Algorithm for Cost-Aware Business Processes Execution in the Cloud

Guillaume Rosinosky(✉), Samir Youcef, and François Charoy

Université de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France
{guillaume.rosinosky,samir.youcef,francois.charoy}@loria.fr

Abstract. With the generalization of the Cloud, software providers can distribute their software as a service without investing in large infrastructure. However, without an effective resource allocation method, their operation cost can grow quickly, hindering the profitability of the service. This is the case for BPM as a Service providers that want to handle hundreds of customers with a given quality of service. Since there are variations in the needed load and in the number of users of the service, the allocation and scheduling methods must be able to adjust the cloud resource quantity and size, and the distribution of customers on these resources. In this paper, we present a cost optimization model and an heuristic based on genetic algorithms to adjust resource allocation to the needs of a set of customers with varying BPM task throughput. Experimentations using realistic customer loads and cloud resources capacities show the gain of these methods compared to previous approaches. Results show that using our algorithm on split groups of customers can provide even better results.

Keywords: BPM · Elasticity · Cloud · Optimization

1 Introduction

Consuming "Business Process Management as a Service" (BPMaaS) offer benefits that IT people widely acknowledge. It reduces the operational burden and allows to rely on the provider for the maintenance and provisioning of the service. However, from the BPMaaS provider point of view, it increases the operational complexity. The provider must ensure that all his customers receive the same attention and a defined quality of service at all time. He must also ensure that it operates at the best possible cost.

A Business Process Management System (BPMS) deployment is complex: it requires application servers, process engines and database management systems. Clustered installations requiring load balancers can also be deployed for high availability. Each customer has a different usage pattern and the number of tasks to execute evolves constantly. Each cloud compute resource is costly and has a capacity corresponding to its CPU, memory, storage and network speed, for a defined response time. In order to maintain an optimal infrastructure cost, processes executions must be distributed on different cloud resources. Figure 1 shows an example with two customers (or tenants). They require different capacity at each hour. We want to allocate the cheaper Cloud resources and distribute the process execution of tenants on these resources to optimize the operational cost.

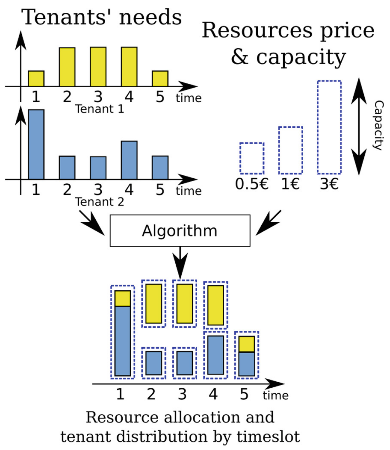


Fig. 1. Multi tenant resource allocation and distribution tenants

In order to optimize resource usage, we must sometimes migrate tenants from an installation to another. In Fig. 1, at time 5, tenant 1 is moved on the same resource as tenant 2 because both require less capacity and fit on this resource. A migration generates disruption of service on the customer side. We must stop the processes and move the data from one installation to the other [1]. We must find the best distribution of tenants on resources for each interval of time while controlling the number of migrations for each tenant. This problem becomes complex for a high numbers of tenants.

We propose an integer linear programming (ILP) model and a genetic algorithm that aims at finding the best allocation strategy while limiting the number of migrations per tenant. We improve our previous results [2] with a method that provides a better cost elasticity.

In the next section, we describe our migration strategy model, our genetic algorithm approach coupled to a solving of our model, and our previous iterative heuristic. We then study experimental results on both of the approaches compared to a baseline approach. We also show how we achieve interesting experimental results. Then, we compare our results with the state of the art. The last part concludes and presents our future work.

2 A Migration Strategy Based Model

In this Section we present the context for BPMaaS, our hypothesis and our method to optimize resource cost. This can be seen as a resource allocation problem with a constraint regarding the number of reallocations. Moving a tenant from a resource to another generates a service disruption. We want to limit their number to ensure stable quality of service for the customers. Thus, one of our problems is to find the right time to move tenants. We call it the migration strategy.

2.1 Context and Constraints

Our approach is tenant-centric. All customers (tenants) processes are executed on the same BPMS installation. It is easier to manage deployments by customers rather than by processes. Customer processes share business data and security configuration that we need to manage together. Our assumptions are the following:

BPMS Do Not Scale Infinitely. There is no such thing as an infinitely scalable BPMS installation. Even clustered installation reach a limit due to the transactional nature of the database interactions. Thus, we use several BPMS installations. In our approach, we assume our tenants can fit on the "bigger" resource available.

Provisioning and Deprovisioning Takes Time. We cannot change instantly tenant distribution, as computing instances instantiation, software installation, and data migration takes time. Thus we compute resource allocation in a discretized manner at fixed time interval or time slot. A time slot is a significant period of time for the provider: it could be a few seconds or an hour.

A Tenant Is a Customer of the BPMaaS. Tenants run BPM processes composed of tasks. To execute them, the BPMS needs computing power, network bandwidth, disk and memory. It relies on separated compute instances for database systems for data persistence, and load balancers for clustered installations.

Task Throughput Is Our Performance Metric. It corresponds to the number of BPM tasks executed for one period of time (e.g. per second). This metric is meaningful for the customers.

Our Approach Is Offline. We assume we know the required BPM task throughput for each tenant and each time slot.

A Cloud Resource Has a Capacity Expressed in BPM Task Throughput. A *cloud resource* (or *resource*) is one or several cloud compute instances that we use for the database tier, BPM system tier, load balancer tier, etc. It supports a full BPMS installation. A cloud resource can host several tenants. In our case, we assume a tenant fits on one resource: tenants won't be distributed on several cloud resources.

The Number of Migrations Must be Limited. We name *migration* the action of moving a tenant data and processes from one cloud resource to another. It requires the target cloud resource to be up and running. This action takes time to be executed. If all tenants of a cloud resource are migrated, it can be released. Migrations generate QoS breaks for the customers [1]. We limit their number for each tenant depending on the Service Level Agreement.

Without an optimization method, a solution to allocate tenant to resources would be to allocate each tenant on a resource that supports its maximum task throughput. We call this solution, *baseline method*. This method is often used in production, but can become very expensive.

We proposed in [2] a method based on an iterative heuristic and time series segmentation. It computes the list of necessary cloud resources and a mapping of tenants on each resource, time slot per time slot. We present it in the next Section.

2.2 Allocation with an Iterative Heuristic and Time Series Segmentation

In this part, we recall briefly our previous method [2]. It has two parts: first, we choose a *migration strategy* i.e the time slots where each tenant can migrate, and, second, we apply a heuristic using resources prices and capacity, tenants needs, and their migration strategy in order to obtain the cloud resources and the placement of tenants.

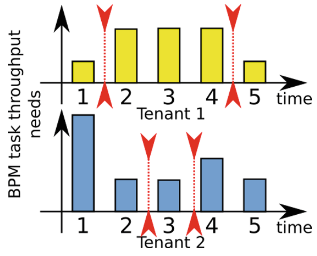


Fig. 2. Example of a migration strategy (Color figure online)

For the first part, a time series segmentation [3] allows to select good migration times for each tenant. We present in Fig. 2 two examples of migration strategies, in red for two tenants. In this figure, *Tenant 1* can only be migrated at the end of time slots 1 and 4. For *Tenant 2*, it is time slots 2 and 3. This means both tenants will be migrated at most 2 times. They will stay on their origin resource if there is no migration authorization.

For the second part, our iterative timeslot heuristic is based on Variable Cost and Size bin packing in which we added repacking steps. It takes as input the tenant throughput by timeslot, the cloud resource prices and capacity, and the migration strategies for the tenants.

This coupled approach provides better results than the baseline approach. Still, the comparison with the solution computed with a solver showed that it is far from the optimal cost. It is possible to enhance this solution. We propose to use a genetic algorithm to find migration strategies that reduce the resource cost, and propose an alternative to the iterative timeslot heuristic based on integer linear programming. We present this model in the next Section.

2.3 An Efficient Model for Migration Strategies

Our problem is to find resource and tenant distributions for each time slot, for given tenants' loads, resource prices and capacity, and migration strategy. The model we propose answers to this problem. Our model principle is based on the absence of tenant migration when there no authorization to move. In this case, simple placement constraints should exist, and no constraint exist when there is an authorization to move in the migration strategy.

Let the following variables:

- $\mathcal{T}$, the set of cloud resource types, with t its cardinality.
- $\mathcal{I}$, the set of tenants with n its cardinality.
- $\mathcal{J}$, is $\mathcal{T} \times \mathcal{I}$ the set of all possible cloud resources associated with each tenant. Its cardinality is $m = t \times n$.
- C_j, and W_j represent respectively the cost and the capacity in terms of BPM task throughput for the configuration j, with j in $\mathcal{J}$.
- $w_i(\mathrm{k})$, the required capacity in terms of BPM task throughput for the tenant i during time slot k.
- $\mathcal{K}$ defines all the time slots, from 0 to D, where $D+1$ is the number of time slots.
- ${x_j}^i(k)$, the assignment of tenant i to configuration instance j during time slot k.
- $y_j(k)$, the activation of configuration j during time slot k.
- M, the maximum number of migrations of tenants between cloud resources on all time slots.
- $h_i(k)$ with $0 \leq k \leq D-1$. $h_i(k)$ is equal to 0 if the tenant i is not allowed to be migrated between time slot k and $k+1$, and equal to 1, if it is allowed. The set of all $h_i(k)$ (for each tenant and each time slot) is a migration strategy.
- migration strategies assume the maximum number of migrations allowed per tenant: $\forall i \in \mathcal{I} \sum_{k}^{k \in \mathcal{K}} h_i(k) = M$ where M is the number of migrations. This number depends on the SLA.

The objective is to minimize the total cost for all active cloud resources, for each time slot, as shows Eq. 1.

$$\min \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K}} C_j y_j(k) \tag{1}$$

We must ensure that the following constraints are enforced:

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K} \sum_{j}^{j \in \mathcal{J}} {x_j}^i(k) = 1 \tag{2}$$

$$\forall j \in \mathcal{J}, \forall k \in \mathcal{K} \sum_{i}^{i \in \mathcal{I}} w_i(k){x_j}^i(k) \leq W_j y_j(k) \tag{3}$$

$$\forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K} | h_i(k) = 0, {x_j}^i(k) = {x_j}^i(k+1) \tag{4}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, {x_i}^j(k) \in \{0, 1\}, y_j(k) \in \{0, 1\} \tag{5}$$

Equation 2 represents the obligation for a tenant to be placed at each time slot on an active cloud resource. Equation 3 means that the sum of the required capacity for each tenant on one cloud resource cannot exceed the capacity of the cloud resource. Equation 5 represents the variables we use. Equation 4 represent the migration strategy. The equality constraint means that for a tenant i and a time slot k, assignation values ${x_i}^j(k)$ will stay the same on time slots k and $k+1$. When a tenant is authorized to migrate between resources, there is no constraint for this tenant. Generalizing this on all resources produces the desired effect: tenants will be migrated from one resource to another only during the time slots specified by the migration strategy. The pre-defined migration strategy is symbolized here by the variable $h_i(k)$.

Finding cheap migrations strategies is primordial in our approach. This is the goal of our genetic algorithm. We present it in the following Section.

2.4 Cost Optimization via Genetic Algorithms

A genetic algorithm is a well known meta-heuristic belonging to the family of evolutionary algorithms, and inspired by natural selection [4]. Its principle is to produce directed random evolutions on a population of individuals until it obtains one or several individuals with a good fitness value. Individuals are usually vectors of boolean values, whose corresponding fitness can be evaluated. Iterations are triggered until an end condition is reached.

We want to find the best migration strategy for all the tenants and time slots. Figure 3 shows our approach. The general principle is to use our iterative time slot heuristic (or the restricted model we have described in the previous Section) for evaluating migration strategies, until we find the best. To represent an individual, we vectorize a migration strategy by concatenating migration strategies of each $|\mathcal{I}|$ tenants (each one corresponding to a vector of D boolean values, $D+1$ being the number of studied time slots). The size of the vector will be $D \times |\mathcal{I}|$, with each element being equal to zero or one. For instance with two tenants and three time slots, the first migrating on the second time slot and the second tenant on the third time slot, we will have the following representation: $\begin{bmatrix} 0 \ 1 \ 0 \ 0 \ 0 \ 1 \end{bmatrix}$.

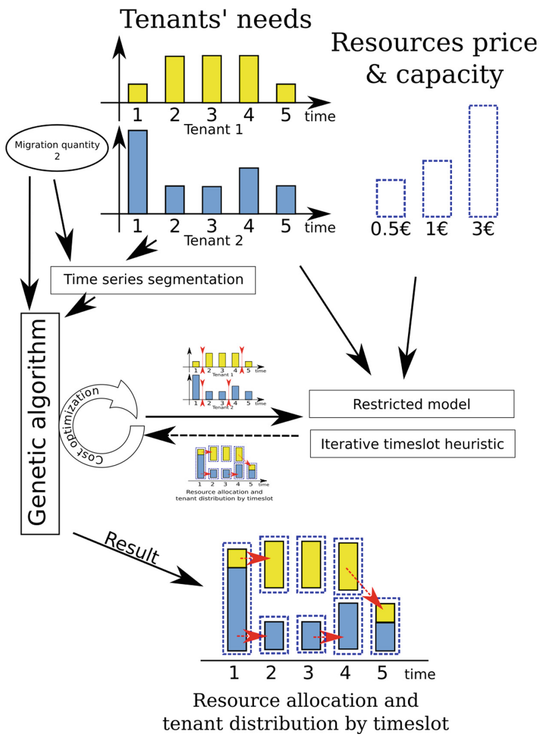


Fig. 3. A genetic algorithm to find better strategies

The reader will find in Fig. 4 a brief description of the different steps of a genetic algorithm. Compared to the traditional approach, we have switched the mutation phase and the crossover phase. The co-hosted mutation requires to know the cost of the migration strategy in the population. We compute the cost in the fitness evaluation phase. The crossover phase generates potentially unknown (not yet computed) migration strategies. Thus, we do it after the mutation phase. In our case, "parents" are mutated instead of the offspring. In the following, we describe the solution we designed.

Population Initialization. We initialize the population with several segmentation algorithm combinations (as we described in Sect. 2.2), and with random individuals with the correct number of migrations for each tenant.

Fitness Evaluation. We want to find the migration strategies that produces the lowest cost. The fitness score corresponds to the total cost of all the active resources on the time slots. To evaluate it on the different individuals, we compute the cloud resources allocation and placement of the tenants on it. In our case, we run our iterative time slot heuristic [2] or a solver on our model presented in Sect. 2.4, for each individual (migration strategy) we need to evaluate.

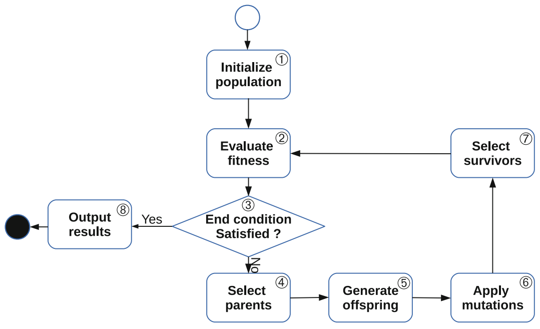


Fig. 4. Genetic algorithm phases

We keep the cloud resources and tenants assignation distribution in memory for the next steps, and the fitness score.

Termination Condition. We use a time limit termination condition. This allow us to compare different solutions based on this limit.

Parent Selection. For this step, we use a classical rank selection strategy. We sort the population by fitness and we select randomly, and with a higher priority, the individuals with the higher rank (lower fitness or price in our case) for parents.

A Specific Mutation: Co-hosted Tenant Migration Mutation Strategy. In classic approaches, mutation updates randomly individuals, depending on a mutation rate, switching scalar values from zero to one or the other way around [4]. Here, the goal is to generate brand new individuals in the population, with non tested configurations. In our case, we cannot use a totally random approach, as the number of migrations for each tenant is bounded. However, even a randomized approach keeping a fixed number of migrations will not provide the desired effects as we can see on the left side of Fig. 5. We have noticed that most of the times, resources are not liberated as only one of their tenants is authorized to migrate. It limits the savings of resources liberation. We developed an alternative mutation more suited to our problem.

It consists in shifting the authorization to migrate for each co-hosted tenant at the indicated time slot for the reference tenant's resource. To achieve this goal, for each tenant, we browse the past time slots until we find an authorization to migrate or until we reach the beginning of the time slot space. If we find one, we set it to zero while setting to one the "destination" time slot. If the "destination" time slot is already set to one, we ignore this behavior. The example on the right side of Fig. 5 describes this principle. There, it is possible to migrate all the tenants of resource R1 to the cheaper resource R2, and thus reduce costs[1].

[1] The simple approach on the left of Fig. 5 keeps tenants on resource R1, it can never be freed.

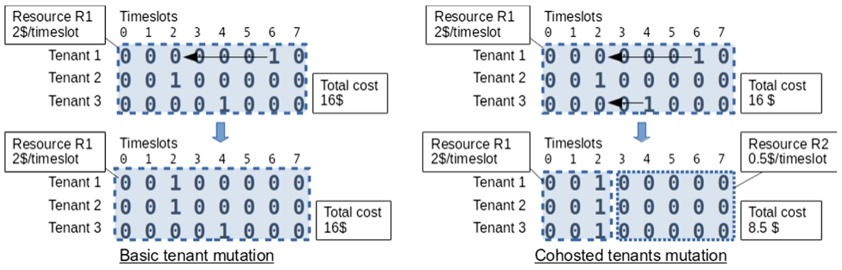


Fig. 5. Basic tenant mutation vs cohosted mutation

Specific Offspring Generation: The Tenant Crossover Strategy. The crossover phase consists in randomly mixing individuals (parents) of the current population in order to generate new individuals (children) having characteristics of both parents. The crossover technique that we use consists in switching the migration time of random tenants. First, two children identical to two migration strategies parents are generated. Then, depending on the number of tenants specified, each one will see its migration times switched in the children.

Generational Replacement. We replace the entire population with the offsprings, except for the best individuals from the original population (named elites). They replace the less fit offspring in the future population.

In the next Section, we present our experiments and the results.

3 Experimentation

We have conduct tests with the cloud resource prices and sizes, and the seeds of our previous work [2]. We consider 12 configurations, each composed of two Amazon Web Services compute resources: one database resource (RDS) for the database, and one compute instance (EC2) for the application server. Prices are comprised between 0.177$ per hour for a BPM task throughput of 16.4 tasks per second, and 4.126$ per hour for a BPM task throughput of 129.279 tasks per second.

For the customer part, we vary the number of tenants (10, 25, 50 and 100), and we use different throughputs in terms of BPM task per second. These throughputs are based on usage of anonymous customers of the BPMS Bonita[2]. We consider 6 configurations, needing a throughput respectively between 1 and 120, 14 and 16, 0 and 120, 1 and 3, 5 and 120, and 0 and 4[3].

We generate each tenant's initial time slot throughput randomly following an uniform distribution between the two throughputs. Our next step is to generate

[2] http://www.bonitasoft.com/.

[3] The data and the results are available at: https://doi.org/10.5281/zenodo.1173617. The source code of the framework is not public, except for the segmentation library, available at https://github.com/guillaumerosinosky/Segmentation/.

the variation of throughput between time slots by adding or removing a random value limited to one quarter of the difference between the maximum and the minimum throughput. For our experiments, we used the Python library Inspyred [5] for the genetic algorithm that integrated well with our environment.

Experiment Parameters. In order to obtain significant and realistic results, we used the following parameters:

– each test is launched for 10 different random seeds (i.e tenants' loads)
– a time slot size of one hour, as it was the reference duration of AWS cost model for computing instances at the time of the experiment.
– we choose to consider 4 migrations per day. A migration produces an interruption of around 10 s depending on the quantity of data.
– we consider a 2 days period (thus limiting migrations to 8, for 48 time slots).
– we consider the following parameters for the genetic algorithm: the number of elites individuals to 5, a mutation rate of 0.4, a population size of 20, a number of mutation points corresponding to the number of tenants divided by 5. These parameters were chosen following tests on multiple values for a limited number of seeds each. Details on this choice cannot be included for space restriction reasons.
– we limit the genetic algorithm computation to 600 or 1800 s and the solver computation time to 5 s.

3.1 Results

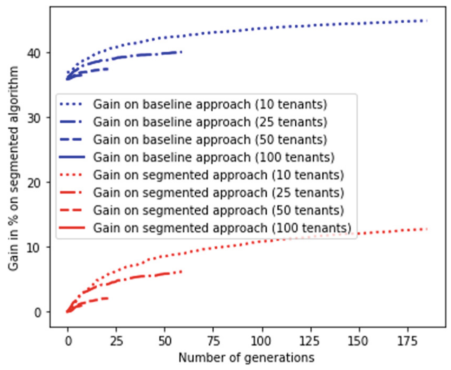


Fig. 6. Mean genetic algorithm gain on best initial segmented population for 600 s of running time (Color figure online)

In Fig. 6, we show the relative gain of this approach compared to our previous approach (segmented approach) in red (in the upper part of the figure), and to a baseline approach in blue (in the lower part of the figure). The gain is better for 10 tenants than for 100 tenants since the system has more time to search for the cheapest solution. For 10 tenants, we obtain more than 10% enhancement compared to the previous approach, and more than 45% compared to the baseline approach. However for 100 tenants, we have only a 1% enhancement.

It appears that either the iterative usage of the heuristic, the genetic algorithm or the two of them is more efficient for a small number of tenants for the same number of generations. This is why we conducted experiments where we apply the proposition to subsets of the tenants and we aggregated the results as described in the next Subsection.

3.2 The Splitting Strategy

For this solution, we split the set of tenants into small groups selected randomly. We have tested different size of groups with various number of tenants and we applied the previous method keeping the same total computation time. Figure 7 shows the results we obtained with the genetic algorithm and the iterative heuristic. The x axis corresponds to the size of the groups of tenants. The y axis shows the relative gain compared to the results with no partition. A subset size with the same size as the number of tenants corresponds to no split, the gain is zero.

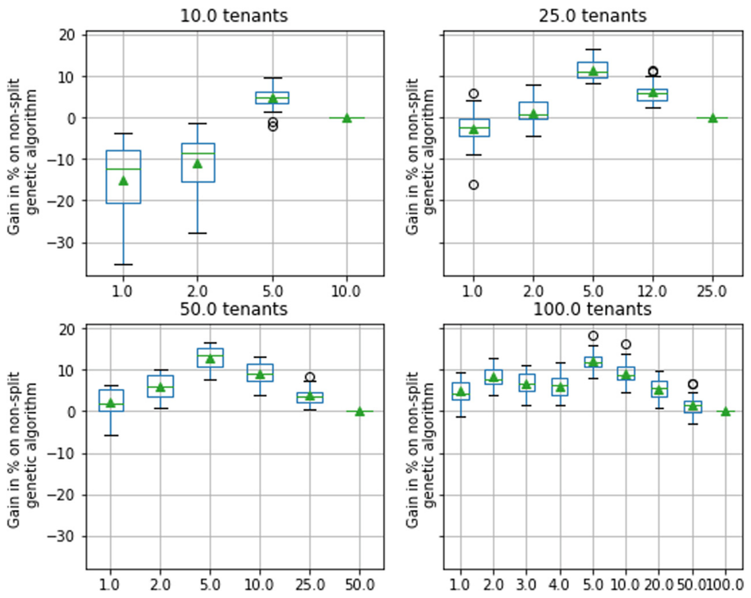


Fig. 7. Gain depending on splitting strategy for various split quantities.

We obtain the best results with partitions of 5 tenants in all cases. For the experiments we ran, the gain varies from 5% to 15%. We have no good explanation for this result that we can reproduce. Our tests with the solver give the same results for the size of the groups as with the heuristic. In the next Section we present our results with groups of 5 tenants.

3.3 Results for Solver and Iterative Heuristic

We implemented our model (presented in Subsect. 2.4) using the optimization library PuLP[4] with the Gurobi solver[5]. For execution time and cost reasons, we were not able to test every set of parameters. For instance, with our current implementation, we managed to obtain results with the solver only up to a size of 25 tenants for the partition. Indeed, the duration of the initialization part and the required memory makes it impossible to run with more tenants. Thus we have limited our tests to parts of 5 tenants, for a total of 50 and 100 tenants. As we can see, the results stay close to the results of the heuristic. Figure 8 shows the absolute gain we obtained, and the corresponding percentage compared to the *baseline approach* cost, for 600 s and 1800 s of running time. We also present the non-split result for the segmented approach (results of the previous paper), and the split segmented approach where we apply time series segmentation on the groups of 5 tenants instead of all the tenants simultaneously.

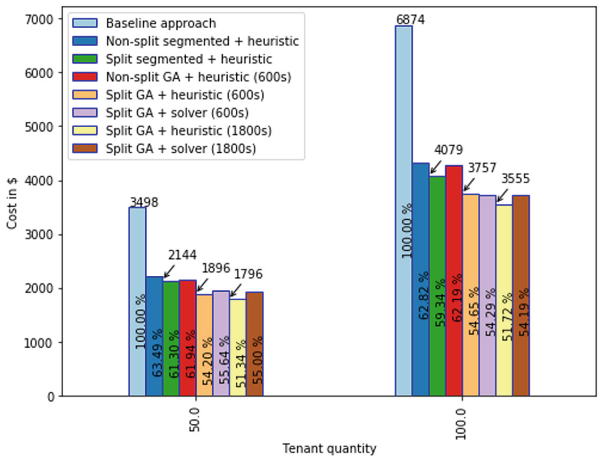


Fig. 8. Mean cost comparison for 50 and 100 tenants per group of 5

For 1800 s of execution time of the genetic algorithm, split heuristic give the best results. Mean distribution costs are 51.34% for 50 tenants, and 51.72% for 100 tenants compared to the cost of the intuitive approach. Using the solver gives good results but more expensive (respectively 55% and 54.19%). For 600 s of execution time, the results are more balanced: they vary between 54.2% and 55.64%.

[4] https://pythonhosted.org/PuLP/.
[5] http://www.gurobi.com/.

The genetic algorithm does not enhance the results a lot for both approaches after 600 s: 3% for the heuristic and less than 1% for the solver. Still, it enhances the initial split segmented results from 61.3% to 51.34% for 50 tenants, and from 59.34% to 51.72% for 100 tenants.

We observe that the split segmented approach allows to gain more than 2%, and to unleash the results of the genetic algorithm. Without splitting we gain around 1% for 600 s of genetic algorithm compared to the original population (non split segmented). When splitting, the genetic algorithm results in a gain of 7.1% for 50 tenants, and 4.69% for 100 tenants compared to the split segmented strategy. The absolute gain compared to the intuitive approach remains worthwhile for 2 days: we save 1702$ for 50 tenants and 3319$ for 100 tenants for a cost of respectively 3498$ and 6874$. The respective gain compared to our previous work is 425$ and 763$.

4 Related Work

Many researchers have studied elasticity in the cloud and elasticity for BPM or orchestration systems. Schulte et al. [6] did a general review on the topic and gave directions for future research. In this paper, we focus on the resource allocation and scheduling problem and use a tenant-centric approach based on BPM task throughput, instead of the BPM process-centric from other approaches. Rekik et al. [7] propose an integer programming model based on general hardware metrics for BPM elasticity on the cloud. They base their approach on resource allocation and BPM task scheduling. They do not consider multiple time slots, tenant migration or multi-tenancy. Other attempts on BPM elasticity in the cloud exist such as [8–10]. Though not cloud-related, Djedovic et al. propose in [11] a genetic algorithm for BPM task scheduling to their corresponding resources. It uses a representation of each resource. They want to minimize the waiting time and the global resource cost. Junhke et al. [12] propose a task focused genetic algorithm for BPEL workflows scheduling in distributed Clouds.

On other subjects than BPM, the machine reassignment problem[6] is close to our problematic. It considered software reassignment problem on virtual machines including the migration cost. Gavranović et al. [13] obtained the better results to this challenge. However, this problem is based on hardware metrics, and aggregate migration cost in the objective function. Our problem is not exactly virtual machine allocation since the hardware is already defined. Numerous other attempts target virtual machines, such as [14]. Automated approaches based on cloud offers retrieval and hardware requirement for software such as [15] are also valuable.

These works do not consider simultaneously multi-tenancy, multiple instance types, and migrations, except in the form of data transfer cost for [10] or aggregated migration cost for [13]. In this paper, we present an evolution of our previous work [2]. We have based our approach on time series segmentation for deducing the "good" time slots to migrate tenants, and on the iterative use of

[6] http://www.roadef.org/challenge/2012/en/.

an enhanced version of our time slot heuristic [16]. We also presented the corresponding ILP (Integer Linear Programming) model. Results were encouraging compared to a baseline approach, but could be improved regarding the results that we obtain with a solver. We could not compare with other approaches since most of them do not consider migrations of data as an issue. They scale up by adding compute resources to the process engine, considering that access to the database is not a problem. From our experience, at some point, the database is always a bottleneck.

5 Conclusion

In this paper, we proposed a method for cost optimization of BPMaaS deployment based on tenant migration strategies and a genetic algorithm. We presented a new integer programming optimization model. Both allows to obtain substantial gains for BPMaaS providers. The result we obtain when we group the tenants is interesting. It may be explained by the size of the objective space. The fact that it is reproducible for different number of tenants shows that testing multiple sizes may allow providers to save on the operation cost. Moreover, using other metaheuristics such as simulated annealing or hill climbing could provide even better results.

Our method was tested with BPM task throughput but could work with other metrics that can be expressed as a scalar for both the cloud resources and the tenants. We can consider for instance the number of processes, or the number of HTTP requests that lead to transactional processing. Our methods can then be generalized on systems non related to BPMS using multi-tenancy and tenant-related persisted data. A BPMS execution engine behaves more or less like a transactional web application. Our approach is offline and require to anticipate on the tenant load. For many business cases, this is a valid assumption. The server load is relative to the number of employees and the number of cases they can execute everyday or hour with little variations. A next obvious step would be to couple our algorithm with prediction systems. This would provide an effective online algorithm. It could adapt to unforeseen variations.

Acknowledgment. The authors would like to thank Gurobi for the usage of their optimizer, and Amazon Web Services for the EC2 instances credits (this paper is supported by an AWS in Education Research Grant Award).

References

1. Das, S., Nishimura, S., Agrawal, D., El Abbadi, A.: Live database migration for elasticity in a multitenant database for cloud platforms. Technical report, CS, UCSB, Santa Barbara, CA, USA (2010)
2. Rosinosky, G., Youcef, S., Charoy, F.: Efficient migration-aware algorithms for elastic BPMaaS. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 147–163. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_9

3. Lovrić, M., Milanović, M., Stamenković, M.: Algoritmic methods for segmentation of time series: an overview. J. Contemp. Econ. Bus. Issues **1**(1), 31–53 (2014)
4. Whitley, D.: A genetic algorithm tutorial. Stat. Comput. **4**(2), 65–85 (1994)
5. Garrett, A.: Inspyred: Bio-inspired Algorithms in Python - Inspyred 1.0 documentation (2014). http://pythonhosted.org/inspyred/
6. Schulte, S., Janiesch, C., Venugopal, S., Weber, I., Hoenisch, P.: Elastic Business Process Management: state of the art and open challenges for BPM in the cloud. Future Gener. Comput. Syst. **46**, 36–50 (2014)
7. Rekik, M., Boukadi, K., Assy, N., Gaaloul, W., Ben-Abdallah, H.: A linear program for optimal configurable business processes deployment into cloud federation, pp. 34–41. IEEE, June 2016
8. Euting, S., Janiesch, C., Fischer, R., Tai, S., Weber, I.: Scalable business process execution in the cloud. In: 2014 IEEE International Conference on Cloud Engineering, IC2E, pp. 175–184, March 2014
9. Xu, J., Liu, C., Zhao, X., Yongchareon, S., Ding, Z.: Resource management for business process scheduling in the presence of availability constraints. ACM Trans. Manag. Inf. Syst. **7**(3), 1–26 (2016)
10. Hoenisch, P., Hochreiner, C., Schuller, D., Schulte, S., Mendling, J., Dustdar, S.: Cost-efficient scheduling of elastic processes in hybrid clouds, pp. 17–24. IEEE, June 2015
11. Djedović, A., Žunic, E., Avdagić, Z., Karabegović, A.: Optimization of business processes by automatic reallocation of resources using the genetic algorithm. In: XI International Symposium on Telecommunications, BIHTel, pp. 1–7. IEEE (2016)
12. Juhnke, E., Dornemann, T., Bock, D., Freisleben, B.: Multi-objective scheduling of BPEL workflows in geographically distributed clouds, pp. 412–419. IEEE, July 2011
13. Gavranović, H., Buljubašic, M., Demirović, E.: Variable neighborhood search for Google machine reassignment problem. Electron. Not. Discrete Math. **39**, 209–216 (2012)
14. Chen, J., Wang, C., Zhou, B.B., Sun, L., Lee, Y.C., Zomaya, A.Y.: Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In: Proceedings of the 20th International Symposium on High Performance Distributed Computing, HPDC 2011, pp. 229–238. ACM, New York (2011)
15. García, J.M., Martín-Díaz, O., Fernandez, P., Ruiz-Cortés, A., Toro, M.: Automated analysis of cloud offerings for optimal service provisioning. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 331–339. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_23
16. Rosinosky, G., Youcef, S., Charoy, F.: An efficient approach for multi-tenant elastic business processes management in cloud computing environment, pp. 311–318. IEEE, June 2016

Edge + IoT Services

Latency-Aware Placement of Data Stream Analytics on Edge Computing

Alexandre da Silva Veith(✉), Marcos Dias de Assunção, and Laurent Lefèvre

Inria, LIP, ENS Lyon, University of Lyon, 46 Allée d'Italie, 69364 Lyon, France
{alexandre.veith,marcos.dias.de.assuncao,laurent.lefevre}@ens-lyon.fr

Abstract. The interest in processing data events under stringent time constraints as they arrive has led to the emergence of architecture and engines for data stream processing. Edge computing, initially designed to minimize the latency of content delivered to mobile devices, can be used for executing certain stream processing operations. Moving operators from cloud to edge, however, is challenging as operator-placement decisions must consider the application requirements and the network capabilities. In this work, we introduce strategies to create placement configurations for data stream processing applications whose operator topologies follow series parallel graphs. We consider the operator characteristics and requirements to improve the response time of such applications. Results show that our strategies can improve the response time in up to 50% for application graphs comprising multiple forks and joins while transferring less data and better using the resources.

Keywords: Data stream processing · Edge computing
Cloud computing · Resource management · Scheduling
Series parallel graphs

1 Introduction

Today's instruments and services are producing ever-increasing amounts of data that require processing and analysis to provide insights or assist in decision making. Much of this data is received in near real-time and requires quick analysis. In the Internet of Things (IoT) [18], for instance, continuous data streams produced by multiple sources must be handled under very short delays. Under several data stream processing engines, a stream processing application is a directed graph or dataflow whose vertices are operators that execute a function over the incoming data and edges that define how data flows between the operators. A dataflow has one or multiple sources (*i.e.*, sensors, gateways or actuators), operators that perform transformations on the data (*e.g.*, filtering, and aggregation) and sinks (*i.e.*, queries that consume or store the data).

In a traditional cloud deployment, the whole application is placed on the cloud to benefit from virtually unlimited resources. However, processing all the data on the cloud can introduce latency due to data transfer, which makes near real-time processing difficult to achieve. In contrast, *edge computing* has become an attractive solution for performing certain stream processing operations, as many *edge devices* have non-negligible compute capacity.

The deployment of data stream processing applications onto heterogeneous infrastructure, however, has proved to be NP-hard [2]. Moreover, moving operators from cloud to edge devices is challenging due to limitations of edge devices [1]. Existing work often proposes placements strategies considering user intervention [19] whereas many models do not support memory and communication constraints [6,12]. Existing work also considers all data sinks to be located in the cloud, with no feedback loop to actuators located at the edge [5,16]. There is hence a lack of solutions covering scenarios involving smart cities, precision agriculture, and smart homes comprising various heterogeneous sensors and actuators, as well as, time-constraint applications that may contain actuators often placed close to where data is collected.

In this paper, we introduce a set of strategies to place operators onto cloud and edge while considering characteristics of resources and meeting the requirements of applications. We consider analytics applications with multiple sources and sinks distributed across cloud and edge. In particular, we first decompose the application graph by identifying behaviors such as *forks* and *joins*, and then dynamically split the dataflow graph across edge and cloud. Comprehensive simulations considering multiple application settings demonstrate that our approach can improve the response time in up to 50%.

The contributions of this work are: (i) it presents a model for Distributed Stream Processing (DSP) applications in heterogeneous infrastructure (§2); (ii) it introduces placement strategies for dynamically identifying how to split the application graph across cloud and edge (§3); and (iii) it evaluates the strategies against traditional and state-of-the-art schemes (§4).

2 System Model and Problem

This section introduces preliminaries and then describes the placement problem.

2.1 System and Application Models

The network topology is a graph $\mathcal{N} = (\mathcal{R}, \mathcal{L})$ with a vertex set $\mathcal{R} = (r_1, r_2, r_3, \dots)$ of computational resources (*i.e.*, cloud servers and edge devices) and links $\mathcal{L} = (l_1, l_2, l_3, \dots)$ interconnecting the resources. Each $r_i \in \mathcal{R}$ has capabilities in terms of CPU cpu_{r_i} and memory mem_{r_i} expressed respectively in Millions of Instructions per Second (MIPS) and bytes. A network link $i \leftrightarrow j \in \mathcal{L}$ interconnecting resources i and j has bandwidth $bdw_{i \leftrightarrow j}$ and latency $lat_{i \leftrightarrow j}$ represented in bits per second (bps) and seconds respectively. We consider the latency of a resource i to itself (*i.e* $lat_{i \leftrightarrow i}$) to be 0. The network topology is known as we consider scenarios using Software Defined Network solutions [4] or discovery algorithms such as Vivaldi [7] to determine and maintain the topology information.

A DSP application is viewed as a graph $\mathcal{G} = (\mathcal{O}, \mathcal{S})$ whose vertices $\mathcal{O}$ are *operators* that perform operations on the incoming data, and edges $\mathcal{S}$ that are streams of events/messages flowing between the operators. The set of operators $\mathcal{O}$ comprises data *sources* $\mathcal{O}^{src}$, *sinks/outputs* $\mathcal{O}^{out}$ where data is stored or published, and *transformations* $\mathcal{O}^{trn}$ performed over the data.

Each operator $o_i \in \mathcal{O}$ has CPU cpu_{o_i} and memory mem_{o_i} requirements for processing incoming events, given respectively in Instructions per Second (IPS) to handle an individual event and number of bytes to load the operator in memory. The rate at which operator i can process events at reference resource k is denoted by $\mu_{\langle i,k \rangle}$ and is essentially $\mu_{\langle i,k \rangle} = cpu_{r_k} \div cpu_{o_i}$. When performing a transformation on the incoming data, an operator can, for instance, parse data or filter events hence reducing the number of events or their size. The ratio of number of input events to output events is determined by the operator's *selectivity* ψ_{o_i}. The data compression/expansion factor is the ratio of the size of input events to the size of output events, and is represented by ω_{o_i}.

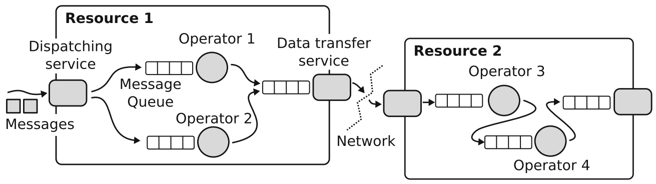


Fig. 1. Example of four operators and their respective queues placed on two resources.

Each event stream $s_{i \to j} \in \mathcal{S}$ connecting operator i to j has a probability $\rho_{i \to j}$ that an output event emitted by i will flow through to j. Here we consider that such information is obtained via profiling techniques or from previous executions of the application. Existing work has demonstrated how such information can be obtained [14]. The rate at which operator i produces events is denoted by λ_i^{out} and is a product of its input event rate λ_i^{in} and its selectivity. The output event rate of a source operator $k \in \mathcal{O}^{src}$ depends on the number of measurements it takes from a sensor or another monitored device. We can then recursively compute the input and output event rates for downstream operators j as follows:

$$\lambda_i^{in} = \lambda_k^{out} \qquad \forall s_{k \to i} \in \mathcal{S}, k \in \mathcal{O}^{src} \tag{1}$$

$$\lambda_j^{in} = \sum_{s_{i \to j} \in \mathcal{S}} \lambda_i^{in} \times \psi_{o_i} \times \rho_{s_{i \to j}} \qquad \forall i \in \mathcal{O}, i \notin \mathcal{O}^{src} \tag{2}$$

$$\lambda_j^{out} = \lambda_j^{in} \times \psi_{o_j} \qquad \forall j \in \mathcal{O}, j \notin \mathcal{O}^{out} \tag{3}$$

Likewise, we can recursively compute the average size ς_i^{in} of events that arrive at a downstream operator i and the size of events it emits ς_i^{out} by considering the upstream operators' event sizes and their respective compression/expansion factors (*i.e.*, ω_{o_i}). In other words:

$$\varsigma_i^{in} = \varsigma_k^{out} \qquad \forall s_{k \to i} \in \mathcal{S}, k \in \mathcal{O}^{src} \tag{4}$$

$$\varsigma_j^{in} = \varsigma_i^{in} \times \omega_{o_i} \qquad \forall i \in \mathcal{O}, i \notin \mathcal{O}^{src} \tag{5}$$

$$\varsigma_j^{out} = \varsigma_j^{in} \times \omega_{o_j} \qquad \forall j \in \mathcal{O}, j \notin \mathcal{O}^{out} \tag{6}$$

When placed onto available resources, operators within a same host communicate directly whereas inter-node communication is done via a communication service as depicted in Fig. 1. If more events arrive than an operator can handle when placed at a given resource, queues will form and the overall service time will increase. Events are handled in a First-Served (FCFS) fashion both by operators and the computation service that serialises messages to be sent to another host. This guarantees the time order of events; an important requirement in many data stream processing applications. We model both operators and the communication service as M/M/1 queues which allows for estimating the waiting and service times for computation and communication. The computation or service time $stime_{\langle o_i, r_k \rangle}$ of an operator i placed on a resource k is hence given by:

$$stime_{\langle i,k \rangle} = \frac{1}{\mu_{\langle i,k \rangle} - \lambda_i^{in}} \tag{7}$$

while the communication time $ctime_{\langle i,k \rangle \langle j,l \rangle}$ for operator i placed on a resource k to send a message to operator j on a resource l is:

$$ctime_{\langle i,k \rangle \langle j,l \rangle} = \frac{1}{\left(\frac{bdw_{k \leftrightarrow l}}{\varsigma_i^{out}} \right) - \lambda_j^{in}} + l_{k \leftrightarrow l} \tag{8}$$

Furthermore, the number of events waiting to be served, being processed, or waiting to be transferred to another resource, enable us to compute the memory requirements of operators at the resources onto which they are placed. The number of events in service at an operator i at resource k is given by:

$$\varphi_{\langle i,k \rangle}^{comp} = \frac{\frac{\lambda_i^{in}}{\mu_{\langle i,k \rangle}}}{1 - \frac{\lambda_i^{in}}{\mu_{\langle i,k \rangle}}} \tag{9}$$

while the number of events waiting in the communication service to be transferred from operator i on resource k to operator j placed on resource l is:

$$\varphi_{\langle i,k \rangle \langle j,l \rangle}^{comm} = \frac{\frac{\lambda_j^{in}}{\left(\frac{bdw_{k \leftrightarrow l}}{\varsigma_i^{out}} \right)}}{1 - \frac{\lambda_j^{in}}{\left(\frac{bdw_{k \leftrightarrow l}}{\varsigma_i^{out}} \right)}} \tag{10}$$

The overall memory required by an operator i allocated on a resource k comprises the memory needed to load it as well as the memory required by in-service events, and events waiting to be serviced or waiting to be transferred to another resource:

$$mem_{\langle i,k\rangle} = \varphi^{comp}_{\langle i,k\rangle} \times \varsigma^{in}_i + mem_{o_i} + \sum_{\substack{j\in\mathcal{O}\\ l\in\mathcal{R}}} \varphi^{comm}_{\langle i,k\rangle\langle j,l\rangle} \times \varsigma^{in}_i \tag{11}$$

A mapping function $\mathcal{M} : \mathcal{O} \rightarrow \mathcal{R}$, $\mathcal{S} \rightarrow \mathcal{L}$ indicates the resource to which an operator is assigned and the link to which a stream is mapped. The function $mo_{\langle i,k\rangle}$ returns 1 if operator i is placed at resource k and 0 otherwise. Likewise, the function $ms_{\langle i\rightarrow j,k\leftrightarrow l\rangle}$ returns 1 when the stream between operators i and j has been assigned to the link between resources k and l, and 0 otherwise. A *path* in the DSP application graph is a sequence of operators from a source to a sink. A path p_i of length n is a sequence of n operators and $n-1$ streams, starting at a source and ending at a sink:

$$p_i = o_0, o_1, \ldots, o_k, o_{k+1}, \ldots, o_{n-1}, o_n \tag{12}$$

Where $o_0 \in \mathcal{O}^{src}$ and $o_n \in \mathcal{O}^{out}$. The set of all possible paths in the application graph is denoted by $\mathcal{P}$. The end-to-end latency of a path comprises the sum of the computation time of all operators along the path and the communication time required to stream events on the path. More formally, the end-to-end latency of path p_i, denoted by L_{p_i}, is:

$$L_{p_i} = \sum_{\substack{o\in p_i\\ r\in\mathcal{R}}} mo_{\langle o,r\rangle} \times stime_{\langle o,r\rangle} + \sum_{r'\in\mathcal{R}} ms_{\langle o\rightarrow o+1,r\leftrightarrow r'\rangle} \times ctime_{\langle o,r\rangle\langle o+1,r'\rangle} \tag{13}$$

2.2 Operator Placement Problem

The problem of placing a distributed stream processing application consists of finding a mapping that minimises the aggregate end-to-end latency of all application paths and that respects the resource and network constraints. In other words, find the mapping that minimises the aggregate end-to-end event latency:

$$\min \sum_{p_i\in\mathcal{P}} L_{p_i} \tag{14}$$

Subject to:

$$\lambda^{in}_o < \mu_{\langle o,r\rangle} \qquad \forall o \in \mathcal{O}, \forall r \in \mathcal{R} | mo_{\langle o,r\rangle} = 1 \tag{15}$$

$$\lambda^{in}_o < \left(\frac{bdw_{k\leftrightarrow n}}{\varsigma^{out}_{o-1}}\right) \qquad \forall o \in \mathcal{O}, \forall k \leftrightarrow n \in \mathcal{L} | mo_{\langle o,k\rangle} = 1 \tag{16}$$

$$\sum_{o\in\mathcal{O}} mo_{\langle o,r\rangle} \lambda^{in}_o \leq c_r \qquad \forall r \in \mathcal{R} \tag{17}$$

$$\sum_{o \in \mathcal{O}} mo_{\langle o,r \rangle} \times mem_{\langle o,r \rangle} \leq mem_r \qquad \forall r \in \mathcal{R} \tag{18}$$

$$\sum_{\substack{s_{i \to j} \in \mathcal{S} \\ k \leftrightarrow l \in \mathcal{L}}} ms_{\langle i \to j, k \leftrightarrow l \rangle} \times \varsigma_i^{out} \leq bwd_{k \leftrightarrow l} \qquad \forall k \leftrightarrow l \in \mathcal{L} \tag{19}$$

$$\sum_{r \in \mathcal{R}} mo_{\langle o,r \rangle} = 1 \qquad \forall o \in \mathcal{O} \tag{20}$$

$$\sum_{k \leftrightarrow l \in \mathcal{L}} ms_{\langle i \to j, k \leftrightarrow l \rangle} = 1 \qquad \forall s_{i \to j} \in \mathcal{S} \tag{21}$$

Constraint 15 guarantees that a resource can provide the service rate required by its hosted operators whereas Constraint 16 ensures that the links are not saturated. The CPU and memory requirements of operators on each host are ensured by Constraints 17 and 18 respectively. Constraint 19 guarantees the data requirements of streams placed on links. Constraints 20 and 21 ensure that an operator is not placed on more than a resource and that a stream is not placed on more than a network link respectively.

3 Application Placement Strategies

This section explains how patterns in the DSP application graphs are identified and then introduces strategies that employ the patterns to devise placement decisions.

3.1 Finding Application Patterns

As depicted in Fig. 2, a dataflow can comprise multiple patterns such as (i) forks, where messages can be replicated to multiple downstream operators or scheduled to downstream operators in a round-robin fashion, using message key hashes, or considering other criteria [16]; (ii) parallel regions that perform the same operations over different sets of messages or where each individual region executes a given set of operations over replicas of the incoming messages; and (iii) joins, which merge the outcome of parallel regions.

We consider Series-Parallel-Decomposable Graphs (SPDG) and related techniques to identify graph regions that present these patterns [8]. This information is used to build a hierarchy of region dependencies (*i.e.* downstream and upstream relations between regions) and assist on placing operators across cloud and edge resources. The streams in the graph paths that separate the operators are hereafter called the *split points*. Figure 2 illustrates the phases of the method to determine the split points (green circles), where red circles represent operators placed on edge resources whereas blue ones are on the cloud: (i) The method starts with sources and sinks whose placements are predefined by the user; (ii) split points are discovered (green circles) as well as sinks that correspond to actuators that can be placed on the edge; (iii) the branches between the existing

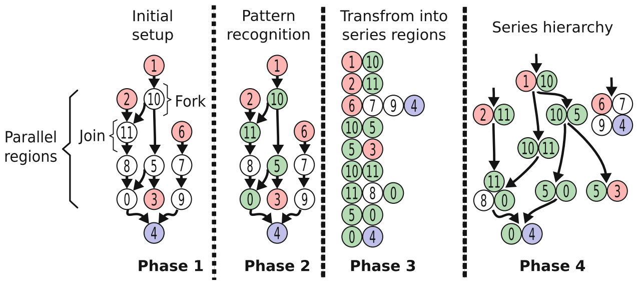


Fig. 2. Method for finding the dataflow split points, where red means placed on edge, blue represents placed on cloud, and green delimits forks and joins. (Color figure online)

patterns (green, red and blue circles) are transformed into series regions; and (iv) a hierarchy following the dependencies between regions is created.

Algorithm 1 describes the function *GetRegions* used to identify the patterns and obtain the series regions. First, the function adds two virtual vertices to the graph, one named *virt_src* connected to all data sources and another named *virt_sink* to which all sinks are connected (line 2–4). The virtual vertices allow for recognizing all paths between sources and sinks. Second, each path is iterated moving operators to a temporary vector and classifying the operators as upstream and downstream according to the number of input and output edges (lines 5–8). If the operator is a split point, the temporary vector is converted to a subset of regions set, and the temporary vector receives the current operator (lines 9–10). Third, the function removes the redundant values (line 11). At last, the region set is iterated comparing the regions by the first and the last position values (equal values represent a connection) and consequently, they are stored in the hierarchy set (lines 12–16).

3.2 Operator Placement Strategies

The region hierarchy allows us to determine the data paths and operator deployment priorities, based on which two strategies are applied: Response Time Rate (RTR) that iterates the deployment sequence and for each operator estimates the *end-to-end latency* (*i.e.* response time); and Response Time Rate with Region Patterns (RTR+RP) which uses the hierarchy to split the application graph across edge and cloud, optimizing only the response time on the edge.

Response Time Rate (RTR) is a greedy strategy that places operators incrementally by evaluating the end-to-end latency of paths (Eq. 13) while respecting the resource constraints (Eqs. 15–21). The response time of an operator in a path comprises the time taken to transfer data and to compute an event. As an operator can be in multiple paths, the RTR strategy accumulates the time taken to transfer data from multiple paths rather than evaluating each path individually.

RTR organizes the operators into deployment sequence and consecutively calculates the response time for each operator by considering the previous map-

Algorithm 1. Algorithm to detect forks and joins.

```
1  Function GetRegions($\mathcal{G} = (\mathcal{O}, \mathcal{S}), \mathcal{O}^{src}, \mathcal{O}^{out}$)
2      $\mathcal{O} \leftarrow \mathcal{O} \cup virt\_src \cup virt\_sink$
3      $\mathcal{S} \leftarrow \mathcal{S} \cup s_{virt\_src \rightarrow o}, \forall o \in \mathcal{O}^{src}$
4      $\mathcal{S} \leftarrow \mathcal{S} \cup s_{o \rightarrow virt\_sink}, \forall o \in \mathcal{O}^{out}$
5      for $p \in$ GetAllPaths($\mathcal{G}, virt\_src, virt\_sink$) do
6          for $o \in p$ do
7              $temp \leftarrow temp \cup \{o\}, \forall o \notin \{virt\_src, virt\_sink\}$
8              $ups \leftarrow |\langle *, o \rangle \subset \mathcal{S}|, downs \leftarrow |\langle o, * \rangle \subset \mathcal{S}|$
9              if $ups > 1$ or $downs > 1$ and $o \notin \{virt\_src, virt\_sink\}$ then
10                 $regions \leftarrow regions \cup temp$, $temp \leftarrow \{o\}$
11     Delete duplicate $regions$
12     for $src\_series \in regions$ do
13         for $dst\_series \in regions$ do
14             if $src\_series \neq dst\_series$ then
15                 if $src\_series[|src\_series| - 1] = dst\_series[0]$ then
16                     $hierarchy \leftarrow hierarchy \cup \{src\_series, dst\_series\}$
17     return $hierarchy$
```

Algorithm 2. Calculating the computational response times.

```
1  Function EstimateResponseTimes($\mathcal{N} = (\mathcal{R}, \mathcal{L}), \mathcal{G} = (\mathcal{O}, \mathcal{S}), o$)
2      for $child \in \langle o, * \rangle \subset \mathcal{S}$ do
3          $upstreams \leftarrow \langle child, r \rangle, \forall r \in \mathcal{R}$ and $mo_{\langle child, r \rangle} = 1$
4      for $r \in \mathcal{R}$ do
5          $comm \leftarrow 0$
6          for $mapping \in upstreams$ do
7              if GetHost($mapping$) $\neq r$ then
8                  $com \leftarrow comm + ctime_{\langle mapping \rangle \langle o, r \rangle}$
9          if MeetConstraints then
10             $rt \leftarrow rt \cup \langle r, stime_{\langle o, r \rangle} + comm \rangle$
11     return $rt$
```

pings, resource capabilities, and operator requirements. The approach initially obtains the region hierarchy and then establishes the deployment sequence employing a breadth-first search traversal algorithm [17] to give priority to upstream operators. Each operator of the deployment sequence has its response time estimated for non-constrained computational resources (Algorithm 2). After that, the resources are sorted in ascending manner by their response times and the host with the shortest response time is picked, and the host's residual capabilities are updated.

Response Time Rate with Region Patterns (RTR+RP) is a strategy that handles complex dataflows that contain multiple paths from sources to sinks. It explores the operator patterns (split points) and the sink placement (cloud or edge) respecting the environment constraints (Eqs. 15–21). Based on the region hierarchy (Fig. 3), the operators are classified and allocated. Operator 5, for instance, was reallocated since the edge does not respect the resource constraints. RTR+RP aims to allocate operators across edge and cloud meeting the response time rate only for operators located in the edge, in contrast to the RTR strategy that evaluates the response time rate for all operators.

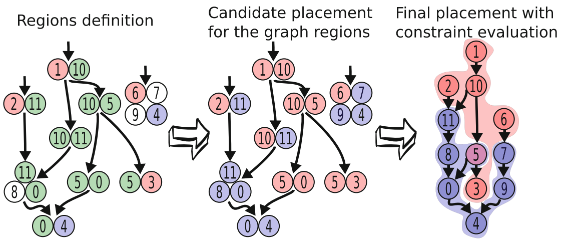

Regions definition
Candidate placement for the graph regions
Final placement with constraint evaluation

Fig. 3. Blue circles are operator candidates to be deployed on cloud whereas red circles are candidates for edge. The right-hand graph shows the final deployment. (Color figure online)

RTR+RP defines the deployment sequence similar to RTR, but it builds upon the classification of the operators considering the served sink infrastructure (candidate infrastructure). The classification is (i) *cloud-only* if the operator serves only sinks placed on the cloud, and (ii) *edge* if the operator shares paths with sinks located on edge. Each operator on the deployment sequence has its candidate infrastructure evaluated. Edge candidates have their response time estimated for non-constrained edge devices where the device with the shortest response time is picked. On the other hand, cloud candidates do not have their response time estimated, hence, the cloud hosts its operator candidates and those that do not meet the constraints on edge. At last, after the operator mapping, the resources have their residual capabilities updated.

4 Evaluation

In this section, we first describe the experimental setup and performance metrics and then discuss experimental results.

4.1 Experimental Setup

We built a framework atop OMNET++[1] to model and simulate distributed stream processing applications. A computational resource is an entity with CPU,

[1] Visit http://www.omnetpp.org/ for further details on OMNET++.

memory and bandwidth capabilities whereas operators comprise waiting queues and transformation operations that pose demands in terms of CPU, memory and bandwidth.

We model our edge devices as Raspberry PI's 2 (RPi) (*i.e.*, 4,74 MIPS[2] at 1 GHz and 1 GB of RAM), and the cloud as AMD RYZEN 7 1800x (*i.e.*, 304,51 MIPS[3] at 3.6 GHz and 1 TB of memory). The infrastructure comprises two cloudlets [13] with edge computing nodes (*Cloudlet 1* and *Cloudlet 2*) and a *Cloud*. Each cloudlet has 20 RPi's, whereas the cloud consists of 2 servers. A gateway interfaces each cloudlet's LAN and the external WAN [11] (the Internet). The LAN has a latency drawn from a uniform distribution between 0.015 and 0.8 ms and a bandwidth of 100Mbps. The WAN has latency drawn uniformly between 65 and 85 ms, and bandwidth of 1 Gbps [13].

As stream processing applications exist in multiple domains with diverse topologies (*e.g.*, face recognition, speech recognition, weather sensing), where sensors/actuators ingest a variety of events (*e.g.*, text, video, pictures, voice record) in the system, we aim to capture this diversity my modeling and simulating two scenarios with various application workloads:

Microbenchmarks: As in previous work [13], we first perform a controlled evaluation using a set of message sizes (10 bytes, 50 KB, and 200 KB) corresponding to multiple data types such as text, pictures/objects, and voice records. Each application, depicted in Fig. 4, has three input event rates (Table 1), a set of CPU requirements according to the message sizes (10 bytes - 3.7952 IPS, 50 KB -18976 IPS, and 200 KB - 75904 IPS) and a configuration of fork/join operators to explore the path sizes. The operators have multiple selectivity and data compression rates (100, 75, 50 and 25%). Sources ingest messages from sensors and sinks act as actuators on cloudlets and databases/message brokers on cloud.

Table 1. Input event rate.

App	10 bytes	50 KB	200 KB
App1	124999, 624999, 1249999	24, 124, 249	6, 31, 62
App2	124999, 374999, 624999	24, 74, 124	6, 19, 31
App3	124999, 218749, 300000	24, 43, 62	6, 10, 15
App4	124999, 137499, 150000	24, 27, 30	6, 7

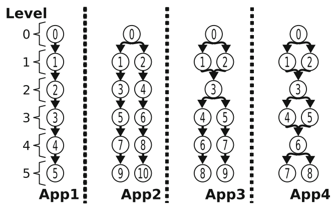


Fig. 4. Six-hop applications.

More Complex Applications: This scenario presents multiple operator behaviors and larger numbers of operators. We crafted a set of application graphs (Fig. 5) using a Python library[4] and varying the parameters of the operators using a uniform distribution with the ranges presented in Table 2. The cloudlets

[2] https://hackaday.com/2015/02/05/benchmarking-the-raspberry-pi-2/.
[3] https://reddit.com/r/BOINC/comments/5xog5v/boinc_performance_on_amd_ryzen.
[4] https://gist.github.com/bwbaugh/4602818.

Table 2. Operator attributes.

Parameter	Value
cpu	1–100
Data compression rate	10%–100%
mem	100–7500
Input event size	100–2500
Selectivity	10%–100%
Input event rate	1000–10000

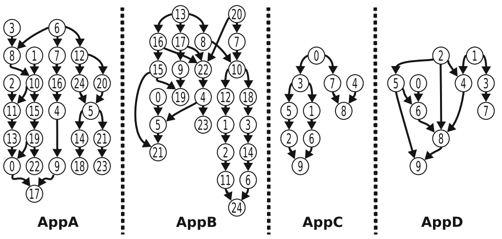


Fig. 5. Complex applications.

host the sink and source placements, except for the sink on the critical path which will be hosted on the cloud. We generated 1160 graphs randomly applying multiple selectivities, data compression rates, sink and source locations, input event sizes and rates, memory, and CPU requirements. Inspired on the size of RIoT-Bench [20] applications, a Realtime IoT Benchmark suite, we proposed two sets of applications, namely: (i) **large** (AppA and AppB) containing 25 operators; and (ii) **small** (AppC and AppD) holding 10 operators.

Metrics: The main performance metric is the *application response time*, which is the end-to-end latency from the time events are generated to the time they are processed by the sinks. To demonstrate the gains obtained by our approach, we compared the proposed strategies against a traditional approach (cloud-only) and a solution from the state-of-the-art [21] (LB). *Cloud-only* deploys all operators in the cloud, apart from operators provided in the initial placement. *Taneja et. al. (LB)* iterates a vector containing the application operators, gets the middle host of the computational vector and evaluates CPU, memory, and bandwidth constraints to obtain the operator placement.

4.2 Performance Evaluation

Figure 6 summarizes the response times for all microbenchmarks. For App1 we carried out 432 experiments (4 selectivities, 4 data compression rates, 3 input event rates, 3 sink locations and 3 input event sizes) for each solution with a pipeline application that may have messages with text, video, pictures, and voice record. Each experiment ran for 300 s in simulation time. RTR and RTR+RP have shown to be over 95% more efficient than cloud-only approach and LB. Initially, LB had its performance comparable to cloud-only, but LB lost performance afterward due to its specific modeling (*i.e.*, health care, and latency-critical gaming) and method (computational ordering).

Cloud-only achieved 5% better results (when the blue line crosses the red at ≈200 ms) when handling voice records (200 KB), selectivity, and data compression rate equal to 1 (without reducing the size of the messages and discarding events) and when the sink was placed on Cloudlet 2 and the source was located on Cloudlet 1 (traverse WAN). For the scenario mentioned before, the operators were CPU-intensive where Cloudlet 1 or 2 can host only one operator per edge device at a time, which increases the communication costs. Moreover, RTR+RP

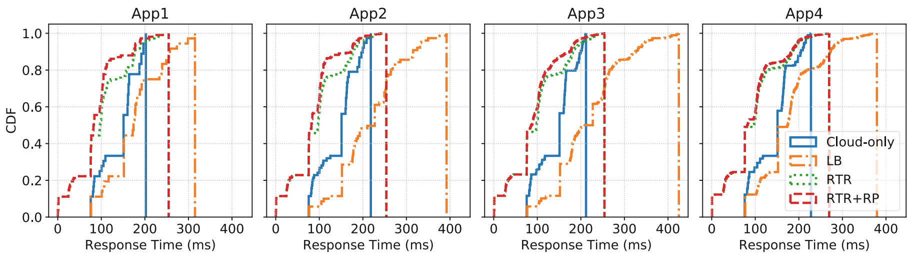


Fig. 6. CDF of response time for microbenchmarks. (Color figure online)

outperformed RTR for sinks placed on cloud, mainly without message discarding and no reduction on message sizes. Even further, to investigate the impacts generated by the split points, we launched App2, App3, and App4 and observed a gradual performance loss (decreasing on the distance between green and red line - ≈100 ms) according to the position between the split points and sinks, and the location of sinks. When sinks and sources require events to traverse the WAN and there is a low number of hops between the split point and sink, the proposed strategies cannot define a reasonable dataflow split because of the assumption to prioritize the sinks on the edge.

The complex application scenario investigates the outcomes for generic and multiple path applications using various dataflow configurations. We launched each experiment during 60 s of simulation time, and the sources and sinks were distributed uniformly and randomly across the infrastructure, except for operator 17 on AppA, operator 24 on AppB, operator 9 on AppC, and operator 9 on AppD placed in cloud due to the critical path. Figure 7 shows the CDF of response times. Even under large applications RTR+RP was able to reduce the response time by applying the region pattern identifications and recursively discovering the operator dependencies with a given sink placement onto dataflows with various paths.

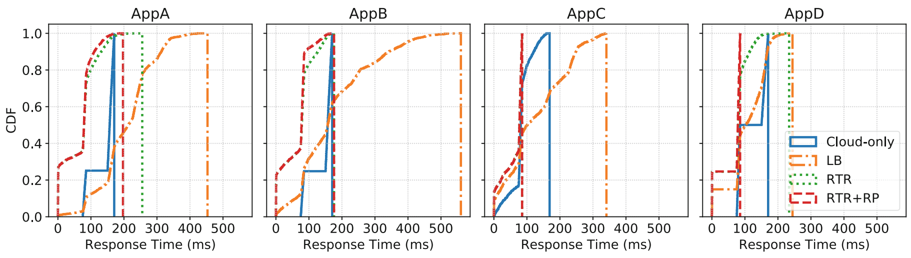


Fig. 7. CDF of response times for complex applications.

Our strategies outperformed cloud-only in over 6% and 50% under small and large applications, respectively. Cloud-only poses high communication overhead when the sink is located on cloudlets due to messages having to traverse the internet at least twice. Similarly, we improve the response times in over

23% (small) and 57% (large applications) compared to the LB approach. This occurs because LB does not estimate the communication overhead and assumes a shorter response time on cloudlets.

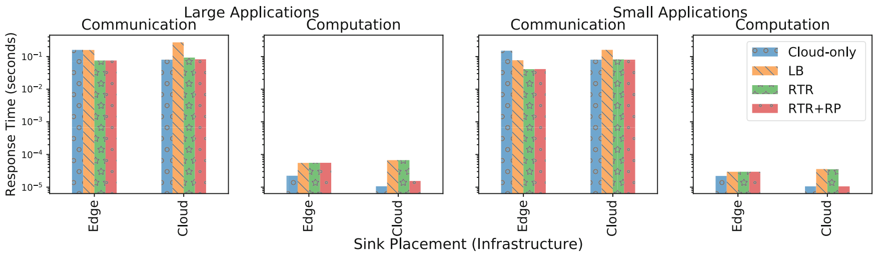


Fig. 8. Communication and computation time for sinks placed on cloud and cloudlets.

Figure 8 shows the communication latency which comprehends the total time to transfer a message between the resources, and the computation that corresponds to the total time to compute all operators. The communication cost for sinks placed on cloudlets at cloud-only was about 160 ms, and RTR+RP was 76 ms. Our solution outperformed cloud-only in up to 52% by putting operators closer to cloudlet sinks, but sinks on the cloud. RTR+RP had a slight performance loss of 3%. Hence, our approach is effective in reducing the communication cost, and, by doing so, it compensates the edge limitations and reaches good results in minimizing the total response time.

5 Related Work

IoT services are increasingly being employed on environments that span multiple data centers or on the edges of the Internet (*i.e.*, edge and fog computing). Existing work proposes architecture that places certain stream processing elements on micro data centers located closer to where the data is generated [3] or employs mobile devices for stream processing [9,15]. The problem of placing DSP applications onto heterogeneous hardware is at least NP-Hard as shown by Benoit *et al.* [2]. To simplify the placement problem, communication is often neglected [6], although it is a relevant cost in geo-distributed infrastructure [13]. Likewise, the operator behavior and requirements are oversimplified using static splitting decisions as proposed by Sajjad *et al.* [19].

Meanwhile, many efforts have been made on modeling the placement problem of DSP application on heterogeneous infrastructure [16] using Petri nets to formalize the application regions and the multiple response times that they produce. On the other hand, Eidenbenz *et al.* [8] evaluated SPDG from its parallelism degree to decompose the application graph and by an approximation algorithm to determine the placement. Cloud and edge have been explored to supply application requirements. For instance, Ghosh *et al.* [10] proposed a model evaluating dynamically the time taken to process an operator into an exclusive computational resource as well as the communication times. Similarly, Taneja et. al. [21] offer a naive approach deploying the application graph across cloud and edge using a constraint sensitive approach.

All contributions to the problem of placing DSP applications evaluate edge devices for improving the network efficiency by treating the infrastructure and application constraints to unleash the full potential of applications triggered by IoT and smart scenarios. The present work considers edge and cloud, and the restrictions created by their interactions. The target scenario includes real-time analytics which comprises multiple forks and joins building multiple paths between data sources and sinks. Our solution exploits the construction of the paths to optimize the end-to-end application latency by decomposing the dataflow and defining the operator's placement dynamically.

6 Conclusions and Future Work

In this paper, we modeled the problem of placing DSP applications onto heterogeneous computational and network resources. We proposed two strategies to minimize the application response time by splitting the application graph dynamically and distributing the operators across cloud and edge. Our solutions were evaluated considering key aspects to identify application behaviors. The RTR strategy estimates the response time for each operator in all computational resources while RTR+RP strategy splits the dataflow graph using region patterns and then calculates the response time only for operators that are candidates to be deployed on edge.

We simulated the strategies' behavior and compared them against the state-of-the-art. The results show that our strategies are capable of achieving 50% better response time than cloud-only deployment when applications have multiple forks and joins. For future work, we intend to investigate further techniques to deal with CPU-intensive operators and their energy consumption.

Acknowledgements. This work was performed within the framework of the LABEX MILYON (ANR-10-LABX-0070) of the University of Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007).

References

1. de Assunção, M.D., da Silva Veith, A., Buyya, R.: Distributed data stream processing and edge computing: a survey on resource elasticity and future directions. J. Netw. Comput. Appl. **103**, 1–17 (2018)
2. Benoit, A., Dobrila, A., Nicod, J.M., Philippe, L.: Scheduling linear chain streaming applications on heterogeneous systems with failures. Future Gener. Comput. Syst. **29**(5), 1140–1151 (2013)
3. Buddhika, T., Pallickara, S.: Neptune: real time stream processing for internet of things and sensing environments. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1143–1152, May 2016

4. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: 10th ACM International Conference on Distributed Event-Based Systems, pp. 69–80. ACM, New York (2016)
5. Cardellini, V., Grassi, V., Presti, F.L., Nardelli, M.: Distributed QoS-aware scheduling in Storm. In: 9th ACM International Conference on Distributed Event-Based Systems, DEBS 2015, pp. 344–347. ACM, New York (2015)
6. Cheng, B., Papageorgiou, A., Bauer, M.: Geelytics: enabling on-demand edge analytics over scoped data sources. In: IEEE International Congress on BigData, pp. 101–108 (2016)
7. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: a decentralized network coordinate system. SIGCOMM Comput. Commun. Rev. **34**(4), 15–26 (2004)
8. Eidenbenz, R., Locher, T.: Task allocation for distributed stream processing. In: IEEE INFOCOM 2016, pp. 1–9, April 2016
9. Elbamby, M.S., Bennis, M., Saad, W.: Proactive edge computing in latency-constrained fog networks. In: European Conference on Networks and Communications, pp. 1–6, June 2017. https://doi.org/10.1109/EuCNC.2017.7980678
10. Ghosh, R., Simmhan, Y.: Distributed scheduling of event analytics across edge and cloud. ACM Trans. Cyber-Phys. Syst. **2**(4), 24 (2017, to Appear)
11. Ha, K., et al.: The impact of mobile multimedia applications on data center consolidation. In: IEEE International Conference on Cloud Engineering (IC2E), pp. 166–176, March 2013
12. Hochreiner, C., Vogler, M., Waibel, P., Dustdar, S.: VISP: an ecosystem for elastic data stream processing for the internet of things. In: 20th IEEE International Enterprise Distributed Object Computing Conference, pp. 1–11, September 2016
13. Hu, W., et al.: Quantifying the impact of edge computing on mobile applications. In: 7th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys 2016, pp. 5:1–5:8. ACM, New York (2016)
14. Kaur, N., Sood, S.K.: Efficient resource management system based on 4VS of big data streams. Big Data Res. **9**, 98–106 (2017)
15. Morales, J., Rosas, E., Hidalgo, N.: Symbiosis: sharing mobile resources for stream processing. In: IEEE Symposium on Computers and Communications Workshop, pp. 1–6, June 2014
16. Ni, L., Zhang, J., Jiang, C., Yan, C., Yu, K.: Resource allocation strategy in fog computing based on priced timed petri nets. IEEE IoT J. **4**(5), 1216–1228 (2017)
17. Peng, B., Hosseini, M., Hong, Z., Farivar, R., Campbell, R.: R-storm: resource-aware scheduling in storm. In: 16th Annual Middleware Conference, Middleware 2015, pp. 149–161. ACM, New York (2015)
18. Ravindra, P., Khochare, A., Reddy, S.P., Sharma, S., Varshney, P., Simmhan, Y.: Echo: an adaptive orchestration platform for hybrid dataflows across cloud and edge. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-69035-3_28
19. Sajjad, H.P., Danniswara, K., Al-Shishtawy, A., Vlassov, V.: Spanedge: towards unifying stream processing over central and near-the-edge data centers. In: 2016 IEEE/ACM Symposium on Edge Computing, pp. 168–178, October 2016
20. Shukla, A., Chaturvedi, S., Simmhan, Y.: Riotbench: an IoT benchmark for distributed stream processing systems. Concurr. Comput.: Pract. Exp. **29**(21), e4257 (2017)
21. Taneja, M., Davy, A.: Resource aware placement of IoT application modules in fog-cloud computing paradigm. In: IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228, May 2017

Optimal Edge User Allocation in Edge Computing with Variable Sized Vector Bin Packing

Phu Lai[1], Qiang He[1](✉), Mohamed Abdelrazek[2], Feifei Chen[2], John Hosking[4], John Grundy[3], and Yun Yang[1]

[1] Swinburne University of Technology, Hawthorn, Australia
{tlai,qhe,yyang}@swin.edu.au
[2] Deakin University, Burwood, Australia
{mohamed.abdelrazek,feifei.chen}@deakin.edu.au
[3] Monash University, Clayton, Australia
john.grundy@monash.edu
[4] The University of Auckland, Auckland, New Zealand
j.hosking@auckland.ac.nz

Abstract. In mobile edge computing, edge servers are geographically distributed around base stations placed near end-users to provide highly accessible and efficient computing capacities and services. In the mobile edge computing environment, a service provider can deploy its service on hired edge servers to reduce end-to-end service delays experienced by its end-users allocated to those edge servers. An optimal deployment must maximize the number of allocated end-users and minimize the number of hired edge servers while ensuring the required quality of service for end-users. In this paper, we model the edge user allocation (EUA) problem as a bin packing problem, and introduce a novel, optimal approach to solving the EUA problem based on the Lexicographic Goal Programming technique. We have conducted three series of experiments to evaluate the proposed approach against two representative baseline approaches. Experimental results show that our approach significantly outperforms the other two approaches.

Keywords: Optimization · Resource management · Edge computing Bin packing

1 Introduction

In recent years, the world has witnessed a surge in the number of cloud and mobile network connected end-devices, including mobile phones, wearables, sensors and a wide range of Internet of Things (IoT) devices. According to Ericsson's Mobility Report [4], it is predicted that there will be around 32 billion of such connected devices by 2023. This has produced a great challenge for online service providers in terms of guaranteeing a reliable and low-latency connection to end-users, which is one of the key quality-of-service (QoS) requirements [12].

To tackle this issue, Cisco [1] has proposed the fog computing paradigm – also called *edge computing* – in which computation, storage, and networking resources are pushed closer to the edge of the network by deploying a number of intermediate *edge servers* with closer proximity to *end-devices*. This paradigm offers lower network latency and greater scalability than the conventional centralized cloud computing paradigm. This is particularly important for high volume streaming applications or critical systems such as autonomous traffic systems, health care, or cloud gaming, which require real-time decision making. In edge computing, online service providers hire existing edge servers to host their services to serve their end-users. Thin clients – such as wearables, sensors or smart phones – all that have limited storage and computing capability, benefit from this architecture by the capability to offload intensive computing tasks to the distributed edge servers near them [17]. In this way, the central cloud is not required to perform all the computing tasks single-handedly, which is highly resource demanding and generally incurs long network latency for end-users. Usually, an edge server covers a specific geographical area so that the users within its coverage can connect to it via LTE, 4G or Radio Network [5]. A number of edge servers would be deployed in a distributed fashion (usually near cellular base stations [5]) so that they can cover different geographical areas. The coverages of adjacent edge servers usually partially overlap to avoid blank areas not covered by any edge server. A user located in the overlapping area can connect to one of the edge servers covering them (*proximity constraint*) that has sufficient computing resource (*capacity constraint*) such as CPU, bandwidth, or memory.

Edge servers' capacity, current workloads, coverages, the number of users to allocate and their proximity to end-users can be obtained or calculated at any time. Based on this information, while fulfilling the above constraints, an optimization goal must be achieved from a service provider's perspective – to minimize the number of edge servers used – in order to attain an optimal solution to the allocation of the service provider's users due to the pay-as-you-go pricing model applied in edge computing [12,17], which might incur higher costs when the number of edge servers used increases. Additionally, due to the aforementioned constraints, there might be a number of users that cannot be assigned to any edge servers. Those users will be connected directly to a central cloud server. Therefore, another optimization objective is to maximize the number of users allocated to hired edge servers.

We refer to the above problem as an *edge user allocation* (EUA) problem then model it as a *variable sized vector bin packing* (VSVBP) problem, a non-geometric generalization of the classical BIN PACKING (BP) problem. The EUA problem is critical in edge computing, however, has not been properly investigated. Solutions to the task allocation problem in cloud computing have been investigated in [8,14]. However, the edge computing architecture is different from cloud computing, i.e., distributed vs. centralized. In addition, the various constraints and dynamic information discussed above significantly differentiate the edge computing environment from the traditional cloud computing environment with many unique characteristics. Thus, the approaches for task allocation in cloud computing are not suitable for solving the EUA problem, hence the need for a new approach. In this paper, we make the following major contributions:

- we have modeled and formulated the EUA problem as a VSVBP problem;
- we have developed an optimal approach for solving the EUA problem using the Lexicographic Goal Programming technique; and
- we have evaluated our approach against two representative baseline approaches with extensive experiments to demonstrate its effectiveness.

The remainder of the paper is organized as follows. Section 2 motivates this research with an example. In Sect. 3, we give a background of the VSVBP problem. Section 4 discusses the proposed approach, which is evaluated in Sect. 5. Section 6 reviews the related work. Section 7 concludes this paper.

2 Motivating Example

A representative example of edge computing applications is large-scale mobile gaming [6] - the fastest growing gaming model [10]. The cloud gaming model has made online game platforms, such as Hatch[1] and Sony PlayStation Now[2], more accessible for thin-client mobile players since the resource-expensive game instance is running on a remote cloud server. Consider an increasingly popular virtual reality game G, which requires a great amount of computing power for graphic rendering. Employing the traditional centralized cloud model helps thin-clients offload the heavy computation tasks; however, this approach introduces a huge network delay due to the long distance between players and cloud servers. Therefore, pushing the processing power closer to players with edge computing is a promising solution to this problem. Figure 1 shows an example of edge computing architecture that can be implemented in this scenario.

Assume there are four edge servers in a specific area that can be used to host game G. Each edge server covers a particular geographical area. Users who are outside the coverage of an edge server will not be able to connect to it (*proximity constraint*). For example, user u_4 cannot be assigned to edge server s_1 or s_4 and has to be allocated to either server s_2 or s_3. Furthermore, we need to take into account various *capacity constraints* such as bandwidth, memory, processing power, etc. In Fig. 1, each edge server has a limited computing capacity denoted as a vector $\langle CPUcore, memory, VRAM, bandwidth\rangle$. The aggregate workload generated by users on a server must not exceed the remaining capacity of that server. There are seven users within the coverage of edge server s_2 with a total workload of $\langle 7, 7, 3.5, 28\rangle$, exceeding the remaining capacity of server $s_2(\langle 7, 8, 4, 25\rangle)$. Thus, the game provider cannot assign all of these users to a single server s_2. Since users $u_1, u_2, \ldots, u_5$ are also covered by other edge servers, it is possible to allocate them to other servers to share the workload with server s_2. One potential solution would be to allocate users u_1, u_2 to server s_1, users

[1] https://www.hatch.live/.
[2] https://www.playstation.com/en-gb/explore/playstation-now/.

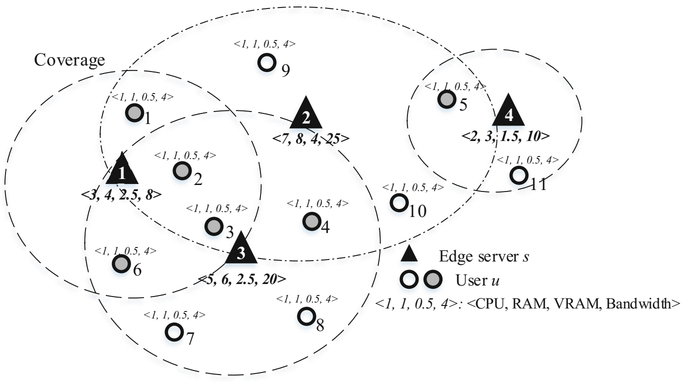


Fig. 1. Edge computing deployment example

u_3, u_6 to server s_3 and users u_4, u_5 stay with server s_2. No proximity or resource constraint is violated this way, but this might not be the optimal solution. If we assign users u_1, u_2, u_4 to server s_2, users u_3, u_6 to server s_3, and user u_5 to server s_4, server s_1 will no longer be required so the service provider can choose not to hire it to lower the total cost of hiring edge servers. This solution satisfies all the aforementioned constraints, uses the least servers to serve the most users, as well as guarantees the QoS.

3 Background

Definition 1 (Classical Bin Packing (BP) Problem). *Given an infinite supply of identical bins $S = \{s_1, s_2, \ldots, s_i\}$ with maximum capacity $C = 1$ and a set of n items $U = \{u_1, u_2, \ldots, u_j\}$. Let a value $w_j \equiv w(u_j)$ be the size of item u_j that satisfies $0 < w_j \leq C$ and $1 \leq j \leq n$. The objective is to pack all the given items into the fewest bins possible such that the total item size in each bin must not exceed the bin capacity C: $\sum_{u_j \in U(s_i)} w_j \leq C, \forall s_i \in S$.*

In the classical BP problem, one can normalize $C = 1$ without loss of generality since the bin capacity is just a scale factor. Aggregating item sizes not exceeding the capacity of the corresponding bin is the only constraint. This problem is known to be an $\mathcal{NP}$-hard combinatorial optimization problem [3].

Definition 2 (Variable Sized Bin Packing (VSBP) Problem. *Given a limited collection of bin sizes such that $1 = size(s_1) > size(s_2) > \ldots > size(s_k)$, there is an infinite supply of bins for each bin type s_k. Let $L = \{s^1, s^2, \ldots, s^l\}$ be the list of bins needed for packing all items. Given a list of items $U = \{u_1, u_2, \ldots, u_j\}$ with $size(u_j) \in (0, 1]$, the objective of the VSBP problem is to find an item-bin assignment so that the total size of the bins required $\sum_{b=1}^{l} size(s^b)$ is minimum.*

In the classical BP problem, all bins are homogeneous with a similar bin capacity. VSBP is a more general variant of the classical BP in which a limited collection of bin sizes is allowed. VSBP aims at minimizing the total size of the bins used, which is slightly different compared to the objective of the classical BP problem as discussed above.

Definition 3 (Vector Bin Packing (VBP) Problem). *Given a set of n items $U = \{u_1, u_2, \ldots, u_j\}$, the size of an item u_j is denoted as a d-dimensional vector $w_j = \langle w_j^1, w_j^2, \ldots, w_j^d \rangle$, $w_j \in [0,1]^d$. One is given an infinite supply of identical bins $S = \{s_1, s_2, \ldots, s_i\}$ with maximum capacity $C = \langle 1^1, 1^2, \ldots, 1^d \rangle$. The objective is to pack the set U into a minimum number of bin s such that $\| \sum_{u_j \in U(s_i)} w_j \|_\infty \leq 1, \forall s_i \in S$.*

In the classical BP problem, the size of an item is presented as a single aggregation measure. By contrast, the size of an item in the VBP problem is associated with a multi-dimensional vector. The objective remains similar, in which the sum of packed item size vectors must not exceed the bin capacity vector in each dimension, which is normalized to 1 without loss of generality. The VBP problem is also known as multi-capacity BP problem in some literature.

In the EUA problem, a bin is referred to as an edge server with the maximum bin capacity being the remaining computing resource of that edge server. An item is referred to as an edge user, which can be a mobile phone or any IoT device; the size of an item is the amount of workloads generated by that user, measured by the computing resource needed to perform the requested task. In this paper, we tackle the EUA problem from a service provider's perspective. Thus, all users of an application generate the same amount of workload. In the real world, different edge servers may have different hardware specifications and host different applications for different numbers of users. Thus, they have different remaining server capacities, or computing resources. In addition, a computing task has various resource requirements such as CPU core, memory, video RAM, bandwidth, and so forth. Therefore, the amount of computing resource needed for a task should not be calculated by a just a single aggregate measure; instead, it can be represented as a d-dimensional vector where each dimension represents a resource type. The proposed VSVBP problem for EUA is $\mathcal{NP}$-hard since the classical BP, which is $\mathcal{NP}$-hard [3], is a special case of VSVBP where $d = 1$ and all the bins are identical in their capacity dimensionality.

4 Our Approach

4.1 Definitions

Edge servers have differentiated remaining capacity and multi-dimensional resource requirements for computation tasks. Therefore, the EUA problem can be modeled as a mixture of the VSBP problem and the VBP problem, hence a variable sized vector bin packing (VSVBP) problem. Our objective is to maximize the number of allocated users and minimize the number of hired edge servers.

We first introduce relevant notations and definitions used in our model in Table 1. In the EUA problem, every user covered by any edge server must be allocated to an edge server unless all the servers accessible to the user have reached their maximum capacities. If a user cannot be allocated to any edge servers, or is not positioned within the coverage of any edge servers, they will be directly connected to a service provider's central cloud server.

Table 1. Key notations

Notation	Description
$S = \{s_1, s_2, \ldots, s_i\}$	Finite set of edge server s_i, where $i = 1, 2, \ldots, m$
$C_i = \langle C_i^1, C_i^2, \ldots, C_i^d \rangle$	$d-$dimensional vector with each dimension C_i^k being a resource type, such as CPU utilization or disk I/O, representing the remaining capacity of an edge server s_i, $k \in \{1, 2, \ldots, d\}$
$U = \{u_1, u_2, \ldots, u_j\}$	Finite set of user u_j, where $j = 1, 2, \ldots, n$
$w_j = \langle w_j^1, w_j^2, \ldots, w_j^d \rangle$	$d-$dimensional vector representing the size of the workload incurred by user u_j. Each vector component w_j^k is a resource type, $k \in \{1, 2, \ldots, d\}$
$U(s_i)$	Set of users allocated to server s_i. $U(s_i) \subset U$
d_{ij}	Geographical distance between server s_i and user u_j
$cov(s_i)$	Coverage radius of server s_i

The total workload generated by all users allocated to an edge server must not exceed its remaining capacity (1). Otherwise, the server will be overloaded, causing service disruptions or performance degradation. Take Fig. 1 for instance. The aggregate workload incurred by users u_5 and u_{11} is $\langle 2, 2, 1, 8 \rangle$ does not exceed the remaining capacity of server s_4, $\langle 2, 3, 1.5, 10 \rangle$; therefore, this is a valid user-server assignment. If we allocate users $u_1, u_2, u_3, u_4, u_5, u_9, u_{10}$ to server s_2, it will be overloaded since the aggregate user workload $\langle 7, 7, 3.5, 28 \rangle$ exceed the server's remaining capacity $\langle 7, 8, 4, 25 \rangle$.

$$\sum_{u_j \in U(s_i)} w_j \leq C_i, \quad \forall s_i \in S \tag{1}$$

In the classical BP problem, an item can be placed in any bins as long as the bin has sufficient remaining capacity. However, in our problem, an edge server covers a limited surrounding geographical region. Thus, an item (user) can be assigned to specific bins (edge servers) since an edge server can only serve users who are located within its coverage (2). Take Fig. 1 for example. Server s_4 can serve users u_5 and u_{11} only. Since users might position in the overlapping areas of different edge servers, there is an optimal solution to allocate as many users as possible to as few servers as possible, which is the main focus of our research.

$$d_{ij} \leq cov(s_i), \forall i \in \{1, 2, \ldots, m\}; \forall j \in \{1, 2, \ldots, n\} \tag{2}$$

Our primary objective is to maximize the number of users allocated to all hired edge servers, which ensures the QoS from the service provider's perspective:

$$maximize \sum_{s_i \in S} |U(s_i)|, \tag{3}$$

Our secondary objective is to find a user-server assignment $\{u_1, \ldots, u_j\} \longrightarrow \{s_1, \ldots, s_i\}$ such that the number of servers hired E is minimum:

$$minimize\, E = |\{s_i \in S| \sum_{u_j \in U(s_i)} w_j > 0\}| \tag{4}$$

4.2 EUA Model

In this paper, we address the EUA problem with two optimization objectives: (1) maximizing the number of users allocated and (2) minimizing the number of edge servers hired, while satisfying the *capacity constraint* and *proximity constraint.* Accordingly, we model the EUA problem as a Lexicographic Goal Programming (LGP) problem [9]. In a lexicographic goal program, there are multiple optimization objectives with a number of constraints. These objectives are ranked by their levels of importance, or priorities. The solver will attempt to find an optimal solution that satisfies the primary objective and then proceed to find a solution for the next objective without deteriorating the previous objective(s). An LGP program can be solved as a series of connected integer linear programs. The LGP formulation of the EUA problem is as follows:

$$maximize\ \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} \tag{5}$$

$$minimize\ E = \sum_{i=1}^{n} y_i \tag{6}$$

subject to:

$$\sum_{j=1}^{n} w_j^k x_{ij} \le C_i^k y_i, \forall i \in \{1, \ldots, n\}; \forall k \in \{1, \ldots, d\} \tag{7}$$

$$d_{ij} \le cov(s_i), \forall i \in \{1, \ldots, n\}; \forall j \in \{1, \ldots, n\} \tag{8}$$

$$\sum_{i=1}^{m} x_{ij} \le 1, \forall j \in \{1, \ldots, n\} \tag{9}$$

$$y_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \tag{10}$$

$$x_{ij} \in \{0, 1\}, \forall i \in \{1, \ldots, n\}; \forall j \in \{1, \ldots, n\} \tag{11}$$

where

$y_i = 1$ if server s_i is hired.
$x_{ij} = 1$ if user u_j is allocated to server s_i.
$cov(s_i)$ is provided by edge computing providers.

The objective (5) maximizes the number of users that are assigned to hired edge servers. The objective (6) minimizes the number of hired edge servers. Here, objective (5) is ranked higher than objective (6) in terms of priority. There are two groups of binary variables, i.e., x_{ij} (11) and y_i (10).

Capacity Constraint: As described by (7), each edge server s_j has a remaining capacity of $C_i = \langle C_i^1, C_i^2, \ldots, C_i^d, \rangle$, a d-dimensional vector. The aggregate workload of each resource type incurred by all allocated users must not exceed the corresponding remaining capacity of their assigned server. Take Fig. 1 for example. Assigning users u_5, u_{11} to server s_4 is valid since $\langle 2, 2, 1, 8\rangle < \langle 2, 3, 1.5, 10\rangle$.

Proximity Constraint: As described by (8), only users located within the coverage of an edge server can be allocated to the edge server. A user may be located in the overlapping coverage of multiple edge servers. For instance, users u_2, u_3 can be allocated to servers s_1, s_2 or s_3.

Constraint family (9) ensures every user is allocated to at most one edge server. In other words, a user can be allocated to either an edge server or service provider's cloud server.

5 Experimental Evaluation

In this section, we evaluate the performance of our approach by extensive experiments with a comparison to two baseline approaches. All the experiments were conducted on a Windows machine equipped with Intel Core i5-7400T processor (4 CPUs, 2.4 GHz) and 8 GB RAM. The LGP problem modeled in Sect. 4.2 was solved using IBM ILOG CPLEX Optimizer.

5.1 Baseline Approaches

Our approach will be benchmarked against two baseline approaches for user-to-server assignment, namely *random* and *greedy* approaches:

- *Random*: Each user will be allocated to a random edge server as long as that server has sufficient remaining capacity to accommodate the user and has the user within its coverage.
- *Greedy*: Each user will be allocated to an edge server that has the most remaining capacity and has the user within its coverage.

5.2 Experiment Settings

In this paper, we conduct experiments on data of base stations and end-users within the Melbourne central business district area in Australia, which has a total area of 6.2 km^2.

Experiment Data: We collect the location data of edge servers and end-users. Australian Communications and Media Authority (ACMA) publishes the radiocomms license dataset that contains the geographical location of all cellular base stations in Australia, which we will use as the locations of edge servers [5]. The coverage of each edge server is randomly set within a range of 450–750 m. In terms of end-users' locations, Asia Pacific Network Information Centre (APNIC) provides all IP address blocks allocated to Australia. We use an IP lookup service[3] to convert the obtained IP addresses into geographical locations. Since IP addresses in the last octet are likely to have identical geographical addresses returned by the IP lookup service, more end-users are uniformly generated around each of the obtained geographical locations. The raw experimental data has been made publicly available (EUA-dataset[4]).

Experimenting Parameters: In the experiments, we vary three setting parameters that may have an impact on our approach:

(1) **Number of end-users:** We randomly select different numbers of end-users $n = 4, 8, 16, \ldots, 512$. For each setting, we run the experiment 100 times to get 100 different random end-user distributions so that extreme cases, such as overly dense or sparse user distributions, are properly neutralized.
(2) **Number of edge servers:** The n end-users are located within the combined coverage of M edge servers. We assume that a total of m servers, where $m = 10\%, 20\%, \ldots, 100\% * M$, are available for accommodating those n end-users.
(3) **Remaining server capacity:** We experiment various levels of remaining server capacity based on the combined user workload. To be specific, we calculate 100%, 150%, ..., 300% of the combined user workload, then normally distribute it to M edge servers collectively covering the n end-users.

Performance Metrics: We evaluate the three approaches, namely our VSVBP, the random and the greedy baseline approaches, using the following metrics: (1) the percentage of allocated end-users of all end-users, the higher the better; (2) the percentage of hired edge servers of all available edge servers, the lower the better; and (3) the execution time (CPU time), the lower the better.

Given the data and the experiment parameters, we conduct three sets of experiments. The corresponding settings are described in Table 2. For each set, we vary one parameter and keep the other two fixed to observe the impact of each parameter on the approaches in the evaluation metrics.

In experiment set 1, the number of users vary from 4, 8, 16, 32, 64, 128, 256 to 512. All the edge servers, which have end-users within their coverage, can serve those end-users. The total remaining server capacity is 300% of the combined user workload. In experiment set 2, the number of users is fixed at 512, and the total remaining server capacity is fixed at 300% of the combined user workload.

[3] http://ip-api.com/.
[4] https://github.com/swinedge/eua-dataset.

Table 2. Experiment settings

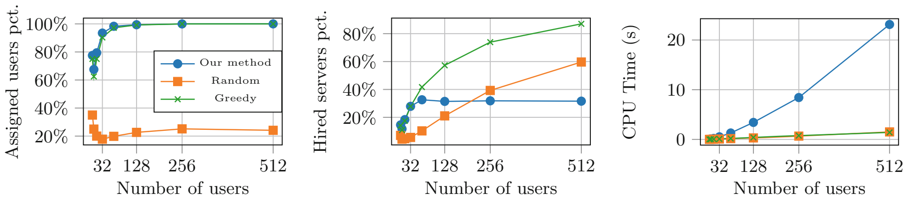
Factor	Number of users	Percentage of the total number of servers	Remaining server capacity
Set #1	$4, 8, \ldots, 512$	100%	300%
Set #2	512	$10\%, 20\%, \ldots, 100\%$	300%
Set #3	512	100%	$100\%, 150\%, \ldots, 300\%$

We change the number of edge servers that would be used to accommodate end-user, i.e., $10\%, 20\%, \ldots, 100\%$ of all edge servers to be made available for hire. In the last experiment set, we keep the number of users fixed at 512 and make all edge servers available for hire. The changing factor is the remaining server capacity – $100\%, 150\%, \ldots, 300\%$ of all users' workload combined.

5.3 Experimental Results and Discussion

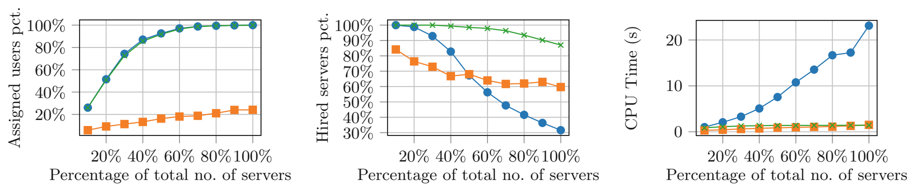
Figures 2, 3 and 4 show the results of the experiment set 1, 2 and 3, respectively. The three performance metrics are depicted in each sub-figure: (a) percentage of user allocated, (b) percentage of servers hired, and (c) execution time.

Figure 2 shows that in experiment set 1, as we increase the number of end-users from 4 users to 512, the random approach performs poorly in terms of allocated users percentage (only 20%–25% of the users are allocated) compared to the greedy approach and our approach, which give an equal performance with all users having been allocated. However, in terms of the number of edge servers hired, our approach starts to outperform the greedy approach as the number of end-users exceeds 32. The percentages of servers hired by the greedy and the random methods keep growing as the number of end-users increases, up to around 87.04% when serving 512 end-users. By contrast, our approach stably uses only around 32% of all available edge servers, 2.7 times less than that of the greedy approach, and remains steady even when the number of end-users increases from 32 to 512.



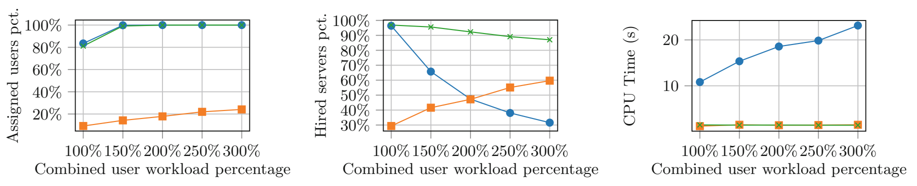
(a) Pct. of users allocated (b) Pct. of servers hired (c) Execution time

Fig. 2. Resulted metrics of set #1 (number of users changing)



(a) Pct. of users allocated (b) Pct. of servers hired (c) Execution time

Fig. 3. Resulted metrics of set #2 (number of servers changing)



(a) Pct. of users allocated (b) Pct. of servers hired (c) Execution time

Fig. 4. Resulted metrics of set #3 (remaining server capacity changing)

In experiment set 2, we change the number of edge servers available for hire. As depicted in Fig. 3(a), the allocated user percentage follows a similar trend as in experiment set 1. Regarding the percentage of hired edge servers (Fig. 3(b)), our approach continues to outperform the other two approaches as the number of edge servers increases.

Figure 4 shows that, in the last set of experiments, where the edge servers' total remaining capacity increases, we can observe the same trending patterns with our approach being the most effective out of the three approaches studied. As we increase the combined user workload percentage from 100% to 300%, our approach uses significantly fewer servers, dropping from 96.4% to 31.6% while the greedy method has to use around 90% of the all available servers.

Note that in all three set of experiments, the random approach seems to perform better than ours with fewer hired servers on some occasions. For example, when the number of users varies between 4 and 128 (Fig. 2(b)), when the total number of hired edge servers percentage varies between 10% and 50% (Fig. 3(b)), and when the total remaining server capacity changes between 100% and 200% (Fig. 4(b)). In fact, the random approach does not produce better results in these scenarios because although it uses fewer servers, the number of users allocated is extremely small (only 6%–20% of all end-users experimenting) compared to the other approaches, as shown in Figs. 2, 3, 4(a).

In terms of efficiency, the computation time of our approach increases considerably as we increase any one of the three parameters. In experiment set 1 with 512 users, the greedy and random methods take only approximately 1.5 s while our approach takes around 23.1 s to solve an instance of the EUA problem. This can also be observed in experiment sets 2 and 3, where we increase the number of servers available for hired and the total remaining server capacity respectively. Since the EUA problem is an $\mathcal{NP}$-hard problem, it is expected that our approach, which optimally solves the problem, will take the most time as opposed to the other approaches, which can only make local decisions without considering the problem globally.

In general, increasing one of the three experimental parameters will increase the complexity of the EUA problem, which is an $\mathcal{NP}$-hard problem, and thus take more time to find an optimal solution. Our experimental results show that the random approach is not able to maximize the number of users allocated (the first optimization objective) as it can assign around only 20% of all the end-users in the experiments. The greedy approach is able to assign a similar number of end-users as our approach; the edge servers' adequate remaining capacities allow the greedy approach to find a capable edge server to accommodate most end-users. However, as shown in Figs. 2, 3, 4(b), our approach hires much fewer edge servers (the second optimization objective) than the greedy method to accommodate all the end-users. This is shown in all three experiment results, especially as the EUA problem scales up.

5.4 Threats to Validity

Threats to Construct Validity. The main threat to the construct validity in our study lies in the comparison with the two baseline methods, i.e., the random and greedy methods. The EUA problem studied in this research is a problem that has not been investigated before in this domain. Thus, we selected these two common and intuitive methods as baselines in our evaluation. Their designs are simple, especially the random method, which employs no heuristics. As a result, our approach is likely to obtain better experimental results, leading to a threat where the comparison with the selected baselines might not properly demonstrate the effectiveness of our approach in solving the EUA problem. To minimize this threat, we conducted experiments with three changing parameters as described in Table 2 to simulate different service deployment scenarios in the real world. This way, we could reliably evaluate our approach through both comparison with the baseline methods and also impacts of varying each experimental parameters on our approach.

Threats to External Validity. A major threat to external validity is whether our findings based on the experimental dataset can be generalized to other application domains in edge computing. Since there is currently no real-world dataset for this type of edge computing problems, we synthesized a dataset of edge servers and end-users based on reliable real-world data sources (ACMA and APNIC). However, this is a generic dataset, and it is possible that different application domains might have different factors that could impact the experimental results, such as the density and distributions of edge servers and end-users. Thus, our approach was evaluated across a breadth of problem scoping, varying in size, i.e., number of end-users, and complexity, i.e., number of edge servers and edge servers' remaining capacities, to simulate as many types of edge server and end-user density and distribution as possible, as well as their combinations. This helped reduce the threat to the external validity of our evaluation and increased the generalizability of our results.

Threats to Internal Validity. A threat to internal validity of our work is the comprehensiveness of our experiments and whether or not the results are not biased by the experimental parameter settings. To mitigate this threat, we carried out extensive experiments with systematically selected parameters. The three experimental parameters (discussed in Sect. 5.2) are the three representative parameters that directly impact the outcomes of the approaches. Also, for each experiment set, we experimented with 100 different user distributions randomly selected from the pool of users to eliminate the potential bias caused by highly special scenarios such as overly dense or sparse distributions. Another threat to the internal validity of our evaluation is where more sophisticated scenarios could be simulated, e.g., those where two or more of those parameters change at the same time. In those scenarios, the results can be predicted in general based on the results that we have obtained. For example, if the total number of available edge servers and their total remaining capacities increase at the same time, the percentage of used servers of all hired by our approach will decline with a trend similar to but more significant than those shown in Figs. 3(b) and 4(b).

Threats to Conclusion Validity. The lack of statistical tests is the biggest threat to our conclusion validity. Statistical tests will be included in our future work to prove a statistically significant relationship between the experiment settings and the results. In this paper, we have compensated for this with meaningful comparison baselines and extensive experiments that cover many different scenarios, varying in both size and complexity. When an experimental parameter changes, the results are averaged over 100 runs of the experiment.

6 Related Work

Resource management in cloud computing has been extensively investigated in the last decade in many research tracks such as load balancing [7], virtual machine placement and provisioning [14], server and task allocation [8], etc.

Edge computing, or fog computing, is a new computing paradigm coined by Cisco in 2012 [1]. Edge computing is a natural extension of cloud computing with regard to the network topology and infrastructure deployment, where the architecture is more geographically distributed compared to cloud computing. This new architecture pushes the cloud resources closer to end-users. Barcelona, Spain is one of the first cities implementing edge computing with many applications, including power monitoring in public spaces, access control and telemetry of sensors, event-based video streaming, traffic analysis and regulation, and connectivity on demand [15]. There are more than 3,000 edge servers deployed across the city serving thousands of IoT devices. The sheer number of edge servers and end-devices, with the horizontal scaling nature of edge computing, leads to the need for effective and efficient resource allocation solutions.

Chen et al. [2] proposed a distributed game theoretic computation offloading algorithm that was able to achieve a Nash equilibrium, minimizing the total energy consumption and offloading latency in the multi-channel mobile edge computing environment. By the proposed approach, they were able to optimally decide whether the users should offload computing tasks to an edge server and if yes, which wireless channel to be used for the computation offloading. In [16], Yao et al. tackled the problem of cost-effective edge server deployment using integer linear programming. They took into account the factors of resource capacity, user-server latency, and deployment costs. In their research, each edge server might not have the service installed to fulfill the requests from end-users. Thus, users' requests might have to travel across different edge servers until executed. However, they assumed that each server covers a region exclusively with other servers. They also assumed a predetermined edge server that first receives the user request. Our research targets more realistic edge computing scenarios where different edge servers' coverages might partially overlap. The authors of [13] also made an assumption that each small geographical area will only receive coverage from only a single edge server, which will be unlikely to happen in real-world scenarios. In [11], the authors formulated a problem similar to the EUA problem but with different objectives, which are to reduce task completion time and energy consumption. Yin et al. [18] addressed the edge server placement and provisioning problem with the objective of maximizing users coverage and minimizing network latency.

To the best of our knowledge, our work is the first to tackle the EUA problem in scenarios with multiple edge servers and end-users that possess and require multi-dimensional computing capacities. We also realistically and innovatively address this problem with respect to proximity constraints with the aims to maximize the number of allocated users and minimize the number of hired servers.

7 Conclusion

Edge computing is a promising new computing architecture, especially for high volume, data processing-intensive, latency-sensitive applications and services. However, when an edge computing scenario scales up, an ineffective edge user allocation solution will greatly increase the operational costs for service providers. To address this problem, we formulated the edge user allocation (EUA) problem as a variant of the bin packing problem named variable sized vector bin packing, an $\mathcal{NP}$-hard problem. We solved this problem using a Lexicographic Goal Programming technique with two optimization objectives, i.e., to maximize the number of users allocated and minimize the number of edge servers hired. We then conducted extensive experiments in scenarios with various service deployment requirements. Our experimental results show that our approach significantly outperforms two baseline approaches, greedy and random.

It is capable of allocating the most end-users with significantly fewer edge servers – nearly three times less than the greedy method – as the EUA problem scales up.

This research has established a basic foundation for the EUA problem and opened up a number of research directions. In our future work, we will take into account the users' mobility as well as the dynamics of users' computation tasks. In addition, apart from the proximity and capacity constraints, there are several elements that also play an important role such as network latency, service availability, pricing, and security.

Acknowledgments. This research is funded by Australian Research Council Discovery Projects (DP170101932 and DP18010021).

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16. ACM, New York (2012). https://doi.org/10.1145/2342509.2342513
2. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**(5), 2795–2808 (2016). https://doi.org/10.1109/TNET.2015.2487344
3. Garey, M.R., Johnson, D.S.: Computers and Intractability, vol. 29. W. H. Freeman and Company, New York (2002)
4. Heuveldop, N.: Ericsson mobility report. Technical report, Ericsson, November 2017. https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-november-2017.pdf
5. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing a key technology towards 5G. Technical report 11, European Telecommunications Standards Institute (2015). http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf
6. Lin, Y., Shen, H.: CloudFog: leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service. IEEE Trans. Parallel Distrib. Syst. **28**(2), 431–445 (2017). https://doi.org/10.1109/TPDS.2016.2563428
7. Mitzenmacher, M.: The power of two choices in randomized load balancing. IEEE Trans. Parallel Distrib. Syst. **12**(10), 1094–1104 (2001). https://doi.org/10.1109/71.963420
8. Ren, R., Tang, X., Li, Y., Cai, W.: Competitiveness of dynamic bin packing for online cloud server allocation. IEEE/ACM Trans. Netw. **25**(3), 1324–1331 (2017). https://doi.org/10.1109/TNET.2016.2630052
9. Romero, C.: Handbook of Critical Issues in Goal Programming. Elsevier, Amsterdam (2014)
10. Smith, J.: The mobile gaming report. Technical report, Business Insider Intelligence (2016). http://www.businessinsider.com/the-mobile-gaming-report-market-size-the-free-to-play-model-and-new-opportunities-to-market-and-monetize
11. Tran, T.X., Pompili, D.: Joint task offloading and resource allocation for multi-server mobile-edge computing networks. CoRR abs/1705.0 (2017). http://arxiv.org/abs/1705.00704

12. Varghese, B., Wang, N., Nikolopoulos, D.S., Buyya, R.: Feasibility of fog computing. CoRR abs/1701.0 (2017). http://arxiv.org/abs/1701.05451
13. Wang, L., Jiao, L., Li, J., Mühlhäuser, M.: Online resource allocation for arbitrary user mobility in distributed edge clouds. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1281–1290, June 2017. https://doi.org/10.1109/ICDCS.2017.30
14. Wolke, A., Tsend-Ayush, B., Pfeiffer, C., Bichler, M.: More than bin packing: dynamic resource allocation strategies in cloud data centers. Inf. Syst. **52**, 83–95 (2015). https://doi.org/10.1016/j.is.2015.03.003
15. Yannuzzi, M., et al.: A new era for cities with fog computing. IEEE Internet Comput. **21**(2), 54–67 (2017). https://doi.org/10.1109/MIC.2017.25
16. Yao, H., Bai, C., Xiong, M., Zeng, D., Fu, Z.: Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing. Concurr. Comput. **29**(16), 1–9 (2017). https://doi.org/10.1002/cpe.3975
17. Yi, S., Li, C., Li, Q.: A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata 2015, pp. 37–42. ACM, New York (2015). https://doi.org/10.1145/2757384.2757397
18. Yin, H., et al.: Edge provisioning with flexible server placement. IEEE Trans. Parallel Distrib. Syst. **28**(4), 1031–1045 (2017). https://doi.org/10.1109/TPDS.2016.2604803

RA-FSD: A Rate-Adaptive Fog Service Delivery Platform

Tiehua Zhang(✉), Jiong Jin, and Yun Yang

School of Software and Electrical Engineering,
Swinburne University of Technology, Melbourne, Australia
{tiehuazhang,jiongjin,yyang}@swin.edu.au

Abstract. As the Internet of Things (IoT) technologies permeate people's daily lives, the sheer number of IoT applications has been developed to provide a wide range of services. Among all, real-time IoT services start to draw increasing attentions. Conventionally, cloud plays the role as the service provider in IoT but is no longer considered as the rational option for the real-time services due to service transmission latency and communication overhead. Therefore, we propose a novel rate-adaptive fog service delivery platform, namely RA-FSD, aiming at real-time service provisioning and network utility maximization (NUM) of the underlying IoT resources based on the newly emerged fog computing paradigm. The platform leverages fog nodes as either fog service provider to offer timely services for end users, or service intermediaries to help track network conditions and mitigate communication overhead. By doing so, service consumers would always benefit from the fact that services produced by IoT applications are in their proximity and thus delivered to destination in a prompt manner. A service rate-adaptive algorithm is also developed as the key component of the RA-FSD platform to handle the abrupt changes happened in IoT network, dynamically adjust service delivery rate based on the network condition while retaining satisfactory Quality of Service (QoS) to each service consumer, and support both elastic and inelastic network services from heterogeneous IoT applications.

Keywords: Fog computing · Internet of Things (IoT)
Service-oriented networking · Network utility maximization (NUM)
Quality of Service (QoS)

1 Introduction

The proliferation of IoT technology has brought the unprecedented convenience to people and draw widespread attentions from both academia and industry [3]. Currently, cloud plays a major role in providing these highly personalized, context-aware IoT services because of the advantages like cost reductions, easy deployment and strong reliability. However, the new challenges presented by real-time IoT applications like stringent latency and Quality of Service (QoS) requirement are not well addressed by the standalone cloud computing paradigm.

In order to address aforementioned issues and overcome inadequacy of the cloud, fog computing is introduced. The idea of extending cloud to the edge of network and closer to end users has been viewed as an alternative with the overarching goal of "off-loading" from the cloud. Fog nodes, acting as the proxy of both cloud and end devices/users (*things*), could be equipped with computing, storage and networking resources to accommodate various IoT applications. Therefore, these applications could be deployed in fog rather than the conventional approach on either resource-constrained IoT devices or remote cloud. In this regard, fog and cloud complement each other to form a service continuum that distributes respective services to end users [2].

Along with the rapid growth of IoT applications, heterogeneous services are tailored for service consumers with certain QoS guarantee. In reality, stable service delivery rate is a key component to meet the QoS and it could act as a major part in service consumers perception with regard to overall performance of service invocation [1]. In our work, the utility function is used to model the QoS performance, which increases as the increasing of service delivery rate. From the utility point of view, the services can be categorized into two main groups, i.e., traditional elastic services (e.g., file transfer, data analysis and web browsing), in which each service attains a strictly increasing and concave utility function to measure its QoS performance, and real-time inelastic services provided by audio, video and multimedia real-time applications. Such inelastic services have an intrinsic bandwidth threshold in nature and adopt the sigmoid-like functions to describe the corresponding QoS [5].

There are several previous efforts made towards developing service delivery architecture to connect service consumers and providers in IoT environment. A vehicular data cloud platform is proposed in [4] to provide real-time information, yet concerns like service transmission latency and transportation cost are not discussed. Some service platforms aim to utilize underlying IT resources to achieve good QoS through NUM, e.g, the automatic service routing platform proposed in [1] and the multicast multirate service delivery platform in [8], but their works either are not designed for IoT environment, or fail to support inelastic, real-time IoT services. Given that, the advantages of RA-FSD platform are highlighted as follows: (1) it seamlessly integrates fog computing into IoT environment, and utilizes fog node as service provider equipped with more computation and storage resources than traditional IoT devices; (2) it offers services that are generated in the proximity of users, and both service transportation cost and delivery latency are largely reduced; (3) beside elastic services, it also supports real-time inelastic services.

2 Architecture of RA-FSD Platform

In this section, we introduce the architecture of RA-FSD platform and the components inside. The platform is composed of heterogeneous *things*, fog nodes and

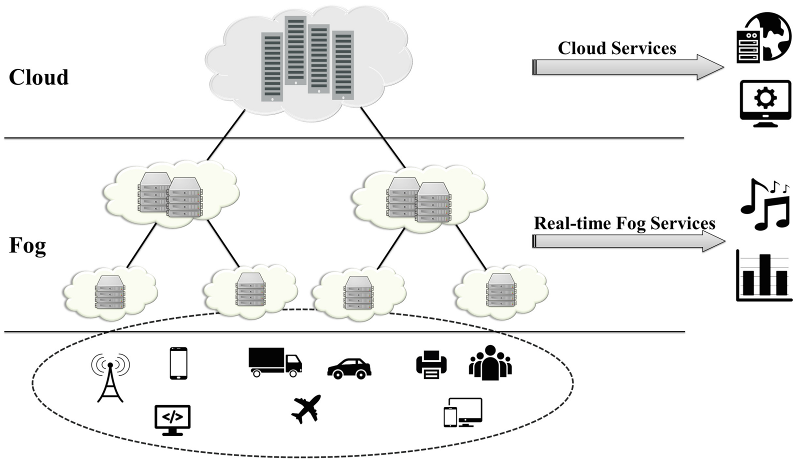


Fig. 1. Fog computing architecture, and example services provided by cloud and fog nodes

the cloud. From the service-oriented computing perspective [7], *things* normally play the role of service consumers/requesters, fog nodes equipped with computation, storage and networking power could serve as either service providers or service intermediaries, in which service intermediaries help collect service requests from bottom-level *things*, track network conditions, cooperate with providers to adapt service transmission rate, and forward services back to requesters. Since the cloud treats fog as the proxy in the edge network, it is noticeable that the use of cloud is no longer mandatory under this platform, but one could choose to continue using cloud as service provider for particular services, e.g., advanced analytic service for big data.

Figure 1 presents a simple fog architecture with a multi-level hierarchy, including different types of service provisioned inside. The bottom layer consists of the end devices or users. The fog nodes have been positioned in the middle two layers, and similar to the traditional IoT network, the cloud is at the top layer. Taking advantage of the geographical locations, IoT applications could deploy on the fog nodes in the vicinity of *things* and fog node is thus able to serve as the real-time service provider in IoT network. In addition, fog node selected as provider would form the corresponding service group in which a particular service could be disseminated. Specifically, the provider in each group will gather feedback regarding downstream network conditions reported by service intermediaries to adjust transmission rate for different service receivers. Service intermediaries, on the other hand, collaborate with providers and could be converted to the providers if needed, which increases the scalability and flexibility of the platform. In general, the cloud is only used as service provider if that service associates with a large volume of data processing and storage, or service requested is delay-tolerant.

3 Analytic Framework and Optimization Problem

The NUM resources allocation problem in the fog-based IoT environment is formulated in this section to make it support both elastic and real-time inelastic services. In addition, we characterize the utility in terms of allocated service transmission rate deriving from the underlying bandwidth of IoT network. Consider a fog service delivery network consisting of a set of links $L = \{1, 2, ..., l\}$, each of which has respective capacity c_l. There is a set of $S = \{1, 2, ..., s\}$ service groups, and each service group is devoted to providing one service. For each service group s, there is only one unique service provider, which is either a fog node or the cloud. A set of receivers in service group s could be defined as $R_s = \{r_{s,1}, r_{s,2}, ..., r_{s,n}\}$, and along with a set of links $L_s \subset L$. They together form the corresponding service delivery tree for that service group, where the provider stays at the root of the tree, and each receiver in R_s is connected to the IoT network through the leaf fog nodes.

For each service receiver $R_{s,i} \in R_s$ in a service group, $L_{s,i} \subset L_s$ describes the service delivery path from the provider of service group s to relevant receiver i. Utility function $U_s(x_{s,i})$ has been selected on per-service basis, which is customized to describe the QoS requirement. In addition, utility function should be strictly increasing and continuously differentiable, but needs not be concave in our work. $x_{s,i}$ represents the service delivery rate to receiver i in service group s.

We then formulate the following optimization problem, on the basis of multicast, multirate model similar to [8]:

$$P1: \quad \max_{x \geq 0} U(x) = \sum_{s \in S} \sum_{i=1}^{n_s} U_s(x_{s,i}) \tag{1}$$

$$\text{subject to} \quad \sum_{s \in S} x_s^l \leq c_l, \quad \forall l \in L \tag{2}$$

$$x_s^l = \max_{\{i|l \in L_{s,i}\}} x_{s,i} = \lim_{N \to \infty} \left(\sum_{\{i|l \in L_{s,i}\}} x_{s,i}^N \right)^{\frac{1}{N}} \tag{3}$$

In Eq. (3), $\{i|l \in L_{s,i}\}$ is a set of receivers that uses link l to receive the corresponding service in service group s. This equation states that in service group s, the service rate on link l is the same as the rate of the fastest downstream receiver in this group. In addition, constraint (2) in this optimization problem means that the aggregate service rate on link l across all service groups should not exceed the link capacity (network condition). Then we replace (3) in (1) and the Lagrangian multiplier could be yielded:

$$L(x, p) = \sum_{s \in S} \sum_{i=1}^{n_s} U_s(x_{s,i}) - \sum_{l \in L} p^l \left[\sum_{s \in S} \left(\sum_{\{i|l \in L_{s,i}\}} x_{s,i}^N \right)^{\frac{1}{N}} - c_l \right] \tag{4}$$

In order to make our platform support both elastic and inelastic services, a pseudo utility function [5] is constructed as (5) to substitute the original optimization problem $P1$:

$$\mathcal{U}_s(x_{s,i}) = \int_{m_s}^{x_{s,i}} \frac{1}{U_s(y)} dy, \quad m_s \leq x_{s,i} \leq M_s \tag{5}$$

where m_s and M_s represent minimum and maximum service delivery rate, respectively. Based on the characteristic of original utility function $U_{s,i}(x_{s,i})$, this pseudo utility function must be strictly increasing and concave as $\mathcal{U}_s''(x_{s,i}) < 0$. By solving the optimization problem, the derived results could be further incorporated into the service rate-adaptive algorithm.

4 Service Rate-Adaptive Algorithm and Implementation

The algorithm is devised to take advantage of the architectural support from the platform and is deployed on all fog nodes to periodically check the network conditions as well as adapt the service delivery rates accordingly. By choosing appropriate fog node as service provider, the corresponding service generated could traverse less hops than the cloud scheme and thus make it suitable for delay-sensitive IoT applications.

Algorithm 1 presents a summary of the algorithm, which incorporates the analytic results derived in Sect. 3 and comprises two phases. The service intermediaries would firstly gather relevant downstream links information, then report it upwards recursively to form the fog service continuum (lines 2–3). Afterwards, several service groups have been established, and whenever a service requester joins or leaves a service group (network changes), the bottom-level fog node is able to detect the change and report it upwards, which consequently starts another round of phase 1. In phase 2, fog nodes would firstly iterate through each downstream links and calculate the current link status. More specifically, lines 5–10 deal with the link price updating process, followed by the calculation of price weighting coefficient in lines 11–17, which implies that this coefficient would continue to increase for receiver with the largest service rate, while decreasing among other receivers.

Algorithm 1 Service rate-adaptive algorithm

```
Phase 1: Initialization
 1: Fog or cloud will collect network condition data reported from child nodes,
    update relevant $W$ and $P$, then communicate back.
 2: Establish Service group $S = [s^1, s^2....s^n]$ and gather link capacity $C =
    [c^1, c^2....c^l]$
 3: $W, P \leftarrow R^S \times L$ matrix,where $w^l_{s,i} = \frac{1}{\{j|l \in L_{s,j}\}}, p^l_{s,i} = 0$ at initial stage
Phase 2: Adjusting service rate
 1: Starting at bottom fog nodes at every interval $t$
 2: repeat
 3:     $f \leftarrow$ current node
 4:     for each downstream link $l$ of $f$ do
 5:         1. select largest service delivery rate of each service on that link
 6:             $x^l_s = \max_{\{i|l \in L_{s,i}\}} x_{s,i}$
 7:         2. aggregate delivery rate of each service on link $l$
 8:             $x^l = \sum_{s \in S} x^l_s$
 9:         3. calculate the current link price of $l$
10:             $p^l = [p^l + \lambda(x^l - c^l)]^+$
11:         4. calculate the price weighting coefficients for each downstream
    receiver $i$, whose service transverses link $l$
12:             $w^l_{s,i} = [w^l_{s,i} + \lambda(x_{s,i} - x^l_s)]^+$
13:         if receiver $i$ receives service s at rate $x^l_s$ then
14:             $w^l_{s,i} = 1 - \sum_{\{j|j \neq i, l \in L_{s,i}\}} w^l_{s,j}$
15:         end if
16:         5. calculate link price $p^l_{s,i}$ for each downstream receiver $i$ on link $l$
17:             $p^l_{s,i} = w^l_{s,i} p^l$
18:         6. update corresponding $w^l_{s,i}$ in $W$ and $p^l_{s,i}$ in $P$, respectively
19:         if $f$ is a provider of service s then
20:             for each receiver $i$ that receives service s do
21:                 7. calculate relevant path price
22:                     $p_{s,i} = \sum_{l \in L_{s,i}} p^l_{s,i}$
23:                 8. adjust service rate
24:                     $x_{s,i} = U_s^{-1}\left(\left[\frac{1}{p_{s,i}}\right]^{U_s(M_s)}_{U_s(m_s)}\right)$
25:             end for
26:         end if
27:     end for
28:     if there is any upstream service coming to $f$ then
29:         9. propagate network condition upward, and repeat phase 2
30:     end if
31: until (all providers have been reached)
```

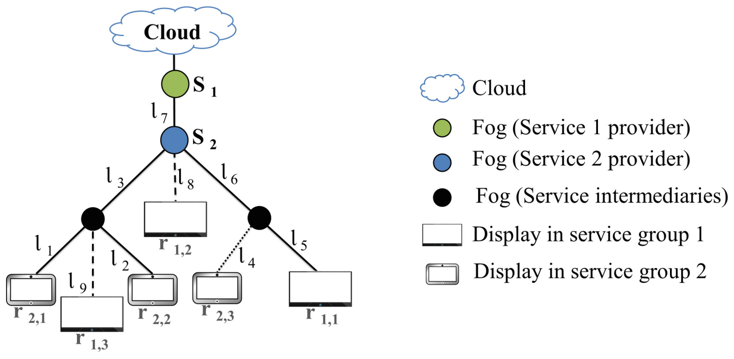


Fig. 2. Fog architecture in the shopping mall use case

Lines 18–27 handle the service rate adjusting process, if and only if current node f is a service provider.

5 Performance Evaluation on a Case Study

In this section, we verify the feasibility of proposed platform with the algorithm through a real-world shopping mall use case. Besides, the numerical results are meanwhile used to demonstrate the applicability and robustness of the RA-FSD platform. It is worthwhile mentioning that, as elastic service is relatively easier to be accommodated, our focus on the use case is to implement inelastic services originated from real-time applications.

Figure 2 illustrates the topology of the IoT network empowered by fog architecture in the shopping mall. In this topology, all fog nodes represented as the dot points are placed inside the shopping mall, along with two different sizes of digital displays acting like *things*. The topmost fog node is configured as the most powerful node among all, and operates as the main gateway of this autonomous network. In addition, cloud is only used for data backup purpose.

The performance of RA-FSD platform is evaluated through simulations. The fog network originally contains 7 links labeled as $l_1, l_2, ..., l_7$ with relevant capacities c = (6, 4, 10, 8, 8, 12, 18) (in Mbps), and these links have been shared between two service groups s_1, s_2. In addition, two types of screens require different services transmission rates to satisfy respective QoS requirement. The utility function $U_1(x_{s=1,i}) = \frac{1}{1+e^{-2(x-6)}}$ is used to reflect the QoS performance of inelastic service 1, and $U_2(x_{s=2,i}) = \frac{1}{1+e^{-2(x-4)}}$ for service 2.

The simulations start at time $t = 0$, and service group 2 contains relatively small screens $r_{2,1}, r_{2,2}, r_{2,3}$, while service group 1 only has one big screen $r_{1,1}$ at the start. The minimum and maximum service delivery rates to each screen are set to be 0 and 10 Mbps but will be adapted quickly based on the feedback of network condition. The platform triggers the algorithm at the bottom-level fog node at every time interval of 0.1 s and experimental results including service

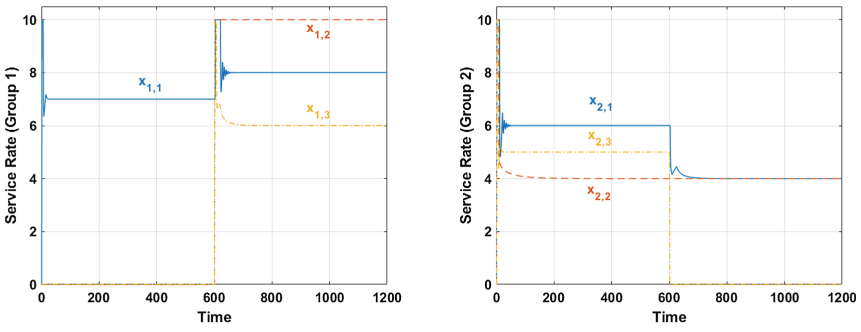


Fig. 3. The service delivery rates in different service groups

rate and utility are expected to reach a stable state rapidly to demonstrate the applicability of the platform. It is also noticeable that at $t = 60$ s, two new displays $r_{1,2}$ and $r_{1,3}$ have joined service group 1 and establish connections to fog nodes through links l_8 and l_9 (dashed-line links) with capacities of 10 and 8, respectively, while $r_{2,3}$ has suddenly left service group 2 along with a disconnection of link l_4. The abrupt changes of topology is not uncommon in real-world situation, which is hereby used to validate the robustness of the platform.

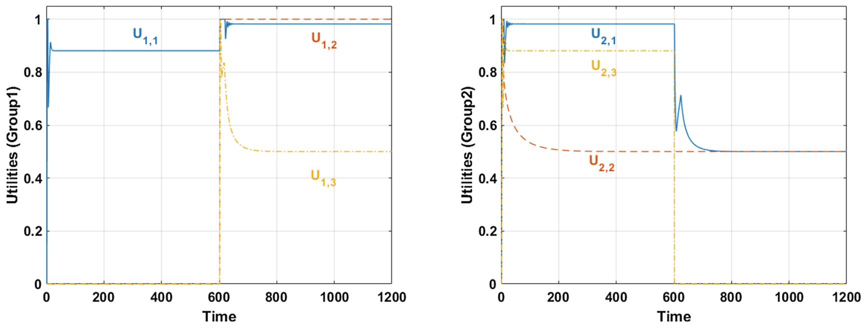


Fig. 4. The utility results in different service groups

The simulation results of service delivery rates ($x_{s,i}$) and utility results (QoS) in service groups 1 and 2 are shown in Figs. 3 and 4, respectively. It is clearly observed that all service rates converge to the optimum under the complex IoT network conditions even with the abrupt network changes. The platform is capable of eliminating the instability, and relevant fog service providers can concretely adapt service rate for each receiver to maintain a relatively good QoS. The minimum service delivery rate achieved in this scenario is 4 Mbps and the overall QoS achieved by the platform retains the value of more than 0.5, which substantially suffices the service requirements of displays in the shopping mall use case [6].

To conclude, the simulation results reconfirm that our proposed platform is applicable and robust in real-world scenario, and also capable of handling abrupt network changes. Furthermore, the convergence of service delivery rates and corresponding utilities clearly demonstrate that the platform could support real-time inelastic services offered by the fog service providers.

6 Conclusion and Future Work

In this paper, we have proposed a novel rate-adaptive fog service delivery platform that is applicable for current IoT network. Important issues in traditional service-oriented IoT network such as service transmission latency and huge bandwidth waste have been well addressed by applying RA-FSD. Heterogeneous IoT services now offered in the vicinity of *things* as fog node could serve as providers that are only a few hops away. Additionally, fog nodes in our platform work collectively to maintain the stability of the IoT network.

Our case study verifies that, on the basis of fog architecture, RA-FSD seamlessly integrates service rate-adaptive algorithm, and copes with real-world scenarios effectively even with the abrupt changes occurred in the IoT network (new joiner or leaver). Moreover, both elastic and inelastic IoT services are well supported in the platform. Our next phase of research will focus on developing the provider "pick up" strategy, in which RA-FSD platform should dynamically select a fog node as the service provider based on its characteristic such as computing power, geographic location or network condition.

Acknowledgements. This work is partly supported by Australian Research Council Discovery Project Grant DP180100212 and Australian Government Research Training Program Scholarship.

References

1. Callaway, R.D., Devetsikiotis, M., Viniotis, Y., Rodriguez, A.: An autonomic service delivery platform for service-oriented network environments. IEEE Trans. Serv. Comput. **3**(2), 104–115 (2010)
2. Chiang, M., Zhang, T.: Fog and IoT: an overview of research opportunities. IEEE Internet Things J. **3**(6), 854–864 (2016)
3. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D.: Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services. IEEE Trans. Serv. Comput. **3**(3), 223–235 (2010)
4. He, W., Yan, G., Da Xu, L.: Developing vehicular data cloud services in the IoT environment. IEEE Trans. Ind. Inform. **10**(2), 1587–1595 (2014)
5. Jin, J., Wang, W.H., Palaniswami, M.: Application-oriented flow control for wireless sensor networks. In: International Conference on Networking and Services, p. 71. IEEE (2007)
6. Karam, M., Payne, T., David, E.: Evaluating bluscreen: usability for intelligent pervasive displays. In: International Conference on Pervasive Computing and Applications, pp. 18–23. IEEE (2007)

7. Tsai, W.T., Sun, X., Balasooriya, J.: Service-oriented cloud computing architecture. In: International Conference on Information Technology: New Generations, pp. 684–689. IEEE (2010)
8. Wang, W.H., Palaniswami, M., Low, S.: Necessary and sufficient conditions for optimal flow control in multirate multicast networks. IEE Proc.-Commun. **150**(5), 385–390 (2003)

A Service-Based Declarative Approach for Capturing Events from Multiple Sensor Streams

Zhongmei Zhang[1,2,3](✉), Chen Liu[2,3], Xiaohong Li[1], and Yanbo Han[1,2,3]

[1] School of Computer Science and Technology, Tianjin University, Tianjin 300350, China
gloria_z@126.com, xiaohongli@tju.edu.cn, yhan@ict.ac.cn

[2] Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, North China University of Technology, Beijing 100144, China
liuchen@ncut.edu.cn

[3] Institute of Data Engineering, School of Computer Science and Technology, North China University of Technology, Beijing, China

Abstract. Existing event processing models require defining events in details beforehand. It is thus challenging to handle uncertainty associated with various sensor streams having dynamic interventions and correlations. In this paper, we improve our previous service abstraction which can increase the value density of primitive sensor streams in two aspects. To deal with the uncertainty, we add declarative rules in our service abstraction for adaptively generating events from different sensor streams that reflect various external stimuli. For extracting events dynamically, we utilize the correlation analysis method to treat events as variations of correlations. This paper reports the tryout use of our approach in Chinese power grid for detecting abnormal situations of power quality.

Keywords: Sensor stream · IoT service · Declarative rule
Correlation analysis

1 Introduction

With the rapid development of Internet of Things (IoT), numerous sensors are deployed and produce an overwhelming amount of stream data [1]. Individual sensor streams are usually fine-grained, have relatively lower value density, and intervene with each other dynamically. For obtaining more valuable information, many IoT applications try to capture events from sensor streams from multiple sources. And it is challenging to extract meaningful events from sensor streams with high efficiency and accuracy.

Event detection techniques [2–4] are already in use in various real-time enterprise solutions. State of the art event processing models provide frameworks to represent and reason events, but require working on fixed set of sensor streams and detailed definitions of event based on raw sensor streams [5]. However, the stream sources of events are usually uncertain. And facing large amount of sensor stream, it is impractical to define all detailed definitions of each variety of event. Hence, it is crucial to adaptively choose relative stream sources and capture events without relying on pre-defined event definitions.

We proposed a service abstraction, called proactive data service in our previous works [6, 7]. In this paper, we aim to improve it and propose a service-based declarative approach to deal with the above stated problems. Firstly, for dealing with the uncertain stream sources, we utilize declarative rules, which indicate the mapping between external stimuli and stream sources, to realize adaptive accessing and processing of sensor streams in proactive data services. Secondly, we utilize correlation analysis method to capture events as correlation's variations. In many fields, such as anomaly detection, electronic trading, etc., variation of correlation among sensor streams can be regarded as a kind of abnormal events. Facing the inconsistency of frequency and shift of correlation among multiple sensor streams, we propose a Dynamic Time Warping (DTW)-based algorithm to obtain lag-correlation. Relying on our declarative rules and the DTW-based event capturing algorithm, our proactive data service can capture events from dynamic sensor streams.

2 Scenario

In most modern electric devices of Chinese power grid, a series of sensors are deployed, such as *voltage, current, frequency*, etc. Each sensor is in charge of monitoring one indicator and produces a corresponding sensor stream. These sensor streams are affected by various disturbance sources, such as *wind farm*, *photovoltaic power station*, *electrified railway* etc., and show some abnormal statuses, i.e. originally correlated sensor streams become uncorrelated, which can be regard as events. Figure 1 shows some affections in sensor streams when various disturbances occur. The *three phase voltages* (*stream₁*, *stream₂*, and *stream₃*) are generally correlated. While when two trains are passing by an electrified railway, the correlation is changed, and an abnormal event "*unbalanced three phase voltage*" occurs.

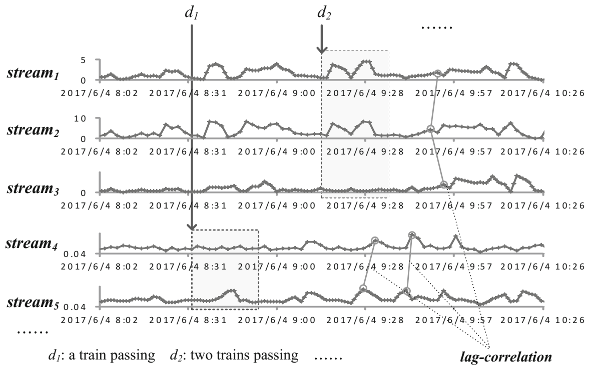


Fig. 1. Examples of disturbances' affection in sensor streams

It is challenging to capture events from multiple sensor streams. Firstly, the sensor stream sources of events are changeable with different disturbances. For example, disturbance d_1 affects $stream_4$, and $stream_5$, while disturbance d_2 affects $stream_1$, $stream_2$, and $stream_3$. One event usually occurs instantaneously with the occurrence of certain disturbance. Since the sensor stream is time-varying and irreversible, the choosing and processing of sensor streams must be reflected immediately. Secondly, the correlation among sensor streams may shift. Figure 1 shows that the trends of rising and falling of $stream_4$ and $stream_5$ are similar but not synchronous strictly. And the frequency and timestamps of data records in different sensor streams are different generally. For example, $stream_2$ and $stream_4$ have lower frequency than other streams.

3 The Declarative Service-Based Approach

3.1 The Proactive Data Service Model

Figure 2 shows the core artifacts and important terms in our previous service abstraction. **Sensor streams** are emitted by **stream sources**, e.g. *sensor*, *producer*, *publisher*, *emitter*, and so on, and consumed by clients, e.g. *users* or *applications*.

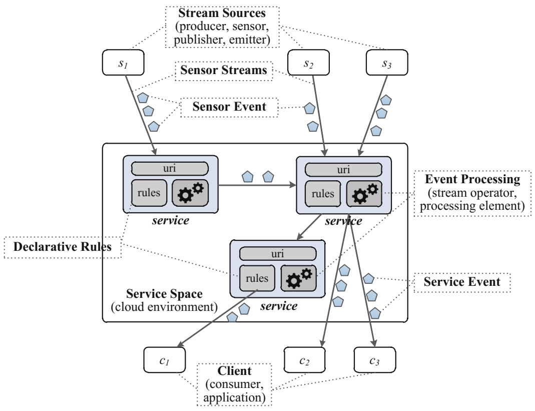


Fig. 2. Core artifacts and terminology in our method

A sensor stream can be defined as follow:

Definition 1 Sensor Stream. A Sensor stream can be represented as $ssd = \langle source_{id}, A, R\rangle$, in which $source_{id}$ is the id of stream source, A is the attribute sets, and $R = \{r_1, r_2, \ldots, r_i, \ldots\}$ is an infinite series of data record $r = \langle\{\langle a_i, v_i\rangle | a_i \in A\}, t\rangle$ which is a set of key-value pairs with a timestamp.

Typically, events are of a certain type, have a timestamp, and hold specific data. We regard each record with its source and timestamp in a sensor stream as a **sensor event**, which is generally fine-grained and with low value density. We abstract the ability of event processing as **proactive data services**, which can access and process multiple sensor streams and generate **service events** with higher value density.

We regard the external stimulus as a special kind of event that can affect the stream sources. For dealing with various stimuli, we utilize declarative rules to indicate the mapping between stream sources and external events. For capturing events from sensor streams, we utilize correlation analysis method which can work directly on sensor streams. Figure 3 shows the structure of our proactive data service model.

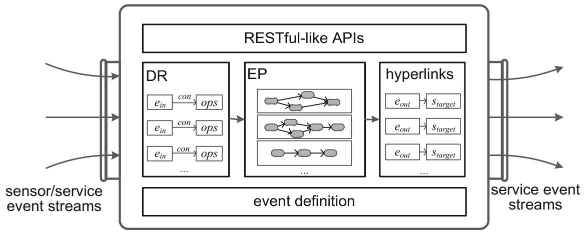


Fig. 3. The structure of the proactive data service

To simplify the service modeling, we limit possible stream sources that can be defined in declarative rules at present.

Definition 2 Proactive Data Service. A proactive data service can be formalized as $pds = \langle pds_{id}, SS, event_{in}, event_{out}, DR, EP, hyperlinks \rangle$, in which pds_{id} is the unique identifier, $event_{in}$ represents the input event streams, $event_{out}$ represents the output event streams generated by EP, SS is the possible stream source set, DR is the declarative rules, EP means the event processing functions, which include both the algorithm proposed in this paper and operations realized before, and the $hyperlinks$ is an optional parameter which indicates the targets of service events.

We define declarative rules based on Event-Condition-Action (ECA) rules, which are originally used in active database systems to provide event-driven, instaneous responses for conventional database systems [8]. In this way, the entire process requires no intervention from users or external applications. Presently, we utilize the Pearson Correlation Coefficient (PCC) [9] to measure the sensor streams' correlations for capturing the changes of correlation as events. The correlation among sensor data usually shifts in time, which can be regarded as lag-correlation [10]. We consider the lag-correlation analysis problem as to find the time lag vector to maximize the PCC, and propose a DTW-based event capturing algorithm in our service abstraction.

3.2 The Setting of Declarative Rules

The triggered action in a declarative rule needs to be a continuous process with an end point. Hence, we utilize an enhancement of ECA rule to represent the declarative rules. We designed and realized several operations, which can be divided into three classes: Transformation, Aggregation and Fusion [7]. Figure 4 shows an example of the declarative rule, in which the action is indicated based on these operations. The stimuli generated by a certain source can form an event stream. The condition is optional which indicates the constraint of certain kind of stimulus. The actions indicate the stream sources and corresponding processing behaviors. And the time period indicates how long the actions continue.

```
on     disturbance1
if     type equals one train passing
do     filter(rate_deviation,
              rate_deviation, >, deviationLimit)
       and filter(v_d,v_de, >, deLimit)
within 10 minutes
```

name	structure
disturbance1	(sourceid, type, timestamp)
rate_deviation	(rate_deviation,timestamp)
v_de	(v_de,timestamp)

Fig. 4. An example of the declarative rule

The bottom part of Fig. 4 shows the sensor streams and their structure involved in the declarative rule. The declarative rule includes one triggered action when an event occurs, i.e. a train is passing by certain disturbance source. The action indicates the sensor streams to be accessed and processed, i.e. **rate_deviation** and **v_de**, and the operations on them, i.e. filtering events which are out of limit.

3.3 The Event Capturing Based on Lag-Correlation Analysis

At present, we calculate the correlations between each two sensor streams, and once any two sensor streams' correlation is changed, an event is captured. For obtaining the correlation between two sensor streams, we keep analyzing limited sensor data in the latest window based on time-based slide window. The correlation among sensor streams usually shifts in time. For obtaining accuracy correlations, we aim to find the time lag vector to amend the PCC, and formalize the lag-correlation analysis problem as follows:

Definition 3 Lag-correlation Analysis Problem. given two sensor data sets E_i and E_j in a slide window, suppose $E_i = \{e_{i(t1)}, e_{i(t2)},\ldots,e_{i(tn)}\}$ and $Ej = \{e_{j(t1)}, e_{j(t2)},\ldots,e_{j(tn)}\}$, if there exists a time lag vector $\Delta = \{t'_1, t'_2, \ldots, t'_n\}$ makes

$$cor_{(E_i,E_j,\Delta)} = \frac{\sum_1^n \left(e_{i(t)} - \overline{e_i}\right)\left(e_{j(t+tk')} - \overline{e_j}\right)}{\sqrt{\sum_1^n \left(e_{i(t)} - \overline{e_i}\right)^2} \times \sqrt{\left(e_{j(t+tk')} - \overline{e_j}\right)^2}} \geq \delta_{cor},$$

in which δ_{cor} is a given threshold, we regard E_i and E_j are correlated.

To analyze the correlation between two sensor data, we can firstly obtain a time lag vector that makes $MAX\left(cor_{E_i,E_j,\Delta}\right)$, and then justify if there is a lag-correlation through comparing with given threshold.

We can calculate the time lag vector which makes the minimum Euclidian distance of the normalized series, and then calculate the PCC of the original series based on the time lag vector. DTW algorithm [11] is a robust method used to measure similarity of time series, which can shift and distort the time-series to ignore the problem of time axis scaling and shifting. In this paper we adopt a DTW-based algorithm to find the time lag vector of two sensor data series. To ignore the problem of amplitude scaling, we align sensor data series with DTW algorithm based on normalized sensor data series. To avoid excessive time warping, we also set a maximum value for the time lag vector.

We integrate the event capturing algorithm in our service abstraction, and set actions in the declarative rules based on it. In this way, when receiving certain stimulus, relative sensor streams can be accessed, the event capturing algorithm can be executed, and events can be captured once the correlation is changed.

4 Evaluation

In this section, we mainly evaluate the efficiency and effectiveness of our service abstraction.

4.1 Experiment Setup

The data set used in our experiments is from real sensor data in Chinese smart grid. We selected sensor data from 2017-05-01to 2017-06-05, and from 281 electric devices in certain province. We simulated the sensor stream for each sensor, strictly conforming to the real time stamps, and design following criteria:

(1) System load. The system load of the proactive data service can be divided into the load of CPU and memory.
(2) Event execution time. For each generated event, its execution time is the difference between the time when its corresponding sensor event is firstly received and the time when it is generated.
(3) Accuracy. The accuracy includes precision $precision = \frac{|D \cap T|}{|D|} \times 100\%$ and recall $recall = \frac{|D \cap T|}{|T|} \times 100\%$, in which, $D = \{d_1, d_2, \ldots, d_m\}$ is detected event list, and $T = \{t_1, t_2, \ldots, t_n\}$is the actual event list.

4.2 Experiment Result and Analysis

We firstly evaluate the efficiency of our service abstraction through comparing services with and without declarative rules. We set the numbers of stream source set in proactive data service as 250, 500, 1000, 1500, 2000, and 2500.

Figure 5 and 6 show the system loads and execution times of different service abstractions. It is obvious that the service abstraction with declarative rules has much lower system load and less execution time, and the increase of system load and execution time of our service abstraction are much slighter. This is because that the senor streams need to be processed in our service abstraction is much less.

For evaluating the effectiveness of our service abstraction, we calculated the precision and recall of events captured by DTW-based algorithm and normal correlation

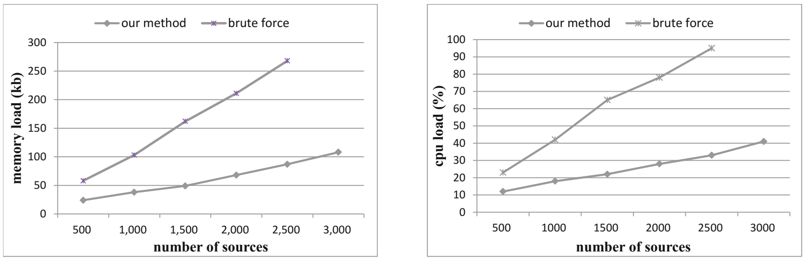


Fig. 5. The system loads of different service abstractions

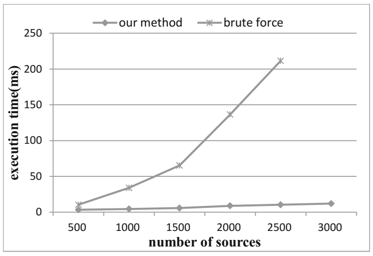


Fig. 6. The execution times of different service abstractions

analysis algorithm. We obtained 1783 abnormal events as the actual event list by consulting domain experts.

Figure 7 shows the average precision and recall of different event capturing algorithms. It is obvious that our event capturing algorithm has much higher precision and recall. Because we presently obtained the declarative rules from history sensor data analysis, our service abstraction cannot handle the dynamic correlations between disturbances and abnormal power quality events. It may affect the accuracy of our method. But still, our method has an average precision over 89%, an average recall over 91%, and increases the accuracy by about 20%.

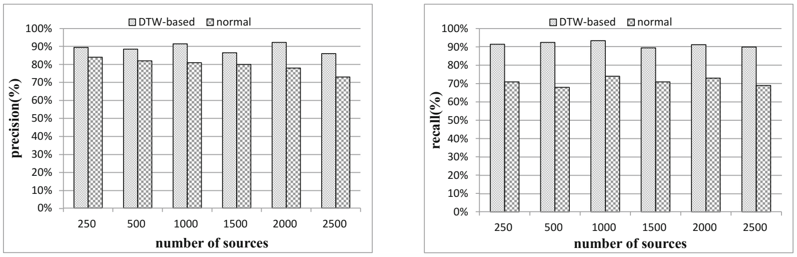


Fig. 7. The accuracy of different event capturing algorithms

5 Related Work

Event detection is one of the main objectives in sensor networks. A basic method is to detected events based on preset threshold value defined by domain expert [2]. To describe more complex event, Xue et al. [3] integrated the pattern-based approach with an in-network sensor query processing framework, and defined 5 common basic patterns to describe events. Various machine learning techniques have also been used to decide whether an event has occurred or not [4, 12]. However, the above pattern-based methods mostly based on pre-defined event patterns which indicated precise sources of sensor streams.

One trend to promote the development of IoT is to regard IoT as Web of Thing (WoT) where the open Web standards are supported for information sharing and device interoperation [13, 14]. Zeng et al. [13] provided a survey on research works in WoT domain, and providing composable services to enable physical-virtual mashups. Federia et al. [14] proposed a framework which supported developers to model smart things as web resources, and exposed them through RESTful APIs. Many works provided data service to provide the event detection ability [15, 16]. Most existing service-based methods needed to predefine the collaboration goals for service composition or collaboration, and didn't support adaptive collaboration.

6 Conclusion

It is meaningful to set up suitable abstraction to increase value density of sensor stream and promote its widespread use. Our previous service abstraction encapsulated fixed sensor streams, and only generated events based on predefined event definition. In this paper, we aim to propose a service-based declarative approach for capturing events from uncertain sensor streams through improving our previous service abstraction in two respects. We utilize declarative rules to deal with the uncertain stream sources, and utilize correlation analysis method to capture events as the variation of correlation. To verify the feasibility of our approach, we applied it for power quality event detection in Chinese Power grid. A series of experiments demonstrate that our method has much higher efficiency with high accuracy.

Acknowledgement. This paper is supported by National Natural Science Foundation of China (No. 61672042), Models and Methodology of Data Services Facilitating Dynamic Correlation of Big Stream Data, and National Key R&D Plan (No. 2017YFC0804406).

References

1. Heidemann, J., Stojanovic, M., Zorzi, M.: Underwater sensor networks: applications, advances and challenges. Philos. Trans. **370**(1958), 158–175 (2012)
2. Kapitanova, K., Son, S.H., Kang, K.D.: Using fuzzy logic for robust event detection in wireless sensor networks. Ad Hoc Netw. **10**(4), 709–722 (2012)

3. Xue, W., Luo, Q., Wu, H.: Pattern-based event detection in sensor networks. Distrib. Parallel Databases **30**(1), 27–62 (2012)
4. Singh, Y., Saha, S., Chugh, U.: Distributed event detection in wireless sensor networks for forest fires. In: Computer Modelling and Simulation, pp. 634–639. IEEE, Cambridge (2013)
5. Patri, O.P., Panangadan, A.V., Sorathia, V.S.: Sensors to events: semantic modeling and recognition of events from data streams. Int. J. Semant. Comput. **10**(04), 461–501 (2017)
6. Han, Y., Wang, G., Yu, J.: A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in China. IEEE Trans. Intell. Transp. Syst. **17**(9), 2648–2657 (2016)
7. Han, Y., Liu, C., Su, S.: A decentralized and service-based approach to proactively correlating stream data. In: S2 International Conference on Internet of Things, pp. 93–100 (2016)
8. Paschke, A.: ECA-RuleML: an approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. CoRR, abs/cs/0610167 (2006)
9. Wu, S., Lin, H., Wang, W.: RLC: ranking lag correlations with flexible sliding windows in data streams. Pattern Anal. Appl. **20**(2), 601–611 (2017)
10. Guo, T., Sathe, S., Aberer, K.: Fast distributed correlation discovery over streaming time-series data. In: ACM International on Conference on Information and Knowledge Management, pp. 1161–1170 (2015)
11. Kate, R.J.: Using dynamic time warping distances as features for improved time series classification. Data Min. Knowl. Discov. **30**(2), 283–312 (2016)
12. Cui, W., Wang, P., Du, Y.: An algorithm for event detection based on social media data. Neurocomputing **254**, 53–58 (2017)
13. Zeng, D., Guo, S., Cheng, Z.: The web of things: a survey. J. Commun. **6**(6), 424–438 (2011)
14. Paganelli, F., Turchi, S., Giuli, D.: A web of things framework for RESTful applications and its experimentation in a smart city. IEEE Syst. J. **10**(4), 1412–1423 (2017)
15. Li, S., Son, S.H., Stankovic, J.A.: Event detection services using data service middleware in distributed sensor networks. In: Zhao, F., Guibas, L. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 502–517. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36978-3_34
16. Gao, F., Curry, E., Ali, M.I., Bhiri, S., Mileo, A.: QoS-aware complex event service composition and optimization using genetic algorithms. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 386–393. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_28

Response Time Aware Operator Placement for Complex Event Processing in Edge Computing

Xinchen Cai[1], Hongyu Kuang[1], Hao Hu[1](✉), Wei Song[2], and Jian Lü[1]

[1] State Key Lab for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China
smldcxx@126.com, {khy,myou,lj}@nju.edu.cn
[2] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu, China
wsong@njust.edu.cn

Abstract. A typical complex event processing (CEP) service is composed by a set of operators organized as a directed acyclic graph. This kind of service is usually used to handle large amounts of real-time data. Meanwhile, edge computing has been widely accepted as a new paradigm to improve the QoS of deployed services by making the services closer to the data. Thus, the response time, which is a crucial QoS metric for CEP services, can be significantly reduced by deploying CEP services on the edge network. However, it is often unlikely for a single node of the edge network to host all operators of a CEP service due to the limited computing resources. Therefore, it is desirable for a CEP service to place its operators on different nodes of the edge network to keep the response time low, especially when the input rate of the CEP service significantly increases. In this paper, we reduce the average response time of CEP services by deploying the operators on the edge nodes dynamically according to the predicted response time of CEP services. Specifically, we first propose a system model to capture the response time of the CEP services, based on which we formulate the problem of the optimal placement of CEP operators in the edge network. We then propose an algorithm that predicts the response time of CEP services and deploys the operators on the edge nodes with the minimum predicted delay. A simulation-based evaluation demonstrates that, compared with two state-of-the art algorithms, our algorithm can reduce the total response time by 33% and 45% on average, respectively.

Keywords: Complex event processing · Edge computing
Operator graph · Operator placement · Response time

1 Introduction

Typically, a complex event processing (CEP) service is used to handle large amounts of real-time data by using a set of operators organized as a directed acyclic graph called *operator graph*. In practice, CEP services are usually deployed on cloud data centers. However, this deployment results in delivering data from the edge of the network to remote cloud data centers and thus seriously reduces QoS of CEP services due to the long distance between data sources and remote cloud data centers, as well as the mass data transmission.

To address this issue, a new paradigm, called edge computing [12], has been proposed to improve the QoS of deployed services by moving services from cloud data centers to the edge of network. Consequently, a growing body of work [2,4,14] focuses on deploying CEP services on the edge network (i.e., nearest to the user [14]) to improve the QoS of CEP services, e.g., to reduce the response time. However, due to the limited computing capacity, lots of users queries, and large amounts of input events, congestion can easily occur at the nearest edge node and increase the response time of CEP services dramatically. Therefore, it is important to be aware of the response time of a CEP service during the placement of its operators on suitable nodes of the edge network to keep the response time low, especially when the input rate of the CEP service significantly increases. We use the following example to demonstrate this situation.

Consider the example shown in Fig. 1. This figure depicts the placement of a CEP service that queries traffic accident events, which are defined as a lane switch event after a decreased speed event. The CEP service has an operator graph consisting of a lane operator, a speed operator, and an accident operator. The edge network contains three edge nodes: v_0, v_1, and v_2. The resources capacity of the nodes v_0, v_1, and v_2 are 1, 2, and 2, respectively. The resources refers to computing resources, such as CPU, etc. Connected devices deliver and cache original data to the nearest node v_0 in the edge network. An intuitive strategy of reducing the response time is to deploy operators on the nearest edge node [14]. As Fig. 1 shows, the lane operator is deployed on the nearest edge node v_0. At this moment, v_0 has no resources to host other operators. Thus, the speed operator and the accident operator are deployed on the edge node v_1 which is the second nearest edge node except v_0. However, when the input rate of the CEP services largely increases, large amounts of data have to be delivered from the edge node v_0 to v_1. The congestion will occur and result in increasing the response time. On the other hand, another placement strategy is to deploy operators distributed in the edge network to balance the load between edge nodes in advance [6]; for example, deploying the accident operator to the edge node v_2 to reduce the load of v_1. When the input rate is high, this placement can reduce the response time compared to the first placement. However, when the input rate remains low, this placement increases the transmission delay between edge nodes and makes the response time even worse. In this paper, we argue that it is important to balance the load of different edge nodes during the placement of CEP operators by being aware of the response time of the operator graphs.

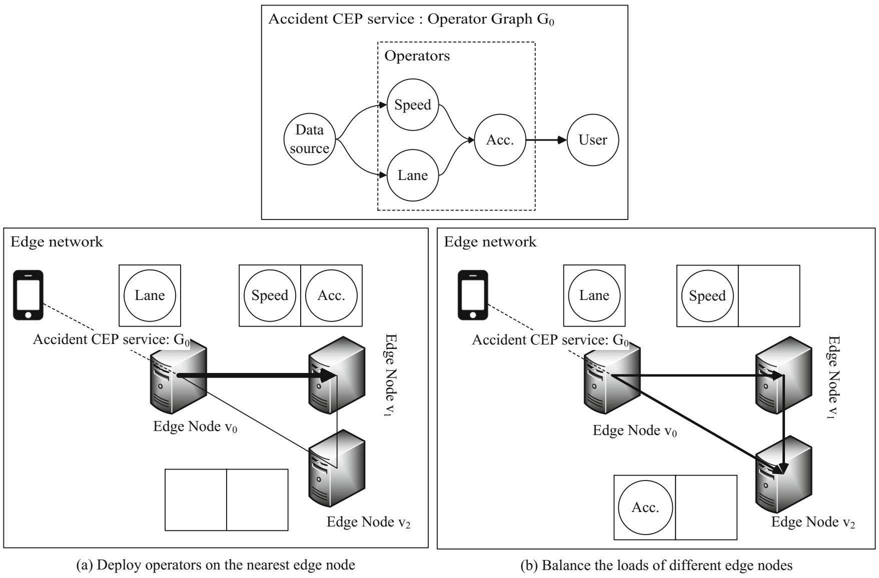

Accident CEP service : Operator Graph G_0
Operators
Data source
Speed
Lane
Acc.
User
Edge network
Lane
Speed
Acc.
Accident CEP service: G_0
Edge Node v_0
Edge Node v_1
Edge Node v_2
(a) Deploy operators on the nearest edge node
(b) Balance the loads of different edge nodes

Fig. 1. Deploy operators of a CEP service that queries accident events on the edge network.

To achieve this goal, we first propose a combined model including CEP model and edge model to capture the response time of CEP services. Based on the models, we predict the delay of these edge nodes according to the monitored information. We then prove the optimal operator placement for CEP in edge computing is NP-hard. Finally, we propose a novel approximation-based algorithm that deploys the operators on the edge node with the lowest predicted delay. We generate 100 operator graphs and 30 edge networks for our simulation-based evaluation. The result demonstrates that our approach is able to maintain low response time when the input rate of CEP services significantly increases.

This paper makes the following three contributions: (1) We propose a system model to capture the response time of CEP services, and formulate a novel optimization problem, that is, to find an optimal placement of CEP operators in the edge network such that the response time of the CEP services is minimized. (2) We prove the optimal operator placement problem for CEP in edge computing is NP-hard. (3) We propose a novel response time aware operator placement algorithm that keeps the average response time of the operator graphs low.

The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 describes the system models and problem formulation. Section 4 presents our response time aware operator placement algorithm. Section 5 reports the evaluation results, and finally, Sect. 6 concludes the paper.

2 Related Work

Operator placement problem has been widely studied in the last decades. The basic form of the *operator placement problem* is stated as: given a network of N nodes with some or all of them generating data processed by an operator, place the operator in the network so as to reduce the network traffic [16]. Several placement algorithms [2,10,11,16] have been proposed. These algorithms are characterized by different assumptions and optimization goals. Pietzuch et al. [10] design a stream-based overlay network for operator placement in distributed stream-processing (DSP) systems that minimizes the network usage of a query, while keeping the query delay low and picking nodes with adequate bandwidth. However, their operator model is not organized as operator graphs. Rizou et al. [11] present a distributed placement algorithm that minimizes the bandwidth-delay product of data streams between operators. However, their approach does not take the resources of the nodes into consider. Tziritas et al. [16] propose an approach enabling both single and group operator migrations using evictions of hosted operators. Although this work takes resources of nodes in the network into consider, it focuses on how to migrate operators to improve a placement of operators. Cardellini et al. [2] propose a general formulation of DSP placement in distributed network. This work focuses on the scenes that data sources are distributed in large scale network and do not take the input rate of CEP service into consideration. In contrast, we focus on deploying CEP services to process data near an edge node to infer meaningful events, especially when the input rate of CEP services significantly increases.

Edge computing is the computational infrastructures that make services closer to the end users [12]. The same concept is also called as cloudlet [13] or fog computing [1]. Taneja et al. [15] conduct experiments to demonstrate that edge computing can effectively improve its QoS. Hong et al. [4] proposed an opportunistic spatio-temporal CEP hosted by edge computing. As far as we know, this is the earliest work to deploy CEP on edge computing. How to offload applications to edge nodes [8,18] is another research interest that quite close to the operator placement problem in edge computing. The goal of this work is to minimize the cost of a user. However, our goal is to minimize the average response time of the CEP services queries. Saurez et al. [14] propose an incremental deployment approach to greedily deploy operators on the nearest edge node. However, they all do not consider the change of the input rate. Jia et al. propose an user to edge computing allocation in wireless metropolitan area networks [5] and a load balance approach to finding an optimal redirection of tasks between edge nodes [6]. However, the task model of this work has no graph structure. Our work is designed for CEP operator graphs whose input rate change according to the data sources.

3 Models and Problem Formulation

In this section, we first discuss our CEP model and edge computing model. Then we use these models to calculate the response time of CEP services, to give the

Table 1. Main notation adopted in the paper

Symbol	Description
$G_{edge} = (V_{edge}, E_{edge})$	The edge network G_{edge} consists of the edge nodes V_{edge} and the connections between edge nodes E_{edge}
v_i	The ith edge node in V_{edge}
(v_i, v_j)	The connection between the edge nodes v_i and v_j
c(v_i)	The resources capacity of the edge node v_i
$\lambda(v_i)$	The execution rate of the edge node v_i
w(v_i, v_j)	The transmission rate between edge nodes v_i and v_j
$G_{cep} = (\Omega$, L)	An operator graph consists of the operators Ω and the event streams L
c(ω_i)	The resources requirement of the operator ω_i
$T_r(X)$	The response time of the operator graph according to a placement X
T_p	The end-to-end delay of a path
$X(\omega_i) = v_u$	Deploy the operator ω_i on the edge node v_u
θ_t	The thread of response time to redeploy operator graphs
Δ_t	The time interval of replacement judgement

operator placement problem statement, and to prove this problem is NP-hard. Table 1 summarizes the symbols that are used in this paper.

3.1 CEP Model

We assume that N_{cep} CEP services queries are sent to an edge node. A CEP service can be represented as an operator graph $G_{cep} = (\Omega, L)$. Ω denotes the set of operators. L denotes the event streams between the data sources, the operators, and the consumer. Each operator $\omega_i \in \Omega$ has the attributes c(ω_i), denoting the amount of resources required for the operator ω_i execution. The resources required to execute an operator include CPU, memory, etc. We simplified these resource models as unit resources, like other works [2,6].

For example, in Fig. 1, Ω contains the speed operator ω_{speed}, the lane operator ω_{lane}, and the accident operator $\omega_{accident}$. L contains (source, ω_{speed}), (source, ω_{lane}), (ω_{speed}, $\omega_{accident}$), (ω_{lane}, $\omega_{accident}$), and ($\omega_{accident}$, user). c(ω_{speed}), c(ω_{lane}), and c$(\omega_{accident})$ are all one-resource units.

3.2 Edge Computing Model

Edge nodes are represented as vertexes V_{edge}. The network connections between edge nodes are represented as E_{edge}. The edge network are represented as G_{edge} = (V_{edge}, E_{edge}). Every node v_i in the edge network has two attributes: (1) c(v_i), the amount of resources available in the edge node v_i; (2) $\lambda(v_i)$, the events execution rate of the edge node v_i. Every network connection (v_i, v_j) has the attribute w(v_i, v_j), the transmission rate between the edge nodes v_i and v_j.

When a user sends a query to the nearest edge node for computing resources, the edge node manages the placement of the operator graph. This edge node is called as *manage edge node*. The manage edge node monitors the information of the nearby edge nodes within H hops. These nearby edge nodes are called *candidate edge nodes*. The operators can be placed on either the manage edge node or candidate edge nodes. Thus, the size of the edge network is limited by the number of hops H. H becomes larger when the operators have no suitable edge nodes to be placed.

3.3 Response Time

The response time of an operator graph G_{cep} can be calculated as:

$$T_r(G_{cep}) = \max_{p \in \pi_{G_{cep}}} T_p \tag{1}$$

where $\max_{p \in \pi_{G_{cep}}} T_p$ means the worst end-to-end delay from a data source to the consumer. T_p denotes the delay of a path in the operator graph G_{cep}. The path p can be represented as $(\omega_{p_1}, \omega_{p_2}, \ldots, \omega_{p_{n_p}})$, where n_p denotes the number of operators in p. These operators are deployed on the edge nodes $(v_{p_1}, v_{p_2}, \ldots, v_{p_{n_p}})$.

For example, in Fig. 2, the end-to-end delay of $path_1$ is the worst. Thus, the response time of operator graph G_0 is $T_r(G_0)$ that equals to 120 ms. $path_1$ is (source, ω_{speed}, $\omega_{accident}$, user). In Fig. 1(a), X($path_1$) = (v_0, v_1, v_1, v_0).

For T_p, we have:

$$T_p = \sum_{i=0}^{n_p-1} d(v_{p_i}, v_{p_{i+1}}) + \sum_{i=0}^{n_p} T_E(\omega_i) + \sum_{i=0}^{n_p} T_q(\omega_i) \tag{2}$$

where $d(v_{p_i}, v_{p_{i+1}})$ denotes the transmission delay between the edge nodes v_{p_i} and $v_{p_{i+1}}$. $T_E(\omega_i)$ denotes the processing delay of events in the operator ω_i. $T_q(\omega_i)$ denotes the queuing delay of events in the operator ω_i. In addition to the above three delays, there is also the propagation delay. However, in the edge network, the propagation delay, which is too low, can be ignored.

The transmission delay $d(v_{p_i}, v_{p_{i+1}})$ can be calculated as:

$$d(v_{p_i}, v_{p_{i+1}}) = \frac{sz}{w(v_{p_i}, v_{p_{i+1}})} \tag{3}$$

where sz is the size of an event packet.

The processing delay of events in the operator ω_i can be calculated as:

$$T_E(\omega_i) = \frac{1}{\lambda_(v_{p_i})} \tag{4}$$

where the average event process rate of v_{p_i} is $\lambda(v_{p_i})$ events per second (eps).

The queuing delay mainly depends on the congestion level of the router. The event rate from v_{p_i} to $v_{p_{i+1}}$ can be represented as $r(p_i, p_{i+1})$ eps. The number of channel between edge node v_{p_i} and $v_{p_{i+1}}$ is n$(v_{p_i}, v_{p_{i+1}})$. The queuing delay of an operator ω_i in a period t is calculated as:

$$T_q(\omega_i) = \begin{cases} \frac{t \times sz \times r(p_i, p_{i+1})}{w(v_{p_i}, v_{p_{i+1}})}, & \text{if } E_c(p_i) > 1 \\ \frac{E_c(p_i)}{w(v_{p_i}, v_{p_{i+1}}) - r(p_i, p_{i+1})} + \frac{1}{w(v_{p_i}, v_{p_{i+1}})}, & \text{if } E_c(p_i) <= 1 \end{cases} \tag{5}$$

$$E_c(p_i) = E_c(n(v_{p_i}, v_{p_{i+1}}), \frac{sz \times r(p_i, p_{i+1})}{w(v_{p_i}, v_{p_{i+1}})}) \tag{6}$$

$$E_c(n, u) = \frac{\frac{u^n}{n!}}{\frac{u^n}{n!} + (1 - u/n) \sum_k^{n-1} \frac{u^k}{k!}} \tag{7}$$

Equation 7 is known as Erlang's C formula to calculate the queuing delay [7].

If the bandwidth is larger than the event rate in the connection, the queuing time is calculated as Erlang's C formula. Otherwise, the queuing time is calculated as $\frac{N_t}{w(v_{p_i}, v_{p_{i+1}})}$, $N_t = t \times sz \times r(p_i, p_{i+1})$. N_t is the size of the data sent to the queue in a period t. To avoid frequent replacements caused by excessive changes in the input rate of operator graphs, we calculate the average event rate in the last ten seconds as r(p_i, p_{i+1}).

Initially, when an operator ω_i has not been decided where to be deployed, the transmission delay is calculated as $\frac{sz}{\bar{w}}$, where $\bar{w}$ denotes the average transmission rate of the connections in the edge network. The execution time is calculated as $\frac{1}{\bar{\lambda}}$, where $\bar{\lambda}$ denotes the average event process rate of edge nodes.

3.4 Operator Placement Problem in Edge Computing

When the manage edge node receives a CEP service query, the operator graph is pushed into an operator-graphs-queue Q = $\{G_{cep}(1), \ldots, G_{cep}(N_{cep})\}$. Optimal operator placement problem in edge computing consists in determining a suitable mapping between operators Ω and edge nodes V_{edge} to minimize the average response time of operator graphs. For every operator ω_i, we can get an edge node v_u that place ω_i on v_u, to minimize the average response time $T_r(Q)$:

$$T_r(Q) = \frac{\sum_{k=1}^{N_{cep}} T_r(G_{cep}(k))}{N_{cep}} \tag{8}$$

and to satisfy:

$$\forall v_u \in V_{edge}, c(v_u) \geq \sum_{i=1}^{|\Omega|} ((X(\omega_i) == v_u)?c(\omega_i) : 0) \tag{9}$$

Then, we prove this optimal problem is a NP-hard problem.

Theorem 1. *Optimal operator placement problem for CEP in edge computing is an NP-hard problem.*

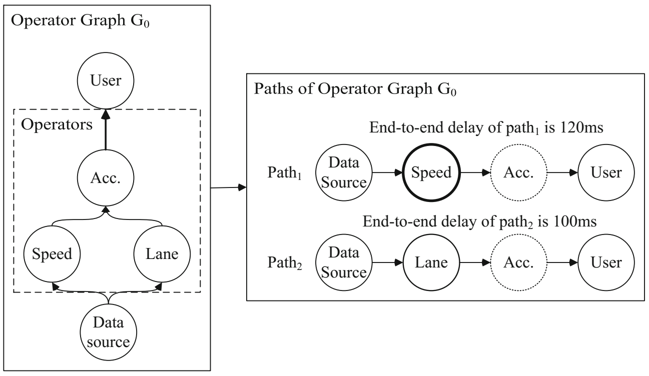


Fig. 2. Algorithm 1 preferentially deploys the speed operator.

Proof. First, we prove the decision problem of the optimal operator placement problem is NP-hard. The decision problem is stated as: N_{cep} operator graphs containing $|\Omega|$ operators and an edge network, *can the operator graphs have a feasible placement?* In the special case: the input rate of N_{cep} operator graphs are the same; the edge network has only two edge nodes, $V_{edge} = \{v_i, v_j\}$, with capacity $c(v_i) = c(v_j) = (\sum_{k=1}^{|\Omega|} c(\omega_k))/2$; $w(v_i, v_j)$ is infinite, $\lambda(v_i) = \lambda(v_j)$. The resulting problem is the Partition problem [3] which is known to be NP-hard. Finally, because the special case is NP-hard, the general decision problem is NP-hard as well. The optimization problem is at least as hard as the decision problem. Thus, optimal operator placement problem for CEP in edge computing is an NP-hard problem.

Thus, if P $\neq$ NP, optimal operator placement problem for CEP in edge computing cannot be solved in polynomial time. Thus, we propose an approximation-based algorithm to solve this problem.

4 Algorithm

In this section, we propose a novel algorithm that balances the response time of different paths to achieve the minimum average response time of operator graphs (Eq. 8). The basic idea is to predict the end-to-end delay of all paths, and then improve the placement of the path with the worst end-to-end delay, which can directly improve the response time of the operator graph. The algorithm deploys the operator logically closest to data source in the path on the edge node with the minimum delay.

4.1 Response Time Aware Operator Placement Algorithm

The response time of the operator graphs is calculated by Eq. 1. Algorithm 1 first calculates the response time of different paths in an operator graph, and

Algorithm 1. Response Time Aware Operator Placement

```
Input: Q = ($G_{cep}(1)$,$G_{cep}(2)$,...,$G_{cep}(N_{cep})$),$G_{edge}$
Output: X(Q)
 1: //initial parameters
 2: for all i = 1 $\rightarrow N_{cep}$ do
 3:     Get all paths p[i][] in $G_{cep}(i)$ //the operators in p is in the order of from the
        user to the data source
 4:     $N_p(i)$ = total number of paths in $G_{cep}(i)$
 5: end for
 6: X = $\emptyset$
 7: for i = 1 $\rightarrow N_{cep}$ do
 8:     for j = 1 $\rightarrow N_p(i)$ do
 9:         In[i][j] = $G_{cep}(i).Source$ ;
10:     end for
11: end for
12:
13: // make placement decision for every operator
14: while $\exists$ X($\omega \in \Omega$) == $\emptyset$ do
15:     $g_i$ = the index of the operator graph with the max response time;
16:     $p_i$ = the index of the path with the max response time in $G_{cep}$(i)
17:     $\omega_{src}$ = In[$g_i$][$p_i$] //the operator whose output is the input of $\omega_{tar}$
18:     $\omega_{tar}$ = (p[$g_i$][$p_i$]).top //the operator connecting to the output stream of $\omega_{src}$
19:     $v_{tar}$ = the edge node with the minimum delay to $\omega_{src}$
20:     if $v_{tar}$ == $\emptyset$ then //cannot find an edge node that has resources
21:         H = H + 1 ;
22:         Restart Algorithm.
23:     else
24:         X($\omega_{tar}$) = $v_{tar}$ // place $\omega_{tar}$ on $v_{tar}$
25:         In[$g_i$][$p_i$] = $\omega_{tar}$ //update input streams
26:         p.erase($\omega_{tar}$) //$\omega_{tar}$ has been deployed
27:         c($v_{tar}$) = c($v_{tar}$) - c($\omega_{tar}$) //calculate new capacity of $v_{tar}$
28:     end if
29: end while
30: return X
```

then improves the placement of the path with the largest end-to-end delay. We first find the operator graph g_i with the maximum response time (line 15) and the path p_i with the maximum end-to-end delay in gi (line 16). The end-to-end delay of the path p_i limits the response time of the operator graph g_i. Thus, we try to improve the placement of p_i. Then, we find the operator ω_{tar} whose input stream is the output stream of last deployed operator ω_{src} in the path p_i, and the edge node v_{tar} that has the minimum delay of the connection to ω_{src}. Algorithm 1 deploys the operator ω_{tar} (line 18) on the edge node v_{tar} (line 19).

For example, in the case of Fig. 1, at the moment that only the operator ω_{lane} is deployed, Fig. 2 shows how to select the operator ω_{tar} in the operator graph. Because the response time of $path_1$ is the largest response time in the paths ($path_1$ and $path_2$) and ω_{speed} is the undeployed operator connecting to the output stream of data source in $path_1$, the Algorithm 1 preferentially makes placement decision on ω_{speed}. Then, only the edge nodes v_1 and v_2 have capacity to host ω_{speed}. The algorithm first assumes that ω_{speed} is deployed on v_1 and v_2, calculates the response time of these two placements, and selects the placement with the minimum response time. Thus, ω_{speed} is deployed on the edge node v_1.

If all edge nodes within H hops have no resources, return $\emptyset$ (line 20). The hops number of candidate edge nodes should be increased (line 21). the algorithm is then restarted based on the new monitored information (line 22).

When the algorithm gets ω_{tar} and v_{tar}, the request is sent to the edge node v_{tar} for hosting the operator ω_{tar}. If v_{tar} accepts the request, system deploys the operator ω_{tar} on the edge node v_{tar} (line 24). Otherwise, the algorithm deletes the edge node and finds v_{tar} again.

The manage edge node monitors information every Δ_t second. If the response time of an operator graph exceeds the thread θ_t, the manage edge node redeploys operators to improve the response time of the operator graphs. To avoid fluctuation in the edge network transmission or the input rate of the operator graphs, this replacement judgement is performed every Δ_t second.

4.2 Time Complexity

Algorithm 1 makes placement decision for every operator. The number of operators is $|\Omega|$. For every operator graph, Algorithm 1 calculates its response time. The number of operator graphs is N_{cep}. The length of an path p_i is n_{p_i}. The time complexity of calculating the response time for an operator graph is O($\sum_{i=1}^{N_p} n_{p_i}$. The largest number of operators in an operator graph is $|G_{cep}|$. The worst case time complexity of calculating the response time for an operator graph is O($|G_{cep}|^2$). The time complexity of finding the edge node with minimum delay is O($|V_{edge}|$). Thus, the time complexity of Algorithm 1 is O($|\Omega| \times |G_{cep}|^2 \times N_{cep} + |\Omega| \times |V_{edge}|$).

5 Evaluation

We conduct simulations by using Omnet++ [17] that is a network simulation tool widely used in the field of the discrete event simulation, e.g. the simulation of MCEP [9]. There is no widely accepted real data set for CEP currently. By using Omnet++, we change the input rate of the CEP services to evaluate the performance of our algorithm in different workloads. We discuss the results of the simulations in this section.

5.1 Reference Algorithms

As a reference for the results achieved by response time aware operator placement algorithm, we compare our algorithm with two baseline algorithms: greedy algorithm and load balance algorithm. Because both the load balance algorithm and the greedy algorithm do not consider the application structure, we formulate the following rule for both algorithms: the order of placing operators is determined by the shortest logical distance from the data source.

Greedy Algorithm. Greedy algorithm [14] deploys operators on the edge node closest to the input streams of the operators except the edge nodes which do not have sufficient resources.

Load Balance Algorithm. Load balance algorithm [6] calculates the average response time of every edge node, and then reduces the load of the overloaded edge nodes by redirecting some of the operators to the underloaded edge nodes.

5.2 Simulation Environment

We randomly generate 30 different topologies of edge networks and 100 operator graphs to evaluate the response time aware operator placement algorithm. We run two sets of simulation. In the first set of simulations, we run our algorithm and two baseline algorithms in the different edge networks where the same operator graphs are deployed. The initial capacity of edge nodes are 3, 4, and 5 resource units, respectively. In the second set of simulations, we run the algorithms in a network where different operator graphs are deployed. The network is the one whose simulation result is the most similar to the average result in the first set simulation. In this network, the resources capacity of the edge nodes is 5 resource units. We report the average results of these simulations.

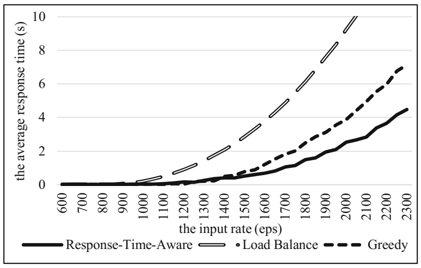
We set $\theta_t = 1\,\mathrm{s}$ and $\Delta_t = 20\,\mathrm{s}$. Every 20 s, if the response time of the operator graphs exceeds 1 s, the manage edge node redeploys operators to achieve lower response time.

Edge Networks. Considering the goal of this paper and the core of our proposed algorithms, i.e., deploying operators on the nearby edge nodes to keep the response time low, relatively small sized edge networks are sufficient for our evaluation. Specially, we set up three different kinds of 10-node edge networks. The first kind is a ring; the second kind is generated randomly with 15 links; and the third kind is generated randomly with 20 links. We repeat 10 times simulations on each network. In each simulation, we randomly choose a node in the edge network as the manage edge node. The transmission rate is generated randomly between 10 Mbps and 20 Mbps. The manage edge node initially monitors nearby edge nodes within 2 hops.

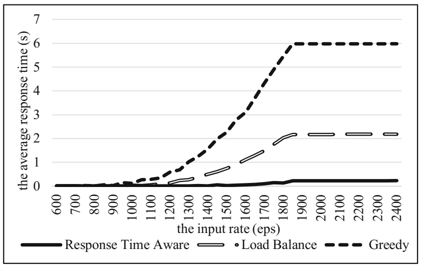
CEP Operator Graphs. To simulate different structures of operator graphs, 100 operator graphs are generated randomly, 10 of which contains 3 operators, 30 of which contains 4 operators, and 60 of which contains 5 operators. In each simulation, only one kind of operator graphs is sent to the manage edge node and the number of operator graphs is 3. For each operator, 25% of events conform to user-defined complex events sent to output streams. The resources capacity required for an operator execution is one-resource units. We gradually increase the input rate of the operator graphs to simulate a system from idle to busy.

5.3 Simulation Results

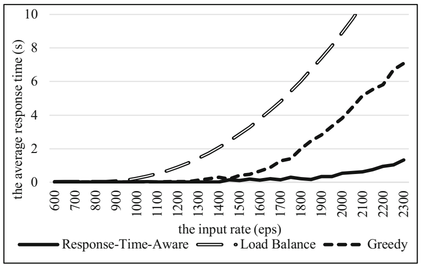
As Fig. 3(a), (b), and (c) shows, in the first set of simulation, our algorithm performs the best among the three algorithms. On average, our algorithm can reduce the total response time by 76% compared to the greedy algorithm and by 82% compared to the load balance algorithm. Because of the extra monitored information about the edge network, our algorithm adapts well to different networks.



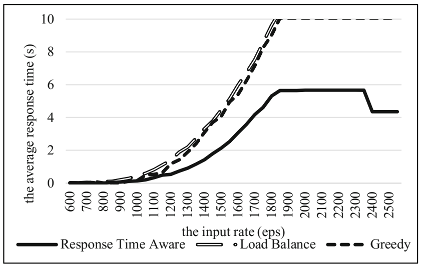
(a) The edge node capacity is 3.



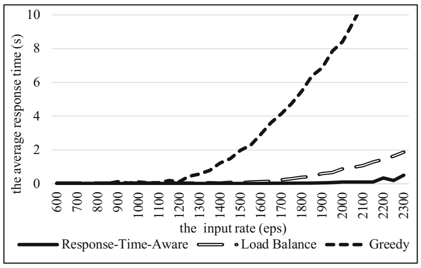
(a) An operator graph has 3 operators.



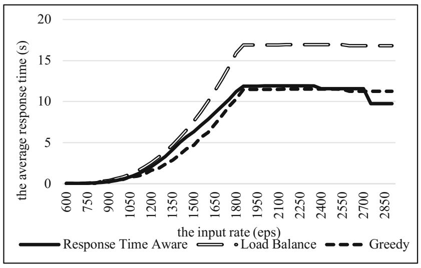
(b) The edge node capacity is 4.



(b) An operator graph has 4 operators.



(c) The edge node capacity is 5.



(c) An operator graph has 5 operators.

Fig. 3. Place the same operator graphs on the different edge networks.

Fig. 4. Place different operator graphs on the same network.

When the capacity of edge nodes is limited, load balance algorithm performs the worst (Fig. 3(a), (b)). In this case, congestion is inevitable. Load balances algorithm redirects operators to other edge nodes, which makes data have to be delivered to the operators distributed in the congested edge nodes. The extra network traffic aggravates the congestion. In contrast, our algorithm predicts the delay to the edge nodes, and deploys operators on the edge node with the lowest delay to keep the low response time. When the capacity of the edge nodes becomes sufficient, greedy algorithm performs the worst. Because the average load of edge nodes can be reduced by more resources, the load balance performs much better in this situation. Greedy algorithm makes operators gathered around the data source to reduce transmission delay. However, when the input rate increases, the congestion occurs due to large amounts of data are sent to the same edge node. Our algorithm avoids this by predicting the delay to different edge nodes and deploying operators on the edge node with low delay.

As Fig. 4(a), (b), and (c) shows, in the second set of simulation, our algorithm performs the best among the three algorithms. On average, our algorithm can reduce the response time by 33% compared to greedy algorithm and by 45% compared to load balance algorithm. We find that congestion is more likely to happen if the operator graph has more paths, because more events from different input streams are sent to an operator.

When the number of operators increases, the response time becomes higher because there are more transmission between operators. The performance of load balance algorithm becomes the worst in Fig. 4(c), because the algorithm distributes more operators than before to balance the load of edge nodes, which increases the transmission delay.

In Fig. 4(c), when the event input rate is low, our algorithm performs better than the other two algorithms. When the input rate increases, our algorithm performs worse than greedy algorithm, because the input rate increases faster than the prediction of our algorithm. However, In Fig. 4(b) and (c), when response time exceeds thread θ_t, the manage edge node updates the information about the edge network and the CEP services (including the input rate), and then redeploys the operator graphs resulting in the response time decreasing. These results show that our algorithm can adapt to the dynamic environment, improve the operator placement, and reduce the response time of CEP services.

Besides the response time, we also use bandwidth-delay to measure network usage as an additional criterion. The lower bandwidth-delay product indicates that the network load is lower [11]. Figure 5 shows the total network usage after running the three algorithms. With operator number increasing in an operator graph, due to the network transmission between different operators increases, the network usage increases. Our algorithm performs the best because the algorithm gives priority to the operator graphs with large input rate. Load balance algorithm performs the worst because the algorithm distributes the operators in the edge network resulting in more network transmission.

5.4 Threats to Validity

Construct Validity. In the simulations, we observe that our algorithm can reduce the response time of the operator graphs deployed on the edge networks. Because the problem is NP-hard, we cannot get the optimal solution in polynomial time. Our algorithm greedily deploys the selected operators on the edge

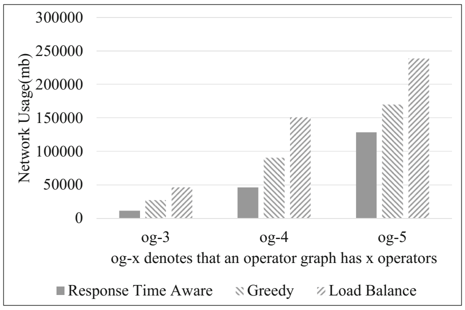


Fig. 5. Network usage versus different operator number.

node with the minimum delay, which is an approximate solution. We can reduce threats to construct validity by searching more edge nodes instead of only the edge node with the minimum delay.

External Validity. The threats to external validity come from our simulation environment. Although we gradually increase the input rate of operator graphs, dynamic network environment cannot be fully simulated.

6 Conclusions

We study response time aware operator placement for CEP in edge computing to reduce the average response time of operator graphs. Since the optimal operator placement problem is NP-hard, we present an approximation-based algorithm to ensure the response time of operator graphs low. The evaluation results show that our algorithm outperforms other approaches in the average response time of operator graphs. We plan to improve our algorithm by searching more placements and use real-world data in the experiments on actual systems in the future.

Acknowledgments. This work is supported by the National Key R&D Program of China (Grant No. 2017YFB1001801), National Natural Science Foundation of China (Grant No. 61690204, No. 61761136003), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16 (2012)
2. Cardellini, V., Grassi, V., Presti, F.L., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pp. 69–80 (2016)

3. Hartmanis, J.: Computers and intractability: a guide to the theory of NP-completeness (Michael R. Garey and David S. Johnson). SIAM Rev. **24**(1), 90–91 (1982)
4. Hong, K., Lillethun, D.J., Ramachandran, U., Ottenwalder, B., Koldehofe, B.: Opportunistic spatio-temporal event processing for mobile situation awareness. In: Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, DEBS 2013, pp. 195–206 (2013)
5. Jia, M., Cao, J., Liang, W.: Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. IEEE Trans. Cloud Comput. **5**(4), 725–737 (2017)
6. Jia, M., Liang, W., Xu, Z., Huang, M.: Cloudlet load balancing in wireless metropolitan area networks. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016)
7. Kleinrock, L.: Queueing Systems: Theory, vol. 1. Wiley-Interscience, Hoboken (1975)
8. Liu, Y., Lee, M.J., Zheng, Y.: Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. IEEE Trans. Mob. Comput. **15**(10), 2398–2410 (2016)
9. Ottenwalder, B., Koldehofe, B., Rothermel, K., Hong, K., Lillethun, D.J., Ramachandran, U.: MCEP: a mobility-aware complex event processing system. ACM Trans. Internet Technol. **14**(1), 6 (2014)
10. Pietzuch, P.R., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., Seltzer, M.I.: Network-aware operator placement for stream-processing systems. In: 22nd International Conference on Data Engineering (ICDE 2006), p. 49 (2006)
11. Rizou, S., Durr, F., Rothermel, K.: Solving the multi-operator placement problem in large-scale operator networks. In: 2010 Proceedings of 19th International Conference on Computer Communications and Networks, pp. 1–6 (2010)
12. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017). https://doi.org/10.1109/MC.2017.9
13. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
14. Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., Ottenwalder, B.: Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS 2016, pp. 258–269. ACM (2016)
15. Taneja, M., Davy, A.: Resource aware placement of IoT application modules in fog-cloud computing paradigm. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 8–12 May 2017, Lisbon, Portugal, pp. 1222–1228 (2017). https://doi.org/10.23919/INM.2017.7987464
16. Tziritas, N., Loukopoulos, T., Khan, S.U., Xu, C.Z., Zomaya, A.Y.: On improving constrained single and group operator placement using evictions in big data environments. IEEE Trans. Serv. Comput. **9**(5), 818–831 (2016)
17. Varga, A., Hornig, R.: An overview of the OMNET++ simulation environment, p. 60 (2008)
18. Zhang, Y., Niyato, D., Wang, P.: Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans. Mob. Comput. **14**(12), 2516–2529 (2015)

Enacting Emergent Configurations in the IoT Through Domain Objects

Fahed Alkhabbas[1,2], Martina De Sanctis[3], Romina Spalazzese[1,2], Antonio Bucchiarone[3](✉), Paul Davidsson[1,2], and Annapaola Marconi[3]

[1] Department of Computer Science and Media Technology, Malmö University, Malmo, Sweden
{fahed.alkhabbas,paul.davidsson,romina.spalazzese}@mau.se
[2] Internet of Things and People Research Center, Malmö University, Malmo, Sweden
[3] Fondazione Bruno Kessler, Via Sommarive, 18, Trento, Italy
{msanctis,bucchiarone,marconi}@fbk.eu

Abstract. The Internet of Things (IoT) pervades more and more aspects of our lives and often involves many types of smart connected objects and devices. User's IoT environment changes dynamically, e.g., due to the mobility of the user and devices. Users can fully benefit from the IoT only when they can effortlessly interact with it. To accomplish this in a dynamic and heterogenous environment, we make use of Emergent Configurations (ECs), which consist of a set of things that connect and cooperate temporarily through their functionalities, applications, and services, to achieve a user goal. In this paper, we: (i) present the IoT-FED architectural approach to enable the automated formation and enactment of ECs. IoT-FED exploits heterogeneous and independently developed things, IoT services, and applications which are modeled as Domain Objects (DOs), a service-based formalism. Additionally, we (ii) discuss the prototype we developed and the experiments run in our IoT lab, for validation purposes.

1 Introduction

Since the technology becomes more and more affordable and connectivity widespread, most objects and devices that would gain from being connected to the Internet are being connected. We refer to the (possibly smart) devices and connected objects as things [1,2]. Big market players, e.g., Ericsson[1], foresee that in the coming years, things will form large heterogeneous and highly distributed systems. As a result, the Internet of Things (IoT) will pervade and potentially improve many aspects of our lives.

From a user perspective, the IoT environment changes dynamically, e.g., when the user moves or due to mobile devices. Given the high diversity of things dynamically available in different and often unknown places, it is not feasible to define a priori all the possible combinations of things to reach specific user

[1] https://www.ericsson.com/res/docs/2015/mobility-report/ericssonmobility-report-nov-2015.pdf.

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 279–294, 2018.
https://doi.org/10.1007/978-3-030-03596-9_19

goals. Additionally, users can fully benefit from the IoT only when they can effortlessly interact with the dynamically available things to satisfy their goals. To accomplish this, a significant engineering effort is needed to abstract from the low-level interactions with the things. To this aim, we make use of Emergent Configurations (ECs), which consist of a set of things that connect and cooperate temporarily through their functionalities, applications, and services, to achieve a user goal [1,2].

In this paper, we present (i) IoT-FED: an approach for **F**orming and enacting **E**mergent configurations through **D**omain objects in the **IoT**. It exploits heterogeneous and independently developed things, IoT services, and applications modeled as Domain Objects (DOs), a service-based formalism [3]. Specifically, we present a process and an architecture realizing IoT-FED by leveraging an IoT platform, and a developer guideline. Additionally, we present (ii) a prototype we developed for validation purposes. The prototype is used to run experiments on two scenarios: one including (real/hardware) things is realized in our IoT lab[2] and the other where the hardware is simulated.

ECs are goal driven IoT systems, i.e., a user goal is the main driver to form and enact an EC. In this paper we assume that: *(A1)* ECs are formed and enacted within well-defined spatial boundaries (e.g., a room, a building). Therefore, we envision the number of things that might potentially form an EC to be in the scale of hundreds. This remarkably mitigates the well-known scalability issue in the IoT [4]. *(A2)* ECs achieve non-time critical user goals, i.e., ECs are formed and enacted within the time scale of seconds. *(A3)* ECs can be successfully formed and enacted, i.e., some needed things and services to achieve the goal are available and working until it is reached. Enabling the automated adaptation of ECs is out of the scope of this paper.

The Adjust Light Scenario. A concrete example of IoT systems we deal with is the Adjust Light (AL) Scenario that we will use throughout this paper.

(*AL-office*) Lara enters an office which is equipped with several IoT things including light sensors, and connected curtains and lights. The things, with their functionalities, are configured to be controllable by people in the office. Preparing for a meeting, she requests as goal to increase the light level using an application running on her smartphone. An EC formed by her smartphone, a light sensor, connected curtain and light is enacted to achieve her goal. The light sensor measures the light level in the room, the light is turned on and the curtain is partially opened. We realized this scenario, including both its hardware and software, in our IoT lab.

(*AL-hotel*) Lara enters a hotel room which is equipped with three light sensors, connected curtain and two connected lamps. The things, with their functionalities, are configured to be controllable by guests living in the room. Using the same application running on her smartphone, Lara expresses her goal to decrease light level in the room. Given that the curtain is closed, an EC formed by her smartphone, one of the light sensors and the two lamps is enacted to reach her goal. The light sensor measures the light level in the room, one of the two lamps is turned off and the light intensity level of the second lamp is reduced.

[2] http://iotap.mah.se/lab/.

To realize the AL-scenario we leverage the Amazon AWS-IoT platform[3], which provides RESTful and publish-subscribe based APIs.

The remainder of the paper is organized as follows. Section 2 describes background notions of DOs. Section 3 illustrates IoT-FED in terms of its process, architecture, and a guideline. Section 4 presents the AL-office Scenario realized including hardware things, and the (software) prototype at work. Section 5 illustrates the validation of IoT-FED. In Sect. 6 lessons learned are discussed. Section 7 surveys related works and Sect. 8 concludes the paper and draws future research directions.

2 Background on the Domain Object Model

The Domain Object model is the building block of a design for adaptation approach [3]. It allows to define independent and heterogeneous things and services in a *uniform way*, allowing developers to work at an abstract level without need to deal with the heterogeneity of things and protocols. In this section, we report the main features of the DO model that support the ECs formation and enactment, while other details can be found in [3,5].

The intuition behind the DO model is to separate *what* a system is designed to do (e.g., adjust the light level) from *how* to do it (e.g., by combining available light sensor S and actuators A_1 and A_2). The how can vary in different execution contexts since can be provided by disparate things dynamically available. The AL scenario highlights a high degree of *heterogeneity* in terms of: types of things (e.g., sensors, actuators, smartphones) and their protocols (e.g., communication, data, infrastructure).

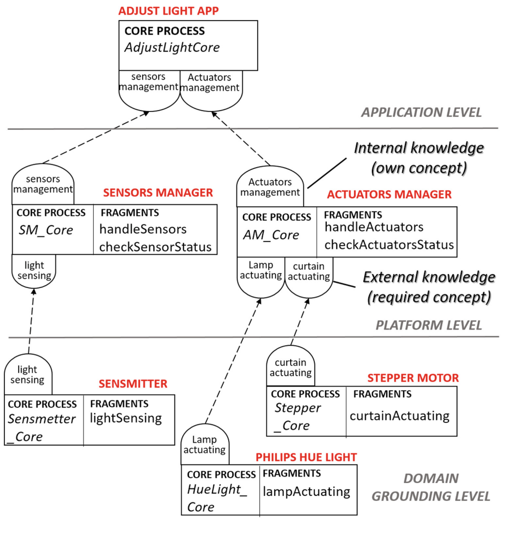


Fig. 1. Adjust Light Scenario: a partial overview of the DOs model of the system.

Things can be wrapped as DOs, e.g., by applications developers. This is done *una tantum*. After its wrapping, a thing is seamlessly part of the framework and exploited for the dynamic formation and enactment of ECs. In Fig. 1, we provide a partial overview of the DOs model of the AL Scenario. Each DO implements a specific *domain property* which models a thing capability. For instance, the `light sensing` property models the light sensing capability of a light sensor. Each DO has a *core process* which defines its behavior. In addition, it exposes one or more fragments (e.g., `checkActuatorsStatus`

[3] https://aws.amazon.com/iot/.

and `handleActuators`) which describe the functionalities it provides. Both core process and fragments are modeled as processes, by means of the *Adaptive Pervasive Flows Language* (APFL) [5]. Furthermore, each DO has *domain knowledge*, which represents its view on the environment in which it is. A DO's domain knowledge consists of: (i) internal knowledge that is the domain concept it implements e.g., the `Actuators management` in the `Actuators Manager`; (ii) external knowledge that is domain concepts it requires for its execution e.g., the `Lamp actuating` and `Curtain actuating` provided by other DOs.

Domain concepts are modeled as State Transition Systems (STS) [5]. The execution of fragments and core process of a DO updates STS state in its knowledge. *Domain-specific* concepts (e.g., light sensing) are usually implemented by DOs wrapping real things (e.g., sensors, actuators). *General-purpose* concepts (e.g., adjust light level), instead, might be realized by defining *value-added DOs* (e.g., a DO modeling the AL application of our scenario), which exploit the functionalities provided by the domain-specific ones.

Unlike traditional services, APFL allows the partial specification of the expected behavior of a DO by defining *abstract activities*: activities which the DO requires but it does not implement by itself. They are defined only in terms of a *goal* (e.g., sense the light). At runtime, the execution of abstract activities is performed through a *specialization* process, also called *refinement* mechanism. It allows abstract activities to be *refined* according to the fragments offered by other DOs, whose execution allows the abstract activity's goal to be reached. The so called *Higher Order Abstract Activities* (HOAA) enable a higher level of abstraction. They are used to dynamically generate abstract activities deriving from a performed reasoning activity, as explained in Sect. 4.2.

It is important to notice that fragments can be partially specified, too. Thus, they can also contain abstract activities. This enables a *chain of refinements* that we will detail in Sect. 4.2. The abstract activity refinement mechanism we deal with in this paper is performed through the application of advanced techniques for the dynamic and incremental service composition based on AI planning [5]. The AI planner takes as input the abstract activity's goal, the available fragments, the domain properties defined in APFL, and it returns a fragments composition process whose execution guarantees to reach the goal. A dynamic network of DOs is established at runtime to achieve the user goal. For instance, the dashed arrows in Fig. 1 stand for *potential* runtime relations that DOs might establish through the exchange of fragments (see Sect. 4).

3 The IoT-FED Approach

The IoT-FED approach includes (a) a process, (b) an architecture, and (c) a guideline. The IoT-FED is a refinement of the abstract architectural approach presented in [1] for enabling the automated formation and enactment of ECs in the IoT. Another possible refinement of [1] is presented in [6] where the focus is different compared to this paper, i.e., it is on the overall approach including adaptation, there is no use of DOs, and it exploits a different software prototype and scenario for experiments.

3.1 The IoT-FED Process

Figure 2 shows the IoT-FED process to enable the automated formation and enactment of ECs exploiting DO technologies. Labels A to H on process activities map them to one or more architectural components handling them in Fig. 3. The process starts by specifying the user *goal type* and the *goal spatial boundaries*. As already mentioned, the user expresses her/his goal via an application running on one of the available smart device called the user agent. The goal type corresponds to the type of functionality provided by the application running on the user agent (e.g., adjust light level). The goal spatial boundaries correspond to the location where the EC must be formed and enacted (e.g., Lara's office). We consider the goal spatial boundaries to be the same boundaries of the user agent. Given these inputs, the core process of the DO corresponding to the specified goal type (e.g., `adjust light app` DO in Fig. 1) is loaded, and its simulation starts. This simulation is needed to check that an EC can be formed to achieve the user goal in the given spatial boundaries. The goal is achievable if all abstract activities in the core process can be refined successfully. All the core process activities are traversed, without being executed. When an abstract activity is found, a refinement process is performed to generate a plan (i.e., composition of fragments) which refines it - if any.

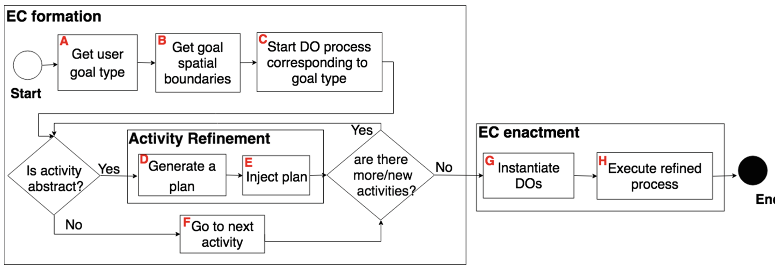


Fig. 2. The EC formation and enactment process

The generated plan, which might itself contain abstract activities, is then injected in place of the abstract activity it refines. If all found abstract activities are refined successfully, an EC exists and is formed. The formed EC is made up by the set of things whose corresponding DOs have been involved in the refinement process, through the selection of their fragments. The formed EC can then be enacted by instantiating the DOs forming it. Finally, the initial core process where all the generated plans are injected, can be executed. In this work, the fragments selection is purely functional. We plan to extend the process to support some situational context and dependencies in selecting fragments and consequently EC constituents.

3.2 An Architecture Realizing IoT-FED

Figure 3 shows a possible refinement of the abstract architecture presented in [1]. The architecture realizes the IoT-FED process of Fig. 2, exploits existing components of the DOs technologies, and presents newly developed components. In the following, we describe the components and their responsibilities.

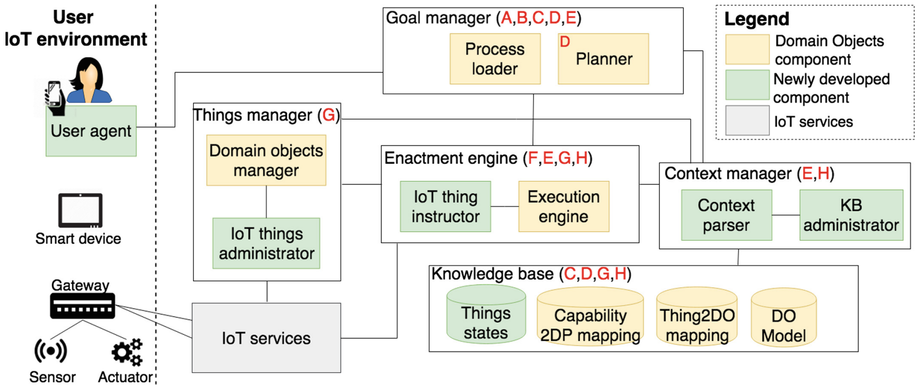


Fig. 3. The IoT-FED architecture

Goal Manager. It is responsible for parsing the user goal and starting the EC formation process. It has two sub-components: the *Process loader*, responsible for specifying the user goal type and the spatial boundaries, and for loading the DO process corresponding to the specified goal type; the *Planner* responsible for the refinement of abstract activities in the loaded process.

Things Manager. It is responsible for managing available IoT things and DOs. It includes: the *IoT things administrator* responsible for answering queries about available IoT things, their capabilities and locations; the *Domain objects manager* responsible for instantiating needed DOs and handling co-relations among them.

Enactment Engine. It is mainly responsible for enacting the EC and includes two sub-components: the *Execution engine* is responsible for both (i) simulating and forming the EC - labels E and F in Fig. 2, and (ii) enacting it -labels G and H in Fig. 2. During the EC formation, it handles the injection of the plans received by the Planner in place of the abstract activities they refine, and during the EC enactment it executes the final refined process that achieves the user goal. The *IoT thing instructor* is responsible for passing the Execution engine instructions to IoT things (e.g., get sensor readings).

Context Manager. It is responsible for maintaining the system knowledge. It includes *KB administrator* responsible for retrieving data from the KB and the *Context parser* responsible for parsing received context from the Execution engine (e.g., new things states) and passing it to the KB administrator which updates the KB.

Knowledge Base. It holds the internal system knowledge and includes four repositories: the *Things states* stores knowledge about things operational states (e.g., if lights are turned on or off); the *DO Model* that stores all the designed DOs; the Thing2DO mapping stores associations between things and the DOs representing them (e.g., the `Sensmitter` DO is a *Light Sensor* thing); the *Capability2DP mapping* that stores information about capabilities and domain properties relations. Different domain properties can relate to the same capability. For instance, the *actuate light level* capability can be mapped on both the `Curtain Actuating` and `Lamp Actuating` domain properties.

IoT Services. The IoT-FED approach relies on a cloud platform to provide IoT services which enable the management and interaction with things. We leverage the Amazon AWS-IoT platform.

In general, realized components can be deployed to the Cloud (see Sect. 3.3), standalone servers or to hybrid infrastructures. A potential infrastructure shall posses capabilities of consuming available IoT services. The deployment model and aspects about security, connectivity, interoperability, and so on are out of the scope of this paper.

3.3 A Guideline About IoT-FED

To make IoT things and services available to be used through IoT-FED, a *developer* needs to do two main operations, here mapped on the development of the AL prototype.

1. *Register things in the AWS-IoT platform:* using the notion of templates in the AWS-IoT platform, developers can register different types of IoT things. For each thing, three searchable attributes should be specified: (thing) location, capabilities and REST endpoint automatically generated by the platform. When a thing is set up for the first time, a developer needs to store the thing initial state in the Things state repository. Finally, by exploiting the AWS-IoT services, the Context manager can dynamically answer queries about which things are available in a specific location, which of them have a specific capability and how to communicate with them.
2. *Model things, services and applications as DOs:* we illustrate this task by describing the modeled AL prototype in terms of DOs (see Fig. 1). The operational environment of the target system is defined through domain properties representing capabilities (e.g., light sensing, lamp actuating) that have to be registered in the Capability2DP mapping repository. Then, DOs implementing the specified domain properties have to be designed. The REST endpoints generated by the platform are invoked in the DOs processes activities. We can distinguish three levels in the DO model. At *domain grounding level* there are all the DOs of things offering functionalities and possibly requiring the interaction with third-party systems and things. DOs wrapping things have to be

designed for each thing type and brand. For instance, the DOs of the things that can be part of a smart room such as the `Sensmitter` light sensor -with its fragments, e.g., `senseLight`. At *platform level* we have value-added DOs offering value-added services, such as the `Actuators Manager` in Fig. 1. These are defined through inter-dependencies among DOs implementing domain-specific functionalities, e.g., the handling of actuator devices. At *application level* we have the `AdjustLightApp` DO modeling as value-added DOs the user application used for adjusting the light level in different locations -while the user moves around. Each DO model has to be stored in the DO Model repository, and for those wrapping things, the mapping thing-DO is registered in the Thing2DO mapping repository.

4 The AL-office Scenario and AL Prototype Running on IoT-FED

This section presents: the realized AL-office Scenario including the (real/hardware) things in our IoT lab exploited in the developed (software) prototype, and the DOs dynamic mechanism leveraged for forming and enacting ECs.

4.1 The Realized AL-office Scenario

The IoT things we used to develop the AL-office Scenario are: three climate Sensmitters[4] including light sensors, a Philips Hue light[5] which has three bulbs connected by a bridge, and motorized blinds we built consisting of a stepper motor[6] integrated with off the shelf blinds[7]. As said, to enable the management and interaction with the things, we leveraged the Amazon AWS-IoT platform and we deployed the architectural software components to an Amazon Elastic Beanstalk[8] instance running a Glassfish web server. Moreover, we use *gateways* as intermediaries between the AWS-IoT platform and the used things that do not have the processing and storage capabilities needed to connect directly to the platform (see Fig. 3). Particularly, the Sensmitters transmit data through a smartphone via Bluetooth technology. The hue light are connected through an Arduino[9] board, which also invokes the Philips Hue API services to adjust the bulbs light levels. Likewise, the motorized blinds connect through another Arduino board, which also sends the low level commands to rotate the stepper motor, thus opening or (partially) closing the blinds. The communications among the Enactment engine, Things manager and the IoT platform are MQTT[10]

[4] https://www.senssolutions.se/.
[5] https://www2.meethue.com/en-us.
[6] https://components101.com/motors/28byj-48-stepper-motor.
[7] https://jysk.se/gardiner/persienner/aluminium/alu-persienn-60x80cm-vit.
[8] https://aws.amazon.com/elasticbeanstalk.
[9] https://www.arduino.cc/.
[10] http://mqtt.org/.

based. To develop the AL prototype, we followed the guideline about IoT-FED, as anticipated in Sect. 3.3. In the following section we provide an example of the AL prototype execution.

4.2 Running the AL Prototype on IoT-FED

When a user expresses a goal (at runtime) via a user agent, this triggers the execution of the IoT-FED process (Fig. 2). The execution starts from the `AdjustLightApp` DO, corresponding to the AL application running on the user's smartphone. Here, we describe the key mechanism contributing to the formation and enactment of ECs, i.e., the dynamic refinement of DOs' abstract activities through the dynamic discovery, injection, and execution of fragment-based plans.

Figure 4 depicts the situation in which Lara enters her office and she wants to adjust the light level in the room. We suppose that this can be done by setting a slider bar (e.g., from 0 to 10 level) on the AL application interface. Figure 4(a) describes the *EC formation* process of Fig. 2 performed by simulating the execution of the AL core process. All its activities are traversed, looking for abstract activities and checking that they can all be successfully refined through some composition of fragments. If this is possible, at least one EC exists and the generated refinement is injected in the original AL core process. Figure 4(b), instead, describes the *EC enactment* process of Fig. 2 where the generated refined AL core process can be executed. Note that, for the sake of description, in the following we mix details of ECs formation and enactment. However, remind that the EC enactment starts only if and after an EC is successfully formed. The EC formation starts from the core process of the AL application[11]. When its execution starts, information about the room where the user is located are retrieved, in order to get the goal spatial boundaries. Then, among others, a sequence of two abstract activities (represented with dotted lines and labeled with a goal defined on top of domain properties) need to be refined. They refer to the sensing and the actuating of the light level in the room, respectively. Then, the refinement mechanism is triggered. In the following we list the refinement steps performed while forming the EC.

Step 1 consists in finding a plan for the goal `G1: SM = Sensor Discovered`. To this aim, the fragment `HandleSensors` from the `SensorsManager` is selected for the refinement and injected in the AL process, in place of the `Detect Light Level` abstract activity. During the EC enactment phase, the execution of this fragment allows to send a `Sensing Request`, by specifying the required capability (e.g., sense *light* level), and receive information about the list of available things that might be exploited for the request. The `Sensing Higher Order Abstract Activity` dynamically generates the `Light Sensing` abstract activity, after the application understands that the required capability deals with the sensing of the light, given that the `SensorsManager` handles different kinds of sensors.

Step 2 consists in finding a plan for the dynamically generated goal `G3: LS = Light Data Sent`, which has been defined over the `Light Sensing` domain

[11] For presentation purposes, in Fig. 4 we omit some details.

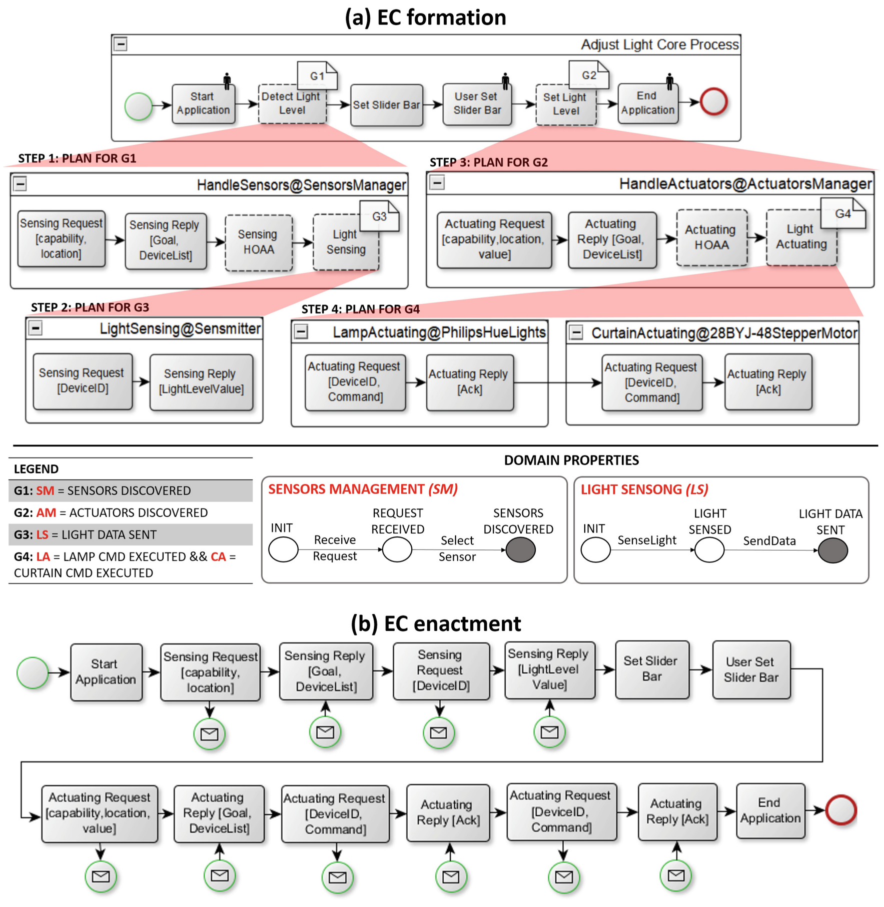

(a) EC formation
Adjust Light Core Process
Start Application
G1
Detect Light Level
Set Slider Bar
User Set Slider Bar
G2
Set Light Level
End Application
STEP 1: PLAN FOR G1
HandleSensors@SensorsManager
Sensing Request [capability, location]
Sensing Reply [Goal, DeviceList]
Sensing HOAA
G3
Light Sensing
STEP 3: PLAN FOR G2
HandleActuators@ActuatorsManager
Actuating Request [capability,location, value]
Actuating Reply [Goal, DeviceList]
Actuating HOAA
G4
Light Actuating
STEP 2: PLAN FOR G3
LightSensing@Sensmitter
Sensing Request [DeviceID]
Sensing Reply [LightLevelValue]
STEP 4: PLAN FOR G4
LampActuating@PhilipsHueLights
Actuating Request [DeviceID, Command]
Actuating Reply [Ack]
CurtainActuating@28BYJ-48StepperMotor
Actuating Request [DeviceID, Command]
Actuating Reply [Ack]
LEGEND
G1: SM = SENSORS DISCOVERED
G2: AM = ACTUATORS DISCOVERED
G3: LS = LIGHT DATA SENT
G4: LA = LAMP CMD EXECUTED && CA = CURTAIN CMD EXECUTED
DOMAIN PROPERTIES
SENSORS MANAGEMENT (SM)
INIT
Receive Request
REQUEST RECEIVED
Select Sensor
SENSORS DISCOVERED
LIGHT SENSONG (LS)
INIT
SenseLight
LIGHT SENSED
SendData
LIGHT DATA SENT
(b) EC enactment
Start Application
Sensing Request [capability, location]
Sensing Reply [Goal, DeviceList]
Sensing Request [DeviceID]
Sensing Reply [LightLevel Value]
Set Slider Bar
User Set Slider Bar
Actuating Request [capability,location, value]
Actuating Reply [Goal, DeviceList]
Actuating Request [DeviceID, Command]
Actuating Reply [Ack]
Actuating Request [DeviceID, Command]
Actuating Reply [Ack]
End Application

Fig. 4. AL Scenario: an example of the *EC formation* (a) and the *EC enactment* (b).

property. It is interesting to notice that the `Sensors Manager` is not able to provide itself the required fragment for the light sensing, because the sensing process depends on the thing used to do it. Only during the execution, the system can discover and select the proper things it needs, in the specific context in which it is running. In the Lara's office room, the `Light Sensing` fragment is given by the `Sensmitter` sensor.

Step 3 and **step 4** are performed by following the same procedure of steps 1 and 2, with the only difference that the system needs now to discover actuators to set Lara's desired light level. Specifically, a composition made by fragments offered by the `Philips Hue lights` and the `Stepper Motor` actuators installed in the office is used to reach the user goal.

The final AL application process executed to enact the AL EC is the one in Fig. 4(b).

5 Validation

To evaluate the *feasibility* of IoT-FED, we conducted two experiments aiming at answering: *can IoT-FED be used to handle the dynamic formation and enactment of ECs at runtime?* Table 1 presents the specifications of two rooms corresponding to the *AL-office* and the *AL-hotel* scenarios (see Sect. 1). For each room, we defined the installed things, i.e., light sensors and actuators. Then, we wrapped the things as DOs and made them usable by the IoT-FED following the guidelines presented in Sect. 3.3. Note that the feasibility of the domain object-based approach is analyzed in [7]. The time for a developer to wrap a service as DO is paid only una tantum; it varies depending on her/his knowledge of DOs and expertise level and in the worst case is few hours.

Table 1. Office and hotel room's stationary things.

Device type	Office room	Hotel room
Light sensors	Sensmitter, LightMeter, WiStar, SensLight	Sensmitter, DayLight, WiStar
Lamp actuators	PhilipsHueLight, SmartLamp	PhilipsHueLight, IkeaLamp
Curtain actuators	HS-422-ServoMotor, StepperMotor	HS-422-ServoMotor

Our experiments include 11 different things of various brands. Besides the real world things (e.g., Philips Hue Light) we used to implement the AL-office scenario, we modeled additional things to simulate more complex still realistic settings. The office and hotel rooms are equipped with 8 and 6 different stationary things, respectively, in addition to Lara's smartphone. We performed sequential executions of <100> runs of the AL application per each room to: (i) evaluate the IoT-FED ability to dynamically generate different ECs based on available things; (ii) get insights on the IoT-FED performances. We evaluated our approach using a dual-core CPU running at 2.7 GHz, with 8 Gb memory.

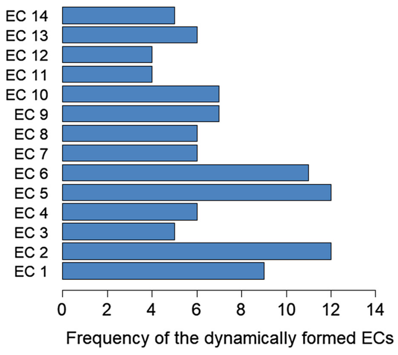


Fig. 5. ECs in the hotel room.

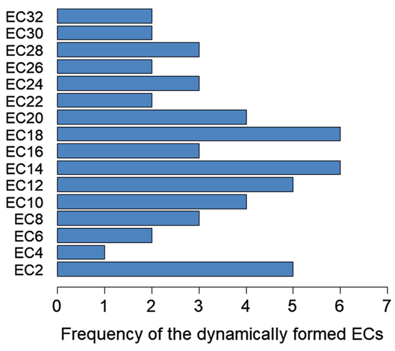


Fig. 6. ECs in the office room.

Figure 5 shows the different dynamically generated ECs and their occurrences over the 100 runs of the AL application in the hotel room. Based on randomly requested light levels at each run, 14 different ECs were formed and enacted. For instance, the EC number 1 is made as follows: $EC1 = \langle smartphone, DayLight, HS-422-ServoMotor, IkeaLamp\rangle$ and it occurs 9 times over the 100 runs. Compared with the hotel room, the office room has one more light sensor and one more curtain actuator. Figure 6 illustrates that with only two more devices in the domain space, 32 different ECs can be formed[12]. For instance, the EC number 2 is made as follows: $EC2 = \langle smartphone, LightMeter, HS-422-ServoMotor, PhilipsHueLight\rangle$ and it occurs 5 times over the 100 runs. For each EC occurrence, we also measured the time required to form and enact it. Fig. 7 shows the average time for forming and enacting ECs over the 100 runs of the AL scenario for both the hotel and office rooms, i.e., 2.39 s and 2.44 s respectively.

One of the advantages of managing IoT domains w.r.t. Internet of Services (IoS) domains is that we deal with simpler DOs processes and fragments (i.e., made by few activities and transitions). This positively impacts on the time required for the dynamic composition of fragments and keeps it in the order of milliseconds. In summary, we can draw positive conclusions about the feasibility of the IoT-FED approach in forming and enacting ECs in dynamic environments. According to the assumption *(A1)* in Sect. 1, we considered a realistic set of things in these experiments, based on the spatial boundaries where ECs are enacted. While for future work we plan to apply the approach to additional real-world scenarios involving more things.

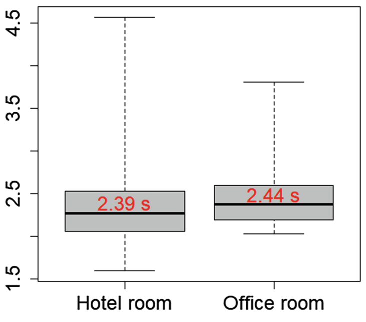


Fig. 7. ECs execution time per room.

6 Lessons Learned

DOs Formalism. IoT things are heterogeneous and operate using different standards. This makes it evident that there is a need for other layers which can handle complex processes such as enabling ECs formation and enactment. Wrapping things as DOs has several *advantages*. For instance, as DOs, things are represented in a uniform way, allowing developers to work at an abstract level where they do not always need to deal with the heterogeneity of things and protocols. Nevertheless, the DOs formalism can be further *extended* to better meet IoT domain's features and requirements. Particularly, when refining an abstract activity, the fragments discovery and selection is currently functional. Available fragments whose execution allows the abstract activity's goal to be

[12] For readability purposes, in Fig. 6, we show only a subset of the 32 ECs.

reached are selected and composed. However, specially in IoT domains, a non-functional selection process would be more appropriate (e.g., to select the device which has a good battery level instead of a random one).

Moreover, the Execution Engine currently operates in a centralized manner. It should evolve to better deal with the execution of distributed systems. Since IoT things are wrapped as DOs manually, the IoT-FED approach supports expressing a set of goals whose types are specified at design time. We plan to support the automated wrapping of things as DOs, thus enabling the definition of new goal types at runtime. Currently, runtime failures require re-executing the EC formation and enactment process to form new ECs which maintain the achievement of user goals. We recall that enabling automated adaptation of ECs is out of the scope of this paper.

Amazon IoT Platform and Deployment Services. On the one hand, the AWS-IoT platform contributes several *advantages* to the IoT-FED approach. For instance, it provides APIs which support managing and interacting registered IoT things. The platform also supports routing a huge number of messages at runtime. On the other hand, the platform imposes some *limitations* and restrictions. For instance, it puts sophisticated security requirements for registering IoT things. Although the AWS Elastic Beanstalk facilitates the deployment process, it may require developers to perform security related configurations when applications are (re-)deployed to web servers.

7 Related Work

In the context of architectures, the IoT-FED architecture is compliant with the IoT reference architecture proposed in [8]. The SOCRADES Integration Architecture (SIA) is a SOA architecture designed to couple the IoT with enterprise services [9]. In SIA, processes are modeled at design time by using BPEL. The authors also extended the BPEL language to enable the dynamic assignment of services at the execution phase. In [10], the authors propose a service-based architectural approach to enable efficient and adaptive composition of services. Composite services are modeled and specified at design time, thus limiting systems flexibility.

The usage of business processes with their related technologies in the IoT context [11] is a novel research field that opened interesting research challenges [12]: from extensions of standard workflow languages (i.e., WS-BPEL, BPMN 2.0) proposed to support suitable communication paradigms for the IoT [13,14], to workflow management systems (WfMS) for industrial IoT [15,16] to execute and monitor IoT-based processes. WfMS suitable in dynamic contexts have also been proposed, to adapt processes in case of failures by replacing the respective resources (things or services) or workflow tasks [17,18]. The use of the APFL language enables the IoT-FED approach to dynamically refine abstract processes with concrete ones provided by available things, thus making it suitable in dynamic contexts, too.

The growing number of online resources, and services led to the rise of methodologies and tools to create applications by combining them, referred to as *mashups*. Most mashups approaches focus on the composition of web-based interfaces and functionalities. In [19], the authors focus on the importance of *context-awareness* and *adaptivity* of service mashups in dynamic environments, otherwise, they tend to be misaligned with their execution environments. In [20], the authors objective is that of overcoming the static nature of IoT applications that, although highly responsive, are usually based on pre-compiled mashups, being thus inflexible. The work is based on a decentralized goal-driven composition of pre-compiled service mashups and, similarly to IoT-FED, it relies on the abstraction of the referring environment allowing to abstract service composition requirements. In last years, to facilitate the development of IoT application, different mashups editors have been proposed, such as e.g., [21,22], providing developers a visual support abstracting both things and services they can compose together. Differently from [22] our approach includes, but is not limited to, the use of REST services. Moreover, it allows applications to define dynamic behaviors, due to the use of abstract activities refined through services composition when the context is known or discovered. The trade-off of the majority of service mashups approaches in the literature is that applications rely on static mashups that cannot deal with open environments. Service-oriented architectures for planning, execution and adaptation of cyber-physical systems (CPS) have been proposed in [23,24]. The approach proposed in [23] is based on a clean separation between domain modeling, planning, execution, monitoring and actuation services enabling the realization of large scale CPSs. In [24], the authors propose a MAPE-K autonomic computing framework to manage adaptivity in service-based CPSs.

Several approaches discuss coalitions of coordinating components, known as *choreography*. For instance, in [25], a formal framework dealing with self-adaptation in the context of choreographies is proposed. A group of interacting components has also be seen as an *ensemble*. Languages to define ensembles have been introduced (i.e., [26,27]) to specify what groups of components should be present in the system together with mechanisms to select at runtime the ones satisfying specific constraints (i.e., predicates). In the IoT-FED approach, the system configuration is less-constrained by design and it leaves components to freely interact to achieve a user goal in a specific context.

8 Conclusion and Future Work

In this paper, we presented IoT-FED, an architectural approach enabling the automated and runtime formation and enactment of ECs in dynamic environments. The approach exploits the DO model and its technologies in the context of IoT environments. We have presented both the process and the architecture of the IoT-FED approach, with a guideline about how to use it. Additionally, we presented the developed prototype used for running initial validation experiments, which showed positive and promising results.

In the future, we plan work in a number of directions including: extending IoT-FED to deal with the adaptation of ECs at runtime (leveraging the adaptation mechanisms of DOs); identifying and realizing additional realistic cases that deals with multiple and possibly competing ECs and using them for more extensive validation; extending the approach to deal with the selection and composition of things based on some non functional aspects.

Acknowledgment. This work was partially funded by the project SmartConstruction (https://enoba.eu/projects/smartconstruction) (EIT Digital activity #18014) and by the Knowledge Foundation through the Internet of Things and People research profile (Malmö University, Sweden).

References

1. Alkhabbas, F., Spalazzese, R., Davidsson, P.: Architecting emergent configurations in the Internet of Things. In: IEEE International Conference on Software Architecture, pp. 221–224. IEEE (2017)
2. Ciccozzi, F., Spalazzese, R.: MDE4IoT: supporting the Internet of Things with model-driven engineering. In: Badica, C. (ed.) IDC 2016, vol. 678, pp. 67–76. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-48829-5_7
3. Bucchiarone, A., De Sanctis, M., Marconi, A., Pistore, M., Traverso, P.: Design for adaptation of distributed service-based systems. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 383–393. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_27
4. Atzori, L., Iera, A., Morabito, G.: The Internet of Things a survey. Comput. Netw. **54**, 2787–2805 (2010)
5. Bucchiarone, A., De Sanctis, M., Marconi, A., Pistore, M., Traverso, P.: Incremental composition for adaptive by-design service based systems. In: IEEE International Conference on Web Services, ICWS. IEEE (2017)
6. Alkhabbas, F., Spalazzese, R., Davidsson, P.: ECo-IoT: an architectural approach for realizing emergent configurations in the Internet of Things. In: Cuesta, C., Garlan, D., Pérez, J. (eds.) ECSA 2018. LNCS, vol. 11048, pp. 86–102. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00761-4_6
7. Bucchiarone, A., De Sanctis, M., Marconi, A.: ATLAS: a world-wide travel assistant exploiting service-based adaptive technologies. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 561–570. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_41
8. Bauer, M., et al.: IoT reference architecture. In: Bassi, A. (ed.) Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model, pp. 163–211. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-40403-0_8
9. Spiess, P., et al.: SOA-based integration of the Internet of Things in enterprise services. In: IEEE International Conference on Web Services, pp. 968–975. IEEE (2009)
10. Dar, K., Taherkordi, A., Rouvoy, R., Eliassen, F.: Adaptable service composition for very-large-scale Internet of Things systems. In: 8th Middleware Doctoral Symposium, pp. 2. ACM (2011)
11. Chang, C., Srirama, S.N., Buyya, R.: Mobile cloud business process management system for the Internet of Things: a survey. ACM Comput. Surv. **49**, 70:1–70:42 (2017)

12. Janiesch, C. et al.: The Internet-of-Things meets business process management: mutual benefit and challenges. In: arXiv preprint arXiv:1709.03628 abs/1709.03628 (2017)
13. Domingos, D., Martins, F., Cândido, C., Martinho, R.: Internet of Things aware WS-BPEL business processes context variables and expected exceptions. J. UCS **20**, 1109–1129 (2014)
14. Tranquillini, S., et al.: Process-based design and integration of wireless sensor network applications. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 134–149. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32885-5_10
15. Mass, J., Chang, C., Srirama, S.N.: WiseWare: a device-to-device-based business process management system for industrial Internet of Things. In: IEEE International Conference on Internet of Things, pp. 269–275. IEEE (2016)
16. Seiger, R., Huber, S., Schlegel, T.: Toward an execution system for self-healing workflows in cyber-physical systems. Softw. Syst. Model. **17**, 551–572 (2018)
17. Seiger, R., Huber, S., Heisig, P.: PROtEUS++: a self-managed iot workflow engine with dynamic service discovery. In: Central European Workshop on Services and their Composition, pp. 90–92 (2017)
18. Wieland, M., Schwarz, H., Breitenbücher, U., Leymann, F.: Towards situation-aware adaptive workflows: SitOPT - a general purpose situation-aware workflow management system. In: IEEE International Conference on Pervasive Computing and Communication, pp. 32–37. IEEE (2015)
19. Dorn, C., Schall, D., Dustdar, S.: Context-aware adaptive service mashups. In: IEEE Asia-Pacific Services Computing Conference, pp. 301–306. IEEE (2009)
20. Ciortea, A., Boissier, O., Zimmermann, A., Florea, A.M.: Responsive decentralized composition of service mashups for the Internet of Things. In: 6th ACM International Conference on the Internet of Things, pp. 53–61. ACM (2016)
21. Giang, N.Ky., Blackstock, M., Lea, R., Leung, V.C.M.: Developing IoT applications in the Fog: a distributed dataflow approach. In: International Conference on the Internet of Things, pp. 155–162. IEEE (2015)
22. Mayer, S., Verborgh, R., Kovatsch, M., Mattern, F.: Smart configuration of smart environments. In: IEEE Transactions on Automation Science and Engineering, pp. 1247–1255. IEEE (2016)
23. Feljan, A.V., Mohalik, S.K., Jayaraman, M.B., Badrinath, R. : SOA-PE: a service-oriented architecture for planning and execution in cyber-physical systems. In: International Conference on Smart Sensors and Systems, pp. 1–6. IEEE (2015)
24. Mohalik, S.K., Narendra, N.C., Badrinath, R., Le, D.: Adaptive service-oriented architectures for cyber physical systems. In: IEEE Symposium on Service-Oriented System Engineering, pp. 57–62. IEEE (2017)
25. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Self-adaptive monitors for multi-party sessions. In: IPDP, pp. 688–696. IEEE (2014)
26. Krijt, F., Jirácek, Z., Bures, T., Hnetynka, P., Gerostathopoulos, I.: Intelligent ensembles - a declarative group description language and Java Framework. In: IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 116–122. IEEE (2017)
27. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. In: TAAS (2014)

Energy-Delay Co-optimization of Resource Allocation for Robotic Services in Cloudlet Infrastructure

Mahbuba Afrin[1](✉), Jiong Jin[1], and Ashfaqur Rahman[2]

[1] School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia
{mafrin,jiongjin}@swin.edu.au
[2] Data61, CSIRO, Sandy Bay, TAS 7005, Australia
ashfaqur.rahman@data61.csiro.au

Abstract. Cloud based robotic services can be adopted for emergency management in smart factory. When multiple robots work collaboratively in such system, optimal resource allocation for executing the tasks of robotic services becomes a challenging problem due to the heterogeneity and energy consumption of resources. Since the tasks of multi-robotic services are inter-dependent, the inconvenience of data exchange between local robots and distant Cloud can significantly degrade the quality of service. Therefore, in this paper, we jointly address the energy consumption and service delay minimization problem while allocating resources in proximate Cloud (Cloudlet) based multi-robot systems for emergency management service in smart factory. A multi-objective evolutionary approach, NSGA-II algorithm is applied to solve this constrained multi-objective optimization problem. We augment the NSGA-II algorithm by defining a new chromosome structure, presorted initial population, mutation operator and selection of minimum distant solution from the non-dominated front to the origin while crossing over the chromosomes. The experimental results on the basis of synthetic data demonstrate that our approach performs significantly well compared to benchmark NSGA-II.

Keywords: Resource allocation · Multi-robot system · Cloudlet Optimization · Evolutionary algorithm

1 Introduction

In Cloud based multi-robot systems, Cloud offers virtualized resources, platform and software services so that both localized robots and remote resources can be utilized to process the tasks of robotic services. In a smart factory, such Cloud-robotic services can manage entire production and supply chain. However, safety assurance in smart factory during hazardous situation like fire occurrence is very crucial. In this case, both robot and Cloud resources should complement each other to process diversified tasks of emergency management service within a stringent deadline. These tasks are usually inter-dependent, latency sensitive and compute intensive. In addition, the resources are heterogeneous in terms of processing capability and energy consumption [3]. In such scenario, an optimal allocation of computing resources to the tasks with the aim of minimizing both service delay and energy consumption is one of the key research challenges.

Taking cognizance of the afromentioned issues, we focus on energy-delay co-optimized resource allocation for processing the tasks of emergency management service such as fire driven emergency service in a smart factory across robot and Cloud infrastructure. Tasks of this service require real-time response and data transmission among robots and Cloud. Therefore, the concept of Cloudlet [4] infrastructure is introduced between robot-Cloud. Cloudlet is a prominent extension of Cloud closer to the data source that can provide multi-robot systems with virtualized resources to execute latency sensitive services.

To the best of our knowledge, this is the first work to design a Cloudlet based multi-robot framework for emergency management service in smart factory. Resource allocation in this context appears as a constrained multi-objective optimization problem when both energy consumption of resources and overall makespan for processing the tasks are targeted to minimize simultaneously. Since, multi-objective evolutionary algorithms help to generate pareto-optimal solutions of such multi-objective optimization problem, in this paper, we extend the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [2] to solve the resource allocation problem for its capability of finding diverse set of solutions. Here, we augment the NSGA-II by defining a new chromosome structure, presorted initial population based on the task size and processing speed of the resources. Besides, while crossing over the chromosomes, rather than selecting arbitrarily, the chromosome having minimum distant solution from the pareto-front to the origin is selected to balance the values of both objectives in subsequent generations. The results of our simulation experiments significantly improves the benchmark NSGA-II in minimizing both objectives.

The rest of the paper is organized as follows. In Sects. 2 and 3, related work and Cloudlet-based framework for multi-robot system are discussed, respectively. The problem formulation and the solution mechanism are described in Sect. 4. In Sect. 5, the efficacy of our proposed approach is validated through simulation and Sect. 6 concludes the paper with future directions.

2 Related Work

In literature, the integration of Cloud services and multi-robot systems has already been investigated. The concept of Robot as a Service through a Service Oriented Architecture is defined in [1] that incorporates robot services in Cloud. A smart city-based Cloud robotic framework for optimal task offloading is designed in [7]. Motion and connectivity-aware offloading for Cloud robotic services using evolutionary algorithm is further introduced in [6]. However, in both [7] and [6], the offloading decision-making is done by a single robot.

Task allocation in resource-constrained multi-robot systems for search and rescue or other emergency and hazardous scenarios is discussed in [8]. Cost-efficient deployment of robots for a search and rescue use case during large-scale disaster management is explored in [5]. However, in the aforementioned works, no multi-robot task allocation policy targeting both local and Cloud resources is pursued. Nevertheless, considering the intrinsic constraints of Cloud and multi-robot assisted smart factory during the execution of emergency management services, neither the combined task allocation problem nor the solutions has been enlightened in the literature so far.

3 Framework for Cloudlet Based Multi-robot System

The Cloudlet infrastructure between robot and Cloud forms a three-tier computational framework and helps to execute emergency management service for smart factory with less communication delay. In this framework, the robots using Zigbee, Bluetooth and WiFi can communicate with each other. They also communicate with Cloudlet and Cloud through gossip protocols [3].

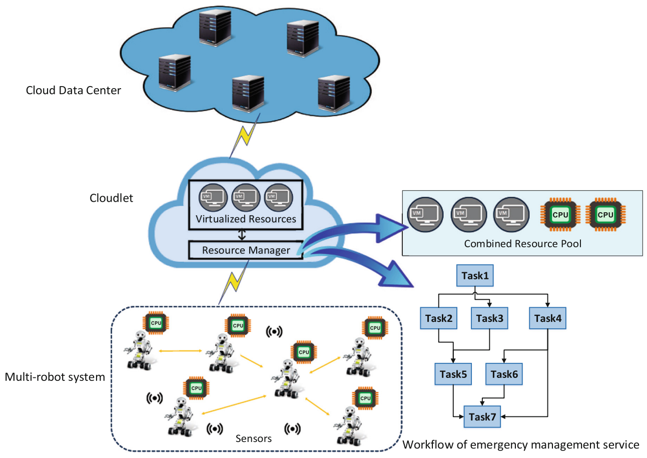


Fig. 1. Framework design for Cloudlet based multi-robot system

In a Cloudlet, *Resource Manager (RM)* creates a combined resource pool with both local and remote resources while executing the emergency management service as shown in Fig. 1. Here, we consider emergency fire management service in smart factory as an example case scenario. This service consists of multiple tasks such as fire origin and cause identification, human victim and hazardous material detection, evacuation planning, navigation as well as management of external help. These tasks are interdependent and orchestrated through a Directed Acyclic Graph (DAG) based workflow model. We assume that the corresponding tasks of a service and the meta-data of the tasks (inter-data dependency delay, Quality of Service requirement etc.) are stored in the Cloudlet. Whenever an event of interest triggers the service initiation, the RM residing in the Cloudlet assigns the tasks to the combined resource pool.

4 Proposed Energy-Delay Optimized Resource Allocation

4.1 Multi-objective Optimization Problem Formulation

The optimal resource allocation for processing the service tasks is subject to multiple constraints driven by the characteristics of tasks (e.g., latency sensitivity, interdependency, resource requirements, etc.). It turns into a multi-objective optimization problem when simultaneous minimization of overall makespan for completing the tasks and energy consumption of the resources are taken into account. In the proposed Cloudlet-based framework, this multi-objective optimization problem can be expressed through Eqs. 1 and 2 for the corresponding set of tasks T and resources R. In Eq. 1, m_r^t denotes the time required to complete a task $t \in T$ on a resource $r \in R$ and it minimizes the overall makespan $\forall t \in T$. At the same time, Eq. 2 minimizes the overall energy consumption e_r^t while processing each task $t \in T$ on particular resource $r \in R$. The solution of both equations points the assignment of task $t \in T$ to resource $r \in R$ through non-zero value of binary decision variable x_{tr}. The constraints of Eqs. 1 and 2 ensure that a task $t \in T$ will not assign to multiple resources (Eq. 3), the total task completion time will not exceed the maximum allowable delay M_T to execute the service (Eq. 4) and the total energy consumption of resources should be within the energy threshold, E_T (Eq. 5). In addition, Eq. 6 signifies that the assignment of a dependent task $t \in T$ to a resource $r \in R$ and its tolerable inter-task data dependency delay δ_t will not be affected by the assignment of all its predecessor tasks $t' \in T_t'$ on computing resources $r' \in R$. In this case, maximum processing time $p_{r'}^{t'}$ of the predecessor tasks and data exchange time $\eta_{r'r}$, $\forall t' \in T_t'$ is taken into account.

$$\min \sum_{t \in T, r \in R} x_{tr} \times m_r^t \tag{1}$$

and

$$\min \sum_{t \in T, r \in R} x_{tr} \times e_r^t \tag{2}$$

subject to

$$\sum_{t \in T, r \in R} x_{tr} = 1 \tag{3}$$

$$\sum_{t \in T, r \in R} m_r^t \leq M_T \tag{4}$$

$$\sum_{t \in T, r \in R} e_r^t \leq E_T \tag{5}$$

$$\max(p_{r'}^{t'} + \eta_{r'r}) \leq \delta_t; \forall t' \in T_t' \tag{6}$$

4.2 Energy-Delay Co-optimization Using Multi-objective Evolutionary Algorithm

To solve the resource allocation problem, we augment multi-objective evolutionary algorithm, NSGA-II. In NSGA-II algorithm, solution of a chromosome contains the values of different objectives obtained from fitness functions and the solutions of a population are classified into different sets according to the ascending level of their domination. A set of solutions S dominates another solution set S' if each solution $x \in S$ is no worse than solution $x' \in S', \forall x'$ in every objectives and each solution $x \in S$ is strictly better than solution $x' \in S', \forall x'$ in at least one objective. If a solution set is not dominated by any other set of solutions, the elements of that set is called the non-dominated solutions. Each set of solutions represents a particular front on the solution space and the chromosomes generating those solutions are treated as the builder of the front. The non-dominated set of solutions provides the optimal front (first front) on the solution space, is termed as the pareto-front. Here, we extend the basic concept of NSGA-II and further refine it to develop an energy-delay co-optimized resource allocation for emergency management service in Cloudlet infrastructure as follows:

Population Initialization. In the augmented NSGA-II, rather than creating randomized initial population, a presorted initial population relying on the heuristics is generated. Here, the set of resources R in the combined resource pool is divided into k categories based on the ascending processing speed $\rho_r, \forall r \in R$. Similarly, the set of tasks T is also classified in k types according to the incremental size $\lambda_t, \forall t \in T$. Thereafter, taking the problem range and constraints (Eqs. 3–6) into account, mathematical combination is used to conceptually assign the tasks of j type to the resources of j^{th} category for generating the chromosomes of initial population. The structure of chromosomes is aligned with the steps of initial population creation, where a Gene Index symbolizes a particular task and the Gene refers to a specific resource.

Domination Count and Ranking. The solution space of a population is determined by the outcome of its member chromosome's fitness value on makespan and energy objectives using Eqs. 1–2. The solutions of combined parent and child population U_i for any generation $g_i; i \in \{1...G\}$, generated by its member chromosomes are ranked in different fronts within corresponding solution space based on the level of their domination according to NSGA-II [2].

Selection of Population. Since the size of combined population U_i for any generation g_i is $2N$, for iterative refinement, it is very important to select the best N number of chromosomes from U_i to form the parent population P_{i+1} for the next generation g_{i+1}. In this case, chromosome slots within P_{i+1} is filled with the builder chromosomes of comparatively better fronts on the solution space.

Crowding Distance Calculation. In selecting N number of chromosomes for the parent population P_{i+1} of next generation g_{i+1}, sometimes the available slot in P_{i+1} can be less enough to accommodate the entire builder chromosome set F_τ of a particular front. In this case, crowding distance of the solutions are calculated to identify the compatible builder chromosomes of that front to fill the available slot in P_{i+1}.

Extension of Genetic Operator. For a particular generation g_i, while generating the child population C_i from the parent population P_i, genetic operators such as fitness calculation, selection, crossover and mutation are applied. The fitness of the population is determined through the objective functions discussed in Eqs. 1–2. To imply mutation on the population, binomial distribution and to make crossover of a particular chromosome with the fittest chromosome of the population, simulated binary approaches are used. To select the fittest chromosome, for each chromosome c, the distance $c.d$ of makespan and energy consumption objective value $(c.M, c.E)$ from the theoretical lowest value of makespan and energy, $(0, 0)$ is calculated. The chromosome having minimum distance value from the origin is selected as the fittest chromosome of the population for making crossover. However, the selection of final solution and characteristics of the corresponding chromosome solely depends on the intention of service providers.

5 Simulation Results and Discussion

5.1 Simulation Environment

To conduct our experiments in Matlab, we use synthetic data, driven from real-world references and selects system parameters carefully for fair evaluation [6]. The parametric values for the simulation environment are summarized in Table 1. The value of simulation parameters within a specific range is set by a pseudo random number generator. For simplicity of the simulation, we consider three types of computing resources based on their heterogeneous processing speed. In addition, we set the ratio of local and virtual resources in the combined resource pool to $\frac{1}{3}$ and the ratio of dependent and independent tasks is set to $\frac{2}{5}$.

5.2 Simulation Scenarios and Result Analysis

The performance of our proposed approach is compared with a benchmark strategy which follows NSGA-II algorithm with randomized initial population and arbitrary chromosome selection during crossover.

Comparison of Pareto-Optimal Solutions and Generations to Meet Stopping Criteria. After 200 generations, the pareto-optimal solutions of our augmented NSGA-II and benchmark NSGA-II on fixed number of heterogeneous tasks (50) and resources (30) are depicted in Fig. 2(a). In this scenario, each pareto-optimal solutions of energy-delay co-optimized approach provides better outcome for both objectives, compared to the benchmark strategy. The initial population of proposed approach that is determined based on the task's size and processing speed of resources, inherently minimizes the overall makespan and energy consumption. In addition, selection of the chromosome having minimum distant solution from the origin for crossover further improves its performance.

Table 1. Simulation parameters

Parameter	Value
Population size	50
No of generations	50–500
Mutation rate	0.5
Crossover rate	0.5
Number of tasks in a service	35–65
Number of computing resources	15–45
Processing speed of virtual resources	10000–30000 MIPS
Processing speed of local resources	8000–15000 MIPS
Per unit energy consumption of virtual resources	50–150 W
Per unit energy consumption of local resources	40–90 W
Tasks data size	5000–10000 MI
Allowable completion time of all tasks	5000 ms
Maximum allowable energy consumption of service	2500 W
Data dependency threshold	1500 ms
Communication bandwidth	128–512 Kbps

Therefore, after a fixed number of generations, it provides significantly better solution than the randomized benchmark. The simulation results on Fig. 2(b) represents the efficacy of proposed approach in achieving no further optimization state within less generations compared to the benchmark strategy. For using pre-sorted population, compared to the benchmark, our approach implicitly advances a certain number of generations in meeting the stopping criteria.

Impacts of Varying Number of Tasks. The average makespan and energy consumption for varying number of tasks while the number of generations (200)

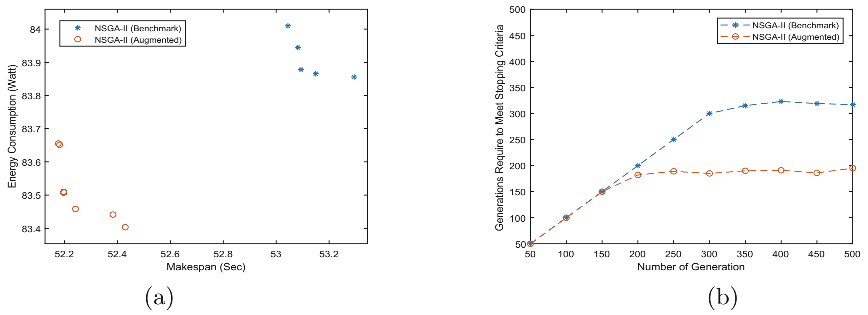
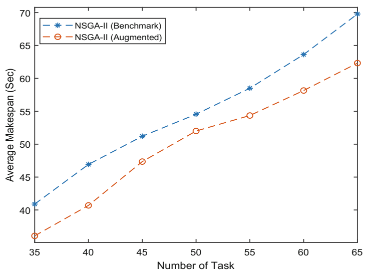
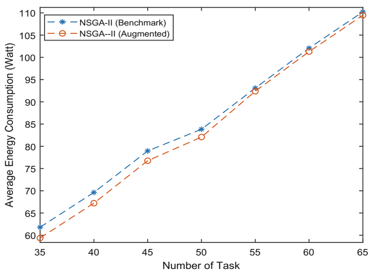


Fig. 2. (a) Pareto-optimal solutions and (b) Number of generations required to meet stopping criteria



(a)



(b)

Fig. 3. (a) Average makespan and (b) average energy consumption of resources for varying number of tasks

and resources (30) are fixed, are illustrated in Figs. 3(a) and (b) correspondingly. Since the resource number is fixed, with increasing of the tasks objective values increases for both approaches. However, compared to benchmark, our proposed approach provides improved solutions in this scenario due to presorted population. In the presorted population, the computing resources are categorized based on their processing speed for assigning the tasks to a particular type of computing resources according to their size. As a consequence, it generates efficient initial population and refines it consistently after each generation.

6 Conclusion and Future Directions

Evaluation results acquired through simulating different scenarios show that our augmented NSGA-II algorithm improves the benchmark NSGA-II algorithm by minimizing both makespan and energy consumption. In future, our proposed approach will be compared with other existing multi-objective problem solving approaches like PAES, SPEA. Furthermore, as our evaluation is based on simulation, sources of uncertainty in the system and their effects on simulation parameters, sensitivity of independent variables will be analysed and implemented in real-world testbed. The impact of complexities on different tasks will also be investigated. Utilizing edge resources for task allocation will be a good extension.

Acknowledgement. The authors would like to thank Data61, CSIRO, Australia for supporting the work.

References

1. Chen, Y., Du, Z., García-Acosta, M.: Robot as a service in cloud computing. In: 5th IEEE International Symposium on Service Oriented System Engineering, pp. 151–158. IEEE Computer Society, Washington (2010)
2. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)

3. Hu, G., Tay, W.P., Wen, Y.: Cloud robotics: architecture, challenges and applications. IEEE Netw. **26**(3), 21–28 (2012)
4. Mahmud, M.R., Afrin, M., Razzaque, M.A., Hassan, M.M., Alelaiwi, A., Alrubaian, M.: Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure. Softw.: Pract. Exp. **46**(11) (2016)
5. Mouradian, C., Yangui, S., Glitho, R.H.: Robots as-a-service in cloud computing: search and rescue in large-scale disasters case study. CoRR (2017)
6. Rahman, A., Jin, J., Cricenti, A., Rahman, A., Panda, M.: Motion and connectivity aware offloading in cloud robotics via genetic algorithm. In: IEEE Global Communications Conference, pp. 1–6 (2017)
7. Rahman, A., Jin, J., Cricenti, A., Rahman, A., Yuan, D.: A cloud robotics framework of optimal task offloading for smart city applications. In: IEEE Global Communications Conference, pp. 1–7 (2016)
8. Wu, D., Zeng, G., Meng, L., Zhou, W., Li, L.: Gini coefficient-based task allocation for multi-robot systems with limited energy resources. IEEE/CAA J. Autom. Sin. **5**(1), 155–168 (2018)

Services in IoT: A Service Planning Model Based on Consumer Feedback

Christian Cabrera(✉), Andrei Palade, Gary White, and Siobhán Clarke

Distributed Systems Group, School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
{cabrerac,paladea,whiteg5,Siobhan.Clarke}@scss.tcd.ie
http://www.dsg.cs.tcd.ie/

Abstract. IoT offers a large number of services from different providers. These services frequently need to be composed to provide novel applications. Current work in IoT service composition can be classified as conversation-based or interface-based. Conversation-based approaches need the manual definition of service plans, which is not feasible in IoT because of the large scale. Interface-based approaches use planning to automate the composition process. Such automation avoids human intervention, but some incorrect services can appear in the discovered plans. The efficiency of these approaches is poor because they perform intensive search in large spaces. This paper proposes a model for service composition with minimal human intervention using consumers' feedback. Results show that our model outperforms its competitors.

Keywords: Internet of Things · Service Oriented Computing
Service composition · Service discovery

1 Introduction

The IoT and Service Oriented Computing (SOC) support cyber-physical systems where things offer capabilities as services that can be composed to create novel applications [2,3]. This composition process is a transformation of consumer requests into the execution of a set of services [9]. A service plan defines the services to be executed but its discovery is challenging in IoT because of the large number of services [4]. The composition approaches in IoT can be classified as conversation-based, or interface-based [12]. Service conversations use process models [11] to represent service plans which are manually defined [8]. This considerable human intervention is not feasible in IoT because users cannot know about all available services. Interface-based approaches compose services based on planning algorithms. They can create service plans automatically, but incorrect services can appear, affecting search accuracy [12]. Search latency is also impacted by large search spaces and exhaustive exploration of all possible services combinations. Novel composition mechanisms for IoT should include historical data in progressive searches, with sensible ranking methods [10].

This paper proposes a model to discover service plans in IoT that uses consumer feedback to guarantee search efficiency minimising human intervention. Consumer feedback (i.e., historical data from previous searches) determines if a discovered plan was correct or incorrect and is used in our model to improve search accuracy. The latency of the process is reduced through two strategies: first, the use of consumers' feedback avoids the exploration of incorrect combinations of services. Second, the model explores the plans according to how well each plan meets the request's requirements. The model explores the most promising plans (i.e., search space reduction), and avoids wasting time on the less promising plans (i.e., progressive search). The proposed approach is evaluated and compared with a classic planning algorithm [6] and a conversation-based approach [12] on both centralised and distributed infrastructures.

This paper summarises the related work in Sect. 2. Section 3 presents the discovery process. Sections 4 and 5 present the evaluation and conclusions.

2 Related Work

Service composition in IoT can be classified as conversation-based and interface-based [12]. Deng et al. [7] propose an architecture where requesters specify a service conversation to select a composition with the lowest latency. Baker et al. [1] propose a service composition algorithm where cloud-providers define service conversations for each offered service. Users requests (i.e., I/O parameters) are matched with the service conversations. Urbieta et al. [12] present a service composition model that selects services whose capabilities conform a predefined service conversation. Conversation-based approaches offer good retrieval accuracy. However, they are time-consuming, error-prone, and require high human intervention [14]. It is not feasible to assume that users will know about all the available services and all their possible combinations in large-scale environments.

Interface-based approaches compose plans automatically. Zhao et al. [13] introduce a mechanism where sensors are grouped into classes that are chained using a planning algorithm to fulfill user goals (i.e., I/O parameters). The resulting plan is used to search services according to energy efficiency constraints. Chen et al. [6] present an approach that supports adaptive service composition. Service discovery is based on a planning algorithm that uses semantic matchmaking of I/O signatures. Interface-based approaches offer automatic planning. However, search precision can be affected negatively by the retrieval of incorrect services [12]. For example, a service $s1$ that provides readings about wind speed and a service $s2$ that consumes data of buses speeds are likely to match because the output type of $s1$ is equivalent to the input type of $s2$ (i.e., both are speeds). Both conversation and interface-based categories present performance problems in terms of response time as they perform complex processes in full search spaces and look for all the possible paths even when they are incorrect [6,12].

3 Composition of Services in IoT Environments

We assume that a network of gateways manages the services offered by IoT environments. Figure 1 shows the service discovery engine (SDE) that is deployed on each gateway. It receives consumer requests and feedbacks, forwards discovery messages to other gateways, and sends response messages to consumers. Each gateway has a registry to store service descriptions that providers advertise through a registry engine described in our previous work [3]. Service descriptions are defined as $\boldsymbol{s}_{desc} = \langle id, I, O, D \rangle$, consisting of an identifier, inputs, outputs, and domains. A request is defined as $r = \langle I, O \rangle$, consisting of inputs and outputs. The engine uses a set of ontologies (i.e., *knowledge model*) to annotate services and requests parameters and services domains. These ontologies are defined by the OWLS-TC V4[1] and our previous work [5]. The *service planner* and *matchmaker* use this semantic information to reduce the search space and identify service relations. The *planner* uses consumers feedback to generate accurate service plans. Consumers mark as correct or incorrect each discovered plan and the *feedback manager* stores this feedback in two data structures:

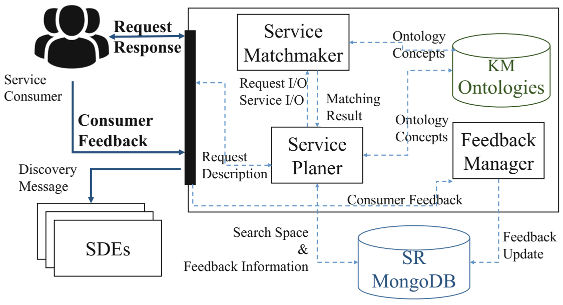


Fig. 1. Service discovery engine.

– Plans: This stores past discovered plans as graphs that include vertices (i.e., services in a plan), and edges (i.e., relations between two services in a plan).
– Relationships: This stores each relation between two services (i.e., an edge between two vertices) discovered by the SDE in past plans. Each relation includes the source parameter type, the source service domain, the target parameter type, the target service domain, the number of plans that have included this relation, and the number of successful plans that have included this relation. This counter of successful plans is updated when a consumer sends the feedback of a plan that includes the relation.

Each identified relation between two services is validated against the consumer feedback in the registry. This knowledge represents relationships between

[1] OWLS-TC V4: http://projects.semwebcentral.org/projects/owls-tc/.

domains and avoids the inclusion of services that have input/output relationships but belong to non-related domains. For example, service S_1 has body temperature as output and belongs to the *health* domain, service S_2 has room temperature as input and belongs to the *buildings* domain. They have an input/output relationship as S_1 produces, and S_2 consumes temperature measurements, but it is incorrect and will be ignored because of the negative feedback.

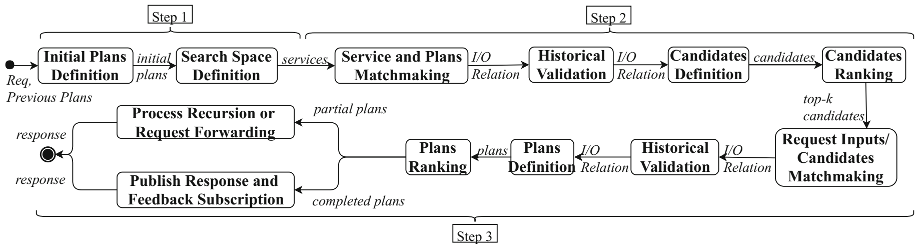


Fig. 2. Service discovery process.

Figure 2 shows the discovery process where the SDE searches for plans that meet a request. This process starts when a gateway receives a request message from the consumer or other gateways. This message includes two inputs: the request that is the consumer requirement and includes the I/O parameters. And a list of previous plans that is a set of graphs that partially solve the request and was discovered previously. It is empty if the message comes from the consumer. The SDE performs the planning process based on these inputs as follows:

Step 1: Initial Plans and Search Space Definition

The *planner* receives the request and previous plans. If the list of previous plans is empty, the planner creates a graph with two vertices: initial and final. The outputs of the initial vertex are the inputs in the request and the inputs of the final vertex are its outputs. The final vertex is an unsolved vertex as there are no vertices that provide their inputs. If the list of previous plans is not empty, these partial plans are used as initial plans. The *planner* defines the search space as a set of services which outputs can match to the unsolved vertices in a plan. The *planner* queries the registry to get these services according to their I/O parameters. This query (i.e., progressive search) improves the search efficiency as the planner does less iterations in next steps.

Step 2: Candidates Plans Discovery

The *planner* tries to complete the initial plans using the search space from the step 1. It uses the *matchmaker* that discovers I/O relations comparing two service parameters and gives a score if the parameters match. This score is used to rank and select the candidate plans (i.e., plans that can solve the request). The *matchmaker* is based on matching methods that compare two parameters (i.e., P_1, P_2) using their annotated types according to an ontology O as follows:

– *equivalent*(P_1, P_2): The type of P_1 is conceptually equivalent to the type of P_2 in O. This method gives a score of 4 to the relation between P_1 and P_2.
– *plugin*(P_1, P_2): The type of P_1 is a sub-concept of the type of P_2 in O. This method gives a score of 3 to the relation between P_1 and P_2.
– *subsume*(P_1, P_2): The type of P_1 is a super concept of the type P_2 in O. This method gives a score of 2 to the relation between P_1 and P_2.
– *similarity*(P_1, P_2): This method calculates the cosine similarity between P_1 and P_2. This method gives a score of 1 to the relation between P_1 and P_2.

The *planner* uses the consumers' feedback to validate each discovered relation. It computes an historic success index (HSI) for a given edge (Eq. 1).

$$HSI(e) = \frac{successfulPlans}{totalPlans} \tag{1}$$

where $successfulPlans$ is the number of correct plans that have included the edge e and $totalPlans$ is the number of plans that have included the edge e. The *planner* uses this index to determine whether a discovered relation is valid. If $HSI >= feedbackThreshold$, the discovered relation is included in the plan. The *planner* creates a candidate plan for each combination of valid discovered relations for an initial plan. Previous planning approaches create and explore candidates for all possible combinations, which increases their latency. The number of candidates can be large even with this constraint. Our model ranks and selects the top k candidates based on the scores defined by the *matchmaker*. Each candidate has a functional matching index FMI that defines how well it meets the request's functional requirements. This index follows the Eq. 2.

$$FMI(G(v, e)) = \frac{\sum_{i=1}^{n} x_i}{4n} \tag{2}$$

where n is the number of edges in the candidate plan G, and x_i is the score for a particular edge according to the *matchmaker*. $4n$ is the maximum value that the plan can receive. The candidate is added to the list of candidates if the length of the list is less than or equal to K and $FMI >= functionalThreshold$.

Step 3: Response Definition and Feedback Management

The *planner* uses the *matchmaker* to define which of the candidate plans solve the request. The *matchmaker* compares the outputs of the initial vertex (i.e., inputs of the request) with the inputs of the non-solved vertices in each candidate. The planner validates the discovered relations calculating the HSI and comparing it with the $feedbackThreshold$. If the request is solved after this process, the list of solutions is ranked using the FMI and published to the consumer. Otherwise, the request and the list of ranked candidates is sent to the next gateway where the process starts again with the candidates as a list of previous plans. The SDE stores the solutions in the registry and subscribes to a feedback message from the consumer. The consumer publishes feedback sending the plan identifier and a boolean mark that states if the plan is correct or not.

4 Evaluation

We implemented the SDE in Python 3.5, services and feedbacks are stored in a MongoDB, and the communication is based on MQTT. We compare our model against a classic planning algorithm [6] and a conversation based approach [12]. We measure search precision, latency in terms of response time (ms), and human intervention in terms of number of bytes in the request and feedback. We deploy the approaches in centralised and distributed infrastructures to evaluate its performance. The centralised infrastructure uses the Kelvin cluster managed at the Trinity Centre for High Performance Computing (TCHPC)[2]. One node hosts the client that sends service requests and the SDE that solves them. The distributed infrastructure is composed by 5 Raspberry Pi3 that run Raspbian, have 1 GB of RAM and an SD card with 16 GB. One of the boards is the consumer and sends requests to the other boards. We run five rounds of 100 requests for each approach with different number of services in the environment. This number varies from 2000 to 10000, incremented by 2000. We repeated this process for all the combinations of the parameters in our model. The K value varies from 5 to 25, incremented by 10, the $feedbackThreshold$ and the $functionalThreshold$ values vary from 0.0 to 1, incremented by 0.2.

The dataset is composed by 946 services and 47 requests from which responses are composed plans that can include from 1 to 5 services. The composed plans were validated to determine the set of relevant responses, this set has 14937 plans. 9054 mockup services are added to evaluate scalability. The implemented approaches, and data sets used in this paper are available on the TCD GitLab[3].

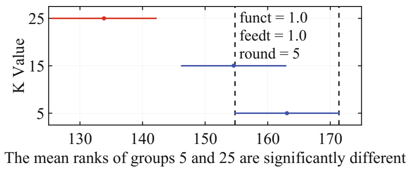
4.1 Results

We evaluate our model (i.e., feedback-based) with different values for its parameters (i.e., Feedback and Functional Thresholds, K Value, and Number of Rounds). We apply a Kruskal-Wallis test with a multiple comparison of ranked means to measure how the values of each parameter affects our model. The feedback and functional threshold do not have a significant impact on the results. However, the best results are achieved when these thresholds have a value of 1.0. The K value has a significant effect in both search precision and response time (Fig. 3). The best results are achieved when K is 5. It improves the precision because only the five most promising candidates are explored. The response time is also minimised as the size of explored plans is small. The number of rounds significantly affects search precision and response time (Fig. 4). It means that our model is more efficient over the time when more consumer feedback is available. The best results are achieved when the number of rounds is 5. The rest of this section compares our model under these parameters against a classic planning algorithm [6] and a conversation-based approach [12].

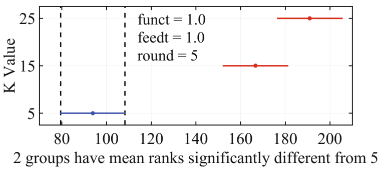
Figure 5 shows the search precision with 10000 services. The classic approach achieves a precision mean of 0.63, the conversation-based approach achieves a

[2] Kelvin Details - https://www.tchpc.tcd.ie/resources/clusters/kelvin.

[3] Smart City SD - https://gitlab.scss.tcd.ie/groups/smartcitySD/subgroups.

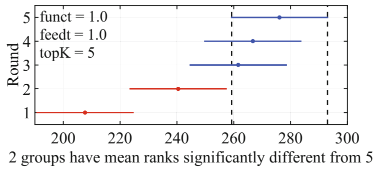


(a) Search Precision

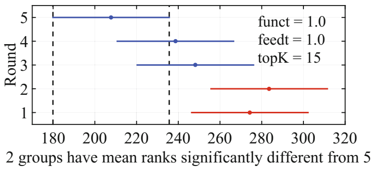


(b) Search Response Time

Fig. 3. K value - Kruskal-Wallis analysis



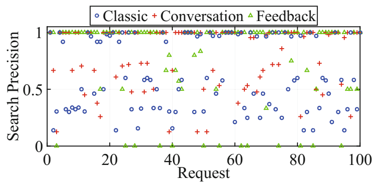
(a) Search Precision



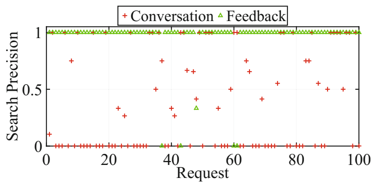
(b) Search Response Time

Fig. 4. Number of rounds - Kruskal-Wallis analysis

mean of 0.81 and the feedback-based approach achieves a mean of 0.8 in the centralised infrastructure (Figure 5(a)). Figure 5(b) shows the search precision in the distributed infrastructure. The conversation-based approach achieves a precision mean of 0.4 and the feedback-based approach achieves a mean of 0.95. Our model has a similar mean for search precision to the conversation-based approach in the centralised infrastructure, but it overcomes its competitors in the distributed infrastructure. The conversation-based approach precision is affected when the registry is distributed because when a task is solved in a gateway it is not explored in others. Our model has a better precision in the distributed infrastructure because in the centralised architecture there is more chance to explore incorrect services. Conversation-based approach and feedback-based approaches overcome the classic planning approach because it does not validate the discovered services. The classic approach did not run properly in the distributed infrastructure as its intensive search overloads the raspberry Pi boards.

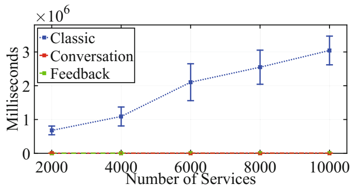


(a) Centralised Infrastructure

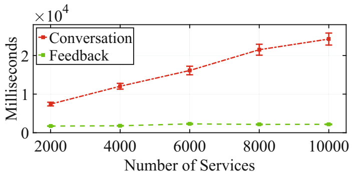


(b) Distributed Infrastructure

Fig. 5. Search precision with 10000 services

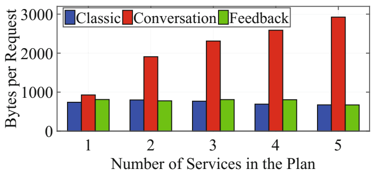


(a) Centralised

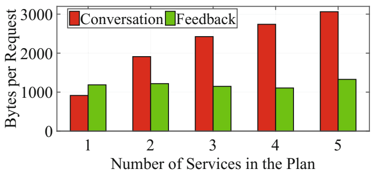


(b) Distributed

Fig. 6. Discovery response time



(a) Centralised



(b) Distributed

Fig. 7. Human intervention in bytes per request

Figure 6 shows the average latency with different number of services. Figure 6(a) presents the results in the centralised architecture. The classic planning approach has the largest average latency, it varies from 11 min with 2000 services to 50 min with 10000 services. The conversation-based approach has the next latency, it varies from 1458 ms with 2000 services to 5191 ms with 10000 services. Our approach is slightly better, its latency varies from 1458 ms with 2000 services to 3048 ms with 10000 services. Figure 6(b) shows the results in the distributed infrastructure. Our approach offers the lowest average latency, it varies from 1723 ms with 2000 services to 2175 ms with 10000 services. The largest latency is 2290 ms with 6000 services. The conversation-based approach latency varies from 7419 ms with 2000 services to 24273 ms with 10000 services. Our approach has the best latency because it iterates over the services that match, the conversation-based approach iterates over all the services in the registry. Our approach outperforms the classic approach because of the K value that limits the number of explored candidates. We analyse the human intervention in each approach (Fig. 7). We measure the number of bytes per request and feedback against the number of services in the discovered plans. The classic and the feedback-based approaches have a human intervention around 700 bytes per request for different plan lengths in the centralised scenario. The human intervention in the conversation-based approach varies from 927 bytes with a plan length of 1 to 2924 bytes with a plan length of 5 (Fig. 7(a)). Our approach also has less human intervention in the distributed infrastructure (i.e., around 1000 bytes). The human intervention in the conversation-based approach varies from 914 bytes with plan length 1 to 3062 bytes with a plan length of 5. The conversation-based approach requires more human intervention when the plans length increases because the number of tasks to be defined is larger. Our approach minimise this intervention because the number of request parameters do not vary with the plans length. The human intervention in our approach is slightly bigger than in the classic approach because it includes the feedback.

5 Conclusions

This paper presents a model for services in IoT which guarantees an efficient service discovery. This model uses consumer feedback to validate the discovered service relations. It avoids the exploration and inclusion of incorrect options. Functional requirements are used to define the search space, and the selection of candidates' plans minimising the number of options to reduce the latency. We compare our model against a classic planning algorithm and a conversation-based approach in centralised and distributed infrastructures. Results show that our model outperforms its competitors regarding search precision and response time with minimal human intervention. We will investigate how to adapt our model to larger environments such as smart cities in future work.

Acknowledgments. This work is supported by Science Foundation Ireland under grant 13/IA/1885. Computational resources have been provided by the TCHPC funded by eINIS.

References

1. Baker, T., Asim, M., Tawfik, H., Aldawsari, B., Buyya, R.: An energy-aware service composition algorithm for multiple cloud-based IoT applications. J. Netw. Comput. Appl. **89**, 96–108 (2017)
2. Borgia, E.: The internet of things vision: key features, applications and open issues. Comput. Commun. **54**, 1–31 (2014)
3. Cabrera, C., et al.: Implementing heterogeneous, autonomous, and resilient services in IoT: an experience report. In: 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM). IEEE (2017)
4. Cabrera, C., Palade, A., Clarke, S.: An evaluation of service discovery protocols in the internet of things. In: Proceedings of the Symposium on Applied Computing, pp. 469–476. ACM (2017)
5. Cabrera, C., Palade, A., White, G., Clarke, S.: The right service at the right place: a service model for smart cities. In: 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom) (2018)
6. Chen, N., Cardozo, N., Clarke, S.: Goal-driven service composition in mobile and pervasive computing. IEEE Trans. Serv. Comput. **11**, 49–62 (2016)
7. Deng, S., Huang, L., Taheri, J., Yin, J., Zhou, M., Zomaya, A.Y.: Mobility-aware service composition in mobile communities. IEEE Trans. Syst. Man Cybern.: Syst. **3**, 555–568 (2017)
8. Klein, M., Bernstein, A.: Toward high-precision service retrieval. IEEE Internet Comput. **1**, 30–36 (2004)
9. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Comput. Surv. (CSUR) **48**(3), 33 (2016)

10. Pattar, S., Buyya, R., Venugopal, K., Iyengar, S., Patnaik, L.: Searching for the IoT resources: fundamentals, requirements, comprehensive review and future directions. IEEE Commun. Surv. Tutor. **20**, 2101–2132 (2018)
11. Thiagarajan, R.K., Srivastava, A.K., Pujari, A.K., Bulusu, V.K.: BPML: a process modeling language for dynamic business models. In: Proceedings Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems 2002. (WECWIS 2002). pp. 222–224. IEEE (2002)
12. Urbieta, A., Gonzalez-Beltran, A., Mokhtar, S.B., Hossain, M.A., Capra, L.: Adaptive and context-aware service composition for IoT-based smart cities. Futur. Gener. Comput. Syst. **76**, 262–274 (2017)
13. Zhao, D., Zhou, Z., Ning, K., Duan, Y., Zhang, L.J.: An energy-aware service composition mechanism in service-oriented wireless sensor networks. In: 2017 IEEE International Congress on Internet of Things (ICIOT), pp. 89–96. IEEE (2017)
14. Zhao, Y., Wang, S., Zou, Y., Ng, J., Ng, T.: Automatically learning user preferences for personalized service composition. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 776–783. IEEE (2017)

Social and Interactive Services

Crowdsourcing Task Scheduling in Mobile Social Networks

Jiahao Fan, Xinbo Zhou, Xiaofeng Gao(✉), and Guihai Chen

Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai 200240, China
{j.h.fan,zxb16161616}@sjtu.edu.cn, {gao-xf,gchen}@cs.sjtu.edu.cn

Abstract. With the growing popularity of mobile devices, a new paradigm called mobile crowdsourcing emerged in the recent years. Mobile users with restricted computational capability and sensing ability are now able to conduct complex tasks with the help of other users in the same mobile crowdsourcing system. In this paper, we consider the mobile crowdsourcing system model based on the spontaneously-formed mobile social networks (MSNs). We introduce two crowdsourcing task scheduling problems under this system model, with one problem aiming to minimize the operating cost of some crowdsourcing tasks and the other focusing on minimizing the overall completion time of tasks belonging to the same project. Correspondingly, under offline settings, we propose an optimal algorithm and an approximation algorithm for these two problems respectively. The optimality and the approximation ratio are analyzed accordingly. Based on these two algorithms, we further design two online algorithms to deal with the problems under online settings and their competitive ratios are computed. Finally, we verify the effectiveness and efficiency of the proposed methods through extensive numerical experiments on synthetic datasets.

Keywords: Crowdsourcing · Task scheduling · Mobile social network

1 Introduction

With the rapid development of technology nowadays, small-sized portable mobile devices are prevailing and this enables mobile users to conduct complex tasks with the embedded powerful mobile sensors. These tasks may include air quality [4] and urban noise [11] measurement, traffic [6] and road surface [13] monitoring, etc. Since the computational capability and sensing ability of a single mobile user may not be enough to carry out a large project which consists of many relatively small and independent tasks, it is necessary to seek the help of other users and distribute these tasks among them in order to complete the whole project as efficiently as possible. This forms the idea of *mobile crowdsourcing*.

Most mobile crowdsourcing activities are carried out in a large mobile crowdsourcing system. This kind of crowdsourcing systems usually requires quite a lot of human and material resources for management and maintenance. If a requester just wants to distribute the crowdsourcing tasks through his social relationships (including families, friends, colleagues, acquaintances, etc.), the requester may conduct his own crowdsourcing activities by taking the advantage of the existing *mobile social networks* (MSNs) to avoid unnecessary overheads and thus save the budget. This idea seems more appealing than the traditional crowdsourcing paradigm to those requesters with limited budgets.

In this paper, we mainly focus on the task scheduling problems for MSN-based mobile crowdsourcing systems. Unlike traditional task scheduling problems in mobile crowdsourcing, we require the requester and the crowd workers to actually make person-to-person contacts, which are some probabilistic events in the mobile social networks, so that crowdsourcing tasks can be sent by the requester and their feedbacks can be delivered by the crowd workers. Each crowdsourcing task requires one contact between the requester and the crowd worker when it is being distributed and another when its feedback is being delivered. Our objectives are strongly related to the completion time of each task, which consists of the time for its distribution, processing, and feedback delivery.

Suppose there is an operating cost related to task s_i when it is being scheduled and processed in the mobile crowdsourcing system, and the cost is proportional to the completion time of s_i. Then, the cost of s_i can be denoted as w_iC_i, where w_i is the proportion of the cost to the task completion time, and C_i is the actual completion time of s_i. Our min-WCT problem aims to minimize the total weighted completion time of all the crowdsourcing tasks, so that the total cost of conducting the crowdsourcing campaign can be minimized. Meanwhile, in some scenarios, the crowdsourcing tasks of a requester may be part of a larger project, and all of them should be completed as soon as possible. Hence, we introduce the min-MCT problem to minimize the maximum completion time of all the crowdsourcing tasks (i.e., $\max C_i$) in order to accelerate the progress of the following stages in the project.

More specifically, the main contributions of this paper include:

1. We introduce two task scheduling problems for crowdsourcing in mobile social networks and formulate them as the min-WCT problem and the min-MCT problem respectively.
2. For the min-WCT problem, we propose an optimal offline algorithm named LWF and give the proof of its optimality. Based on LWF, we design an online algorithm named CosMOS to deal with min-WCT under online settings and give the analysis of its competitive ratio.
3. For the min-MCT problem, we propose an offline approximation name LRSTF and give the analysis of its approximation ratio. Based on LRSTF, we also design another online algorithm named TiMOS to deal with min-MCT under online settings and give the analysis of its competitive ratio.
4. We conduct extensive numerical experiments on synthetic datasets to compare our algorithms with some traditional scheduling algorithms. Both the theoretical analysis and the experimental results validate the effectiveness and efficiency of our designs.

The rest of this paper is organized as follows. Section 2 discusses some related works. Section 3 describes the MSN-based mobile crowdsourcing system model. In Sect. 4, we formulate the min-WCT problem, and propose an optimal offline solution named LWF and an online solution named CosMOS. In Sect. 5, we formulate the min-MCT problem, and introduce an offline approximation named LRSTF and an online solution named TiMOS. In Sect. 6, we conduct numerical experiments to evaluate the performance of our designs. Section 7 is the final conclusion.

2 Related Work

Mobile crowdsourcing is getting more and more research interests in the recent years. Some of the literatures focus on the framework and application design [9,18], while others pay more attention to the specific stages in the process of mobile crowdsourcing, such as task scheduling [16], incentive mechanisms [5], quality control [17], and security and privacy issues [15]. Generally, there are three components or participants in a typical mobile crowdsourcing system (MCS): requesters or crowdsourcers, crowd workers, and crowdsourcing platform [14]. Specifically, to crowdsource a task, a requester submits the task to a crowdsourcing platform, and optionally, after receiving the solutions provided by crowd workers, rates their qualities. Crowd workers choose to work on those tasks and attempt to submit their solutions as feedback. An intermediation platform (i.e., crowdsourcing platform) builds a link between the requesters and workers, which serves as a crowdsourcing enabler and has some rules for the whole lifecycle of crowdsourcing, such as the skill-set, certification level, due date, expected outcomes, and payments for the crowd workers.

With the growing popularity of mobile smart devices, the concept of the mobile social networks (MSNs) gradually comes into our view. As an application scenario, MSN is widely adopted in all kinds of problems including communication [19], social community detection [10], etc. Besides, researchers are also interested in exploring inner features and characteristics regarding MSN itself. In this paper, we focus on the task scheduling problems in MSN-based mobile crowdsourcing systems. There have been plenty of previous literatures concerning the topic of task scheduling problems [1–3]. Our problems may be similar to the traditional scheduling problems on parallel machines, but our system model includes a probabilistic event which is the person-to-person contact between the requester and the crowd workers. In order to solve the problems, we have to make online task scheduling decisions, and this makes our designs more realistic and applicable under the scenario of crowdsourcing in mobile social networks.

3 System Model

In this section, we introduce the participants in the mobile social network (MSN) and how they interact with each other under the paradigm of crowdsourcing.

Here, the mobile social network model in [16] is adopted. Consider a crowd of mobile users in the network, denoted as $U = \{u_0, u_1, \ldots, u_m\}$. Each of these mobile users carries a mobile device which supports wireless communication and possesses certain processing capability to carry out some crowdsourcing tasks. Two mobile users can communicate with each other only if they get close enough for Bluetooth to work, or enter the communication range of some access points respectively to interact indirectly via WiFi. Assume that the connection duration and bandwidth are enough to satisfy our needs to communicate for the purpose of crowdsourcing. We further adopt the mobility model in [7], where the pairwise inter-contact time is exponentially distributed. This means that the inter-meeting time between any pair of mobile users u_i and u_j $(i \neq j)$ follows the exponential distribution with parameter λ_{ij}. This parameter can be estimated from historical communication records between u_i and u_j.

Under the paradigm of crowdsourcing, there is a user in the MSN that hopes to recruit other mobile users to help complete some crowdsourcing tasks. The user with this kind of need is called the *requester*, and those who have the potential to carry out these tasks are called *crowd workers*. Without loss of generality, we assume u_0 is a requester and the other m users $\{u_1, u_2, \ldots, u_m\}$ in the MSN are crowd workers that are encouraged to participate in the crowdsourcing campaign by some incentive mechanisms. Since crowd workers are not supposed to communicate with each other, we denote λ_{0j} as λ_j for simplicity.

Suppose that the requester u_0 has n indivisible crowdsourcing tasks, denoted as $S = \{s_1, s_2, \cdots, s_n\}$. The workload of task $s_i \in S$ is represented by its *Required Service Time* (RST), denoted as τ_i. Each crowdsourcing task should be assigned to only one crowd worker by the requester. The distribution, as well as the feedback, of each task requires one contact between the designated crowd worker and the requester. In practice, it is more efficient for the requester to send a batch of crowdsourcing tasks to a crowd worker at their first contact and receive the feedbacks whenever they are ready at the time of their following contacts. We use $\Lambda = \{S_1, S_2, \ldots, S_m\}$ to denote an scheduling decision of the n crowdsourcing tasks to the m crowd workers, where $\bigcup_{S_j \in \Lambda} S_j = S$ and $S_{j_1} \cap S_{j_2} = \phi (j_1 \neq j_2)$.

A crowd worker will process the tasks assigned to him one by one until there is none left. Thus, the *Completion Time* (CT) of task s_i includes three parts: (a) the time for the requester u_0 and the crowd worker u_j, to whom we assign the task s_i (i.e., $s_i \in S_j$), to meet for the first time and complete the task distribution process (this includes the distribution of all the tasks in S_j), (b) the time for u_j to process all the tasks prior to s_i in S_j and task s_i itself, and (c) the time for u_0 and u_j to meet again after the feedback of s_i becomes available.

Since we cannot foresee the exact time when the requester and the crowd workers meet, we define the *Expected Meeting Time* (EMT), which is $\frac{1}{\lambda_j}$ under the assumption of the exponentially distributed inter-meeting time between u_0 and u_j, to represent the time in part (a) and part (c). Therefore, the completion time of task s_i, which is denoted as C_i, consists of two expected meeting time intervals and the time for the crowd worker u_j to process all the tasks prior to s_i in S_j and task s_i itself.

4 Cost Minimized Scheduling

In this section, we first formulate the cost-related min-WCT problem for crowdsourcing task scheduling. Then, we propose an optimal algorithm LWF to solve the problem under offline settings, followed by the proof of its optimality. Finally, we give an online algorithm CosMOS based on LWF to solve the problem under online settings together with the analysis of its competitive ratio.

4.1 Problem Formulation

Consider the system model described in the previous section (Sect. 3). Assume that all the tasks have the same required service time, i.e., $\tau_1 = \tau_2 = \cdots = \tau_n$. Suppose there is an operating cost related to task s_i when it is being scheduled and processed in the crowdsourcing system, and the cost is proportional to the completion time of task s_i. Our goal is to minimize the total operating cost of carrying out all the tasks in S. This is equivalent to the following definition of the min-WCT problem.

Definition 1 (Minimizing the Weighted Completion Time (min-WCT)). *Given tasks $S = \{s_1, s_2, \ldots, s_n\}$ with the same required service time (i.e., $\tau_1 = \tau_2 = \cdots = \tau_n$) and their corresponding weights $W = \{w_1, w_2, \ldots, w_n\}$, min-WCT aims to find an scheduling decision $\Lambda = \{S_1, S_2, \ldots, S_m\}$ among crowd workers $U = \{u_1, u_2, \ldots, u_m\}$ such that the total weighted completion time of all the tasks $\sum_{i=1}^{n} w_i C_i$ is minimized.*

Denote $\sum_{i=1}^{n} w_i C_i$ as WCT in this section for simplicity.

4.2 Offline Task Scheduling

First, we would like to consider the min-WCT problem under offline settings. In this scenario, the requester is supposed to make the scheduling decision before any contacts with the potential crowd workers and stick to this decision throughout the entire crowdsourcing campaign.

Intuitively, we wish to reduce the completion time for tasks with large weights so that we can reduce the total weighted completion time significantly. Following this idea, we design the *Largest Weight First* (LWF) algorithm (Algorithm 1) to solve the min-WCT problem under offline settings. The concept of a crowd worker's *Expected Workload* (EW) is defined as follows.

Definition 2 (Expected Workload (EW)). *The expected workload EW_j of a crowd worker u_j consists of three parts: (a) the expected meeting time for u_j and the requester u_0 to meet for the first time and complete the task distribution process, (b) the total required service time of all the tasks assigned to u_j, i.e., the tasks in S_j, and (c) the expected meeting time for u_j and u_0 to meet again and deliver the feedback of the tasks. Furthermore, we define $EW_j = \frac{2}{\lambda_j}$ if no task is assigned to u_j and S_j is empty.*

Algorithm 1: The LWF Algorithm

```
Input: $S = \{s_1, s_2, \ldots, s_n : \tau_1 = \tau_2 = \cdots = \tau_n\}$,
       $W = \{w_1, w_2, \ldots, w_n : w_1 \geq w_2 \geq \cdots \geq w_n\}$,
       $U = \{u_1, u_2, \ldots, u_m : \lambda_1, \lambda_2, \ldots, \lambda_m\}$
Output: $\Lambda_{\text{LWF}} = \{S_1, S_2, \ldots, S_m\}$
1 for $j \leftarrow 1$ to $m$ do
2     $S_j \leftarrow \phi$;
3     $EW_j \leftarrow \frac{2}{\lambda_j}$;
4 for $i \leftarrow 1$ to $n$ do
5     $j_{min} \leftarrow \arg\min\{EW_k \mid u_k \in U\}$;
6     $S_{j_{min}} \leftarrow S_{j_{min}} \cup \{s_i\}$;
7     $EW_{j_{min}} \leftarrow EW_{j_{min}} + \tau_i$;
8 return $\Lambda_{LWF} = \{S_1, S_2, \ldots, S_m\}$;
```

The Optimality. We briefly describe the proof of its optimality as follows. Assume $\Lambda_{\text{opt}} = \{S_1^*, S_2^*, \ldots, S_m^*\}$ is the optimal scheduling decision for the min-WCT problem under offline settings.

Lemma 1. *For any $S_j^* \in \Lambda_{opt}$, a task $s_{j_p} \in S_j^*$ with a larger weight w_{j_p} is processed prior to the processing of another task $s_{j_q} \in S_j^* \backslash \{s_{j_p}\}$ with a smaller weight w_{j_q} (ties may be broken arbitrarily).*

According to Lemma 1, we can conclude that in order to achieve the optimal solution, tasks assigned to the same crowd worker must be processed in a non-increasing order according to their weights by this worker.

Lemma 2. *Suppose that we have somehow determined an scheduling decision for $\{s_1, s_2, \ldots, s_{i-1}\}$ and this partial scheduling shall not be further changed, which means all the other tasks must be scheduled behind them, the optimal solution to the min-WCT problem under this condition must assign task s_i to the crowd worker with the currently smallest expected workload, denoted as u_j.*

Then, we are ready to prove the optimality of the LWF algorithm as follows.

Theorem 1. *As for the min-WCT problem, the LWF algorithm achieves the optimal solution by assigning the task with the largest weight to the crowd worker with the currently smallest expected workload in each round until all tasks in S have been assigned.*

Proof. We prove this theorem by induction. The base case where no task has been assigned is trivial. For the inductive step, assume we can obtain the optimal scheduling decision for tasks in $\{s_1, s_2, \ldots, s_{k-1}\}(1 \leq k \leq n)$ by the LWF algorithm. Then, according to Lemmas 1 and 2, we can also obtain the optimal scheduling decision for tasks in $\{s_1, s_2, \ldots, s_k\}$ through another round of scheduling by the LWF algorithm. Finally, we can obtain the optimal scheduling decision for all the tasks in $S = \{s_1, s_2, \ldots, s_n\}$ from the output of the LWF algorithm. This finishes the proof of Theorem 1.

4.3 Online Task Scheduling

Next, we would like to consider the min-WCT problem under online settings. In this scenario, the requester does not need to make the final scheduling decision at first. Instead, he can adjust his decision with the progress of the crowdsourcing campaign. Since the requester gets a better knowledge of the crowd whenever he meets a crowd worker, he can make the scheduling decision regarding this crowd worker at the time of their first contact to improve the final result.

We build the *Cost Minimized Online Scheduing* (CosMOS) algorithm (Algorithm 2) based on our previous design. In the CosMOS algorithm, we make the scheduling decision regarding each crowd worker at the time of his first contact with the requester. When u_0 meets u_j, we determine a partial scheduling decision of $S_j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ and make it final. We denote the process of determining S_j as *decision step* DS_j in the CosMOS algorithm.

Algorithm 2: The CosMOS Algorithm

```
Input: $S = \{s_1, s_2, \ldots, s_n : \tau_1 = \tau_2 = \cdots = \tau_n\}$,
       $W = \{w_1, w_2, \ldots, w_n : w_1 \geq w_2 \geq \cdots \geq w_n\}$,
       $U = \{u_1, u_2, \ldots, u_m : \lambda_1, \lambda_2, \ldots, \lambda_m\}$
Output: $\Lambda_{\text{CosMOS}} = \{S_1, S_2, \ldots, S_m\}$
1  when $u_0$ meets $u_j$ do
2      $S_j \leftarrow \phi$;
3      $EW_j \leftarrow \frac{1}{\lambda_j}$;
4      foreach $k$ such that $u_k \in U \backslash \{u_j\}$ do
5          $S_k \leftarrow \phi$;
6          $EW_k \leftarrow \frac{2}{\lambda_k}$;
7      foreach $i$ such that $s_i \in S$ (in an increasing order of $i$) do
8          $k_{min} \leftarrow \arg\min\{EW_k \mid u_k \in U\}$;
9          $S_{k_{min}} \leftarrow S_{k_{min}} \cup \{s_i\}$;
10         $EW_{k_{min}} \leftarrow EW_{k_{min}} + \tau_i$;
11     $S \leftarrow S \backslash S_j$;
12     $U \leftarrow U \backslash \{u_j\}$;
13 return $\Lambda_{CosMOS} = \{S_1, S_2, \ldots, S_m\}$;
```

Performance Analysis. Without loss of generality, we assume that the requester u_0 meets the crowd workers in the order of $u_1, u_2, \ldots, u_m$. Assume that after decision step DS_j, the total weighted completion time is WCT_j. Obviously, $WCT_{\text{CosMOS}} = WCT_m$. Furthermore, let $WCT_0 = WCT_{\text{LWF}}$.

Theorem 2. *$WCT_{LWF} = WCT_0 \geq WCT_1 \geq \cdots \geq WCT_m = WCT_{CosMOS}$.*

Proof. In any decision step $DS_j (1 \leq j \leq m)$, we will not change the scheduling of any of the tasks in $\bigcup_{1 \leq i \leq j-1} S_i$, and the total weighted completion time of these tasks remains unchanged before and after the decision step. As for the remaining tasks, we can prove that the modified LWF algorithm achieves the optimal solution under the condition that u_0 has met u_j by duplicating the proof of the optimality of our LWF algorithm. Therefore, we have $WCT_{j-1} \geq WCT_j$, and this finishes the proof of Theorem 2.

Now, we give the competitive ratio of the CosMOS algorithm as follows.

Theorem 3. *Assume someone can foresee the mobilities of all the crowd workers, so that he knows exactly at what time each meeting between the requester and the crowd workers will happen. Based on this knowledge, he can give an optimal online task scheduling decision, denoted as $\Lambda_{OPT} = \{S_1^*, S_2^*, \ldots, S_m^*\}$. Then, we have*

$$\frac{WCT_{CosMOS}}{WCT_{OPT}} \leq 1 + \frac{w_{max} \sum_{j=1}^{m} \frac{2}{\lambda_j}}{w_{min}\tau_{min}}.$$

Proof. First, we can give WCT_{OPT} as follows.

$$WCT_{\text{OPT}} = \sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k}(t_j + t'_j + \mathcal{T}_{j_k} + \tau_{j_k}).$$

If we adopt the scheduling decision of Λ_{OPT} in the offline version of min-WCT and denote it as Λ', then we have

$$\begin{aligned} WCT' &= \sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k}(\frac{2}{\lambda_j} + \mathcal{T}_{j_k} + \tau_{j_k}) \\ &= WCT_{\text{OPT}} + \sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k}(\frac{2}{\lambda_j} - t_j - t'_j) \\ &\leq WCT_{\text{OPT}} + \sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k} \cdot \frac{2}{\lambda_j}. \end{aligned}$$

Since Λ_{LWF} is the optimal scheduling for the offline version of min-WCT, then we have $WCT_{\text{LWF}} \leq WCT'$. Combined with Theorem 2, we can get

$$WCT_{\text{CosMOS}} \leq WCT_{\text{LWF}} \leq WCT' \leq WCT_{\text{OPT}} + \sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k} \cdot \frac{2}{\lambda_j}.$$

The lower bound of WCT_{OPT} should be $n \cdot w_{min} \cdot \tau_{min}$, and this gives us

$$\begin{aligned} \frac{WCT_{\text{CosMOS}}}{WCT_{\text{OPT}}} &\leq 1 + \frac{\sum_{j=1}^{m} \sum_{s_{j_k} \in S_j^*} w_{j_k} \cdot \frac{2}{\lambda_j}}{WCT_{\text{OPT}}} \\ &\leq 1 + \frac{w_{max} \sum_{j=1}^{m} \frac{2}{\lambda_j}}{w_{min}\tau_{min}}. \end{aligned}$$

Thus, Theorem 3 holds.

5 Time Minimized Scheduing

In this section, we first formulate the time-related min-MCT problem for task scheduling in mobile crowdsourcing systems. Then, we propose an approximation algorithm LRSTF to solve the problem under offline settings, followed by the performance analysis regarding its approximation ratio. Finally, we give an online algorithm TiMOS based on LRSTF to solve the problem under online settings together with the analysis of its competitive ratio.

5.1 Problem Formulation

In some cases, the crowdsourcing tasks of a requester may be part of a larger project, and all of them should be completed as soon as possible. Hence, we introduce the min-MCT problem to minimize the maximum completion time of all these tasks in order to accelerate the progress of the following stages in the project. In this problem, crowdsourcing tasks now have different values of the required service time and there is no weight related to any of these tasks. Our goal is equivalent to minimizing the completion time of the latest completed task. The problem is formally defined as follows.

Definition 3 (Minimizing the Maximum Completion Time (min-MCT)). *Given tasks $S = \{s_1, s_2, \ldots, s_n\}$ with required service time τ_i related to task s_i, min-MCT aims to find an scheduling decision $\Lambda = \{S_1, S_2, \ldots, S_m\}$ among crowd workers $U = \{u_1, u_2, \ldots, u_m\}$ such that the maximum completion time of all the tasks $\max\limits_{1 \le i \le n} C_i$ is minimized.*

Denote $\max\limits_{1 \le i \le n} C_i$ as MCT in this section for simplicity.

The NP-Hardness. Consider a special case of the min-MCT problem, in which there may exist one requester and two crowd workers in the mobile crowdsourcing system. The expected inter-meeting time between the requester and any one of the crowd workers is assumed to be zero. In this case, the min-MCT problem is equivalent to the *PARTITION* problem, which is traditionally considered to be NP-hard according to [8]. Therefore, the general case of the min-MCT problem is also NP-hard.

5.2 Offline Task Scheduling

The observation we have on the min-MCT problem is that we should leave the tasks with relatively short required service time to the end of our scheduling process so that the expected workloads of all the workers can be balanced and the maximum completion time of all the tasks can be reduced. Unfortunately, this idea only leads to an approximation rather than an optimal solution, which we denote as the *Longest Required Service Time First* (LRSTF) algorithm (Algorithm 3). We claim that its approximation ratio is $(\frac{3}{2} - \frac{1}{2m})$ and omit its detailed proof due to limited space.

```
Algorithm 3: The LRSTF Algorithm
   Input: $S = \{s_1, s_2, \ldots, s_n : \tau_1 \geq \tau_2 \geq \cdots \geq \tau_n\}$,
          $U = \{u_1, u_2, \ldots, u_m : \lambda_1, \lambda_2, \ldots, \lambda_m\}$
   Output: $\Lambda_{\text{LRSTF}} = \{S_1, S_2, \ldots, S_m\}$
 1 for $j \leftarrow 1$ to $m$ do
 2     $S_j \leftarrow \phi$;
 3     $EW_j \leftarrow \frac{2}{\lambda_j}$;
 4 for $i \leftarrow 1$ to $n$ do
 5     $j_{min} \leftarrow \arg\min\{EW_k \mid u_k \in U\}$;
 6     $S_{j_{min}} \leftarrow S_{j_{min}} \cup \{s_i\}$;
 7     $EW_{j_{min}} \leftarrow EW_{j_{min}} + \tau_i$;
 8 return $\Lambda_{LRSTF} = \{S_1, S_2, \ldots, S_m\}$;
```

5.3 Online Task Scheduling

Similar to our design of the CosMOS algorithm, we build the *Time Minimized Online Scheduing* (TiMOS) algorithm (Algorithm 4) based on the LRSTF algorithm. In the TiMOS algorithm, we make the scheduling decision regarding each crowd worker at the time of his first contact with the requester. When u_0 meets u_j, we determine a partial scheduling decision of $S_j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ and make it final. We also denote the process of determining S_j as *decision step* DS_j in the TiMOS algorithm.

```
Algorithm 4: The TiMOS Algorithm
   Input: $S = \{s_1, s_2, \ldots, s_n : \tau_1 \geq \tau_2 \geq \cdots \geq \tau_n\}$,
          $U = \{u_1, u_2, \ldots, u_m : \lambda_1, \lambda_2, \ldots, \lambda_m\}$
   Output: $\Lambda_{\text{TiMOS}} = \{S_1, S_2, \ldots, S_m\}$
 1 when $u_0$ meets $u_j$ do
 2     $S_j \leftarrow \phi$;
 3     $EW_j \leftarrow \frac{1}{\lambda_j}$;
 4     foreach $k$ such that $u_k \in U \backslash \{u_j\}$ do
 5         $S_k \leftarrow \phi$;
 6         $EW_k \leftarrow \frac{2}{\lambda_k}$;
 7     foreach $i$ such that $s_i \in S$ (in an increasing order of $i$) do
 8         $k_{min} \leftarrow \arg\min\{EW_k \mid u_k \in U\}$;
 9         $S_{k_{min}} \leftarrow S_{k_{min}} \cup \{s_i\}$;
10         $EW_{k_{min}} \leftarrow EW_{k_{min}} + \tau_i$;
11     $S \leftarrow S \backslash S_j$;
12     $U \leftarrow U \backslash \{u_j\}$;
13 return $\Lambda_{TiMOS} = \{S_1, S_2, \ldots, S_m\}$;
```

Performance Analysis. Without loss of generality, we assume that the requester u_0 meets the crowd workers in the order of $u_1, u_2, \ldots, u_m$. First, we need to calculate the difference between the solution given by LRSTF and the optimal offline solution in each decision step. Assume that $\Lambda_{\text{opt}}^{(j)}$ is the optimal offline solution in decision step DS_j, and $\Lambda_{\text{LRSTF}}^{(j)}$ is the solution given by the LRSTF algorithm. Obviously, Λ_{TiMOS} is $\Lambda_{\text{LRSTF}}^{(m)}$, and we define $\Lambda_{\text{LRSTF}}^{(0)}$ to be Λ_{LRSTF}. Then we have the following theorem.

Theorem 4. $MCT_{opt}^{(j)} \leq MCT_{LRSTF}^{(j)} \leq MCT_{opt}^{(j)} + \tau_{max}$.

Proof. First, since $\Lambda_{\text{opt}}^{(j)}$ is the optimal offline solution in decision step DS_j, we have $MCT_{\text{opt}}^{(j)} \leq MCT_{\text{LRSTF}}^{(j)}$. Combining the two inequalities below,

$$MCT_{\text{LRSTF}}^{(j)} \leq \tau_n(1 - \frac{1}{m}) + \frac{\sum_{i=1}^{n} \tau_i + \sum_{j=1}^{m} \frac{2}{\lambda_j}}{m}$$

$$MCT_{\text{opt}}^{(j)} \geq \frac{\sum_{i=1}^{n} \tau_i + \sum_{j=1}^{m} \frac{2}{\lambda_j}}{m}$$

we can get that

$$MCT_{\text{LRSTF}}^{(j)} \leq MCT_{\text{opt}}^{(j)} + \tau_n(1 - \frac{1}{m}) \leq MCT_{\text{opt}}^{(j)} + \tau_{max}.$$

This finishes the proof of this theorem.

Similar to Theorem 2, we directly give the following theorem regarding $\Lambda_{\text{opt}}^{(j)}$.

Theorem 5. $MCT_{opt}^{(0)} \geq MCT_{opt}^{(1)} \geq \cdots \geq MCT_{opt}^{(m)}$.

Now, we can compute the competitive ratio of the TiMOS algorithm.

Theorem 6. *Assume someone can foresee the mobilities of all the crowd workers, so that he knows exactly at what time each meeting between the requester and the crowd workers will happen. Based on this knowledge, he can give an optimal online task scheduling decision, denoted as $\Lambda_{OPT} = \{S_1^*, S_2^*, \ldots, S_m^*\}$. Then, we have*

$$\frac{MCT_{TiMOS}}{MCT_{OPT}} \leq 2 + \frac{2}{\lambda_{min}\tau_{max}}.$$

Proof. First, we can give MCT_{OPT} as follows.

$$MCT_{\text{OPT}} = \max_{S_j^* \in \Lambda_{\text{OPT}}} \left\{ t_j + t'_j + \sum_{s_{j_k} \in S_j^*} \tau_{j_k} \right\}.$$

If we adopt the scheduling decision of Λ_{OPT} in the offline version of min-MCT and denote it as Λ', then we have

$$\begin{aligned} MCT' &= \max_{S_j^* \in \Lambda'} \left\{ \frac{2}{\lambda_j} + \sum_{s_{j_k} \in S_j^*} \tau_{j_k} \right\} \\ &\le MCT_{\mathrm{OPT}} + \max \left\{ \frac{2}{\lambda_j} - t_j - t'_j \right\} \\ &\le MCT_{\mathrm{OPT}} + \frac{2}{\lambda_{min}}. \end{aligned}$$

Since $\Lambda_{\mathrm{OPT}}^{(0)}$ is the optimal scheduling for the offline version of min-MCT, we have $MCT_{\mathrm{OPT}}^{(0)} \le MCT'$. Combined with Theorems 4 and 5, we can get

$$MCT_{\mathrm{TiMOS}} = MCT_{\mathrm{LRSTF}}^{(m)} \le MCT_{\mathrm{opt}}^{(m)} + \tau_{max} \le MCT_{\mathrm{opt}}^{(0)} + \tau_{max} \le MCT' + \tau_{max}.$$

Therefore, we have

$$MCT_{\mathrm{TiMOS}} \le MCT_{\mathrm{OPT}} + \frac{2}{\lambda_{min}} + \tau_{max},$$

and further considering the fact that $MCT_{\mathrm{OPT}} \ge \tau_{max}$, we have

$$\frac{MCT_{\mathrm{TiMOS}}}{MCT_{\mathrm{OPT}}} \le 1 + \frac{\frac{2}{\lambda_{min}} + \tau_{max}}{\tau_{max}} = 2 + \frac{2}{\lambda_{min}\tau_{max}}.$$

Thus, Theorem 6 holds.

6 Evaluation

In this section, we carry out extensive numerical experiments to evaluate the performances of the proposed algorithms.

6.1 Algorithms in Comparison

The first scheduling algorithm we would like to compare with is the Water Filling (WF) algorithm described in [12] in the context of mobile computing. More specifically, the WF algorithm assigns tasks to the earliest idle worker in their initial order, which is also called the natural order or the list order.

For the min-WCT problem, we also implement the *Smallest Weight First* (SWF) algorithm to assign tasks in a reverse order of the LWF algorithm. For the min-MCT problem, we compare our design with another algorithm called the *Shortest Required Service Time First* (SRSTF) algorithm to assign tasks in a reverse order of the LRSTF algorithm. The SRSTF algorithm is the optimal algorithm when we consider the problem of reducing the average completion time of all the tasks, and its optimality is proved in [16].

6.2 Simulation Results

Figure 1 shows the simulation results when we change the number of crowd workers, the average inter-meeting time between the requester and crowd workers, the number of tasks and the average workload of tasks respectively. Given the same conditions, LWF can always deliver a better result than WF or SWF. Meanwhile, SWF performs even worse than WF.

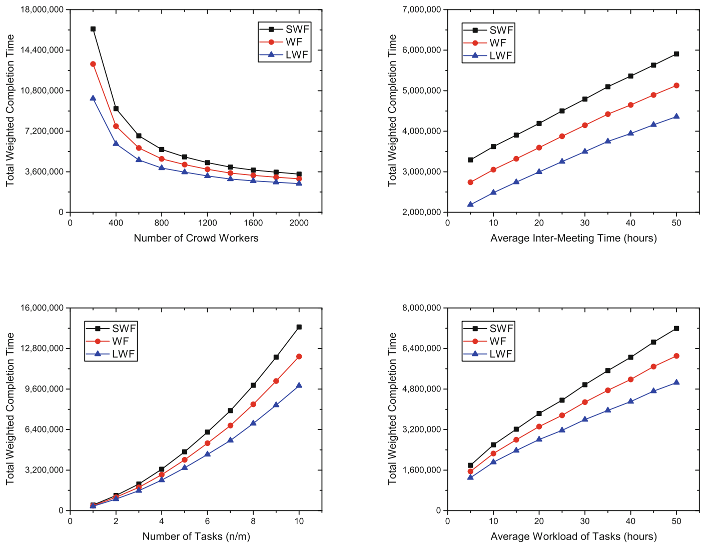


Fig. 1. Performance comparisons for min-WCT on synthetic datasets

Figure 2 shows the simulation result when we change the number of crowd workers, the average inter-meeting time between the requester and crowd workers, the number of tasks and the average workload of tasks respectively. Given the same conditions, LRSTF can always deliver a better result than WF or SRSTF. Meanwhile, SRSTF performs almost as badly as WF.

7 Conclusion

In this paper, we discuss two problems regarding task scheduling for crowdsourcing in mobile social networks. The first one is the min-WCT problem, which aims to minimize the total weighted completion time of some crowdsourcing tasks. We propose an optimal algorithm named LWF to solve this problem under offline settings and give the proof of its optimality. Based on this optimal offline algorithm, we further design an online algorithm named CosMOS to deal with the min-WCT problem under online settings. By computing the competitive ratio,

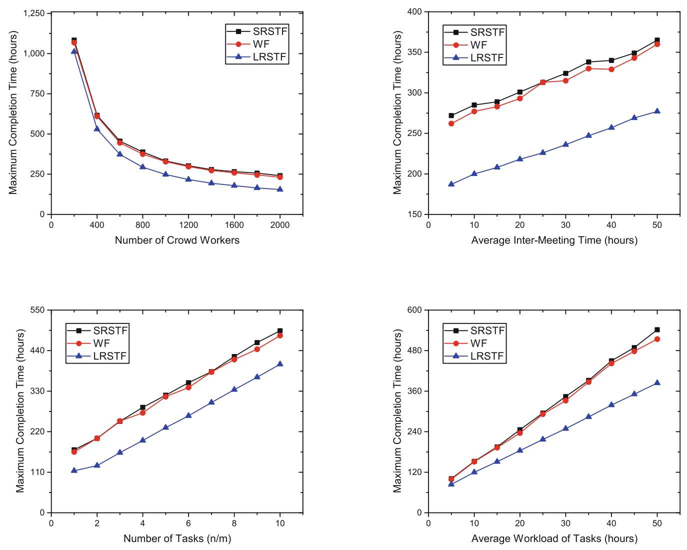


Fig. 2. Performance comparisons for min-MCT on synthetic datasets

we find that CosMOS can achieve near-optimal performance under certain circumstances. The other problem in discussion is the min-MCT problem, which focuses on minimizing the maximum completion time among the tasks belonging to the same project. Since the offline version of this problem is proved to be NP-hard, we propose an approximation algorithm named LRSTF to solve it and the approximation ratio is analyzed. Similarly, we design another online algorithm named TiMOS to deal with the online version of the min-MCT problem and its difference with the optimal solution is bounded by a fixed value. At last, we conduct extensive simulations on synthetic datasets to demonstrate the performance of our algorithms. Both the theoretical analysis and the simulation results validate the effectiveness and efficiency of our designs.

Acknowledgements. This work is supported by the National Key R&D Program of China (2018YFB1004703), the National Natural Science Foundation of China (61872238, 61672353), the Shanghai Science and Technology Fund (17510740200), the CCF-Tencent Open Research Fund (RAGR20170114), and Huawei Innovation Research Program (HO2018085286).

References

1. Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. **187**(3), 985–1032 (2008)
2. Bridi, T., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: A constraint programming scheduler for heterogeneous high-performance computing machines. IEEE Trans. Parallel Distrib. Syst. (TPDS) **27**(10), 2781–2794 (2016)

3. Ding, J., Song, S., Zhang, R., Chiong, R., Wu, C.: Parallel machine scheduling under time-of-use electricity prices: new models and optimization approaches. IEEE Trans. Autom. Sci. Eng. **13**(2), 1138–1154 (2016)
4. Dutta, P., et al.: Common sense: participatory urban sensing using a network of handheld air quality monitors. In: ACM International Conference on Embedded Networked Sensor Systems (SenSys), pp. 349–350 (2009)
5. Fan, Y., Sun, H., Liu, X.: Poster: TRIM: a truthful incentive mechanism for dynamic and heterogeneous tasks in mobile crowdsensing. In: ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pp. 272–274 (2015)
6. Farkas, K., Nagy, A.Z., Tomas, T., Szabó, R.: Participatory sensing based real-time public transport information service. In: IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom), pp. 141–144 (2014)
7. Gao, W., Li, Q., Zhao, B., Cao, G.: Multicasting in delay tolerant networks: a social network perspective. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 299–308 (2009)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
9. Guo, B., Liu, Y., Wu, W., Yu, Z., Han, Q.: Activecrowd: a framework for optimized multitask allocation in mobile crowdsensing systems. IEEE Trans. Hum.-Mach. Syst. **47**(3), 392–403 (2017)
10. Kim, P., Kim, S.: A detection of overlapping community in mobile social network. In: ACM Symposium on Applied Computing (SAC), pp. 175–179 (2014)
11. Rana, R.K., Chou, C.T., Kanhere, S.S., Bulusu, N., Hu, W.: Ear-phone: an end-to-end participatory urban noise mapping system. In: International Conference on Information Processing in Sensor Networks (IPSN), pp. 105–116 (2010)
12. Shi, C., Lakafosis, V., Ammar, M.H., Zegura, E.W.: Serendipity: enabling remote computing among intermittently connected mobile devices. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 145–154 (2012)
13. Singh, G., Bansal, D., Sofat, S., Aggarwal, N.: Smart patrolling: an efficient road surface monitoring using smartphone sensors and crowdsourcing. Pervasive Mob. Comput. **40**, 71–88 (2017)
14. Wang, Y., Jia, X., Jin, Q., Ma, J.: Mobile crowdsourcing: framework, challenges, and solutions. Concurr. Comput.: Pract. Exp. **29**(3), 1–17 (2017)
15. Wu, S., Wang, X., Wang, S., Zhang, Z., Tung, A.K.H.: K-anonymity for crowdsourcing database. IEEE Trans. Knowl. Data Eng. (TKDE) **26**(9), 2207–2221 (2014)
16. Xiao, M., Wu, J., Huang, L., Wang, Y., Liu, C.: Multi-task assignment for crowdsensing in mobile social networks. In: IEEE Conference on Computer Communications (INFOCOM), pp. 2227–2235 (2015)
17. Yan, R., Song, Y., Li, C., Zhang, M., Hu, X.: Opportunities or risks to reduce labor in crowdsourcing translation? Characterizing cost versus quality via a pagerank-hits hybrid model. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1025–1032 (2015)
18. Yuen, M., King, I., Leung, K.: TaskRec: a task recommendation framework in crowdsourcing systems. Neural Process. Lett. **41**(2), 223–238 (2015)
19. Zhang, J., Chen, Y., Hong, S., Li, H.: REBUILD: graph embedding based method for user social role identity on mobile communication network. In: Tan, Y., Takagi, H., Shi, Y. (eds.) DMBD 2017. LNCS, vol. 10387, pp. 326–333. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61845-6_33

Cognitive System to Achieve Human-Level Accuracy in Automated Assignment of Helpdesk Email Tickets

Atri Mandal[1(✉)], Nikhil Malhotra[2], Shivali Agarwal[1], Anupama Ray[1], and Giriprasad Sridhara[1]

[1] IBM Research AI, Bengaluru, India
[2] IBM Global Technology Services, Bengaluru, India
{atri.mandal,nikhimal,shivaaga,anupamar,girisrid}@in.ibm.com

Abstract. Ticket assignment/dispatch is a crucial part of service delivery business with lot of scope for automation and optimization. In this paper, we present an end-to-end automated helpdesk email ticket assignment system, which is also offered as a service. The objective of the system is to determine the nature of the problem mentioned in an incoming email ticket and then automatically dispatch it to an appropriate resolver group (or team) for resolution.

The proposed system uses an ensemble classifier augmented with a configurable rule engine. While design of classifier that is accurate is one of the main challenges, we also need to address the need of designing a system that is robust and adaptive to changing business needs. We discuss some of the main design challenges associated with email ticket assignment automation and how we solve them. The design decisions for our system are driven by high accuracy, coverage, business continuity, scalability and optimal usage of computational resources.

Our system has been deployed in production of three major service providers and currently assigning over 40,000 emails per month, on an average, with an accuracy close to 90% and covering at least 90% of email tickets. This translates to achieving human-level accuracy and results in a net saving of about 23000 man-hours of effort per annum.

Keywords: Cognitive email assignment · Helpdesk automation Ticket resolver group · Smart dispatch · Ensemble classifiers

1 Introduction

The landscape of modern IT service delivery is changing with increased focus on automation and optimization. Most IT vendors today, have service platforms aimed towards end-to-end automation for carrying out mundane, repetitive labor-intensive tasks and even for tasks requiring human cognizance. One such task is ticket assignment/dispatch where the service requests submitted by the end-users to the vendor in the form of tickets are reviewed by a centralized dispatch team and assigned to the appropriate service team i.e. resolver group.

The dispatch of a ticket to the correct group of practitioners is a critical step in the speedy resolution of a ticket. Incorrect dispatch decisions can significantly increase the total turnaround time for ticket resolution, as observed in a study of an actual production system [2]. Several factors make the dispatcher's job challenging such as requirement of knowledge of the IT portfolio being managed, roles and responsibilities of the individual groups, ability to quickly parse the ticket text describing the problem and map it to the right group, which is often not straightforward given the heterogeneous and informal nature of the problem description. A number of different approaches have been proposed for automating ticket dispatch [2,7,10,11]. Although automated email assignment may look like a simple text classification problem at first glance it becomes quite complex and challenging when considered at industry scale.

In this paper we present a deployed end-to-end automatic email dispatch system having the following key features:

1. An ensemble based classification engine that uses supervised data in the form of unstructured email text and resolver groups as labels. The choice of ensemble is based on the results of comprehensive study performed with various machine learning and deep learning models as presented in Sect. 4.2.
2. A rule engine with a customer-independent framework for rule specification to ensure business continuity and handle domain specific content missed by the ensemble classifier.
3. Experimental results with real customer data from three different datasets - the largest of them having more than 700,000 emails and 428 resolver groups. We were able to achieve human level accuracy with more than 90% coverage on all the datasets with the proposed system using minimal computational resources.

The remainder of the paper is organized as follows. Section 2 describes the related work. Section 3 gives a system overview, and Sect. 4 discusses the different components of the system in detail. We present our experimental results in Sect. 5 and conclusion in Sect. 6.

2 Related Work

A semi-automated approach based on confidence scores of Support Vector Machines and discriminative keywords had been proposed in [2] for ticket dispatch. We have surpassed their work to (i) reach human level accuracy using advanced ensemble techniques for automated dispatch, (ii) scale it to hundreds of resolver groups and (iii) incorporate retraining strategies to adapt to changing data. Several other researchers have studied different aspects of the problem of routing tickets to resolver groups [7,10,11]. The work in [11] approaches the problem by mining resolution sequence data and does not access ticket description at all. Its objective is to come up with ticket transfer recommendations given the initial assignment information. The work in [10] mines historical ticket data and develops a probabilistic model of an enterprise social network that represents the functional relationships among various expert groups in ticket routing. Based on this network, the system then provides routing recommendations to new tickets. The work in [7] approaches the problem from a queue perspective, related to the issue of service times and becomes particularly relevant when the ticket that has been dispatched to a group needs to be assigned to an agent. Some works have focused on applying text classification techniques to handle tickets [3,12], by identifying the ticket category helping human dispatchers for faster ticket assignment. The work in [4] attempts to classify the incoming change requests into one of the fine-grained activities in a catalog. Some other works [1,9] talk about a holistic approach of ticket category classification, cause analysis and resolution recommendation. However, they do not automate the process of assignment.

3 System Overview

Figure 1 shows the system architecture along with the data flow diagram. Historical email ticket data is downloaded from the ticketing tool (e.g. Remedy or ServiceNow) using custom-built adapters. The downloaded emails are passed through two stages of pre-processing for data enrichment. The resolver group level pre-processing module uses techniques like resolver group merging, long tail cutoff etc. to reduce the noise in the email data. The training data is further enriched using text pre-processing methods. The enriched email data is then trained using an ensemble of machine learning classifiers and the trained models are stored in a database.

When a user sends an email to the helpdesk account a ticket is automatically generated and stored in the backend ticketing tool. The newly generated tickets are downloaded by the adapter and classified using the runtime that consists of ensemble classifier and the rule engine. The classification system returns a resolver group along with a confidence score. If the confidence score is above a configured threshold the ticket is routed to the returned resolver group. Otherwise the ticket is assigned back to manual queue for inspection by human agent. The combination of ensemble classifier and rule engine ensures that a high percentage of tickets (more than 90%) are classified automatically by our system with a low error rate.

4 Assignment Engine Components

4.1 Preparation of Training Data

Most large companies nowadays use ticketing tools like Remedy or ServiceNow to maintain tickets obtained from various channels (voice, email, web etc.) by the helpdesk. The ticketing tool organizes the email data into structured fields containing relevant information about the ticket e.g. incident number, incident

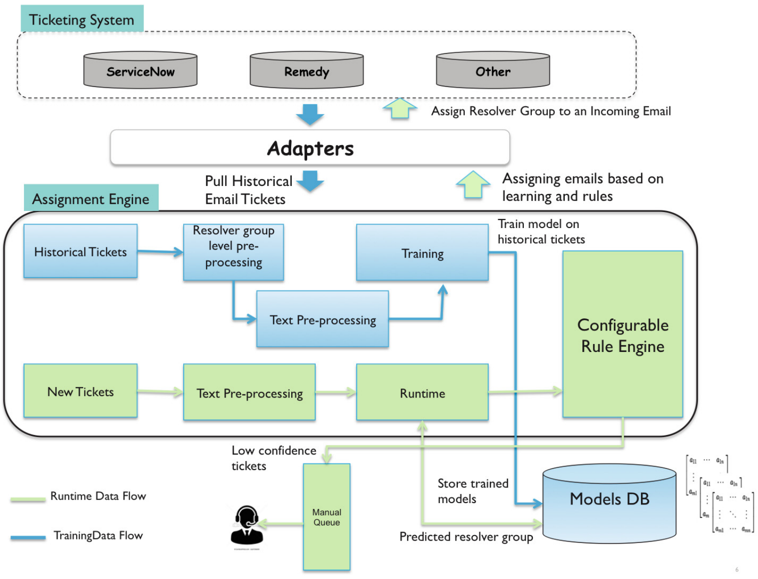


Fig. 1. Architecture of the proposed system

type, date of creation, description, assigned group etc. We use custom adapters to connect to the ticketing tool and extract fields relevant for training. Currently the adapter extracts only the text portion of the email (viz. email subject and body) along with the resolver group for training. The data collected by the adapter is then converted into a format readable by the classifier. The steps involved in training data preparation are described below.

Resolver Group Level Pre-processing. This type of pre-processing is a one-time effort required during customer on-boarding phase. The purpose of this pre-processing is to reduce noise in the training data. We reduce noise and enrich training data for the resolver groups using the following techniques:

Merging Related Resolver Groups. Some of the resolver group labels in the training data can be merged. Merging increases the size of the training data and at the same time reduces the number of unique labels thus improving training accuracy. We found that there are at least two types of resolver groups that can be merged for assignment purpose viz. **(a) Resolver groups with varying escalation levels** and **(b) Region (or zone) specific resolver groups**.

Long Tail Cutoff. We observed that in most of the datasets there are a large number of resolver groups with very few samples. If we plot a histogram of frequencies these groups will constitute more than 80% of the resolver groups but less than 5% of training data. Our studies indicate that, if the long tail is included in training, the overall accuracy of classification goes down along with a significant increase in training time and model size. By restricting the number of resolver groups in training we reduce noise significantly and also avoid class imbalance. Additionally, the resolver groups, which fall in the long tail, can often be better predicted using the rule engine and using some augmentation techniques. As such our strategy was to divide the downloaded historical data into 2 parts viz. IH = IT + IL where IH is the complete data downloaded for training, IT is the data used for training classifiers and IL is the long tail. Resolver groups belonging to IT will be classified using trained models while those belonging to IL will be handled exclusively by the rule engine. In our system we use the above strategy to retain at least 98% of data while cutting down the resolver group count to less than 20%.

4.2 Classification Models

This section presents our study on the performance of various machine-learning classifiers in classification of email data, in terms of accuracy and training time, although the training is offline. For training the classification models, we concatenate the subject and the body of the email(description) with a space in between and use the resulting string as our training data. The resolver group acts as the label for our training data. Tables 1 and 2 show the impact of various traditional machine learning models [6] and deep neural network models that were used. In order to improve accuracy and coverage of the overall service, we use an ensemble [5]. Each pair of models were combined, and the final ensemble classifier was chosen based on the accuracy and coverage. As explained in Sect. 4.1, rule engine is important to handle the long tail in class distribution and the final chosen ensemble classifier in combination with the rule engine forms the classification module of the service.

Table 1. Comparison of various machine learning algorithms w.r.t. accuracy and training time

		LinearSVM	KNN	LR	m-NB	RF	Adaboost	Gradient boosting
Dataset A	Accuracy (%)	87.3	80.12	79.48	72.68	81.41	31.5	75.6
	Train-time (s)	7.8	260.5	43	17.3	363.75	4561	8612
Dataset B	Accuracy (%)	83.42	72.58	79.95	64.19	74.91	32.98	65.1
	Train-time (s)	76.12	2218.65	404.05	22.18	7190.16	332.97	95320.1
Dataset C	Accuracy (%)	86.339	67.57	84.29	63.97	76.99	30.43	61.47
	Train-time (s)	1001.06	1921.7	2992	167.5	20799.6	1288.63	126960

Training the Classifiers. We convert the training data samples into word vector representation before applying machine-learning algorithms. We observed that using tf-idf representation increased the accuracy of traditional machine learning algorithms for all datasets by at least 3–4%. Another observation was that using bigrams also improved the accuracy for some datasets. Intuitively we can argue that this is so because some bigrams like ‘account creation’, ‘account deletion’, ‘password reset’ etc. are useful indicators in deciding the

Table 2. Comparison of various deep neural networks w.r.t. accuracy and training time

		MLP	CNN-WE	LSTM-WE	CNN-G	LSTM-G	CNNLSTM-G
Dataset A	Accuracy (%)	85.8	74	76.94	74.01	71.64	73.24
	Train-time (s)	184.12	183.75	5546.6	160.56	9833.7	1844.8
Dataset B	Accuracy (%)	80.87	77.75	79.35	76.23	80.37	77.7
	Train-time (s)	10858.15	8680.35	86651.57	1926	89280.94	23229.47
Dataset C	Accuracy (%)	83.3	78.8	78.14	79.1	83.51	81.33
	Train-time (s)	2779	4000.9	90149.9	9522.12	687483	116583.22

resolver group. The hyperparameters were chosen experimentally over 10-fold cross-validation on the datasets.

However, for learning deep neural networks, tf-idf representation being extremely sparse is not useful and hence we used word embeddings. There are primarily two methods of learning the word embeddings: one in which word embeddings are learnt while training the neural network; and second using pre-trained word vectors. We experimented with both methods for classification (models learning word embeddings being referred to CNN-WE, LSTM-WE, and CNN-LSTM-WE in Table 2), and pretrained word-vector representations (100-d GloVe vectors) [8] referred to as CNN-G, LSTM-G and CNN-LSTM-G.

4.3 Rule Engine

The rule engine is one of the key components of our end-to-end system and is used to capture domain specific elements as well as to ensure business continuity. Also, the resolver groups belonging to IT can only be predicted using the rule engine as discussed in Sect. 4.1.

The rule engine is designed for ease of usage and configuration. Each rule in the rule engine can be expressed by the following equation:

$$(f1 = \phi 1) \wedge (f2 = \phi 2) \wedge (f3 = \phi 3) \wedge (CE = R) \Rightarrow CF \quad (1)$$

where CF is the resolver group predicted by the engine, ϕi is the value of the ticket parameter i and CE is the resolver group predicted by the ensemble classifier. Here ϕi can take a finite value (or regular expression) or the value X (implying DON'T CARE).

4.4 Email Ticket Dispatcher

The email ticket dispatcher actually assigns the ticket to a specific resolver group and updates the ticket. The dispatcher selects an ensemble of two classifiers to optimize accuracy and coverage as shown in Fig. 3. It combines the results of the ensemble classifier and rule engine using a dispatch algorithm to output the final prediction and confidence score. If the confidence score of the final result is below the configured threshold the ticket is assigned to the manual queue.

Table 3. Dataset details and results

	Dataset A	Dataset B	Dataset C
Number of email tickets (N)	11562	423343	712320
Number of resolver groups	70	403	428
Duration of the dataset	6 months	12 months	15 months
Ensemble Accuracy (X_{acc})	90.07%	86.17%	89.61%
Ensemble Coverage (X_{cov})	93.67%	92.88%	93.83%
Assignment Engine Accuracy (E_{acc})	92.73%	88.66%	92.13%
Assignment Engine Coverage (E_{cov})	97.84%	93.3%	95.5%

5 Experimental Results

This section enumerates the results of the experimental setup of the assignment engine. For evaluation we have used real datasets from three major helpdesk service providers. The datasets are from two different domains viz. telecom and supply-chain/logistics. To preserve client confidentiality we henceforth refer to these datasets as Dataset A, Dataset B and Dataset C respectively. The datasets were divided into training and test sets with a 90:10 split and we used 10-fold cross-validation on the datasets. All our experiments were run on a NVIDIA Tesla K80 GPU cluster with 4 CUDA-enabled nodes. We use open source machine-learning libraries viz. Python scikit-learn and Keras for our experiments. The dataset statistics as well as the final accuracy numbers achieved by our system are described in Table 3. Please note that the deployment setup is similar to our experimental setup but not identical; so numbers in production may vary slightly. The details about production setup and results are not included to preserve confidentiality.

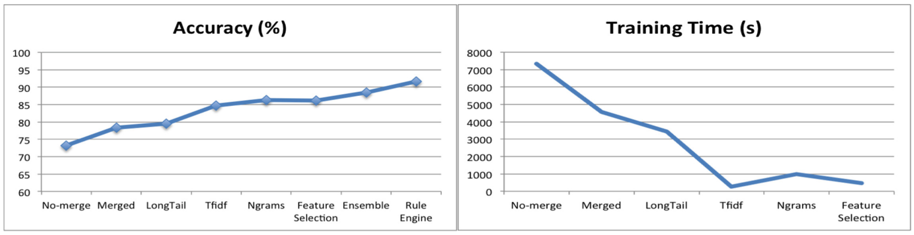


Fig. 2. Effect of different optimization techniques (a) Accuracy trend (b) Training time.

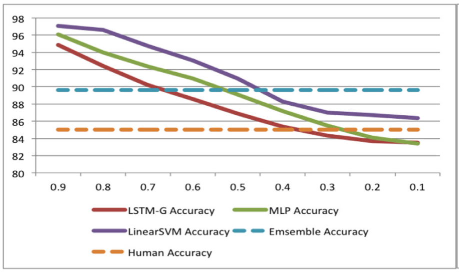
5.1 Accuracy and Training Time

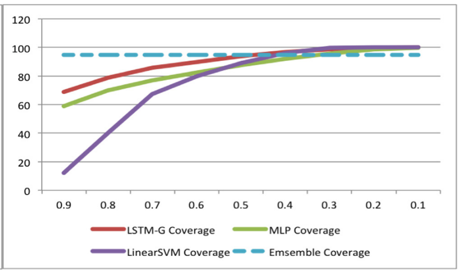
Figure 2 shows how the training techniques and preprocessing affect the accuracy of prediction and training time. It shows the gradual increase in accuracy and corresponding decrease in training time as we apply each technique(shown in X-axis) incrementally. The accuracy and training time charts are shown for only one of the datasets viz. dataset C which is our largest dataset - but the trend is fairly similar across other datasets as well.

5.2 Human Accuracy vs. Assignment Engine Accuracy

We next look at the optimal selection of algorithms that maximize accuracy and coverage. We assert that for business purposes the algorithms need to have at least human-level accuracy with high enough coverage.

To compute human accuracy we mined audit logs of the ticketing systems. Our experiments reveal that across all datasets the accuracy achieved by human agents is about 85%. Therefore we select the confidence threshold such that the expected accuracy of prediction is at least 85%. Figure 3 show the performance of the best three algorithms at different confidence levels (ranging from 0.1 to 0.9). For dataset C a combination of linear SVM (confidence ≥ 0.5) and MLP (confidence ≥ 0.6) gave a slightly higher accuracy (89.61%) than that of LSTM-G (confidence ≥ 0.5) and linear SVM ($X_{acc} = 88.38\%$), although the individual accuracy was marginally higher for LSTM-G compared to MLP. For this reason, as also for other practical considerations like memory and CPU constraints as well as training time our deployment in production uses an ensemble of linear SVM and MLP. For the other two datasets SVM and MLP were clear winners.





(a) (b)

Fig. 3. At different confidence thresholds (a) Assignment accuracy (b) Assignment coverage.

5.3 Observations

There are three main takeaways from our experimental results above. The most important observation is that our assignment engine (ensemble classifier augmented with rule engine) performs better, in terms of accuracy and coverage, than all traditional machine-learning and deep learning algorithms. Secondly we can see that simple machine learning algorithms like SVM and MLP are often better than more computationally expensive deep learning algorithms in the task of helpdesk email assignment automation. This result is somewhat surprising and unexpected, but is very significant from a product development standpoint as these algorithms are easy to implement, require minimal computational resources and provide better performance at runtime. Last but not the least we observe that LSTM accuracy increases with the size of the dataset and for the largest dataset (dataset C) it outperforms MLP. Thus our results indicate that an ensemble of SVM and MLP will be a good trade-off for most practical purposes but if we have a large enough dataset and infrastructure is not a concern then the best choices are SVM and LSTM-glove.

6 Conclusion and Future Work

We have proposed email ticket assignment engine that uses an ensemble of machine learning techniques, combined with a configurable rule engine, to perform automated dispatch. Our system achieves human-level accuracy and has already been deployed for three customers in production. However, there are still some areas in the system like rule engine which need human intervention and can be automated. In future, we want to solve the problem of automatically extracting rules based on data from misclassified emails. We would also like to handle concept drift in utterances for better retraining. Last but not the least, we need to enhance our assignment algorithm to handle email attachments.

References

1. Agarwal, S., Aggarwal, V., Akula, A.R., Dasgupta, G.B., Sridhara, G.: Automatic problem extraction and analysis from unstructured text in it tickets. IBM J. Res. Dev. **61**, 4–41 (2017)
2. Agarwal, S., Sindhgatta, R., Sengupta, B.: SmartDispatch: enabling efficient ticket dispatch in an it service environment. In: 18th ACM SIGKDD 2012 (2012)
3. Dasgupta, G.B., Nayak, T.K., Akula, A.R., Agarwal, S., Nadgowda, S.J.: Towards auto-remediation in services delivery: context-based classification of noisy and unstructured tickets. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 478–485. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_39
4. Kadar, C., Wiesmann, D., Iria, J., Husemann, D., Lucic, M.: Automatic classification of change requests for improved it service quality. In: 2011 SRII (2011)
5. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms (2004)
6. Mitchell, T.M.: Machine Learning, 1st edn. McGraw-Hill Inc., New York (1997)
7. Parvin, H., Bose, A., Van Oyen, M.P.: Priority-based routing with strict deadlines and server flexibility under uncertainty. In: WSC 2009 (2009)
8. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP) (2014)
9. Potharaju, R., Jain, N., Nita-Rotaru, C.: Juggling the Jigsaw: towards automated problem inference from network trouble tickets. In: USENIX (2013)

10. Shao, Q., Chen, Y., Tao, S., Yan, X., Anerousis, N.: EasyTicket: a ticket routing recommendation engine for enterprise problem resolution. In: VLDB 2008 (2008)
11. Shao, Q., Chen, Y., Tao, S., Yan, X., Anerousis, N.: Efficient ticket routing by resolution sequence mining. In: 14th ACM SIGKDD (2008)
12. Zeng, C., Zhou, W., Li, T., Shwartz, L., Grabarnik, G.Y.: Knowledge guided hierarchical multi-label classification over ticket data. IEEE Trans. Netw. Serv. Manag. **14**(2), 246–260 (2017)

Crowdsourcing Energy as a Service

Abdallah Lakhdari(✉), Athman Bouguettaya, and Azadeh Ghari Neiat

The University of Sydney, Sydney, NSW 2000, Australia
alak5184@uni.sydney.edu.au,
{athman.bouguettaya,azadeh.gharineiat}@sydney.edu.au

Abstract. We propose a new framework for crowdsourcing energy services from Internet-of-Things devices. We introduce a new crowdsourced energy as a service and energy-related quality model considering spatiotemporal aspects. We describe a new temporal composition algorithm to compose energy services to satisfy a user's energy requirement. The temporal composition algorithm is a variation of fractional knapsack algorithm. We conduct preliminary experiments to demonstrate the performance and effectiveness of our approach.

Keywords: Crowdsourcing · IoT services · Spatiotemporal service
Temporal service composition · Energy as a service · Wearable device

1 Introduction

Internet of Things (IoT) has the ability to connect any physical object to the Internet. It is expected that more than 50 billion devices will be connected to the Internet by 2020 [10]. The existing infrastructure to deploy IoT applications relies mainly on *cloud computing*. The Cloud offers external computing capabilities to overcome resource constraints of some IoT devices in terms of storage capacity and processing power. Recently, *edge computing* also provides a miniaturized external computing capability for IoT devices. Edge computing consists of micro datacenters implemented one hop away from the end user. There is an increasing trend to moving computation resources even much closer to the user which makes the computation and storage resources available locally [5].

A broad range of IoT devices comprise significant resources like processing, storage, communication, and energy. These capabilities leverage IoT devices to be possible computing resources. *Crowdsourcing IoT resources* provides unprecedented opportunities to create a self-sustained IoT ecosystem. Example of crowdsourcing IoT resources are WiFi hotspot sharing [9]. We propose a new approach for crowdsourcing energy in IoT environment. We consider a particular set of IoT devices called *wearables*. This set refers to any smart thing which can be worn or hand-held like smart shirts, smart glasses, smart watches, smartphones etc [12]. Energy could be easily available from harvested power of wearables or just spare energy from smartphones. Nowadays, wireless power transfer technologies are broadly used in sensor networks and IoT [8]. Crowdsourcing energy relies

© Springer Nature Switzerland AG 2018
C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 342–351, 2018.
https://doi.org/10.1007/978-3-030-03596-9_24

mainly on wireless power transfer for the delivery of energy from the providing wearables to the consuming IoT devices.

We harness the service paradigm as a powerful mechanism to abstract the functional and non-functional properties (QoS) of IoT based energy resources [1, 2]. Therefore, an IoT based energy resource is provisioned as a service, called Crowdsourced Energy as a Service (EaaS). Connectivity and mobility are two key challenges to unlock the full potential of crowdsourcing energy from wearable IoT devices [4]. For an energy request, nearby IoT devices need to be discovered. In addition, delivering energy services from an IoT device to another relies on ensuring the connectivity between IoT devices. A short range distance is required for a successful wireless delivery of energy. As a result, space and time attributes are of paramount importance to define and select energy services. The main contributions of this paper are: (i) a service model for crowdsourcing energy from IoT devices, (ii) A Quality of Service QoS model to distinguish between energy services, (iii) a temporal composition algorithm which is a variation of a fractional knapsack and temporal knapsack algorithms offering required energy and (iv) experiments are conducted based on a real dataset to illustrate the performance and effectiveness of the temporal composition approach.

The remainder of the paper is organized as follow: Sect. 2 presents the crowdsourced energy service and QoS model. Section 3 details the proposed temporal composition algorithm. Section 4 discussed the experimental results. Section 5 concludes the paper and highlights some future work.

Motivating Scenario
We consider a scenario of depleting battery of a smartphone user in a public place like a coffeeshop where it is challenging to find a vacant power point. We assume that the user needs to extend the lifetime of their smartphone's battery to run some critical applications e.g., make a call or look for a destination in the Google map. Users may have different requirements in terms of energy capacity and time availability. In our scenario, energy is offered by nearby wearable IoT devices provided by the crowd. Some wearable IoT devices like smart shirts and smart shoes can harvest and store energy for a long time. Wearables consume a tiny amount of their harvested energy. They can share their energy wirelessly as a service with nearby IoT devices [8]. Getting energy from one single service may not fulfill a user's requirement. In such case, multiple energy services need to be combined to satisfy the user's requirements. We propose to reformulate the problem of finding and crowdsourcing energy in a coffeeshop as a spatiotemporal service composition.

2 System Model and Problem Formulation

In this section, we first define the key concepts of a crowdsourced energy service. We then give a formal definition for the problem of crowdsourced energy service composition.

Definition 1: Crowdsourced Energy Service. A crowdsourced energy service ES is a tuple of $< Eid, Eownerid, F, Q >$ where:

- *Eid* is a unique service ID,
- *Eownerid* is a unique owner ID,
- F is the function of sharing energy by an IoT device owner *Eownerid* via an IoT device d
- Q is a tuple $< q_1, q_2, ..., q_n >$ where each q_i denotes a QoS property of ES e.g., energy capacity.

The energy service is available just when the energy capacity is higher than a predefined threshold by the provider Thr. This capacity will definitely change overtime. It decreases because of the use of the IoT device while delivering the energy to the service consumer. On the other hand, some IoT devices may not be affected by delivering energy because they are either plugged to a power point or self-harvesting. An example of such an IoT device is smart shirt which harvests energy from a body heat. An energy consumption model is required to estimate the real availability of the energy service. The real availability of an energy service is affected by three factors, (i) the usage of the IoT device by the providers themselves, (ii) the energy delivery to the consumer, and (iii) the preference of the provider. Carroll et al. [3] define different usage patterns of IoT devices. According to these patterns the energy consumption of the device can be estimated. Providers who want to share their energy have to follow one of these patterns as follows: *Suspend* i.e., not using their devices , *Casual* i.e., using them casually with few functionalities, or *Regular* i.e., using them with a predictable usage behavior. We define α as a parameter related to the usage pattern. We set $\alpha = 1$ if the usage pattern is "Suspend", $\alpha = 0.75$ for "Casual", and $\alpha = 0.50$ for "Regular".

QoS parameters allow users to distinguish among crowdsourced energy services. In the following, we propose quality attributes for an ES.

Definition 2: Crowdsourced Energy Service Quality Attributes.

- **Intensity:** Intensity shows the intensity of the wirelessly transfered current. The current is transferred under a certain voltage. We assume that all IoT devices related to energy services are functioning under a voltage between 3 and 5 V. These IoT devices are also compatible in term of voltage.
- **Transmission success rate:** Transmission success rate is the ratio between the transmitted energy from an energy provider and the received energy by an energy consumer [8]. The transmission success rate Tsr is calculated based on the following formula.

$$Tsr = \frac{G_t \gamma}{L_p} \left(\frac{\lambda}{4\pi(D + \beta)} \right)^{\theta}$$

where G_t, L_p,γ, λ, β, θ, and D represent the transmission gain, polarization loss, rectifier efficiency, wave length, short distance energy transmission parameter, path-loss coefficient, and the distance between devices, respectively.
- **Deliverable energy capacity:** The energy capacity that a consumer realistically receives is affected by the *Transmission success rate* Tsr and the

state-of-charge of the IoT device battery EC, the usage pattern α, and the predefined threshold by the device user Thr. It is given by *milliAmpere hour mAh*. Deliverable energy capacity DEC is defined as follows.

$$DEC = \alpha(EC - Thr)Tsr$$

- **Location** is the GPS location of ES that can transfer energy.
- **Start time:** Start time st is the time of launching an energy service by an IoT device. It is assumed to be announced by energy service providers.
- **End time:** End time et is the time where the energy capacity remains higher than the predefined threshold. If $et < aet$ (aet means the announced end time of the service by the provider), et is calculated based on an estimation formula and not the end time announced by the service provider aet. Otherwise, et is considered as the real end time of the energy service. Given the initial energy capacity EC, intensity of the transferred current I, the start-time of the energy service st, the usage pattern α, and the predefined threshold Thr, the service real end time et can be estimated by the following formula:

$$et = st + \alpha(EC - Thr)/I$$

Definition 3: Crowdsourced Energy Service Consumer Query. An energy service query is defined as a tuple $Q < t, l, re, d >$:

- t refers to a timestamp when the query is launched,
- l refer to a location of an energy service consumer. We assume that a consumer stays fixed after launching a query,
- re represents the required amount of energy,
- d refers to a user-defined charging period of time designated by its start time and end time.

Definition 4: Temporal crowdsourced energy service composition problem. Given a set of crowdsourced energy services $S_{ES} = \{ES_1, ES_2, \ldots ES_n\}$ and a query $Q < t, l, Re, d >$, the aim of our work is to transfer the required energy Re in the time period d by selecting the optimal composition of nearby energy services $ES_i \in S_{ES}$.

3 Temporal Crowdsourced Energy Service Composition Approach

We propose a deterministic spatiotemporal service composition framework for crowdsourcing energy services. We assume that service providers are fixed in space and they can transfer energy wirelessly inside a micro-region e.g., a coffeeshop. There is also an assumption that crowdsourced energy services are deterministic i.e., there is a knowledge about service availability and QoS values a-priori. First, we need an efficient indexing method for the fast discovery of energy services. Location and time are intrinsic parts of energy services. Therefore, we index energy services based on spatiotemporal characteristics The *3D R-tree* is a spatiotemporal index data structure which deals with range queries of the type "report all objects within a specific area during the given time interval" [7]. Time is added as the third axis to spatial axes. When a query Q arrives, an area is defined by the location of the consumer $Q.l$ and a range allowing the wireless power transmission between IoT devices. In our framework, we use *3D R-tree* to select energy services located in this defined area at the time interval $[Q.t, Q.t + d]$ (see Sect. 3.1). Then, we need a greedy approach to search through all of available services and select an optimal composition plan along the query duration (see Sect. 3.2). Figure 1 shows the time intervals of the five ES and the query Q.

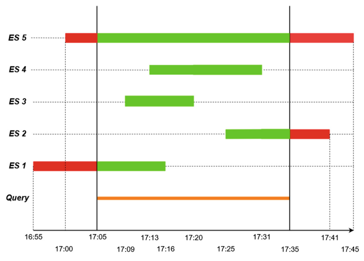


Fig. 1. Time intervals for nearby energy services for a query Q (Color figure online)

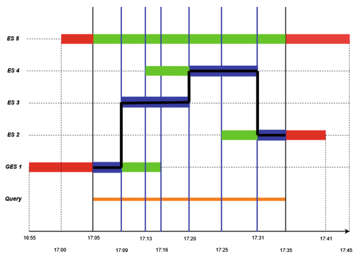


Fig. 2. Energy service composition (Color figure online)

3.1 Crowdsourced Energy Service Selection

Each energy service has a time interval $[st, et]$ and a location loc. Services are filtered spatially by selecting just the services inside the area defined by the consumer location $Q.l$ and the user defined range r which allows a successful wireless energy transfer. Services also are filtered temporally by choosing just those having time interval within or overlapping with the query duration $[Q.t, Q.t + d]$ (see Fig. 1). Each leaf in the *3D R-tree* is considered as ES. A search cube SC is determined by the energy query location $Q.l$ and duration $Q.d$. All leaves inside or which overlap with the search cube are selected. The color in overlapping services in Fig. 1 illustrates their new availability lifetime. For partially available services, we recalculate the provided energy capacity according to their availability lifetime. We consider a uniform distribution in term of energy consumption for an IoT device and its delivery. We use the following formula to update DEC of the partially available services.

$$ES_i.DEC = (ES_i.et - ES_i.st)\ .\ ES_i.I\ .\ ES_i.Tsr$$

where $ES_i.st$ and $ES_i.et$ are the new start time and end time of ES respectively. $ES_i.I$ is the intensity of the wirelessly transferred current. $ES_i.Tsr$ is the wireless transmission success rate. We define a function called *Service Selector*.

The function has as an input, the position of the consumer $Q.l$, the starting time of the query $Q.t$, and the duration of the query $Q.d$. The output is the set of all nearby available services $NearbyS$ within the query duration (Line 3 in Algorithm 1).

Algorithm 1. Crowdsourcer

```
1: **Input:** $Q.t$, $Q.d$, $Q.re$
2: **Output:** $CompServ$    // a set of energy services available during $Q.d$
3: $NearbyS \leftarrow$ Service selector($Q.l$, $Q.t$, $Q.d$)
4: if ($\exists ES_i \in NearbyS \mid ES_i.st \geq Q.t$ and $ES_i.et \leq Q.t + d$) then
5:     if there is one or multiple services satisfying the condition then
6:         $CompServ = \{preferred\ ES_i\}$
7:     else        // There is multiple services but not satisfying the condition
8:     // Fractional temporal service composition
9:     create new $time_slot.st \leftarrow Q.d$ // Divide $Q.d$ into time slots
10:    For $int\ \ t = Q.t\ to\ Q.t + d$ do
11:    if ($for\ all\ ES_i \in NearbyS\ and\ t = ES_i.st\ or\ t = ES_i.et$) then
12:        previous $time_slot.et \leftarrow t$
13:        if $t \neq Q.t + d$ then
14:            create new $time_slot.st \leftarrow t$
15:    EndFor
16:    For each slot
17:    Max $\leftarrow$ ($Max\ miniES.DEC$)
18:    if ($miniES.DEC = Max$ ) then
19:        $CompServ \leftarrow CompServ \cup (\{miniES\})$
20:    EndFor
21: End
```

3.2 Temporal Composition Algorithm

A single energy service may not satisfy the required amount of energy. As a result, we require to compose the optimal nearby energy services to fulfill the required amount of energy within the query duration. We assume that an energy service consumer does not receive more than one service at a time. The composition in this situation is an optimal sequence of services or partial services along a query time which fulfills the required energy.

We formulate our composition problem as a knapsack problem. A knapsack problem [11] is the selection of a set of items having weights and values by maximizing the total value of selected items and considering the limited weight capacity of the knapsack. We interpret each ES as an item. The time intervals $[st, et]$, DEC, and the query duration are considered as weights, values, and the knapsack weight capacity, respectively. Conversely to the classic knapsack problem where the items cannot be fractioned, time intervals of energy services can be chunked. As a result, we reformulate our composition problem as a fractional knapsack problem FKP [6]. We propose a greedy function *Crowdsourcer* which aims to select an optimal set of ES based on their deliverable energy capacity and time intervals. The inputs of *Crowdsourcer* are the query start time $Q.t$, end time e.g., $Q.t + d$, and the required energy $Q.re$. The output of *Crowdsourcer* is an optimal composition plan in which component energy services are ordered based on their start times.

The *Crowdsourcer* first calls *Service Selector*. *Service Selector* returns all available energy services within $Q.d$ (Line 3 in Algorithm 1). In the previous example in Fig. 1, $ES3$, $ES4$, are selected as available services. $ES1$, $ES2$, $ES5$ are selected as partially available services $NearbyS = \{ES1, ES2, ES3, ES4, ES5\}$. After selecting candidate services, the *Crowdsourcer* searches if there exist an energy service which fulfills the required energy $Q.re$. If the result is one or multiple services, the function can take the shortest duration service, the earliest start time service, or a service providing the maximum of energy according to a consumer's preferences (Line 6 in Algorithm 1). It is possible that there exist multiple services providing less than the required energy. A service composition is needed to achieve the required amount of energy. Sometimes, it is not possible to get the required energy from all the available service. In this situation, *Crowdsourcer* returns a set of available services which provides the maximum available amount of energy within the query duration.

Our composition problem is formulated as a fractional knapsack problem to find an optimal set of energy services providing *the maximum amount of energy* within *the query duration*. The proposed approach selects the best composition of services or parts of services (e.g., fractions) which provides the maximum energy in the query duration. Our approach relies on dividing the query duration into several time slots. Each time slot is limited either by the start time of an available service or its end time (Lines 10 to 15 in Algorithm 1), see vertical lines in Fig. 2. For example, start time of $ES3$ and $ES4$ define the first and second time slots, respectively. All energy services are fractioned into mini services, one mini service per time slot. $ES3$ is fractioned into three mini services, because $ES3$ crosses three time slots. Solving fractional knapsack problem relies on choosing mini services with highest density value ratio per each time slot (Lines 16 to 20 in Algorithm 1). In our case, the density value ratio refers to the deliverable energy capacity DEC on a unit of time *hour*. This ratio is the effectively transferred current $effI$ from an energy service:

$$effI = Tsr \ . \ \frac{ES_i.DEC}{ES_i.et - ES_i.st}$$

The blue segments in Fig. 2 represent the mini service providing the maximum DEC in each time slot. The composition is the set of selected mini services from the query start time $Q.t$ to the end of its duration $Q.t + d$.

4 Experiment Results

We evaluate effectiveness and feasibility of the proposed composition framework in terms of the number of served queries. We define different scenarios (see

Table 1. Different query scenarios for **short services** providing **small** amounts of energy

$Q.re/Q.d$	Short	Long
Low	Scen 1	Scen 3
High	Scen 2	Scen 4

Table 2. Different query scenarios for **short services** providing **large** amounts of energy

$Q.re/Q.d$	Short	Long
Low	Scen 5	Scen 7
High	Scen 6	Scen 8

Table 3. Different query scenarios for **long services** providing **small** amounts of energy

$Q.re/Q.d$	Short	Long
Low	Scen 9	Scen 11
High	Scen 10	Scen 12

Table 4. Different query scenarios for **long services** providing **large** amounts of energy

$Q.re/Q.d$	Short	Long
Low	Scen 13	Scen 15
High	Scen 14	Scen 16

Tables 1, 2, 3 and 4). To the best of our knowledge, there is limited research investigating temporal service selection. We compare the proposed temporal composition algorithm and a greedy selection algorithm as our baseline to show the feasibility of our approach. The greedy selection algorithm selects the energy service providing the maximum DEC among the available services within the query duration $Q.d$.

All algorithms are implemented in Java. The experiments are conducted on a 3.60 GHZ Core i7 processor and 8 GB RAM under Windows 10. We use *Yelp* which contains people's *check-ins* into coffee places and restaurants. We assume that people who offer energy services from their wearables are lingering in a coffee place. We define spatiotemporal features of energy services by *check-in* and *check-out* times of customers at the coffee place. These times reflect start time st and end time et of energy services at the coffee place. We augment our real dataset with the synthetic QoS parameters. The energy service QoS parameters including deliverable energy capacity DEC, intensity of the transferred current I, and the transmission success rate Tsr are randomly generated. Similarly, energy queries time $Q.t$ and duration $Q.d$ are generated from *check-in* and *check-out* times of coffee place customers. The required energy $Q.re$ is also randomly generated for each query.

Tables 1, 2, 3 and 4 illustrate different query scenarios in terms of the required energy and the query duration for services with short duration (Scen. 1 to Scen. 8). The same scenarios are defined with long services (Scen. 9 to Scen. 16) in Fig. 4. For each scenario, energy queries are served by either the greedy selection or the temporal composition algorithm. We assess the number of served queries by both algorithms to conclude the best scenarios of our temporal composition algorithm. Figures 3 and 4 show the number of served queries by energy services with short and long duration. The results indicate the necessity of temporal composition for almost all the queries requiring high amounts of energy when the existing services have short duration and provide low amounts of energy like the energy provided by wearables (e.g. Scen 2, Scen 4 in Fig. 3). Paradoxically, the temporal composition is also required for short duration queries requiring small amount of energy from long duration services (e.g. Scen 10, Scen 12 in Fig. 4). This is because of the property of fractioning energy services into mini services (see Sect. 3).

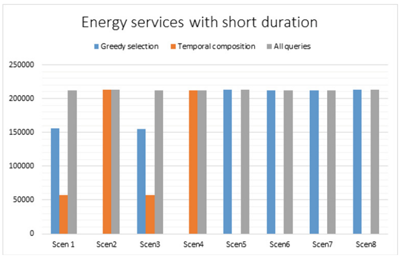


Fig. 3. Served queries with short services

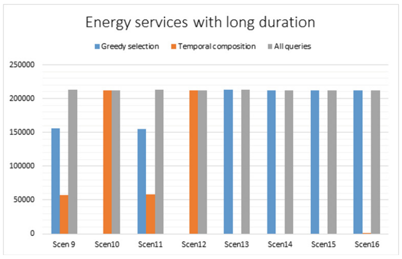


Fig. 4. Served queries with long services

5 Conclusion

This paper proposed a novel approach for crowdsourcing energy services from IoT devices. We defined an energy as a service model for crowdsourcing energy from wearables. We defined a new quality model for energy services. We devised a temporal composition framework to meet users' energy requirements. A set of experiments is conducted on a real dataset to illustrate the feasibility of the temporal composition framework for energy services. The results also show that the proposed framework deals effectively with the resource limitation of IoT devices which is reflected by the services providing small amounts of energy. Future work includes mobility-aware composition of crowdsourced energy services.

Acknowledgments. This research was partly made possible by NPRP 9-224-1-049 grant from the Qatar National Research Fund (a member of The Qatar Foundation) and DP1601 00149 and LE180100158 grants from Australian Research Council. The statements made herein are solely the responsibility of the authors.

References

1. Bouguettaya, A., et al.: End-to-end service support for mashups. IEEE Trans. Serv. Comput. **3**(3), 250–263 (2010). https://doi.org/10.1109/TSC.2010.34
2. Bouguettaya, A., et al.: A service computing manifesto: the next 10 years. Commun. ACM **60**(4), 64–72 (2017). https://doi.org/10.1145/2983528
3. Carroll, A., Heiser, G., et al.: An analysis of power consumption in a smartphone

4. Ghari Neiat, A., Bouguettaya, A.: Crowdsourcing of Sensor Cloud Services. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91536-4
5. Habak, K., Ammar, M., Harras, K.A., Zegura, E.: Femto clouds: leveraging mobile devices to provide cloud service at the edge. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), pp. 9–16. IEEE (2015)
6. Ishii, H., Ibaraki, T., Mine, H.: Fractional knapsack problems. Math. Program. **13**(1), 255–271 (1977)
7. Jun, B., Hong, B., Yu, B.: Dynamic splitting policies of the adaptive 3DR-tree for indexing continuously moving objects. In: Mařík, V., Retschitzegger, W., Štěpánková, O. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 308–317. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45227-0_31
8. Na, W., Park, J., Lee, C., Park, K., Kim, J., Cho, S.: Energy-efficient mobile charging for wireless power transfer in internet of things networks. IEEE IoT J. **5**(1), 79–92 (2018)
9. Neiat, A.G., Bouguettaya, A., Sellis, T., Mistry, S.: Crowdsourced coverage as a service: two-level composition of sensor cloud services. IEEE Trans. Knowl. Data Eng. **29**(7), 1384–1397 (2017)
10. Perera, C., Liu, C.H., Jayawardena, S., Chen, M.: A survey on internet of things from industrial market perspective. IEEE Access **2**, 1660–1679 (2014)
11. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. **32**(9), 2271–2284 (2005)
12. Seneviratne, S., Hu, Y., Nguyen, T., Lan, G., Khalifa, S., Thilakarathna, K., Hassan, M., Seneviratne, A.: A survey of wearable devices and challenges. IEEE Commun. Surv. Tutor. **19**(4), 2573–2620 (2017)

Social-Sensor Composition for Scene Analysis

Tooba Aamir[1](✉), Hai Dong[1], and Athman Bouguettaya[2]

[1] School of Science, RMIT University, Melbourne, Australia
{tooba.aamir,hai.dong}@rmit.edu.au

[2] School of Information Technologies, The University of Sydney, Sydney, Australia
athman.bouguettaya@sydney.edu.au

Abstract. We consider the scene analysis as a service composition problem. A social-sensor cloud services composition model is proposed for the scene analysis. Our proposed model selects and composes social-sensor cloud services based on the user queries. Textual features of the social-sensor cloud services, i.e., description, comments, and meta-data of the social media images are used to reconstruct a scene. Our key contribution is an efficient and real-time composition of related images for scene analysis relying on meta-data and related posted information. Analytical results demonstrate the performance of the proposed model.

1 Introduction

The rich and explosive growth of social media data has resulted in the integration of social data into a range of data-centric applications [1,2]. Recent communication devices like smartphones, i.e., social-sensors provide the ability to embed sensor data directly into cloud-based social networks, i.e., social-sensor clouds [2,4]. Monitoring these social-sensors' activities provide multiple benefits in various domains. For example, urban management requires scene reconstruction and analysis in an area. Suppose the surveillance of the road segment through traditional sensors is limited in coverage. In such cases, social-sensors facilitate to fill in the information gap within events or happenings [5].

Social-sensor data, e.g., social media image meta-data and related posted data (e.g., location, description, and comments) are inherently multi-modal because of the different data formats and sources in those social media platforms. The multifaceted data poses a significant challenge for the efficient and real-time delivery of the social-sensors' data to the users [7,13]. In our previous work, we propose the *social-sensor cloud services* to provide an open, flexible, and reconfigurable platform for monitoring and controlling applications [4,5]. We abstract social-sensor cloud data, e.g., images' annotations (meta-data and related information like description and comments) as a service, i.e., *social-sensor cloud service*, to fulfill the users' information requirement [4,5].

This paper focuses on using the service paradigm as a vehicle to devise a method for scene reconstruction and analysis without carrying out actual image processing. The aim is to provide the similar useful information about the required scene as image processing does [5]. A complete scene analysis needs images from multiple angles and different time intervals. In such cases, a composition of services is required to form multiple viewing angles to fulfill the users' requirement(s) [10,12]. In this regard, we have identified the following challenges:

- *Relevance Model for Spatio-temporal Cloud Service:* The accurate information regarding the context of the service is vital for better utilization and selection of the social-sensor cloud service as per user requirements [8]. The relevance of the service to the given query helps to ascertain whether the service is in the same context as of the query.
- *Spatio-temporal Composition.* Social-sensor cloud services composition becomes even more challenging in dynamic service environments characterized by changing conditions and context. An optimal composite service is a set of social-sensor cloud services, providing the best-suited services at any given time as per the users' query. A spatio-temporal composition aims to execute an optimal composition based on the functional attributes.

This paper accommodates the solution of the challenges mentioned above. We propose a composite service that will provide the user-required view and related information about any event or a happening for the scene analysis. The proposed composition model forms a tapestry in the spatial aspect and a storyboard in the temporal aspect. In the spatial aspect, the composition forms a scene by selecting images from un-coordinated users and placing them in a tapestry-like structure. In the temporal aspect, a timeline is formed by combining various tapestries to form the story of the event.

2 Motivation Scenario

Let us assume an accident occurred on 5th July 2016 around 8:30 pm, on the Pascovale Road, Glenroy. The crash involves two vehicles cars A and B crossing an intersection. The service user, i.e., the police has queried a scene analysis of the accident. The aim is to find the original behavior leading to the crash and the objects of interest, i.e., the vehicles or people involved. In such case, anyone in the area can act as a social-sensor by sharing images over a social network. We rely on these social-media images as social-sensor cloud services in the vicinity during that specific period to reconstruct the desired scene.

This work proposes a model for selection and composition of the social-sensor cloud services based on the user query. The query includes a region of interest, textual description and time of the queried event. The query includes (1) Query phrase, e.g., a car accident involving Car A and B on city-bound Pascovale Road. Car A and Car B are the objects of interest. (2) Query region, i.e., decimal longitude-latitude position. For example, *(−37.694264, 144.9131593)* covering the area of 10 m on all sides of the road. (3) Query time, e.g., 5th July 2016, from 8:25 pm to 8:40 pm.

The basic functional attributes of a social-sensor cloud service *Serv*, are abstracted from the social media image information. These include:

- Time T of the service at which the image is taken.
- Description D is a set of keywords or key-phrases providing additional information regarding the image, e.g., Car A crashes in Car B, Car accident.
- Location $L(x, y)$ is longitude and latitude position where the image is taken.
- Coverage Cov of the image is defined as $VisD$, i.e., the maximum visible distance, covered by the image, $\overrightarrow{dir}$, i.e., the orientation angle of the image and α, the angular extent of the scene covered by the image.

It is assumed that the available services are tagged with location and time. We index all the available services considering their spatio-temporal features using a 3D R-tree [4,6]. The search space is reduced by selecting the services that are spatio-temporally close to the querying location and contextually related to the query description. For example, at time t_{-1}, the descriptions of three images img_1, img_2 and img_3 show that Cars A and B were running along Pascovale Road city-bound, and Car C was taking the exit from M80 Ring Road. Cars A and B are objects of interest and therefore img_1, img_2 and img_3 are selected due to their contextual relevance to the query. Further, at time t_0, the description of an image img_4 shows that Car A stopped and avoided the collision with Car C. Therefore, Car C is considered as interacting with the object of interest Car A and img_4 is selected. Three images img_5, img_6 and img_7 in the spatio-temporal query region show that at the intersection, Car C ran the red light. At time t_1, four images img_8, img_9, img_{10} and img_{11} are selected due to their spatio-temporal and contextual relevance. Images img_8, img_9 and img_{10}'s description says that Car B and Car A crashed. Image img_{11}'s description says Car C escaped the accident scene.

11 services (images) are selected in this scenario. We cluster the selected services according to their spatio-temporal and contextual relationships. The contextual clustering is based on the interaction and relations between the services. The interactions and relationship between the objects of interest of the services are determined on the basis of the semantic similarity between the service description and the query description. The event-specific relationship describing the vocabulary dictionary provided by domain experts is used for this purpose. We assess the services for composability. The composability is assessed by predefined relations, explained in the relevance and composability models (Sect. 4). Finally, we build-up the composition, i.e., a visual summary by forming a tapestry-like scene. The composition is formulated by selecting the composable services covering the accident, the object of interests and the interacting object. The composition depicts the cars crashed and the cars involved, i.e., Car A and B crashed, and Car C escaped the crash scene.

3 Model for Social-Sensor Cloud Service

In this section, we have defined the social-sensor cloud service, selection, and composition model.

3.1 Model for an Atomic Social-Sensor Cloud Service

An atomic social-sensor cloud service *Serv* is defined by:

– *Serv_id* is a unique service id of the service provider *SocSen*.
– F is a set of functional properties of the service *Serv*.

3.2 Functional Model of an Atomic Social-Sensor Cloud Service

The functional requirements capture the intended behavior of an atomic service and form the baseline functionality. The minimal functional requirements associated with an atomic service and their information sources are:

– Social-sensor device: The basic functional attributes of a social-sensor cloud service associated with social-sensor device are time t, location *L(x,y)* and coverage *Cov* of the sensor. We have discussed all these parameters in [5].
– Social-sensor service owner: Context *Con* of a social-sensor cloud service is associates with the service owner. It is the description of a service provided by the service owner. Context *Con* is defined by D and T. Description D of the service provides additional information regarding the image. It is assumed that the service's description includes complete detail of the service specifics related to the scene captured, e.g., objects captured, and their relations. Tags T provide location and focus of the image.
– Social-sensor cloud: Interaction I is the social network provided information regarding objects of interest in the image. It is assumed that the description includes detail of the objects of interests. This description is provided by the users of the cloud, i.e., social media, through comments. We assume that the information collected though comments is trustworthy. The trustworthiness of the comments is dealt in our previous work [3,9].

4 Social-Sensor Cloud Service Composability

In this section, we propose the social-sensor cloud service relevance and composability models for the social-sensor cloud services.

4.1 Model for Social-Sensor Cloud Service Relevance

The relevance of a service to a given query or another service helps to ascertain whether the service is in the same context as of the query or the other service. The relevance between two or more social-sensor cloud services can be described as spatio-temporal closeness, contextual relatedness and interaction relevance. The relevance between two services $Serv_1$ and $Serv_2$ can be defined as:

Spatial Relevance. Rel_S means $Serv_1$ and $Serv_2$ are close in space boundaries and have similar coverage direction. This encompasses ($Serv_1.Cov_{(\alpha,dir)} \cong Serv_2.Cov_{(\alpha,dir)}$), i.e., similar in directions and angles AND $Serv_1.L =$

$Serv_2.L \pm \Delta$, i.e., close in the geo-location. Where, Δ is the max. allowed spatial difference.

Temporal Relevance. Rel_t means $Serv_1$ and $Serv_2$ coincide in time, i.e., $((Serv_1.t_e = Serv_2.t_s \pm \varepsilon) \mid (Serv_1.t_s = Serv_2.t_s \pm \varepsilon) \mid (Serv_1.t_e = Serv_2.t_e) \pm \varepsilon)$. Where, $(Serv_1.t_e = Serv_2.t_s \pm \varepsilon)$ means the end time of $serv_1$ is close to the start time of $Serv_2$. $(Serv_1.t_s = Serv_2.t_s \pm \varepsilon)$ means the start time of $serv_1$ is close to the start time of $Serv_2$. $(Serv_1.t_e = Serv_2.t_e \pm \varepsilon)$ means the end time of $serv_1$ is close to the end time of $Serv_2$. ε is the max. allowed time difference.

Spatio-Temporal Relevance. Rel_{St} means $Serv_1$ and $Serv_2$ have overlap in time and space. This encompasses $Rel_S \cap Rel_t$.

Contextual Relevance. Rel_C means $Serv_1$ and $Serv_2$ share same or almost similar context. This encompasses $(Serv_1.Con \cong Serv_2.Con)$. The contextual relevance is based on the textual similarity of the contextual descriptions of both services. Contextual relevance is calculated as a semantic distance between the descriptions of the services and the query [5]. Event specific relationships are used for the implementation of the similarity measure. These event specific relationships are described in the vocabulary dictionary provided by the domain experts. We have used θ to define $related_{LIN}(Serv_1.Con, Serv_2.Con)$. The higher value of θ shows higher similarity in context.

Interaction Relevance. Rel_I means $Serv_1$ and $Serv_2$ both share objects of interest in the coverage (refer Sect. 4.1). This encompasses $(Serv_1.I \cap Serv_2.I)$

4.2 Model for Social-Sensor Cloud Service Composability

The spatio-temporal and contextual composability of two or more social-sensor cloud services can be defined as four instances:

– $(Rel_{St} \cap Rel_C)$. Two or more services are composable if these services are spatio-temporally and contextually relevant.
– $(Rel_t \cap Rel_C)$. Two or more services are composable if these services are temporally and contextually relevant. In such cases, services might be located outside the region of interest but still capture a scene inside.
– $(Rel_S \cap Rel_C)$. Two or more services are composable if these services are spatially comparable and contextually relevant. In such cases, services are available either before or after the required period.
– $(Rel_C \cap Rel_I)$.Two or more services might be composable if these services share context and objects of interest. In such cases, services might be located outside the region of interest but still capture some related objects of interest.

5 Social-Sensor Cloud Service Composition Approach

We propose an approach to filter, select and compose the best available social-sensor cloud service to form a visual summary according to the user's query. The composition is achieved by constructing the information context of the service with the functional. The composite service comprises a set of selected atomic services to form a visual summary of the queried event. The visual summary offers an arrangement of the 2D images, forming a tapestry-like scene of the required event. Our approach aims to efficiently compose the available services into a single composite service that matches with the users' requirements.

A query q can be defined as $q = (Rgn, des, t_s, t_e)$, giving the region of interest, description and time of the required service(s).

- $Rgn = \{P < x, y >, l, w\}$ [5], where P is a geospatial co-ordinate set, i.e., decimal longitude-latitude position and l and w are length and width distance from P to the edge of region of interest.
- t_s is the start time and t_e is the end time of the query.
- *des* is a phrase describing the query. Query description includes details of the objects of interests *obj*, i.e., objects involved and the context of the query *cont*, i.e., the scene to be captured.

5.1 Social-Sensor Cloud Service Selection

The indexing and spatio-temporal filtering of the services enable the fast discovery of the services. We index all the available services using a 3D R-tree [4] and select the services inside the bounded region of interest [5]. Next, the services are selected and classified based on the relevance between the services, the queried scene and the objects of interest. It might happen that the service does lie spatio-temporally in the query area *Rgn*, but has no contextual relation with the query q or has too much noise concerning unwanted information. In such cases, the object(s) of interest and behavior relations are used for the service filtration. The contextual relevance of all the services to a query's scene and objects of interest are assessed. Using previous research as reference we have set the value of threshold $\theta = 0.5$ for the contextual relevance [14]. The services related to the queried scene and objects of interests are selected. The services are classified in three sets according to their relevance: (1) spatio-temporally and contextually relevant services S_{StC}, (2) spatio-temporally relevant and interacting services S_{StI} and (3) contextually relevant and interacting services S_{CI}.

5.2 Social-Sensor Cloud Service Composability Assessment

The composability rules aims to construct a composite service. Composability assessment among component services is based on their spatio-temporal and contextual parameters. The relevance and overlap is considered to define the composability relations between the services, e.g., $Serv_1$ and $Serv_2$. We aim to define composability of the service as quantitative relations. The relevance between the services is an arithmetic mean of the considered parameters. It is calculated as:

$$\begin{aligned} Rel(Serv_1, Serv_2) = [(&Rel_{St}(Serv_1, Serv_2) + \\ &Rel_C(Serv_1, Serv_2) + Rel_I(Serv_1, Serv_2))] \end{aligned} \tag{1}$$

where, $Rel_{St}(Serv_1, Serv_2)$ is based on the time of the services and their proximity in space. λ is the shortest distance between $Serv_1$ and $Serv_2$ and ϑ is the difference between coverage angles $Serv_1.Cov_{dir}$ and $Serv_2.Cov_{dir}$. The thresholds for the spatial relevance are set as λ_{thr} for distance and ϑ_{thr} for $\overrightarrow{dir}$. Therefore, the services are considered spatio-temporally relevant if difference between the distance and direction of the services is below the threshold. $Rel_C(Serv_1, Serv_2)$ is the semantic distance between the descriptions of $Serv_1$ and $Serv_2$ (Refer Sect. 5.1). $Rel_I(Serv_1, Serv_2)$ is the count of the mutual objects of interest in $Serv_1$ and $Serv_2$. The overlap between the services is considered:

$$Overlap(Serv_1, Serv_2) = Overlap_{spatial}(Serv_1, Serv_2) \tag{2}$$

The quantitative value of the mutual composability is calculated as:

$$Comp(Serv_1, Serv_2) = Rel(Serv_1, Serv_2) - Overlap(Serv_1, Serv_2) \tag{3}$$

A geographic coverage patch *GeoPatch* is formed to assess the composability of each service from the spatio-temporal and contextual selection S_{StC}. A set *N* of the spatio-temporally nearest services is selected for each *GeoPatch.* The mutual composability *Comp* is calculated with each service in *N.* The process of calculating the mutual composability of the services is repeated with the sets *N'* and *N"*. *N'* is the set of the nearest services concerning the spatio-contextual and temporal-contextual relevance. *N"* is a set of the nearest services based on the contextual relevance and interaction. The assessment process of the mutual composability is based on relevance and overlap of the services.

5.3 Social-Sensor Cloud Service Composition

The composition is handled as sewing a tapestry to form the scene. We start with the central piece, concerning space and time, and build a tapestry around it. The build-up is based on selecting the best composable services from the set of nearest services. The best neighbor service is with the maximum relevance and the minimum overlap.

The composition covers the visual summary of the whole queried *scene*, i.e., all objects of interest and their context. We choose the central service $Serv_c$ in terms of space and time from the spatio-temporal and contextual selection. We further add $Serv_c$'s neighbors to a separate pool. We assume that the central service is in the middle of the spatio-temporal dimension. Next, we extract the best neighbor service $Serv_{k.bn}$ from the pool and place it with $Serv_c$ by joining the patch. $Serv_{k.bn}$ is selected according to the maximum composability. We add neighbors of $Serv_{k.bn}$ to the pool. The process of selecting the best neighbor and joining to patch continues until we have any service in the pool. We reassess the composability of the remaining services and start again with the nearest service if the pool is empty. Spatial gaps in the composition are assessed after the utilization of all services from the spatio-temporal and contextual selection. $Comp.C$ is the total coverage of the services in the composition overlapping the bounded region *Rgn* and within time t_s and t_e. The relationship between *Serv* and *q.Rgn* can be illustrated as:

$$Composition \longrightarrow \{Comp \in \cup_{i=1}^{n} Serv \mid (Comp.C \cap Q.Rgn) \cap Rel_C \cap Rel_I, t_s \leq t \leq t\_e\} \tag{4}$$

In our previous work, we have discussed the coverage of the composition and gap assessment [5]. We estimate and select an arbitrary neighbor $Serv_{kc'}$ if there are any spatial gaps. Next, the best nearest service $Serv_{k.bn}$ from the set of the spatio-temporally relevant and interacting services. The process of selecting and joining the services continues until we fill in the gaps and get the maximum available coverage. The composite service is a series of spatial tapestries in time, providing a timeline of the visual summary of the event.

6 Experiment and Evaluation

We focus on evaluating the proposed approach using the real dataset. The set is a collection of 10000 user uploaded images downloaded from social networks (flicker, twitter, google+). We had extracted their geo-tagged locations, the time when an image was captured, post description and tags to create the services. Further, the camera direction $\overrightarrow{dir}$, the maximum visible distance of the image *VisD* and the viewable angle α are abstracted as the functional property *Cov.*

We generated eight different queries based on the locations and events in our dataset. We have evaluated the service composition based on the spatial relevance in the first part of the experiment. The result of these experiments is evaluated upon the traditional image processing technique SIFT (Scale-Invariant Feature Transform) [11]. We used images' geolocation information, associated directions and viewing angles to gather an associated image dataset *I* from Google Street View of the area of interest *R*. We first downloaded 360° views of Google Street View using GPS from the image and collected the views related to the service. Further, we compared the similarity between images in the composition and the image set *I* by SIFT features. This comparison is achieved by individually comparing the key point feature vector of the images in *I* and images in the composition, and finding the images' matching features based on the Euclidean distance of their feature vectors. Further, we assessed if the images in the composition are correctly positioned in spatial relations. The evaluation of the similarity threshold is set around 60%. 40% noise margin is given due to traffic and pedestrian obstruction in the images.

We have assessed how useful the composite service is in completing the contextual storyboard in the second part of the experiment. The assessment is done by manually analyzing the composition for the spatial-temporal and contextual coverage. The effectiveness of the composite service is assessed upon the selection and composition of the related and accurate services. It is assessed if the composite service contains the required object(s) of interest and their behavior according to the user query.

Table 1. Relative accuracy in spatio-temporal coverage

	Queries themes		Accuracy rate of SIFT selection
Event-oriented queries	Q1	Bourke street accident	57.9%
	Q2	F1 race, Melbourne	54.5%
	Q3	Essendon airport crash	36.8%
	Q4	CBD random accident	54.5%
Location-oriented queries	Q5	Melbourne night	81.5%
	Q6	Melbourne central	78.6%
	Q7	Melbourne trams	75.0%
	Q8	Elishbeth street, Melbourne CBD	69.2%
Average			63.5%

Table 2. Precision and recall

	Queries themes		Precision	Recall
Event-oriented queries	Q1	Bourke street accident	51%	72%
	Q2	F1 race, Melbourne	55%	88%
	Q3	Essendon airport crash	73%	79%
	Q4	CBD random accident	77%	66%
	Average		**64%**	**76%**
Location-oriented queries	**Q5**	Melbourne night	88%	79%
	Q6	Melbourne central	68%	55%
	Q7	Melbourne trams	80%	53%
	Q8	Elishbeth street, Melbourne CBD	80%	74%
	Average		**79%**	**65%**

6.1 Evaluation

We have evaluated the proposed approach by (1) accuracy in the spatial coverage of the user required region, (2) effectiveness in selecting the related services (precision), and (3) effectiveness of the composite service in capturing the required context, i.e., the object(s) of interest and their behaviors (recall). All images and the composed services are manually analyzed by a human to form a baseline.

We have assessed the composite services by comparing the similarity between the service image and the Google street view. SIFT image processing is used for the comparison of all the eight queries (Table 1). We observed that approximately 63% of services in the compositions are accurately categorized in space. The 37% error rate was reasonable due to the noise in the images. Noise is an obstruction in the image affecting the scene building. For example, a vehicle obstructing the building of interest can be considered as noise. Further, we have assessed the composite services by manually analyzing the effectiveness of selecting the relevant spatio-temporal services, i.e., precision (Table 2). The average precision of the proposed approach for the location-based queries is 78% and for the event-based queries is 64%. The effective spatio-temporal and contextual coverage are assessed by recall (Table 2). The average recall of the proposed approach for the location-based queries is 65% and for the event-based queries is 76%. The results show that the values of precision are higher for the location-oriented queries, e.g., Melbourne Central (Q6). The values of recall are higher for the event or scene-oriented queries, e.g., Bourke Street Accident (Q1). Therefore, it is concluded that our proposed approach effectively helps in the accurate composition of the services for the scene analysis. The proposed approach considers the related contextual data that describes the situation from various aspects, e.g., what has happened, where it happened, who is involved and what the effects on surrounding area.

7 Conclusion

We propose a social-sensor cloud service composition approach based on the spatio-temporal and contextual relevance. Our experiments evaluate the proposed approach for an accurate and effective composition. We plan to focus on the optimal social-sensor cloud service composition based on the uncertain time, location and context requirements.

Acknowledgement. This research was partly made possible by NPRP 9-224-1-049 grant from the Qatar National Research Fund (a member of The Qatar Foundation) and DP1601 00149 and LE180100158 grants from Australian Research Council. The statements made herein are solely the responsibility of the authors.

References

1. Rosi, A., Mamei, M., Zambonelli, F., et al.: Social sensors and pervasive services: approaches and perspectives. In: Proceedings of PERCOM (2011)
2. Aggarwal, C.C., Abdelzaher, T.: Social sensing. In: Aggarwal, C.C. (ed.) Managing and Mining Sensor data. Springer, Boston (2013). https://doi.org/10.1007/978-1-4614-6309-2_9
3. Aamir, T., Dong, H., Bouguettaya, A.: Trust in social-sensor cloud service. In: Proceedings of IEEE ICWS (2018)
4. Aamir, T., Bouguettaya, A., Dong, H., et al.: Social-sensor cloud service selection. In: Proceedings of IEEE ICWS (2017)
5. Aamir, T., Bouguettaya, A., Dong, H., Mistry, S., Erradi, A.: Social-sensor cloud service for scene reconstruction. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 37–52. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_3
6. Neiat, A.G., Bouguettaya, A., Sellis, T., Ye, Z.: Spatio-temporal composition of sensor cloud services. In: ICWS (2014)
7. Bouguettaya, A., Singh, M., et al.: A service computing manifesto: the next 10 years. In: CACM (2017)
8. Wang, H., Shi, Y., et al.: Web service classification using support vector machine. In: Proceedings of IEEE ICTAI (2010)
9. Aamir, T., Dong, H., Bouguettaya, A.: Stance and credibility based trust in social-sensor cloud service. In: Proceedings of WISE (2018)
10. Ghari Neiat, A., Bouguettaya, A., Sellis, T.: Spatio-temporal composition of crowdsourced services. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 373–382. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_26
11. Lowe, D.G.: Distinctive image features from scale-invariant key points. IJCV **60**, 91–110 (2004)

12. Li, L., Liu, D., Bouguettaya, A.: Semantic based aspect-oriented programming for context-aware web service composition. Inf. Syst. **36**(3), 551–564 (2011)
13. Bouguettaya, A., Nepal, S., et al.: End-to-end service support for mashups. In: IEEE TSC (2010)
14. Mihalcea, R., et al.: Corpus-based and knowledge-based measures of text semantic similarity. In: Proceedings of AAAI (2006)

QITA: Quality Inference Based Task Assignment in Mobile Crowdsensing

Chenlin Liu, Xiaofeng Gao(✉), Fan Wu, and Guihai Chen

Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai 200240, China
ChanningLiu@sjtu.edu.cn, {gao-xf,fwu,gchen}@cs.sjtu.edu.cn

Abstract. With the rapid proliferation of mobile devices, Mobile Crowdsensing (MCS) has become an efficient way to ubiquitously sense and share environment data. Due to the openness of MCS, sensors and workers are of different qualities. Low quality sensors and workers may yield low sensing quality. Thus it is important to infer workers' qualities and seek a valid task assignment with enough total qualities for MCS. To solve the quality inference problem, we adopt *truth inference* methods to iteratively infer workers' qualities. This paper also proposes an task assignment problem called *quality-bounded task assignment with redundancy constraint* (QTAR) based on truth inference. We prove that QTAR is NP-complete and propose a $(2 + \epsilon)$ - approximation algorithm QTA for QTAR. Finally, experiments conducted on real dataset prove the efficiency and effectiveness of the algorithms.

Keywords: Mobile Crowdsensing · Task assignment · Truth inference

1 Introduction

In recent years, the rapid proliferation of mobile devices has changed people's lives. *Mobile Crowdsensing* (MCS) [2] has thus become an efficient way to ubiquitously sense and share environment data with mobile devices. One of the significant advantage of MCS comparing with traditional sensor networks comes from its active involvement of workers to collect and share sensing data. For a typical MCS, one of the most important problem is how to appropriately assign tasks to workers, which has recently been widely studied by researchers [4,6,7].

Due to the openness of MCS, sensors and workers are of different qualities. Low quality sensors and workers may yield low sensing quality. Low quality devices may collect noisy, even inaccurate sensing data. Moreover, low quality workers may randomly collect data ignoring the location constraints in oder to deceive payment. To overcome the quality inference problem, this paper introduce a method called *truth inference* to simultaneously infer the workers' qualities and the truth of each task. Most existing truth inference methods are redundancy-based, which means that one task is assigned to multiple workers.

By guaranteeing a certain amount of redundancy for each tasks, this paper investigates the quality-aware task assignment problem with budget constraint and propose quality bounded task assignment with redundancy constraint (QTAR). In QTAR, the total quality of selected workers exceeds the quality bound while the overall cost is minimized in the task assignment process. Different from tradition task assignment problem, we add redundancy constraint to satisfy the preliminaries of truth inference, which requires that each task should be assigned a certain or more amount of workers.

Therefore, our method is divided into two steps. Firstly, a truth inference method is adopted to infer workers' qualities based on previously aggregated sensing data. Next, we solve QTAR with the inferred workers' qualities in step 1. After workers finish their assigned tasks, the sensing data are aggregated and prepared to be used in the following quality inference step. This method, which is called Quality Inference Based Task Assignment (QITA), is an effective way to improve the overall sensing quality of MCS.

This paper proves that QTAR problem is NP-complete and propose a $(2+\epsilon)$ approximation algorithm QTA for QTAR. Finally, we evaluate our algorithms by conducting a series of experiments on both synthesis data and real dataset.

2 Related Work

In this paper, we focus on minimizing monetary cost for task assignment while satisfying the quality constraint. In order to infer workers' qualities under the lack of real data, we adopt truth inference methods [1,9], which has been widely studied in existing crowdsourcing works. Based on workers' answers, truth inference methods iteratively infer the truth and workers' qualities. Different from online crowdsourcing, in MCS, the aggregated sensing data are numeric data such as air quality and WiFi signal strength. A few researches studied truth inference methods for numeric tasks. [5] assumed that the answers of workers follows certain biases and variance and the paper adopted EM algorithm to iteratively infer the truth and workers' qualities. Moreover, [3] considered the confidence of each worker and proposed a confidence-aware truth discovery method to infer truth by considering the confidence interval of the inference.

3 Problem Formulation

We consider an MCS consists of a cloud server, multiple sensing tasks and multiple workers with mobile devices. When a worker is assigned a sensing task, her/he moves to the specified location to sense data. Based on the moving distance, cloud server allocates monetary reward to the worker. Let C_{ij} denote the monetary cost for assigning worker j to task i.

Suppose that there are m sensing tasks and n workers. Let $T = \{t_1, t_2, \ldots, t_m\}$ denote the set of tasks and $W = \{w_1, w_2, \ldots, w_n\}$ be the set of workers. The quality evaluation of worker j is represented by $q_j \in \mathbb{R}^+$, the larger q_j is, the more accurate the collected data of worker j will be. Let

$Q = \{q_1, q_2, \ldots, q_n\}$ denote the set of workers' qualities. We will further discuss how to compute Q in Sect. 4. In order to satisfy the redundancy requirements R of truth inference methods, for each task, the number of assigned workers is no less than R.

Since we prefer to assign tasks to workers with higher qualities, this paper aims to find a subset of worker set W with one task assigned to each worker, such that the total worker quality is no less than the quality bound Q_B and the overall cost is minimized. For each task, the number of assigned workers is greater or equal to the redundancy requirement R. This problem is defined as *quality bounded task assignment with redundancy constraint* (QTAR).

To solve the problem, we propose a method called *Quality Inference Based Task Assignment* (QITA). There are two processes in QITA as follows:

Quality Inference: Based on the previously aggregated sensing data, QITA adopts truth inference method to iteratively infer both the truth of tasks and workers' qualities. Workers who submit sensing data that are close to the truth will be assigned higher qualities.

Task Assignment: With worker set W, Task set T and quality set Q, QITA selects a subset $S \in W$ as the selected worker set, then assigns one task for each worker $w \in S$. After the workers finish their tasks, the aggregated sensing data, which is denoted as D, will be used for next quality inference process.

Figure 1 shows the workflow of QITA.

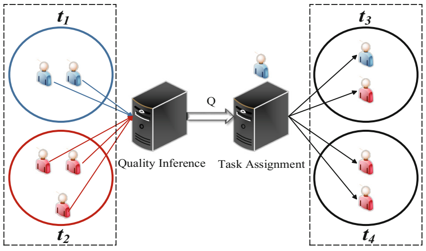


Fig. 1. The Workflow of QITA

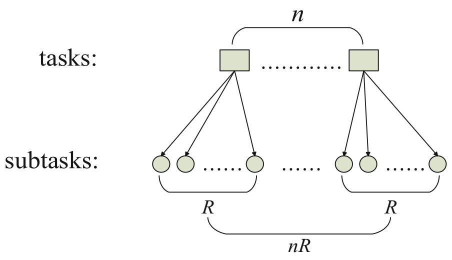


Fig. 2. An example of tasks dividing into subtasks

4 Quality Inference

Considering that the sensing data in MCS are mostly numeric numbers (e.g., WiFi signal strength, air quality index, etc.), we adopt quality inference algorithm based on truth inference method called LFC_N [5]. In this section we introduce the modeling and quality inference algorithm of this method.

For the i^{th} task $t_i \in T$, let $S_i = \{w_1^i, w_2^i, \ldots, w_{m_i}^i\}$ denote the worker set assigned to task t_i. Let $\boldsymbol{d}_j^i = \{d_{j1}^i, d_{j2}^i, \ldots, d_{j\lambda}^i\}$ be the aggregated sensing data of worker w_j^i for task t_i. Assume that d^i is the truth (real sensing data) of task

t_i, our model is that the worker provides a noisy version of the actual true value d^i. For worker w_j^i we assume a Gaussian with mean d^i (the truth) and inverse-variance (quality) q_j, that is, $\Pr(d_j^i \mid d^i, q_j) = \mathcal{N}(d_j^i \mid d^i, 1/q_j)$. To obtain the exact methods of truth inference step and quality inference step, the LFC_N method adopts a *Maximum-Likelihood Estimator* as follows:

Assume the tasks are independent, for task t_i, we suppose that a worker need to sense for λ times of data to complete this task. Let $\boldsymbol{d^i} = \{y_1^i, y_2^i, \ldots, y_\lambda^i\}$ denote the inferred truth for task t_i each time the worker senses the data. $\boldsymbol{q} = \{q_1, q_2, \ldots, q_m\}$ denotes the inferred quality of each worker. The likelihood of the parameter $\boldsymbol{\theta} = (\boldsymbol{d}^i, \boldsymbol{q})$ given the observations $\mathcal{D}_i$ can be factored as $\Pr(\mathcal{D}_i \mid \boldsymbol{\theta}) = \Pr(\boldsymbol{d}_1^i, \boldsymbol{d}_2^i, \ldots, \boldsymbol{d}_{m_i}^i \mid \boldsymbol{\theta})$. By maximizing the log-likelihood, we obtain the update equation for the inferred quality and inferred truth

$$\frac{1}{\hat{q}_j} = \frac{\sum_{k=1}^{\lambda} (d_{jk}^i - \hat{y}_k^i)^2}{\lambda} \quad \hat{y}_k^i = \frac{\sum_{j=1}^{m_i} \hat{q}_j d_{jk}^i}{\sum_{j=1}^{m_i} \hat{q}_j} \tag{1}$$

Since the two parameters $\boldsymbol{q}$ and $\boldsymbol{d}^i$ are coupled together, by using the equations in Eq. (1), we can iteratively infer the qualities and truth until convergence. Therefore we can reasonably infer the workers' qualities by adopting the truth inference based method. The inferred quality set will be used as an input of the task assignment process of QITA.

5 Quality Bounded Minimum Cost Task Assignment

5.1 QTA: An Approximation Algorithm for QTAR

The goal of our assignment is to minimize the overall cost within the quality bound and the redundancy constraint, which is formally formulated as QTAR problem in Sect. 2. Let the redundancy constraint $R = 0$ and suppose the number of workers $m = 1$, the problem can be reducted to a 0-1 knapsack problem and thus QTAR is NP-complete.

To satisfy the redundancy constraint, we divide each task t_i into R subtasks. As is shown in Fig. 2, the subtasks have the same locations with the tasks they are born from. Completing the original tasks with redundancy constraint is equivalent to assigning more than one workers to these subtasks.

This paper proposes a $(2+\epsilon)$ - approximation algorithm called QTA to solve QTAR. This algorithm successively solves two related problems, then combines the results of both problems as the final result.

Minimum Worker Matching (MWM): Assign exactly one worker to each task and the total cost is minimized.

We formulate this problem as a *Minimum Weighted Complete Mathcing* problem. Let task set T and worker set W be the disjoint sets of nodes in a bipartite graph. The edge set E where each edge has one endpoint in each of T and W denotes the assignment between workers and tasks; let $B = (T, W, E)$ denotes such a bipartite graph. If edge $(t_i, w_j) \in E$, then task t_i is assigned to worker w_j. We assume that the weight of each edge $(t_i, w_j) \in E$ is the cost C_{ij}.

By adopting a negative weight $\widetilde{C}_{ij} = -C_{ij}$ for each edge, we convert the problem into a *Maximum Weighted Complete Matching* problem, which can be solved in polynomial time by the Hungarian method (also been known as the Kuhn-Munkres algorithm or Munkres assignment algorithm).

Quality Bounded Minimum Assignment (QBMA): An assignment satisfying that the total qualities of workers exceeds the quality bound and the overall cost is minimized. Each worker can only be assigned one task.

QBMA can be reduced from 0-1 knapsack problem when $m = 1$. Due to the NP-completeness of QBMA, this paper proposes a dynamic programming based FPTAS $(1+\epsilon)$ - approximation algorithm by scaling the cost down enough such that the costs of all assignments are polynomially bounded in n. Let C_{max} denote the maximum cost among workers and tasks, the algorithm is as follows:

Algorithm 1. FPTAS Approximation Algorithm for QBMA

```
1 Given $\epsilon > 0$, let $K = \frac{\epsilon C_{max}}{n}$ ;
2 For each possible assignment $a_{ij}$, define $C'_{ij} = \lceil \frac{C_{ij}}{K} \rceil$ ;
3 Using the dynamic programming to find the minimum cost assignment $S'$
  with these as the costs of assignments ;
4 Output $S'$ ;
```

Suppose that a_{ij} represents an assignment between worker w_j and task t_i, then we introduce our approximation algorithm QTA for QTAR.

Algorithm 2. QTA: An Approximation Algorithm for QTAR

```
  Input: task set $T$, worker set $W$, quality set $Q$ and quality bound $Q_B$
  Output: assignment set $S$
1 Divide the QTAR into an MWM and a QBMA ;
2 Solve the corresponding MWM and QBMA, the output is $A_1$ and $A_2$;
3 Initially $S = \emptyset$ ;
4 foreach $1 \leq j \leq n$ do
5     foreach $1 \leq i \leq mR$ do
6         if $a_{ij} \in A_1$ then $S \leftarrow a_{ij}$ ; break ;
7         else if $a_{ij} \in A_2$ then $S \leftarrow a_{ij}$ ;
8 return S ;
```

In QTA, we divide the QTAR into an MWM and a QBMA. For the MWM, we construct a bipartite graph $B = (T, W, E_1)$. Let T be the task set of mR subtasks and W denote the worker set. The cost C_{ij} denotes the edge weight between t_i and w_j. For the QBMA, we find an assignment for task set T and worker set W. Different from QTAR, each worker can only be assigned one task while each task can be assigned to multiple or zero workers.

5.2 Algorithm Analysis

We define C_1, C_2 and C_S as the total cost of solutions of MWM, QBMA and QTAR respectively. Then we define OPT_2 and OPT_S as the total cost of optimal solutions of QBMA and QTAR. We claim that QTA is a $(2+\epsilon)$-approximation. To prove it, we first prove the following two lemmas.

We define C_1, C_2 and C_S as the total cost of solutions of MWM, QBMA and QTAR respectively. Then we define OPT_2 and OPT_S as the total cost of optimal solutions of QBMA and QTAR.

Theorem 1. *QTA is a* $(2+\epsilon)$*-approximation algorithm.*

Proof. In MWM, we find a complete match for tasks with minimum total cost C_1. Assume each task is only assigned to one worker in QTAR, the result of QTAR is at most as good as MWM, which means the total cost of QTAR can not be smaller than MWM. Thus $C_1 \leq OPT_S$.

Comparing with QTAR, we ignore the redundancy constraint in QBMA, thus the result of QTAR is at most as good as QBMA. Then we have $C_2 \leq (1+\epsilon)OPT_2 \leq (1+\epsilon)OPT_S$.

In the combining step of QTA, we discard some assignment to ensure that one worker is assigned exactly one task. The total cost decreases in this step, which means $C_S \leq C_1 + C_2$. Finally, we have

$$C_S \leq C_1 + C_2 \leq OPT_S + (1+\epsilon)OPT_S = (2+\epsilon)OPT_S,$$

which completes the proof.

6 Experiments

In this Section, we evaluate the performance of QITA by conducting a series of experiment on both synthesis data and real dataset. From the results of our experiments, the effectiveness and efficiency of QITA is proved.

6.1 Quality Inference Simulation Experiments

For each worker, we randomly generate Gaussian distribution data as the sensing data. Suppose the truth is d and the quality of worker w_j is q_j, then the generated data D_j satisfies the Gaussian distribution with d as the mean and $1/q_j$ as the variance. Initially, we have three kinds of workers: (1) bad workers with $q = 1$; (2) normal workers with $q = 10$; (3) good workers with $q = 100$. For totally 30 workers, workers with ID (1–10) are bad workers, the next 10 workers are normal workers and the last 10 workers are good workers. The three kinds fo workers should be clearly distinguished in the result of quality inference.

As is shown in the following figures, for all workers, the average error between inferred qualities and real qualities is less than 10%. Therefore, we can draw the conclusion that the quality inference is accurate and effective (Figs. 3, 4 and 5).

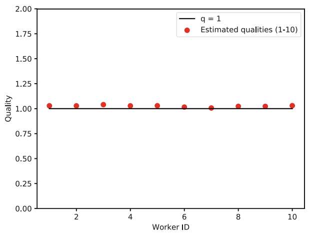


Fig. 3. Worker (1–10)

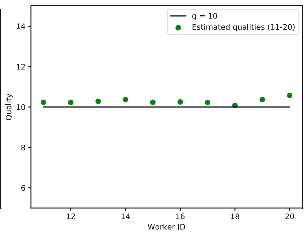


Fig. 4. Worker (11–20)

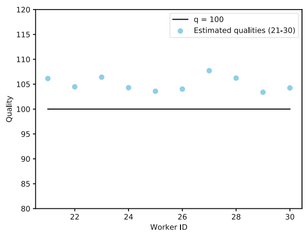


Fig. 5. Worker (21–30)

6.2 Task Assignment Experiments

We evaluate our task assignment algorithms by conducting experiments on real dataset TSMC2014 [8]. This dataset includes long-term (about 10 months) check-in data in New York city and contains 227,428 check-ins. We randomly select check-ins of different users as our worker set. The qualities of workers are generated by Gaussian Distribution with different means. Unless specified otherwise, we suppose that the positions of tasks are randomly placed in the sensing area. For each set of experiment parameter choices, we run the experiment 100 times with randomized qualities for workers and locations for tasks then task the average as the result. Figure 6 provides the evaluation results.

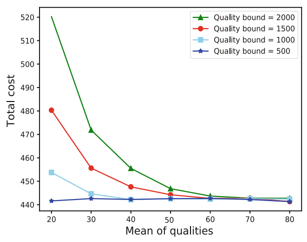


Fig. 6. Performance of QTA

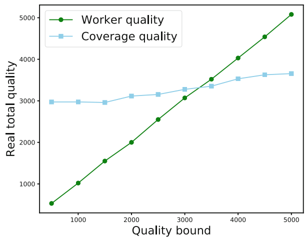


Fig. 7. Quality comparison

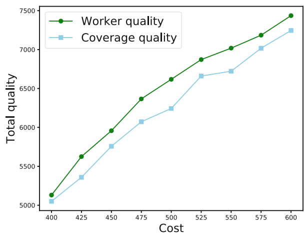


Fig. 8. Cost comparison

The total cost decreases with the increase of quality means. Higher mean of qualities means higher average value of qualities, which makes it easier for QTA to reach the quality bound. Moreover, with the decrease of quality bound, the total cost also declines due to the decreasing demand for workers.

This paper evaluates QTA by implementing a benchmark: considering the coverage quality. The coverage quality has been considered as an optimization goal to improve sensing quality in many previous researches [4,7], which is defined as the number of sensor readings in each MCS sensing cycle. The larger amount of sensor readings in MCS, the higher the coverage quality.

Figure 7 shows the comparison result of QTA and benchmark under different quality bounds. The total quality of benchmark's result becomes lower than QTA with the increase of quality bound. The comparison result under different total cost is illustrated in Fig. 8. The total quality of QTA's result is larger than benchmark when the amounts of total cost are close for both algorithms, which proves the effectiveness and efficiency of QTA.

7 Conclusion

Motivated by the truth that low quality and malicious workers may yield low sensing quality in MCS, we studied quality-aware redundancy-based task assignment problem in MCS. We first adopted truth inference methods to iteratively infer the truth and qualities based on the aggregated sensing data. By taking the quality inference result as an input, we proposed *quality bounded task assignment with redundancy constraint* (QTAR). We proved that QTAR is NP-complete and proposed $(2+\epsilon)$-approximation algorithm QTA for QTAR. By conducting experiment on both synthesis data and real dataset, we compared the performance between our algorithms and benchmarks. Experiment results showed that our algorithms is efficient and effective.

Acknowledgement. This work is supported by the program of International S&T Cooperation (2016YFE0100300), the China 973 project (2014CB340303), the National Natural Science Foundation of China (Grant number 61472252, 61672353), the Shanghai Science and Technology Fund (Grant number 17510740200), and CCF-Tencent Open Research Fund (RAGR20170114).

References

1. Dong, X.L., Berti-Équille, L., Srivastava, D.: Integrating conflicting data: the role of source dependence. PVLDB **2**(1), 550–561 (2009)
2. Ganti, R.K., Ye, F., Lei, H.: Mobile crowdsensing: current state and future challenges. IEEE Commun. Mag. **49**(11), 32–39 (2011)
3. Li, Q., et al.: A confidence-aware approach for truth discovery on long-tail data. PVLDB **8**(4), 425–436 (2014)
4. Philipp, D., Stachowiak, J., Alt, P., Dürr, F., Rothermel, K.: Drops: model-driven optimization for public sensing systems. In: PerCom, pp. 185–192 (2013)
5. Raykar, V.C., et al.: Learning from crowds. JMLR **11**, 1297–1322 (2010)
6. Wu, S., Gao, X., Wu, F., Chen, G.: A constant-factor approximation for bounded task allocation problem in crowdsourcing. In: GLOBECOM, Singapore, pp. 1–6, 4–8 December 2017
7. Xiong, H., Zhang, D., Chen, G., Wang, L., Gauthier, V.: Crowdtasker: maximizing coverage quality in piggyback crowdsensing under budget constraint. In: PerCom, pp. 55–62 (2015)
8. Yang, D., Zhang, D., Zheng, V.W., Yu, Z.: Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. IEEE Trans. Syst. Man Cybern.: Syst. **45**(1), 129–142 (2015)
9. Zheng, Y., Li, G., Li, Y., Shan, C., Cheng, R.: Truth inference in crowdsourcing: is the problem solved? PVLDB **10**(5), 541–552 (2017)

Recommendation

Expert Recommendation via Tensor Factorization with Regularizing Hierarchical Topical Relationships

Chaoran Huang[1](✉), Lina Yao[1], Xianzhi Wang[2], Boualem Benatallah[1], Shuai Zhang[1], and Manqing Dong[1]

[1] UNSW Sydney, Sydney, NSW 2052, Australia
{chaoran.huang,lina.yao}@unsw.edu.au
[2] University of Technology Sydney, Broadway, NSW 2007, Australia
sandyawang@gmail.com

Abstract. Knowledge acquisition and exchange are generally crucial yet costly for both businesses and individuals, especially when the knowledge concerns various areas. Question Answering Communities offer an opportunity for sharing knowledge at a low cost, where communities users, many of whom are domain experts, can potentially provide high-quality solutions to a given problem. In this paper, we propose a framework for finding experts across multiple collaborative networks. We employ the recent techniques of tree-guided learning (via tensor decomposition), and matrix factorization to explore user expertise from past voted posts. Tensor decomposition enables to leverage the latent expertise of users, and the posts and related tags help identify the related areas. The final result is an expertise score for every user on every knowledge area. We experiment on Stack Exchange Networks, a set of question answering websites on different topics with a huge group of users and posts. Experiments show our proposed approach produces steady and premium outputs.

Keywords: Knowledge discovery · Stack Exchange Networks
Expertise finding · Question answering

1 Introduction

Question and Answering (Q&A) websites are gaining momentum as an effective platform for knowledge sharing. These websites usually have numerous users who continuously contribute. Many researchers have shown interests in the recommendation issues on these websites such as identifying experts. Despite the tremendous research efforts on user recommendation, no state-of-the-art algorithms consistently stand out compared with the others. As the recent work increasingly focuses on domain-specific expertise recommendation, there emerges the research on multi-domain (or cross-domain) recommendation in the "Stack Exchange (SE) Networks"[1] repository. SE is a network of 98 Q&A subsites, all

[1] stackexchange.com.

following the same structure. This consistency enables us to expand our approach from one subsites to the all the other subsites on SE. These subsites cover various disciplines from computer science to even the Ukrainian language. Take "Stack Overflow"[2] (SO) as an example (Fig. 4). It is a software-domain-oriented website where users can post and answer questions, or vote up/down to other users' questions and answers. The author of a question (a.k.a., the requester) can mark an answer as accepted and offer a bounty to the answerer.

So far, there are two popular ways to locate experts: collaborative filtering (CF) and content-based recommendation. The former extracts similar people without understanding the contents while the latter focuses on building user profiles based on users' activity history. CF relies merely on ratings (e.g., scores in SE networks) and therefore may not well handle sparse Q&A subsites data, where many questions involve very limited users. Usually, users can vote on questions, and the vote counts can serve as ratings to the questions. An earlier work [1] also suggests that the lack of information can be a challenge for recommendation techniques. The work aims to address the data sparsity issue by selectively using the ratings of some experts. This experts presumed by this approach is exactly the same experts we aim to find. As for content-based approaches, a typical approach (e.g., [17]) builds user profiles based on user's knowledge scores and user authority in link analysis. The knowledge scores are called reputation in [17], which is derived from users' historical question-answering records. Srba et al. [22] point out that some users may maliciously post low-quality content, and those highly active spammers might be taken as experts in a system. Huna et al. [11] solve this problem by calculating question and answer difficulties based on three aspects of hints: the numbers of user-owned questions and answers, time difference of the question being posted and answered, average answering time, and score of the answer with the maximum of score among all the answers provided by the answerer. Although these approach may compute user reputation, they also take considerable cost on building user profiles. Matrix Factorization is one method that works on sparse data , while matrices can only store two dimensions of data, which is not handy in many applications, where users' attributes can be vital to the identification of experts. Recently tensor-based approaches became popular as an alternative to matrix factorization, made it feasible to handle multi-faceted data [27]. For example, Ge et al. in [7] decompose a (Users, Topics, Experts) tensor for the personalized expert recommendation; Bhargave et al. [3] propose a (User, Location, Activity, Time) tensor decomposition along with correlated matrix to make recommendations based on user preferences.

We aim to recommend experts in multiple areas simultaneously (Fig. 1). In particular, we use the Stack Exchange networks dump, which contains various areas, to build up a multi-domain dataset. We propose group lasso [15] that works on a relationship tree formed upon the natural structure of the SE network. The tree is used to guide the decomposition of 4th rank tensor data consisting of questions, topics, voting and expertise information. We additionally factorize selected matrices to provide additional latent information.

[2] stackoverflow.com.

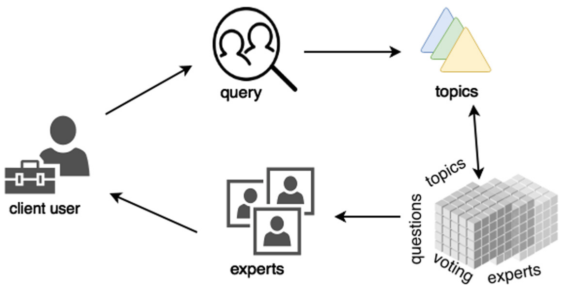


Fig. 1. Work-flow of our proposed methodology: for a given input query, experts are output based on the detected topic of the query combined with our 4th order tensor, which contains latent information like topics, questions, voting, and experts.

Our contributions in this work are as follows:

1. We take the hierarchical relationship between participants and topics into account and build a model that combines tree-guided tensor decomposition and matrix factorization;
2. We introduce the relationship tree group lasso to alleviate the data sparsity problem;
3. We conduct experiments on real-world data and evaluate the proposed approach against state-of-the-art baselines.

2 Related Works

Expert recommendation has been studied extensively in the past decade. Generally, skillfulness and resourcefulness of experts can assist users in making decisions more professionally and solving problems more effectively and efficiently. That is, making appropriate recommendations to users with the different requirement can be important.

The expert recommendation techniques apply to many areas, and different fields may require differently in methodologies to handle different situations. Baloga et al. [2] introduce a generative probabilistic framework for find experts in various enterprise data sources. Daud et al. [4] devise a Temporal-Expert-Topic model to capture both the semantic and dynamic expert information and to identify experts for different time periods. Fazelzarandi et al. [6] develop an expert recommendation system with utilizing the social networks analysis and multiple data source integration techniques. Wang et al. [23] propose a model ExpertRank which take both document profile and authority of experts into consideration to perform better. Huang et al. [10] take advantage of word embedding technology to rank experts both semantically and numerically. More relate works can be found in a survey by Wang et al. [24].

The works mentioned above mostly focus on recommend experts for organizations, enterprises or institutes. There is also some literature on recommending experts in Q&A System, which is more related to our work. Kao et al. [13] propose to incorporate user subject relevance, user reputation and authority of categories into expert finding system in Q&A websites. Riahi et al. [21] investigate two topic model namely Segmented Topic Model and Latent Dirichlet Allocation model to direct new questions in Stack-overflow to related experts. Ge et al. [7] propose a personalized tensor-based method for expert recommendation by considering factors like geospatial, topical and preferences. Liu et al. in [18] propose a method to rank user authority by exploiting interactions between users, which is aimed to avoid potential impacts of users with considerable social influences. They introduced topical similarities into link analysis to rank user authorities for each question. Latent Dirichlet allocation is applied to extract topics from both the questions and answers of users so that topical similarities between questions and answers can be measured, and then related users can be ranked by links. Huna et al. found Q&A communities often evaluate user reputation limited to the number of user activities [11], regardless of efforts on creating high-quality contents. This causes inaccurate measurements in user expertise and their value. Inspired by former works, they calculate user reputations for asking and answering questions. The reputation results from the combination of the difficulty score of a question and the utility score for the question or answer. A utility score measures the distance between a score and the maximum score of the post, and the difficulty measures the times that a user spends on the question. The time spent on questions is normalized on each topic. Fang et al. [5] are well aware of the quantity of social information Q&A website can provide, along with the importance of user-generated textual contents. Their idea to simultaneously model both social links and textual contents leads to the proposed framework named "HSNL" (CQA via **H**eterogeneous **S**ocial **N**etwork **L**earning). The framework adopts random walk to exploit social information and build the heterogeneous social network, and a deep recurrent neural network was trained to give a text-based matching score for questions and answers.

Our proposed model builds on tensor decomposition, which has been applied to various fields such as neuroscience, computer vision, and data mining [16]. CANDECOMP/PARAFAC (CP) and Tucker decomposition are two effective ways to solve tensor decomposition problems. We adopt the former in this work. Tensor decomposition based recommender systems can also be found widespread in recent studies. Rendle et al. [19] introduce a tensor factorization based ranking approach for tag recommendation. They further improve the model by introducing pairwise interaction and significantly improve the optimization efficiency. Xiong et al. [25] propose a probabilistic tensor decomposition model and regard the temporal dynamics as the third-dimension of the tensor. Karatzoglou et al. [14] offer a context-aware tensor decomposition model to integrate context information with collaborative filtering tightly. Hidas et al. [9] investigate approach which combines implicit feedback with context-aware decomposition. Bhargava et al. [3] present a tensor decomposition-based approach to model the influence of multi-dimensional data sources. Yao et al. [26] decompose tensor with contextual regularization to recommend location points of interest.

3 Methodology

CANDECOMP/PARAFAC Tensor Decomposition, or CP Decomposition, is discovered by Kiers and Möcks independently [16]. For a Rank-R size-N tensor $\mathcal{X}$ ($R \in \mathbb{N}$), let $U_1 \in \mathbb{R}^{I_1 \times R}, U_2 \in \mathbb{R}^{I_2 \times R}, ..., U_R \in \mathbb{R}^{I_N \times R}$, we have the decomposition:

$$\mathcal{X} \approx \sum_{r=1}^{R} U_{1 i_1 r} U_{2 i_2 r} \cdots U_{R i_N r} \tag{1}$$

While multiple methods can do tensor decomposition, the most common and effective one shall be the alternating least squares (ALS) [16].

3.1 Relationship Tree Modelling

Our data is naturally divided into subsites, topics, and posts, as shown in Fig. 3. This decomposition forms a tree, with subsites on top, and posts as leaves. As our tensor models the expertise information based on user activities, this tree reserves the relationships of entities. We illustarte the construction of the tree as follows (Fig. 2).

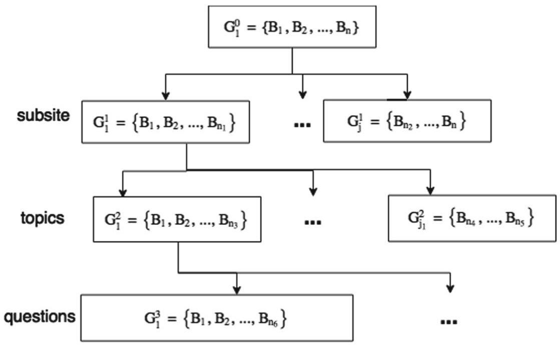

$G_1^0 = \{B_1, B_2, ..., B_n\}$
subsite
$G_1^1 = \{B_1, B_2, ..., B_{n_1}\}$
...
$G_j^1 = \{B_{n_2}, ..., B_n\}$
topics
$G_1^2 = \{B_1, B_2, ..., B_{n_3}\}$
...
$G_{j_1}^2 = \{B_{n_4}, ..., B_{n_5}\}$
questions
$G_1^3 = \{B_1, B_2, ..., B_{n_6}\}$
...

Fig. 2. An example of modeled tree representation of hierarchical relationship

Given the tree $\mathcal{T}$, we assume that the i-th level of $\mathcal{T}$ has n_i nodes and organized as $\mathcal{T}_i = \{G_1^i, G_2^i, ..., G_{n_i}^i\}$. And so, a group G_v where node $v \in V$ is in the tree, and all leaves under v are in G_v. Now we can define a tree-structured regulation as

$$Weight(\mathbf{U}_1) = \frac{\lambda_W}{2} \sum_{k=1}^{J} \omega_j^i \|\mathbf{U}_{1_k}\|_2^2 \quad (\mathbf{U}_{1_k} \in G_j^i) \tag{2}$$

This inspired from Moreau-Yosida regularization, and here λ_W is the Moreau-Yoshida regulation parameter for tree $\mathcal{T}$, $\|\cdot\|$ denotes Euclide an norm, $\mathbf{U}_{1_k}$ is a vector of $\mathbf{U}_1$, where $\mathbf{U}_1$ is the first factor matrix of the tensor $\mathcal{X}$, which corresponding to a question post and detailed explaination can be found in the following subsection. Additionally, ω_j^i is set by following Kim's approach [15] and it means a pre-set weight for j-th node at level i. ω_j^i can be obtained by setting two variables summed up to 1, i.e. s_j^i for the weight of independent relevant covariates selecting and g_j^i for group relevant covariates selecting. We have:

$$\sum_i^d \sum_j^n \omega_i^j \|\mathbf{U}_{1_{G_j^i}}\|_2 = \lambda \omega_0^j \tag{3}$$

where

$$\omega_i^j = \begin{cases} s_j^i \cdot \sum_{c_p^q \in \text{Child}(v_j^i)} |\omega_p^q| + g_j^i \cdot \|\mathbf{U}_{1_{G_j^i}}\|_2 & v_j^i \text{ is a internal node,} \\ |\mathbf{U}_{1_{G_j^i}}| & v_j^i \text{ is a leaf node.} \end{cases} \tag{4}$$

3.2 Proposed Model

Our dataset is obtained naturally categorized by their subdomains, which we call it "subsites" here. Additionally, in each subsite, we can find tags in every post, and such information is often an indicator of the post's topics. Accordingly, after gathering those data, we can build a tree to represent such hierarchical information (shown in Fig. 3).

All Stack Exchange subsites share the same structure. That means, in all this subsites, answerers may propose multiple answers and questioners can adopt only one answer for each question. Also, both question and answers can be commented and voted, and the difference between vote-ups or vote-downs on each question is calculated into a score. Figure 4 shows an example.

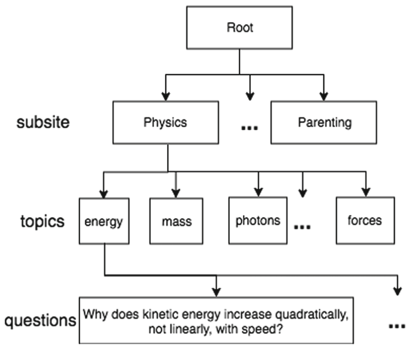
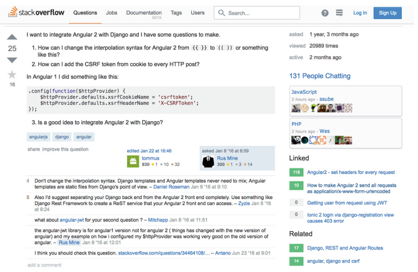


Fig. 3. Tree representation of hierarchical entity relationship



25

16

I want to integrate Angular 2 with Django and I have some questions to make.

1. How can I change the interpolation syntax for Angular 2 from `{{ }}` to `(( ))` or something like this?
2. How can I add the CSRF token from cookie to every HTTP post?

In Angular 1 I did something like this:

```
.config(function($httpProvider) {
    $httpProvider.defaults.xsrfCookieName = 'csrftoken';
    $httpProvider.defaults.xsrfHeaderName = 'X-CSRFToken';
});
```

3. Is a good idea to integrate Angular 2 with Django?

angularjs django angular

share improve this question

edited Jan 22 at 16:46
tommus
930 ● 1 ● 10 ● 32

asked Jan 8 '16 at 8:59
Rus Mine
300 ● 1 ● 3 ● 14

4 Don't change the interpolation syntax. Django templates and Angular templates never need to mix; Angular templates are static files from Django's point of view. – Daniel Roseman Jan 8 '16 at 9:10

5 Also I'd suggest separating your Django back end from the Angular 2 front end completely. Use something like Django Rest Framework to create a ReST service that your Angular 2 front end can access. – Zyzle Jan 8 '16 at 9:24

what about angular-jwt for your second question ? – Mitchapp Jan 8 '16 at 11:51

the angular-jwt library is for angular1 version not for angular 2 (things has changed with the new version of angular) and my example on how i configured my $httpProvider was working very good on the old version of angular. – Rus Mine Jan 8 '16 at 12:01

I think you should check this question. stackoverflow.com/questions/34464108/... – Antano Jun 23 '16 at 9:01

add a comment

asked 1 year, 3 months ago
viewed 20989 times
active 2 months ago

131 People Chatting

JavaScript
2 hours ago - ssube

PHP
2 hours ago - Wes

Linked

118 Angular2 - set headers for every request
10 How to make Angular 2 send all requests as application/x-www-form-urlencoded
0 Getting user from request using JWT
0 Ionic 2 login via django-registration view causes 403 error

Related

17 Django, REST and Angular Routes
14 angular, django and csrf

Fig. 4. An example of Stack Overflow post (postId:34672987), here demonstrates a question with its description and comments, along with score of the question.

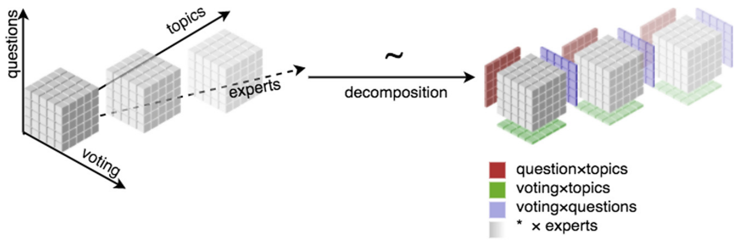


Fig. 5. Proposed decomposition

Instead of the simple score-user matrix based recommendation, we propose a tensor-decomposition based tree-guided method, based on the basic idea of Tree-Guided Sparse Learning [12].

1. A 4th-order-tensor, **Question** $\times$ **Topic** $\times$ **Voting** $\times$ **Expert**. Shown in Fig. 5, we denoted it as $\mathcal{X} \in \mathbb{R}^{I \times J \times K \times L}$, where I is the number of questions, J is the number of Topics, K is the number of voting of question towards questioners, L is the expert users and the value of the tensor is the number of expertise evaluation criterion. With limited users participated in certain domains, it is believed that the tensor is very sparse. Additionally we denote $\mathbf{U_1} \in \mathbb{R}^{I \times R}, \mathbf{U_2} \in \mathbb{R}^{J \times R}, \mathbf{U_3} \in \mathbb{R}^{X \times R}, \mathbf{U_4} \in \mathbb{R}^{L \times R}$ as factor matrices of tensor $\mathcal{X}$.
2. A **subsite** $\times$ **answerer** matrix. We denoted this as $M \in \mathbb{R}^{X \times Z}$, where if answerer z appears in subsite x, $M_{x,z} = 1$ else $M_{x,z} = 0$.
3. A **topics** $\times$ **answerer** matrix. We denoted this as $N \in \mathbb{R}^{Y \times Z}$, similarly here, when answerer z appears in topic y, $M_{y,z} = 1$ else $M_{y,z} = 0$.
4. Hierarchical relationship tree $\mathcal{T}$ of depth d. Due to the isolation of subsites and their topics, our data show clearly a structured sparsity. Thus, we can utilize tree-guided group lasso in our model. That is, besides above two supplement matrices, we also use the tree shown in Fig. 3 to guide the learning.

After modeling the data, we apply CANDECOMP/PARAFAC (CP) tensor decomposition to factorize the tensor and solve the tree-structured regression with group lasso (Table 1).

First, we decompose the 4th-order tensor with regulation by Alternating Least Square (ALS) as follows:

$$\begin{aligned} Tensor(\mathbf{U_1}, \mathbf{U_2}, \mathbf{U_3}, \mathbf{U_4}) = \frac{1}{2}\|\mathcal{X} - [\![\mathbf{U_1}, \mathbf{U_2}, \mathbf{U_3}, \mathbf{U_4}]\!]\|_F^2 \\ + \frac{\lambda_{\mathcal{X}}}{2}(\|\mathbf{U_1}\|_F^2 + \|\mathbf{U_2}\|_F^2 + \|\mathbf{U_3}\|_F^2 + \|\mathbf{U_4}\|_F^2) \end{aligned} \quad (5)$$

Then, we can have the aforementioned 2 matrices decompose as:

$$Networks(\mathbf{S}, \mathbf{A}) = \frac{1}{2}\|\mathbf{M}_{site} - \mathbf{S}\mathbf{A}^T\|_F^2 + \frac{\lambda_S}{2}(\|\mathbf{S}\|_F^2 + \|\mathbf{A}\|_F^2) \quad (6)$$

Table 1. Symbol table

Symbol	Description
$\mathcal{X} \in \mathbb{R}^{I\times J\times K\times L}$	A 4th-order-tensor, I, J, J, L accordingly is the number of **Question**, **Topic**, **Voting** and **Expert**
$\mathbf{U_1} \in \mathbb{R}^{I\times R}$ $\mathbf{U_2} \in \mathbb{R}^{J\times R}$ $\mathbf{U_3} \in \mathbb{R}^{X\times R}$ $\mathbf{U_4} \in \mathbb{R}^{L\times R}$	Factor matrices of tensor $\mathcal{X}$
$M \in \mathbb{R}^{X\times Z}$	**subsite** $\times$ **answerer** matrix where X, Z are the number of **subsite** and **answerer**
$N \in \mathbb{R}^{Y\times Z}$	**topic** $\times$ **answerer** matrix where Y are the number of **topic**
$\mathcal{T}_i = \{G_1^i, G_2^i, ..., G_{n_i}^i\}$	Set of node in the i-th level of tree $\mathcal{T}$ G_{n_i} is the n_i-th node in the level

$$Topic(\mathbf{T}, \mathbf{A}) = \frac{1}{2}\|\mathbf{M}_{topic} - \mathbf{T}\mathbf{A}^T\|_F^2 + \frac{\lambda_T}{2}(\|\mathbf{T}\|_F^2 + \|\mathbf{A}\|_F^2) \tag{7}$$

Since each subsite S_j contains a group of questions $\mathbf{U}_{1_j}$, we expect S_j to be similar to the average $\mathbf{U}_{1_j}$, which can be solved as a regulation:

$$Site(\mathbf{S}, \mathbf{U_1}) = \frac{\lambda_S}{2}\sum_{j=1}^{U}\|\mathbf{S}_j - \frac{1}{G_j^1}\sum_{\mathbf{U}_{1_k}\in G_j^1}\mathbf{U}_{1_k}\|_2^2 \tag{8}$$

By combining those objectives and regulations, we have the following objective function:

$$\begin{aligned} f(\mathbf{U_1}, \mathbf{U_2}, \mathbf{U_3}, \mathbf{U_4}, \mathbf{S}, \mathbf{A}, \mathbf{T}) = &\, Tensor(\mathbf{U_1}, \mathbf{U_2}, \mathbf{U_3}, \mathbf{U_4}) \\ &+ Weight(\mathbf{U}_1) + Networks(\mathbf{S}, \mathbf{A}) \\ &+ Topic(\mathbf{T}, \mathbf{A}) + Site(\mathbf{S}, \mathbf{U_2}) \end{aligned} \tag{9}$$

Equation 5 follows the CANDECOMP/PARAFAC Decomposition, accomplished by the ALS algorithm (see Algorithm 1), which is a popular way to decompose a tensor.

Computational Complexity Analysis. The time complexity of the above decomposition includes two parts. The first concerns initializing the set of $\mathbf{A}^{(n)}$s. We note the average of the dimension of our tensor as D, which we use to represent the size of the tensor as $\mathbf{D}^N$. The initialization is a traverse of $\mathbf{A}^{(n)}$s and has a time complexity of $\mathcal{O}(NDR)$. Assuming that we use index flip to implement the matrix transpose, its time complexity is $\mathcal{O}(1)$. Thus, the total time complexity on N loops is $\mathcal{O}((NDR)^2 + N^2DR)$ time. Combining the two steps, we now have the time complexity of the algorithm as $\mathcal{O}((NDR)^2)$.

Algorithm 1. CP Decomposition via Alternating Least Squares, where N-th order tensor $\mathcal{X}$ of size $I_1 \times I_2 \times ... \times I_N$ is decomposite into R components

```
Input: $\mathcal{X}, R$
Output: $\lambda, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(n)}$

    Algorithm $CP - ALS(\mathcal{X}, R)$
    initialize $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ for $n = 1, 2, ..., N$
 1: for $n = 1, 2, ..., N$ do
 2:     $\mathbf{V} \leftarrow \mathbf{A}^{(1)\top}\mathbf{A}^{(1)} * ... * \mathbf{A}^{(n-1)\top}\mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)\top}\mathbf{A}^{(n+1)} * ... * \mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$
 3:     $\mathbf{A}^{(n)} \leftarrow \mathbf{X}^{n}(\mathbf{A}^{(n)} \odot ... \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot ... \odot \mathbf{A}^{(1)})\mathbf{V}^{*}$
 4:     normalize columns of $\mathbf{A}^{(\mathbf{n})}$ and store norms as $\lambda$
 5:     if fit stops improve or iteration reach threshold then
 6:         break
 7:     end if
 8: end for
 9: return $\lambda, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, ..., \mathbf{A}^{(n)}$
```

4 Experiments and Evaluation

In this section, we report our experiments to evaluate our proposed approach. We first briefly introduce our dataset and the evaluation metrics, and then present the results analysis and evaluation.

Until now, there is no “gold standard” to evaluate our approach regarding expert recommendation, to the best of our knowledge. Also, it is difficult to judgment user’s expertise manually due to the large-scale data (e.g., our test data contains more than 2 million users and nearly 20 million voting activities on 5 million posts) and the lack of ranking information in the dataset—the reputation scores of users in Stack Exchange systems are computed globally, which cannot be utilized to evaluate individual’s ability in specific domains or topics.

Similar to Huna et al. [11], we calculate the reputation score of each user by topics, according to the rules adopted by Stack Exchange[3]. We simplify the rule by removing bounty-related and edition-related reputation differences. Table 2 summarizes the simplification results. A rank can be established based on the

Table 2. Adopetd reputation rules

Activity	Reputation gaines
Answer is upvoted	+10
Question is upvoted	+5
Answer/question is downvoted	−2
Downvote an answer	−1
Answer is accepted	+15

[3] https://stackoverflow.com/help/whats-reputation.

built-in reputation scores of users, following the approach proposed by Huna et al. [11]. The rank serves as a baseline for comparative performance evaluation. Given the lack of a standard to measure verifiable expertise of users, we adopt this idea and conduct comparison experiments.

4.1 Dataset and Experiment Settings

Dataset. As mentioned above, the Stack Exchange Networks includes 98 subsites and massive data. We identified 14,220,976 users, 46,575,393 posts, 178,575 tags, and 178,184,014 votes. Computing at such a scale can be challenging to any existing systems. Thus, in this work, we conducted experiments on several reasonably selected subsets, which contains a feasible yet still decent volume of data.

Table 3. Selected statistics profiles of experiment dataset

	# of Users	# of Posts	# of Tags	# of Votes
apple	153360	202239	1048	720540
askUbuntu	420227	598530	3022	2543467
gis	63977	179507	2221	573263
math	315792	1807772	1518	6046107
physics	95485	234583	876	1055850
serverFault	302850	645711	3514	2048746
stat	111974	195038	1331	782689
superuser	500264	859690	5190	3281616
unix	188934	284114	2438	1276409

Note that, our method is a tree-guided tensor decomposition approach, where the tree models the hierarchical entity relationships including topics information. To keep the variance of the topics, we generate our testing subsets from several independent subsites. These subsites are named as *"apple", "math", "stats", "askubuntu", "physics", "superuser", "gis", "serverfault", and "unix"*. Some selected statistics profiles can be found at Table 3.

Due to the massive scale of our data source and its high degree of sparseness, a random sampling could end up output posts with an enormous number of unrelated users and topics. Hence, we first sample randomly to select a subset of users and then enumerations on posts tags and voting are performed. This ensures the selected posts and votes are all related to the sampled users.

4.2 Results Analysis and Evaluation

Evaluation Metrics

– **Precision@k.** Precision@k is one of standard evaluation metrics in information retrieval tasks and recommender systems. It is defined to calculate the proportion of retrieved items in the top-k set that are relevant. Here our frameworks return a list of users so that the Precision@k can be calculated as follows:

$$P@k = \frac{|\{relevant\_top - k\_users\} \cap \{retrived\_top - k\_users\}|}{|\{retrived\_top - k\_users\}|}$$

– **MRR.** The Mean Reciprocal Rank is a statistic measure for evaluating response orderly to a list, which here is average of reciprocal ranks for all tested questions:

$$MRR = \frac{1}{|Q|} \sum i = 1|Q| \frac{1}{Rank_i}$$

Compared Methods

– **Baselines.** Apart from the reputation value calculated by Stack Exchange rules mentioned earlier in Table 2, it also can be found that some baselines are also often used apart from reputation value. Namely, lists generated by rank by "Best Answer Ratio" of users and rank by "Number of Answers" produced by users.
– **MF-BPR** [20]**.** Rendel et al. introduce pairwise BPR ranking loss into standard Matrix Factorization models. It is specifically designed to optimize ranking problems.
– **Zhang et al.** [28], Z-Score by Zhang et al., is a well-known reputation measure, despite their original work is a PageRank based system and is not aimed at measurements. This feature-based score can be resolved by q the number of questions a user asked and a, the number of answers the user posted. That is,

$$Z - Score = \frac{a - q}{\sqrt{a + q}}$$

– **ConvNCF** [8]**.** Outer Product-based Neural Collaborative Filtering, a multi-layer neural network architecture based collaborative filtering method. it use an outer product to find out the pairwise correlations between the dimensions of the embedding space.

Results Analysis. Figure 6 shows the evaluation results with respect to the Precision and MRR of different methods, where precision measures the ability to find experts and MRR the performance of outputting list of experts in correct order. We observed that our approach generally outperformed other tested approaches, although some other approaches produces more accurate list when the length of the requested list is no more than 3, and this can be claimed less likely to be practical. Our approach yielded better ranks in most cases except some case where very short lists were requested. Yet, It can be argued, in real life applications, a the list of approximately 10 or more experts is largely sensible and our approach will have substantial better performance. Also interestingly, here we can see both precision and MRR decreases by the increase of K, which

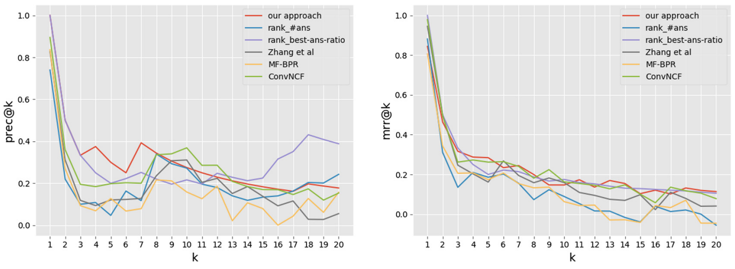


Fig. 6. Preformance comparison of our approach to others, tested with 250 users and their historical data

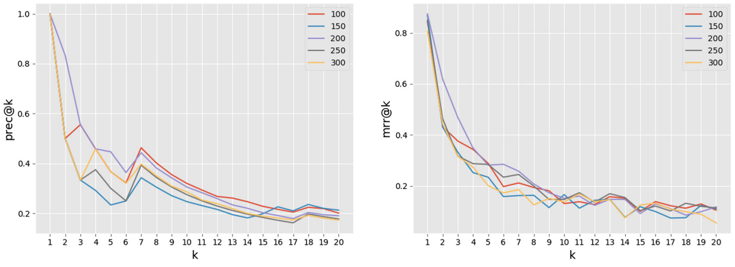


Fig. 7. Precision and MRR of tests at various number of users

differs from our experience of previous work. And a further look at the distribution of reputation in our tested data reveals it actually sensible, as we can see in Fig. 8, the distribution of users' reputation is considerably uneven, given very few people high have reputation, which are our goal of output, and most people in the dataset are reputed at value 1. Additionally, to assess the stability of our approach, we conducted tests with various size of input data, ranging from 100 users to 300 users. Besides acceptable fluctuations, the results demonstrate our approach performs relatively stable, both in accuracy and quality (Fig. 7).

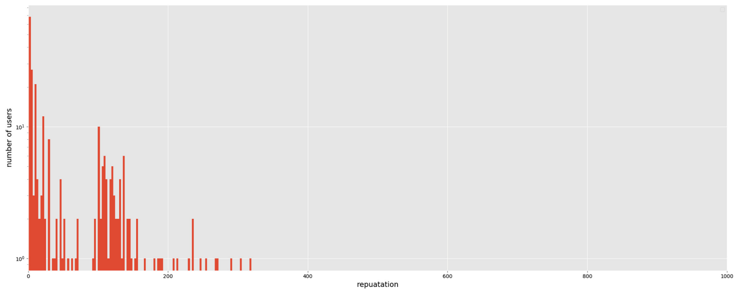


Fig. 8. Distribution of reputation of users in our dataset

5 Conclusion

In this paper, we have proposed a framework to identify experts across different collaborative networks. The framework use tree-guided tensor decomposition to exploit insights from Q&A networks. In particular, we decomposite a 4th rank tensor with tree-guided lasso and matrix factorization to exploit the topic information from a collection of Q&A websites in Stack Exchange Networks to alleviate the data sparsity issue. The 4th rank tensor model of the data ensures to keep as much as information as needed, which confirmed by experiments and evaluation. Due to the lack of "Gold Standard", we compared our approach with baselines accordingly to the rank by the reputation score calculated by Stack Exchange built-in approaches on each topic. The comparison results demonstrate the feasibility of our approach. The proposed approach can be applied to broader scenarios such as finding the most appropriate person to consult on some specific problems for individuals, or identifying the desired employees for enterprises.

Acknowledgment. This research was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI), which is supported by the Australian Government.

References

1. Amatriain, X., Lathia, N., Pujol, J.M., Kwak, H., Oliver, N.: The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 532–539. ACM (2009)
2. Balog, K., Azzopardi, L., de Rijke, M.: A language modeling framework for expert finding. Inf. Process. Manag. **45**(1), 1–19 (2009)
3. Bhargava, P., Phan, T., Zhou, J., Lee, J.: Who, what, when, and where: multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In: Proceedings of the 24th International Conference on World Wide Web, pp. 130–140. ACM (2015)
4. Daud, A., Li, J., Zhou, L., Muhammad, F.: Temporal expert finding through generalized time topic modeling. Knowl.-Based Syst. **23**(6), 615–625 (2010)
5. Fang, H., Wu, F., Zhao, Z., Duan, X., Zhuang, Y., Ester, M.: Community-based question answering via heterogeneous social network learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
6. Fazel-Zarandi, M., Devlin, H.J., Huang, Y., Contractor, N.: Expert recommendation based on social drivers, social network analysis, and semantic data representation. In: Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems, pp. 41–48. ACM (2011)
7. Ge, H., Caverlee, J., Lu, H.: Taper: a contextual tensor-based approach for personalized expert recommendation. In: Proceedings of RecSys (2016)
8. He, X., Du, X., Wang, X., Tian, F., Tang, J., Chua, T.S.: Outer product-based neural collaborative filtering (2018)
9. Hidasi, B., Tikk, D.: Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012. LNCS (LNAI), vol. 7524, pp. 67–82. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33486-3_5

10. Huang, C., Yao, L., Wang, X., Benatallah, B., Sheng, Q.Z.: Expert as a service: software expert recommendation via knowledge domain embeddings in stack overflow. In: 2017 IEEE International Conference on Web Services (ICWS). pp. 317–324, June 2017. https://doi.org/10.1109/ICWS.2017.122
11. Huna, A., Srba, I., Bielikova, M.: Exploiting content quality and question difficulty in CQA reputation systems. In: Wierzbicki, A., Brandes, U., Schweitzer, F., Pedreschi, D. (eds.) NetSci-X 2016. LNCS, vol. 9564, pp. 68–81. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28361-6_6
12. Jenatton, R., Mairal, J., Bach, F.R., Obozinski, G.R.: Proximal methods for sparse hierarchical dictionary learning. In: Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 487–494 (2010)
13. Kao, W.C., Liu, D.R., Wang, S.W.: Expert finding in question-answering websites: a novel hybrid approach. In: Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 867–871. ACM (2010)
14. Karatzoglou, A., Amatriain, X., Baltrunas, L., Oliver, N.: Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In: Proceedings of the Fourth ACM Conference on Recommender Systems, pp. 79–86. ACM (2010)
15. Kim, S., Xing, E.P.: Tree-guided group lasso for multi-task regression with structured sparsity. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML 2010, pp. 543–550. Omnipress, USA (2010). http://dl.acm.org/citation.cfm?id=3104322.3104392
16. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
17. Liu, D.R., Chen, Y.H., Kao, W.C., Wang, H.W.: Integrating expert profile, reputation and link analysis for expert finding in question-answering websites. Inf. Process. Manage. **49**(1), 312–329 (2013). https://doi.org/10.1016/j.ipm.2012.07.002
18. Liu, X., Ye, S., Li, X., Luo, Y., Rao, Y.: ZhihuRank: a topic-sensitive expert finding algorithm in community question answering websites. In: Li, F.W.B., Klamma, R., Laanpere, M., Zhang, J., Manjón, B.F., Lau, R.W.H. (eds.) ICWL 2015. LNCS, vol. 9412, pp. 165–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25515-6_15
19. Rendle, S., Balby Marinho, L., Nanopoulos, A., Schmidt-Thieme, L.: Learning optimal ranking with tensor factorization for tag recommendation. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 727–736. ACM (2009)
20. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence, pp. 452–461. AUAI Press (2009)
21. Riahi, F., Zolaktaf, Z., Shafiei, M., Milios, E.: Finding expert users in community question answering. In: Proceedings of the 21st International Conference on World Wide Web, pp. 791–798. ACM (2012)
22. Srba, I., Bielikova, M.: Why is stack overflow failing? Preserving sustainability in community question answering. IEEE Softw. **33**(4), 80–89 (2016)
23. Wang, G.A., Jiao, J., Abrahams, A.S., Fan, W., Zhang, Z.: ExpertRank: a topic-aware expert finding algorithm for online knowledge communities. Decis. Support. Syst. **54**(3), 1442–1451 (2013)

24. Wang, X., Huang, C., Yao, L., Benatallah, B., Dong, M.: A survey on expert recommendation in community question answering. J. Comput. Sci. Technol. **33**(4), 625–653 (2018)
25. Xiong, L., Chen, X., Huang, T.K., Schneider, J., Carbonell, J.G.: Temporal collaborative filtering with Bayesian probabilistic tensor factorization. In: Proceedings of the 2010 SIAM International Conference on Data Mining, pp. 211–222. SIAM (2010)
26. Yao, L., Sheng, Q.Z., Qin, Y., Wang, X., Shemshadi, A., He, Q.: Context-aware point-of-interest recommendation using tensor factorization with social regularization. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2015, pp. 1007–1010. ACM, New York (2015). http://doi.acm.org/10.1145/2766462.2767794
27. Yao, L., Sheng, Q.Z., Wang, X., Zhang, W.E., Qin, Y.: Collaborative location recommendation by integrating multi-dimensional contextual information. ACM Trans. Internet Technol. (TOIT) **18**(3), 32 (2018)
28. Zhang, J., Ackerman, M.S., Adamic, L.: Expertise networks in online communities: structure and algorithms. In: Proceedings of the 16th International Conference on World Wide Web, pp. 221–230. ACM (2007)

Software Service Recommendation Base on Collaborative Filtering Neural Network Model

Liang Chen(✉), Angyu Zheng, Yinglan Feng, Fenfang Xie, and Zibin Zheng

School of Data and Computer Science,
National Engineering Research Center of Digital Life, Sun Yat-sen University,
Guangzhou, China
{chenliang6,zhzibin}@mail.sysu.edu.cn,
{xieff5,fengylan6}@mail2.sysu.edu.cn, zanyu17@gmail.com

Abstract. With broad application of Web service technology, many users look for applicable Web services to construct their target application quickly or do some further research. Github, as a treasury including a variety of software programs, provides functional code modules for those in need, which has become their characteristic service. However, tremendous Web services have been developed all the time which increase the difficulty to find the target or interested services for users. Service recommendation has become of practical importance. There is few studies in the personalized repository recommendation of Github. In this paper, we present a general framework of PNCF, a preference-based neural collaborative filtering recommender model, and develop the instantiation of PNCF framework in Github repository recommendation with language preference called LR-PNCF. We use a neural network to capture the non-linear user-repository relationships and obtain abstract data representation from sparse vectors. Comprehensive experiments conducted on a real world dataset demonstrate the effectiveness of the proposed approach.

Keywords: Neural network · Recommender system
Github recommendation · Repository recommendation

1 Introduction

In our daily life, various services (e.g., Web service, API, mobile Apps) are commonly used in all walks of life and continue to grow enormously. They exist in various software systems, called by users to meet their needs. Specifically, open source software repositories in social coding sites could be treated as one kind of service provided by developers. Users explore functional code modules in the massive software service to build their web application. In this paper, we will focus on the research of this special service, i.e., software service.

As is widely known to programmers, Github[1] is a large-scale open source hosted platform popular with different developers from all over the world. Plentiful users explore particular code service and use it in accelerating construction of their complex application. As of April 2017, GitHub reports having almost 20 million users and 57 million code programs [6], making it the largest host of source code service in the world [7]. These large-scale software services have undoubtedly increased the difficulty of searching target code service for users. However, there is not any personalized repository recommender for users. They can only find the software service they are interested in by browsing popular repositories or by following other friends. Thus, service recommendation has become of practical importance.

To better offer users the software service of interest, it is crucial to design customized recommendation lists for various users.

Traditional recommendation methods mainly include collaborative filtering approach and content-based recommendation approach [4,20,22]. Jyun-Yu et al. [10] extended the one-class collaborative filtering approach (OCCF) [18] and proposed the Language-Regularized Matrix Factorization model (LRMF) based on Bayesian Personalized Ranking (BPRMF) [19]. Tadej et al. [16] constructed a network by projects and users on GitHub and then use link prediction to generate personalized recommendation lists for various users. Xu et al. [25] considered not only the user behavior, but also the content of repositories to improve the recommendation accuracy. In our work, we take into account the programming language preference from users and repositories as an important input.

In addition to the traditional recommender systems approaches, there are a number of other methods including deep learning, tensor factorization, factorization machines, and so on. These more advanced methods are good for taking the quality of your recommenders to the next level.

The past few decades have witnessed the tremendous success of the deep learning in many application domains such as computer vision, speech recognition and object detection [2,13,21]. Models with deep learning are mostly composed of multiple processing layers. They use the backpropagation algorithm to update the internal parameters of the model, which are used to compute the representation in each layer. Trained in this way, models are able to learn representations of data with multiple levels of abstraction [13].

Deep learning in recommender system has gained more attention [28]. Due to its ability to capture non-linear relationships and better data presentation for sparse high-dimensional vector, deep learning is able to gain good performance in rating prediction in the recommendation.

Most recently, Google proposed the Wide&Deep model [1] to jointly train wide linear models and deep neural network and apply it to Google Play. Guo et al. [8] constructed a new neural network architecture to use deep learning for feature learning in factorization machines for the recommendation. Similarly, He et al. [9] explored the use of deep neural networks based on collaborative filtering (CF). Another extension is CCCFNet (Cross-domain Content-boosted Collab-

[1] Github: https://github.com.

orative Filtering neural Network) [14], which combines CF and content-based filtering in a unified framework. Moreover, they further embed this cross-domain model into a multi-view neural network due to the advantages of capturing hidden representation learning.

In this paper, we explore an emerging technology in the recommendation, which combines the repository recommendation with the use of the neural network. We first introduce the framework of recommendation and later implement it into a specific recommender of Github program.

Our contributions are listed as follows:

1. We propose a general framework of PNCF, a preference-based neural collaborative filtering recommender model, for repository recommender.
2. Due to a great influence of language preference on repository recommendation, we develop the instantiation of PNCF framework with language preference called LR-PNCF in Github repository recommendation. In this model, we abstract the language preference from users and repositories and consider it as one of the inputs.
3. The result of the experiment shows that the model we proposed can outperform the existing state-of-the-art models in repository recommendation.

The remainder of this paper is organized as follows. In Sect. 2, we give an overview of the service recommendation framework. In Sect. 3, we present an instantiation of the PNCF framework for Github repository recommendation with language preference. In Sect. 4, we conduct the model on a real-world dataset compared other methods in the experiment. Section 5 makes a conclusion and give details of the future work.

2 Framework Overview

We first present a general framework of PNCF, which is short for a preference-based neural collaborative filtering recommender model. We adopt a multi-layer representation to show the whole structure, shown in Fig. 1, where the model is divided into four layers: Input layer, Embedding layer, Interaction Layer, and Output Layer.

In the Input Layer, we obtain n sparse features from users and projects respectively, which are set as $\mathbf{x_{u_1}}, \mathbf{x_{u_2}}, ..., \mathbf{x_{u_n}}$ for user u and $\mathbf{x_{i_1}}, \mathbf{x_{i_2}}, ..., \mathbf{x_{i_n}}$ for item i. These feature vectors of user u or item i are then separately compressed to low-dimensional vector through the Embedding layer and will be merged into a concatenated vector, named user concatenated vector c_u or item concatenated vector c_i respectively. This process will preserve the same or similar features in the vectors before interaction training, allowing the same user or vector to have the same suffix in vector and therefore enhancing the similarity between similar users or items.

After merging, user concatenated vector c_u as well as item concatenated vector c_i will be fed into a neural network to model the interaction ϕ between

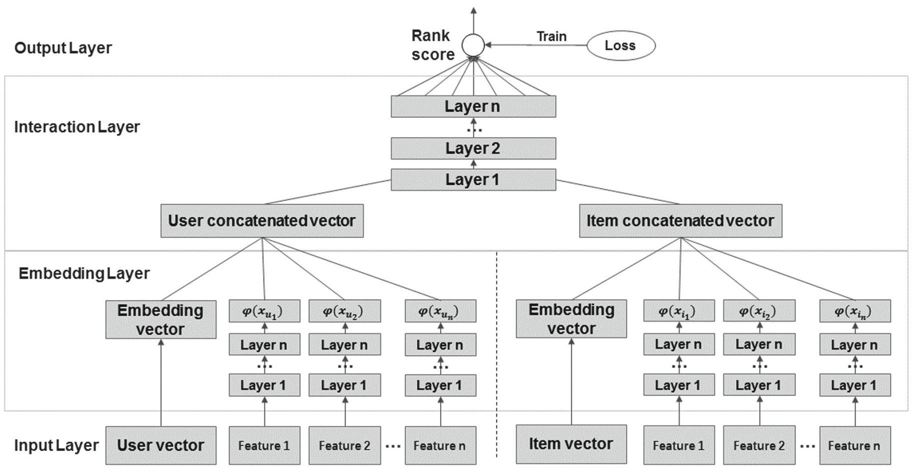


Fig. 1. Preference-based neural collaborative filtering recommender framework

user u and item i in the Interaction layer. Finally, score $\hat{y}$ is output through the Output layer to normalize the result.

Overall, PNCF framework can be formulated as below.

$$c_u = [\varphi(x_{u_1}), \varphi(x_{u_2}), ..., \varphi(x_{u_n})] \tag{1}$$

$$c_i = [\varphi(x_{i_1}), \varphi(x_{i_2}), ..., \varphi(x_{i_ns})] \tag{2}$$

$$\hat{y} = f_{out}(\phi(c_u, c_i)) \tag{3}$$

After calculating the output scores of candidate repositories, we sort it into a ranked list and select top-k items to recommend to user u.

3 Recommendation Model

In this section, we develop the instantiation of PNCF framework in Github repository recommendation with language preference, named LR-PNCF. In this case, our ultimate goal is to recommend suitable repositories in Github. At this point, users have a lot of features, such as language preference, various operations on the repositories or auxiliary information of following users, as well as the repositories. Since users in Github more likely to find programs with language they used, the preference of language is heavily taken into consideration. Additionally, the proposed model attempts to strengthen the similarity between similar users by extending the processed feature vectors at the end of corresponding latent vectors result in closer embedding vectors between similar users or items. As shown above, we divide the model into four layers and give details for each layer in implementation.

3.1 Input Layer

The proposed recommendation model is a kind of sequential model, each layer of which is arranged in sequence. The first layer is the Input layer. In this layer, there consists of two kinds of input: identity and language preference vector for each user or repository (Fig. 2).

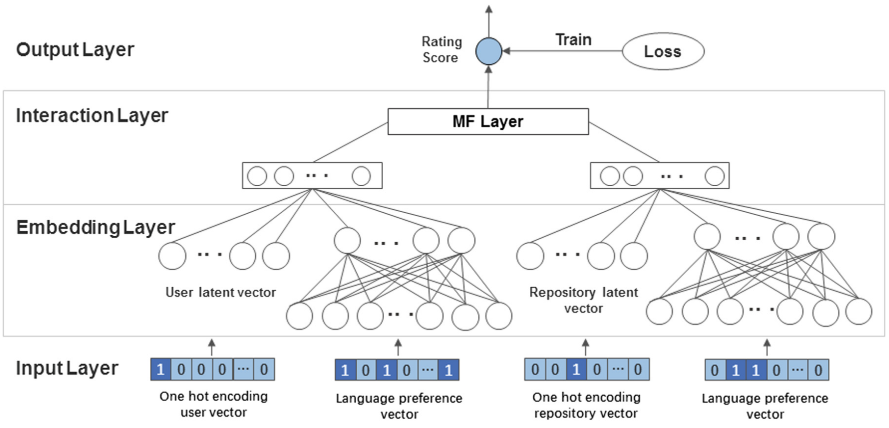


Fig. 2. LR-PNCF, an instantiation of PNCF framework with language preference

For the input of identity, we define u and i as the identities of user and repository. As usual, these two identities for user and repository are encoded by one-hot encoding and is used to obtain the corresponding latent vectors from the embedding matrix when training.

For input vectors of language preference from user or repository, we use $\mathbf{p}_u = \{p_u^1, p_u^2, ..., p_u^Y\}$ and $\mathbf{p}_i = \{p_i^1, p_i^2, ..., p_i^Y\}$ to model the language preference of user u and repository i. It is obvious that both users and repositories are many-to-many relationships with language preference, we use preference vectors to capture their language information. Consider the number of users and repositories as M and N and the number of languages as Y, Here, Y determines the rank of preference vectors $|\mathbf{p}_u| = Y$ and $|\mathbf{p}_i| = Y$. We define the preference matrix $\mathbf{P} \in \mathbb{R}^{(M+N)\times Y}$. For preference vector $\mathbf{p}_a = \{p_a^1, p_a^2, ..., p_a^Y\}$ from user or repository a, there exists

$$p_a^j = \begin{cases} 1, \text{ if the } j^{th} \text{ language is favored by } a \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

For example, if there are totally 5 languages and user a is keen on the language 1,2 and 5, the language preference vector of user a is encoded as $p_a = (1, 0, 1, 0, 1)$. As such, we convert the language preferences of users and repositories into corresponding preference vectors.

3.2 Embedding Layer

Above the Input Layer is the Embedding Layer. This layer aims at learning a dense vector to summarize the profile of each user or repository. In this layer, we process the two kinds of input, identity input and preference vector, into one dense vector. Since the input of identity is binarized sparse vectors with one-hot encoding, we pick out corresponding latent vectors from the embedding matrix $\mathbf{E} \in \mathbb{R}^{(M+N)\times D}$ marking it as $\mathbf{e}_u$ and $\mathbf{e}_i$ where D is the dimension of the latent vectors. These two latent vectors will learn the context from the latent factor model by using the information of an user-item rating matrix [9,12]:

$$\min_{e^*} \sum (y_{ui} - \mathbf{e_i}^T \mathbf{e_u})^2 + \lambda(||\mathbf{e_i}||^2 + ||\mathbf{e_u}||^2) \tag{5}$$

To be more specific, these latent vectors are initialized randomly following the normal distribution and then trained to optimize the loss function with the rating information.

For language preference input vector, it is a sparse, high-dimensional categorial features and thus we converted it into a low-dimensional and dense real-valued vector through a feed-forward neural network. Language preference vectors obtained from the Input layer are then fed into the input layers of a feed-forward neural network:

$$\mathbf{p}_u^{(0)} = \mathbf{p}_u, \ \mathbf{p}_i^{(0)} = \mathbf{p}_i \tag{6}$$

Each hidden layer in this neural network performs:

$$\mathbf{p}^{(l+1)} = f(\mathbf{W}^{(l)} \times \mathbf{p}^{(l)} + \mathbf{b}^{(l)}) \tag{7}$$

where $f(\cdot)$ is a specified activation function (often use ReLUs), $\mathbf{p}^{(l)}, \mathbf{b}^{(l)}$ and $\mathbf{W}^{(l)}$ are respectively the output, bias and weights at l-th layer.

To enhance the representation for embedding vectors of each user or repository, we merge the latent vector and preference vector into one concatenated vector. We formularize the concatenation as follow, $\mathbf{a}_u$ and $\mathbf{a}_i$ respectively for user u or repository i:

$$\mathbf{a}_u = [\mathbf{e}_u, \mathbf{p}_u^{(l)}], \ \mathbf{a}_i = [\mathbf{e}_i, \mathbf{p}_i^{(l)}] \tag{8}$$

3.3 Interaction Layer

The purpose of the Interaction layer is to conduct the interaction behaviours for user or item embedding vectors acquired by the Embedding layer. We use $f(.)$ to model these learning behaviours.

$$\hat{q}_{ui} = f(\mathbf{a}_u, \mathbf{a}_i) \tag{9}$$

where a_u and a_i is the embedding vectors for user u and repository i.

In most of research, f(.) is represented by the inner product of input vectors adding a bias from corresponding user and item. In this paper, we use MF (Generalized Matrix Factorization) to model the interaction between the users and repositories. Let $\hat{y}_{ui}$ denotes the output of the interaction layer and use the inner product of $\mathbf{a}_u$ and $\mathbf{a}_i$ to model the interaction:

$$\hat{q}_{ui} = \mathbf{a}_u^T * \mathbf{a}_i = \sum_{j=1}^{L} a_u^j * a_i^j \tag{10}$$

where L is the dimension of the embedding vector. Interaction function can be replaced by others common, classical or complicated neural network. In this paper, we use the simplest interaction function since it works.

3.4 Output Layer

There raise a problem when we directly use the output of the Interaction layer to be the rating prediction. It is obvious that the rating is calculated by a linear combination of user embedding factors, item embedding factors and bias in Eq. 10. These may fail in capturing the non-linear or more complex structure implied in embedding vectors. Some research works on various fields, such as natural language processing, show enhancement of representation learning when using non-linear transformations.

We let $\hat{y}_{ui}$ denotes the final output of the neural network. Since $\hat{y}_{ui}$ is considered as the probability that the user u likes the repository i, we use sigmoid function to limit the value into [0,1].

$$\hat{y}_{ui} = \sigma(\hat{q}_{ui}) \ , \ \sigma(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

where $\hat{y}_{ui}$ indicates the preference degree of the repository i for user u which can be also considered as the rank score. In this way, we map user embedding vectors and item embedding vectors into real-valued ratings. Based on the scores of candidate repositories, one can sort all the repositories and the final ranked list is generated.

3.5 Model Training

To learn the model parameters, there are many learning-to-rank algorithms can fit into the above framework. Note that different approaches model the process of learning to rank in different ways. These approaches are usually divided into point-wise, pair-wise and list-wise approaches [15]. In this paper, we use point-wise approach to train our model. Among different point-wise approaches, ranking is commonly modeled as regression, classification and ordinal regression. The most common use is Mean squared error which is largely applied in regression in machine learning. However, it is usually better to use cross-entropy error to evaluate the quality due to the structure of neural network [11]. Moreover, this formular much the same as the negative logarithm of the likelihood function when training parameters with probabilistic methods [9].

Let $\hat{y}_{ui}$ is the output of our model and y_{ui} is the desired output. Here $y_{ui} = 1$ represents repository i is favour by user u and otherwise $y_{ui} = 0$. The definition of cross-entropy loss function as follows:

$$\min_{\Theta} -\frac{1}{n} \sum [y \ln \hat{y}_{ui} + (1-y) \ln(1-\hat{y}_{ui})] \tag{12}$$

where Θ denotes the parameters of the model, n denotes the number of samples. The above loss function can be minimized by updating its parameters using optimization techniques like stochastic gradient descent.

Note that pointwise approach has its limitations. Since it does not consider the relationship between any two items, the corresponding loss function does not reflect the influence of different positions of items in the rank list.

4 Experiment

In this section, we evaluate the proposed model for Github Recommendation using a subset of real-world Github dataset. We first give a brief introduction of the dataset used in the experiment. And then describe two kinds of metrics and as well the evaluation protocols. Subsequently, the baseline of the experiments and their parameter setting including proposed LR-PNCF are illustrated. Finally shows the results. The experimental results demonstrate better improvement over competitive baselines.

4.1 Dataset Description

There are over one million people and three million public repositories in Github. We screen out those who have more than 10 repositories and randomly drew a part of it which contains about 3982 users and 4987 repositories. In Github, users have multiple behaviours for public repositories. For example, they may fork, watch or contribute coding to the repositories if they are interested. Therefore, in our paper, we define that a user is considered to prefer a repository if the user folk, watch or contribute coding to it. After the data is processed, there are about 179370 ratings in the dataset, the sparsity of user-item matrix is nearly 0.9%.

4.2 Evaluation and Metrics

For experiment, we apply leave-one-out schema to evaluate the performance of recommendation [9,19]. Leave-one-out validation involves selecting one item for each user as the validation set and the rest observations as the training set. For more efficient verification, we randomly select 100 unobserved items, along with the test item, to generate the recommendation list for each user [5,9]. We assume that the user prefers repositories that have been observed over all other non-observed repositories [19]. E.g. user u interacts with repository i_2 but not repository i_1, so we consider that user u prefers repository i_2 over i_1. The higher score or rank of the test repository indicates the better performance of the recommender.

To judge the ranked list quantitatively, we adopt two common metrics to evaluate the performance of ranking: Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG).

Hit Ratio is defined below:

$$HR@k = \begin{cases} 1, rank_i \ \leq \ \text{k} \\ 0, \text{otherwise} \end{cases} \tag{13}$$

where $rank_i$ denotes the rank of the test repository i of ranked list.

$NDCG$ is ratio of DCG and $iDCG$. Here $iDCG$ is the ideal discounted cumulative gain. The above DCG is calculated by summing up all "gains" along the rank list with a log discount factor and is able to quantify the usefulness of an item based on its position in the result list. Its formula is defined as:

$$DCG@K = \sum_{i=1}^{k} \frac{2^{r(i)} - 1}{log_2(1 + rank_i)} \tag{14}$$

where $r(i)$ denotes the desired rank of i^{th} repository in recommendation list R.

Intuitively, HR measures whether the test repository is appeared in ranked list and $NDCG$ measure how good the quality of recommendation by focusing on the position of the test repository.

4.3 Comparing Methods

We compare our recommendation approach with another recommendation algorithm. These methods, except for the PopN, is related to matrix factorization and are often used in repository recommendation:

1. PopN: Recommender is implementation by pushing the most popular repositories to users. This non-personalized method rate items using some form of popularity measure such as most forked or contributed even watched by users.
2. GMF [12,17]: This is the basic matrix factorization using only user-repository matrix for recommendations.
3. NCF [9]: This is a general framework named NCF, short for Neural network-based Collaborative Filtering. We use one of its instances NeuMF in following experiments.
4. BPRMF [19]: This is a matrix factorization model training with a generic optimization criterion BPR-OPT. It optimizes the measure about rankings directly. BPRMF is one of the common methods for ranking in the OCCF problem.

5. LRMF [10]: This is one of the state-of-the-art approaches for Github Recommendation, short for language-regularised matrix factorization. It based on matrix factorization and is regularized by the relationships between user programming language preferences.

In experiments, we set respective optimal parameters either according to corresponding references or based on our experiments for all methods to be validated. All of these methods executes 100 epochs. The initialization strategy we adopt for all involved embedding matrixes is to be randomly initialized with a uniform distribution within the interval [0, 1]. The radio of numbers of positive and negative samples is 1:1 when training.

For BPRMF method, we train it in batches and execute 100 epochs as well. In each batch, we randomly generate a certain number of pairs $(u, repo_i, repo_j)$ fed into a model where u denotes the user u, $repo_i$ denotes the repository observed by user u and $repo_i$ denotes the non-observed repository. For LRMF method, we apply bootstrapping-based stochastic gradient descent according to the original paper. Refer to their experimental setting, we set $c = 100$ which indicates we randomly select 100 authors for each user to calculate language preference regularization. For LR-PNCF models, the dimension of preference vector $p_u^{(0)}$ and $p_u^{(0)}$ in the following experiments is set to be 100. We use three layers in the feed-forward neural network to dense the preference vector where the number of neurons is the same as that of embedding vector $\mathbf{e}$. Later, we will evaluate the performance of the above model through quantitative indicators NDCG and HR in the next subsection.

4.4 Experimental Results

In this section, we compare the performance of models mentioned above using the metrics of HR and NDCG in the Top-K recommendation. Figure 4 and Fig. 3 shows the best HR and NDCG in evaluation for each competitive approaches. Since the performance of PopN is to weak to be negligible, we remove the HR and HDCG of PopN in comparison. In experiment, we focus on two issues. The first problem is whether this proposed approach performs better than the other comparative approaches. Another is how the performance of proposed method is affected by the dimension of latent vectors or the number of recommendation list. To simplify the explanation, we set *dim* as the dimension of latent vectors and *topk* as the number of recommendation list. We carried out the experiments where *dim* is assigned to 8,16 and 32 respectively with fixed value $topk = 10$ and *topk* is assigned to 10,20 and 30 respectively with fixed value $dim = 8$. All other variables remain the same. In each experiment, we record the best HR and NDCG in evaluation. The results are shown in Fig. 4 and Fig. 3. To show more numerical information, we also present the value of best HR and NDCG onto the Table 2 and Table 1 among the 100 epoches.

From the Fig. 4 and Fig. 3, we can see that the proposed model LR-PNCF has the better HR and NDCG that the other competitive methods. BPRMF and LRMF perform weaker when they are evaluated by the metrics of HR and

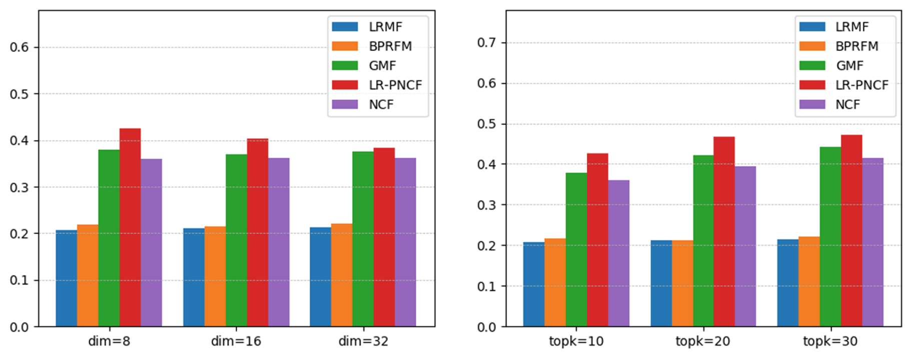


Fig. 3. Best HR in different dimension of latent vectors and in different recommendation list size

NDCG while the proposed model LR-PNCF, as well as GMF and NCF, keeps relatively high scores of metrics in different assignment of variables *topk* or *dim*. GMF and NCF behave better that BPRMF and LRMF. Among them, LR-PNCF outperforms other comparative methods in either HR or NDCG.

Shown in Table 2 and Table 1, LR-PNCF achieves higher HR and NDCG where the maximum value of HR and NDCG can be increased by 13.9% and 18.3% compared with the NCF and by 12.1% and 8.7% compared with GMF method under the same settings of *dim* and *topk*. Comparing the performance differences between comparative models, those difference is the most significant when the variables of *dim* and *topk* are assigned to smaller values. Especially with *topk* increase, these difference are shorten.

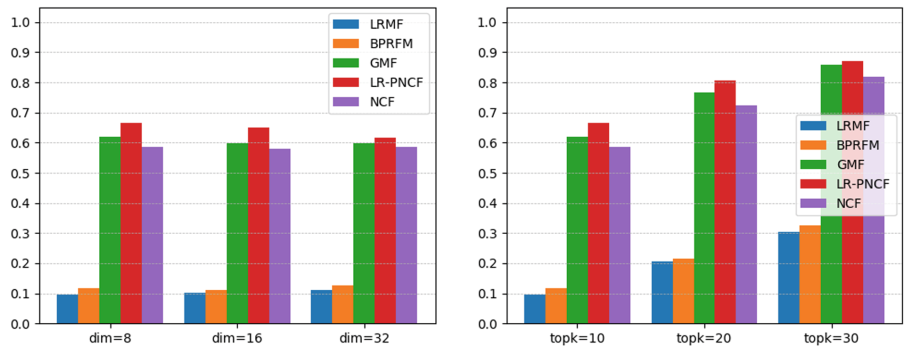


Fig. 4. Best DNCG in different dimension of latent vectors and in different recommendation list size

Additionly, in order to evaluate performance affected by the number of recommendation list, HR and NDCG were recorded under different assignment where *topk* = 10, 20, 30 respectively in Table 1. It can be clearly seen that HR and NDCG go higher when *topk* increase. It is quite understandable that target test items are easier to appear in the recommendation list expanding the list of recommendations under the fixed total number of sorted items in evaluation. In terms of the influence of the dimension of latent vectors, we recorded HR and NDCG under different assignment where *dim* = 8, 16, 32 respectively in Table 2. It can be clearly seen that HR and NDCG have a slight volatility when *dim* increase, which may encounter the problem of overfitting.

Overall we can make the conclusion that our model obtains the best performance and efficiency than other competitive methods.

Table 1. The best HR and NDCG in different *topk*

	topk = 10		topk = 20		topk = 30	
	HR	NDCG	HR	NDCG	HR	NDCG
BPRMF	0.1165	0.2179	0.2147	0.2126	0.3257	0.2215
LRMF	0.0944	0.2071	0.2049	0.2112	0.3039	0.2138
GMF	0.6188	0.3793	0.7652	0.4210	0.8589	0.4419
NCF	0.5853	0.3594	0.7240	0.3937	0.8179	0.4150
LR-PNCF	0.6667	0.4255	0.8064	0.4668	0.8709	0.4707

Table 2. The best HR and NDCG in different *dim*.

	dim = 10		dim = 20		dim = 30	
	HR	NDCG	HR	NDCG	HR	NDCG
BPRMF	0.1165	0.2179	0.1125	0.2149	0.1258	0.2215
LRMF	0.0944	0.2071	0.1015	0.2110	0.1102	0.2133
GMF	0.6188	0.3793	0.5972	0.3702	0.5982	0.3757
NCF	0.5853	0.3594	0.5806	0.3614	0.5848	0.3619
LR-PNCF	0.6667	0.4255	0.6489	0.4041	0.6173	0.3844

5 Related Work

In recent years, there is a variety of research works of service discovery and recommendation, but only a handful of them in Github recommendation. Previous work can be mainly classified into three types, semantics-based, CF-based and network-based. The semantics-based approaches mostly consider the semantic similarity of web services that they define the similarity between two web services due to particular semantic aspects [24]. The CF-based approaches are under the assumption that the target user has similar friends or target service owns similar services, without considering the exceptional scenarios [17,18]. The network-based approaches are to construct the information network based on relations using their history information [16].

Lately, as a special software service, Github repository become a new object of the recommendation system. The idea of matrix factorization represents a feasible resolution for software service discovery and has forced various CF-based approaches, e.g Probabilistic Matrix Factorization (PMF)[17], One-Class Collaborative Filtering (OCCF)[18]. Inspired by OCCF, Jyun-Yu et al. [10] proposed the Language-Regularized Matrix Factorization (LRMF) by utilizing user's programming language preference, which is an improvement of matrix factorization based on Bayesian Personalized Ranking(BPRMF) [19] and have achieved better performance on the real-world dataset of Github. Tadej et al. [16] constructed a network by the contributions based on Github data and then use link prediction to generate personalized recommendation lists for various users. Xu et al. [25] proposed a recommendation system named REPERSP, which is constructed based on both the developers behavior and the content of each project.

With less feature engineering, deep neural networks can generalize better to unseen feature combinations through low-dimensional dense embedding learned for the sparse features. Recently, many people explore deep learning to improve the performance of the recommendation. For the rating matrix, collaborative deep learning has been extended by coupling deep learning for content information and collaborative filtering [23,27].

Google presented the Wide & Deep learning framework [1] to achieve both memorization and generalization in one model. Wide linear models memorize sparse feature interactions using cross-product feature transformations, while deep neural networks generalize to unseen feature combinations through low-dimensional embedding. To avoid expertise feature engineering on the input in the wide part, Guo et al. [8] proposed an end-to-end model called DeepFM combining factorization machines with deep learning techniques for the recommendation. Wang et al. [23] proposed a hierarchical Bayesian model called collaborative deep learning (CDL), which jointly performs deep representation learning for the content information and collaborative filtering for the rating (feedback) matrix. Dong et al. [3] extended the stacked denoising autoencoder to integrate additional side information into the inputs and presents a hybrid collaborative filtering model with the deep structure of recommender systems. Similarly, He et al. [9] proposed the NCF model, which is short for Neural Collaborative Filtering. They use a neural architecture to learn an arbitrary function from data with matrix factorization and multi-layer perception. To make full use of both explicit ratings and implicit feedback, Xue et al. [26] proposed Deep Matrix Factorization models (DMF) that map the users and items into low-dimensional vectors and both of the input matrix and loss function consider both explicit ratings and implicit feedback. CCCFNet [14], namely Cross-domain Content-boosted

Collaborative Filtering Neural Network, is an extension of a unified framework with content-based filtering.

6 Conclusion and Future Work

In this paper, we present a general framework of PNCF, a preference-based neural collaborative filtering recommender model, and develop the instantiation of PNCF framework in Github repository recommendation with language preference. To better illustrate, the model is divided into four layers: Input layer, Embedding layer, Interaction layer, and Output layer. Then we explained the implementation of each layer in detail. Additionally, we extract the language preference from users and repositories to make the better performance of recommendation.

The results of extensive experiments show that our methods can significantly outperform the existing state-of-the-art repository recommender models. In this work, we combine deep learning with collaborative filtering to further enhance the effectiveness of the traditional recommender systems approaches.

In future, we will add more features of users and repositories to the input layer, such as users' actions on the repositories, rather than only using the important but single language preferences. Moreover, due to the pointwise approach has its limitations that it does not consider the relationship between any two items, we will study pairwise learners for PNCF instead.

Acknowledgment. The paper was supported by the National Key Research and Development Program (2017YFB0202200), the National Natural Science Foundation of China (61702568, U1711267), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (2017ZT07X355) and the Fundamental Research Funds for the Central Universities under Grant (17lgpy117).

References

1. Cheng, H.T., et al.: Wide & deep learning for recommender systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 7–10. ACM (2016)
2. Deng, L., Yu, D., et al.: Deep learning: methods and applications. Found. Trends® Signal Process. **7**(3–4), 197–387 (2014)
3. Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., Zhang, F.: A hybrid collaborative filtering model with deep structure for recommender systems. In: AAAI, pp. 1309–1315 (2017)
4. Dou, Y., Yang, H., Deng, X.: A survey of collaborative filtering algorithms for social recommender systems. In: 2016 12th International Conference on Semantics, Knowledge and Grids (SKG), pp. 40–46. IEEE (2016)
5. Elkahky, A.M., Song, Y., He, X.: A multi-view deep learning approach for cross domain user modeling in recommendation systems. In: Proceedings of the 24th International Conference on World Wide Web, pp. 278–288. International World Wide Web Conferences Steering Committee (2015)

6. Github: Celebrating nine years of github with an anniversary sale. https://github.com/blog/2345-celebrating-nine-years-of-github-with-an-anniversary-sale. Accessed 11 Apr 2017
7. Gousios, G., Vasilescu, B., Serebrenik, A., Zaidman, A.: Lean GHTorrent: GitHub data on demand. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 384–387. ACM (2014)
8. Guo, H., Tang, R., Ye, Y., Li, Z., He, X.: DeepFM: a factorization-machine based neural network for CTR prediction. arXiv preprint arXiv:1703.04247 (2017)
9. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web, pp. 173–182. International World Wide Web Conferences Steering Committee (2017)
10. Jiang, J.Y., Cheng, P.J., Wang, W.: Open source repository recommendation in social coding. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1173–1176. ACM (2017)
11. Kline, D.M., Berardi, V.L.: Revisiting squared-error and cross-entropy functions for training neural network classifiers. Neural Comput. Appl. **14**(4), 310–318 (2005)
12. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 330–37 (2009)
13. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
14. Lian, J., Zhang, F., Xie, X., Sun, G.: CCCFNet: a content-boosted collaborative filtering neural network for cross domain recommender systems. In: Proceedings of the 26th International Conference on World Wide Web Companion, pp. 817–818. International World Wide Web Conferences Steering Committee (2017)
15. Liu, T.Y., et al.: Learning to rank for information retrieval. Found. Trends® Inf. Retrieval **3**(3), 225–331 (2009)
16. Matek, T., Zebec, S.T.: GitHub open source project recommendation system. arXiv preprint arXiv:1602.02594 (2016)
17. Mnih, A., Salakhutdinov, R.R.: Probabilistic matrix factorization. In: Advances in Neural Information Processing Systems, pp. 1257–1264 (2008)
18. Pan, R., et al.: One-class collaborative filtering. In: Eighth IEEE International Conference on Data Mining, ICDM 2008, pp. 502–511. IEEE (2008)
19. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp. 452–461. AUAI Press (2009)
20. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, pp. 285–295. ACM (2001)
21. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
22. Song, Y., Zhang, L., Giles, C.L.: Automatic tag recommendation algorithms for social recommender systems. ACM Trans. Web (TWEB) **5**(1), 4 (2011)
23. Wang, H., Wang, N., Yeung, D.Y.: Collaborative deep learning for recommender systems. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1235–1244. ACM (2015)
24. Wu, J., Chen, L., Xie, Y., Zheng, Z.: Titan: a system for effective web service discovery. In: Proceedings of the 21st International Conference on World Wide Web, pp. 441–444 (2012)

25. Xu, W., Sun, X., Hu, J., Li, B.: REPERSP: recommending personalized software projects on GitHub. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 648–652. IEEE (2017)
26. Xue, H.J., Dai, X., Zhang, J., Huang, S., Chen, J.: Deep matrix factorization models for recommender systems. In: IJCAI, pp. 3203–3209 (2017)
27. Yang, X., Guo, Y., Liu, Y., Steck, H.: A survey of collaborative filtering based social recommender systems. Comput. Commun. **41**, 1–10 (2014)
28. Zhang, S., Yao, L., Sun, A.: Deep learning based recommender system: A survey and new perspectives. arXiv preprint arXiv:1707.07435 (2017)

A Weighted Meta-graph Based Approach for Mobile Application Recommendation on Heterogeneous Information Networks

Fenfang Xie, Liang Chen(✉), Yongjian Ye, Yang Liu, Zibin Zheng, and Xiaola Lin

School of Data and Computer Science,
National Engineering Research Center of Digital Life, Sun Yat-sen University,
Guangzhou 510006, China
{xieff5,yeyj7,liuy296}@mail2.sysu.edu.cn,
{chenliang6,zhzibin,linxl}@mail.sysu.edu.cn

Abstract. Explosive growth in the number of mobile applications (apps) makes it difficult for users to find relevant apps. Therefore, it is an urgent task to recommend desired apps for users. Traditional approaches focus on exploiting the context information, user's interest, privacy, security and other features for app recommendation. Most of them do not consider heterogeneous information network (HIN) in the scenario of mobile app recommendation. HIN contains rich structure and semantic information, and it can satisfy various requirements of users and generate better recommendation results. In this paper, we propose a **W**eighted **M**eta-**G**raph based approach for app **Rec**ommendation, called WMGRec, on HIN. Specifically, we firstly introduce the concept of weighted meta-graph, which not only distinguishes different rating scores to depict the subtle semantics but also utilizes meta-graph to capture complex semantics. And then, we apply weighted meta-graph to measure the semantic similarity between users and apps. Furthermore, we leverage non-negative matrix factorization on user-app similarity matrix to obtain user latent features and app latent features. Finally, the concatenated user and app latent features are fed into the factorization machine & deep neural network model to learn the higher-order interactions and get the final prediction score. Extensive experiments conducted on two real-world datasets validate the effectiveness of the proposed approach compared to state-of-the-art recommendation algorithms.

Keywords: Mobile app recommendation · Meta-graph
Heterogeneous information network · Factorization
Deep neural network

1 Introduction

Mobile devices (e.g., smart phones, tablet computers) have been becoming increasingly popular in recent years. They are gradually becoming a part of our daily life for study, entertainment, social intercourse, browsing news and businesses. Mobile devices promote the explosive growth of mobile apps. It becomes considerably difficult and time-consuming for users to find a relevant mobile app from a huge number of mobile apps. Therefore, it is essential to help users to locate their desired apps. Mobile app recommendation is a good choice to solve this problem and enhance user experience.

Previous studies concentrate on considering the context information, user's interest, privacy, security and other features for mobile app recommendation [2,5,8,18,21]. They usually employ collaborative filtering (CF) (e.g., item-based CF, user-based CF and matrix factorization) methods to recommend apps to users. Most of them do not consider the rich structure and semantic information on HIN. HIN has been widely exploited to data mining tasks, such as similarity measure, clustering, classification, link prediction and recommendation [10]. There exist multiple object types (e.g., users, mobile apps, reviews, and rating scores) and rich relations among object types (e.g., use and used by relation between users and apps, write and written by relation between users and reviews) in the scenario of app recommendation, which naturally constitutes an HIN. Exploiting the rich structure and semantic information can reveal subtle relations among objects. Therefore, it will satisfy different kinds of requirements of users and improve recommendation performance. As an important characteristic of HIN, meta-path is usually applied to model the multiple semantic information. HIN accomplishes traditional CF methods by building various meta-paths. For example, if we want to recommend apps to users, we can build a simple meta-path "User-App-User" or "User-APP-APP" and learn from this meta-path to make generalizations. These two kinds of meta-paths achieve the user-based CF and item-based CF methods, respectively. From HIN's schema, we can define more complicated meta-paths like "User-Review-Topic-Review-App". This meta-path defines a similarity to measure whether a user tends to like an app if his/her reviews are similar to those written by other users for the same app. In addition, users only use and rate a limited number of apps in the real-world. Therefore, the user-app rating matrix is considerably sparse. As a result, the recommendation results obtained by matrix factorization may be inaccurate. The rich structure and semantic information on HIN can alleviate this sparsity problem.

Inspired by the significance of HIN, we prefer to apply HIN to solve our app recommendation problem. When utilizing meta-path based similarities to app recommendation, there exist the following two major challenges: (1) Traditional HIN and meta-path do not consider the attribute values (the rating scores in app recommendation) on links. Ignoring the rating scores between users and apps may result in bad similarity discovery and cannot reveal the subtle difference. Figure 1 shows an example, given a meta-path "User-APP-User", if both *U1* and *U2* provide a rating score as 5 to Facebook, then *U1* and *U2* are quite similar due to the same preference. If *U1* gives a 5 score to Facebook and *U2* gives a 1 score to Facebook, then *U1* and *U2* may not be so similar due to the totally different taste. But the conventional meta-path regards these two cases as the same. Therefore, it will result in inaccurate similarity and cannot recognize the

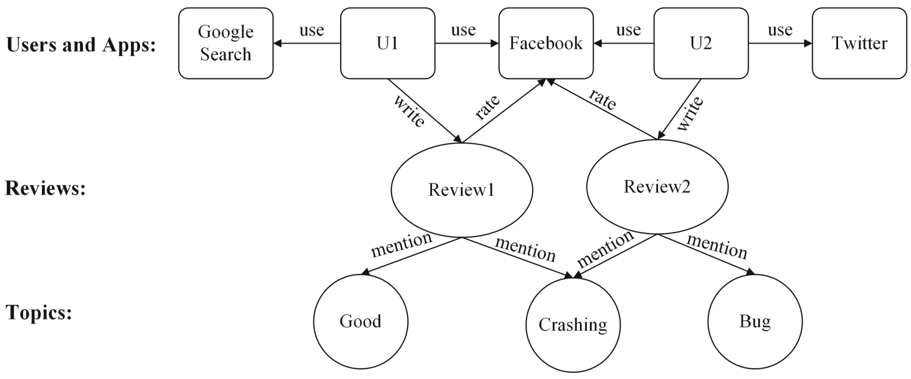


Fig. 1. Example of HIN built by mobile app dataset

subtle difference. However, in the scenario of app recommendation, this difference is very important because it can help to more accurately capture interactions between users and apps. It is essential to extend the conventional meta-path for considering attribute values on links. (2) Meta-path may not be very suitable to describe the rich structure and capture complicated semantics. Figure 1 presents a concrete example of this kind of case. Given a meta-path "User - Review - Topic - Review - User", which can be used to capture users' similarity since both of them write reviews and mention the same topic. However, if we intend to capture the semantic relation that *U1* and *U2* provide rating scores to the same app (e.g., Facebook), and at the same time, they mention the same topic (e.g., crashing), the meta-path fails. As a result, it is essential to find a more suitable way to capture such complex semantics. Meta-graph [3,6,19] is a good way to represent the above mentioned semantic relation. Meta-graph computes similarity between homogeneous type of entities over HINs, which can capture more complex semantics that meta-path cannot.

To alleviate the above challenges, this paper proposes a weighted meta-graph based approach for mobile apps recommendation on HIN. Specifically, we first describe the concept of weighted meta-graph rather than meta-path to capture the complicated semantics. Moreover, weighted meta-graph is used to measure the semantic similarity between users and apps through distinguishing different rating scores. And then, we use all latent features of all meta-graphs. For each meta-graph, we first compute the user-app similarity matrix under the guidance of the weighted meta-graph, and then exploit non-negative matrix factorization to obtain user latent features and app latent features. Finally, by concatenating different user latent feature and app latent feature vectors computed from different weighted meta-graphs, we use the factorization machine & deep neural network (DeepFM) model to learn the low- and high-order interactions between users and apps, and to further learn the rating matrix. Experimental results on two real-world datasets demonstrate the proposed approach outperforms other state-of-the-art recommendation approaches.

In summary, the major contributions of this paper are three folds:

– To the best of our knowledge, we are the first to introduce the concept of weighted meta-graph. The weighted meta-graph considers rating scores on links and capture more complex semantic similarity between users and apps.
– We propose a weighted meta-graph based approach, which not only effectively integrates rich structure and semantic information contained in app recommendation, but also captures the higher order interactions between users and apps. Furthermore, WMGRec can satisfy different kinds of requirement of users and improve recommendation performance.
– Empirical studies on two real-world datasets verify the effectiveness of WMGRec. WMGRec achieves better performance than other recommendation algorithms with the help of weighted meta-graph, non-negative matrix factorization and DeepFM.

The rest of this paper is organized as follows. Section 2 presents the proposed approach in detail. Section 3 analyzes and discusses the experimental results and impact of parameters. Section 4 introduces some related works. Section 5 concludes the paper and gives some future directions.

2 Weighted Meta-graph Based Mobile App Recommendation

In this section, the details of our proposed approach are described. For simplicity, the same definitions of HIN and the corresponding network schema in [14] are adopted in this paper. An illustration of the HIN network schema and meta-graphs are shown in Fig. 2(a) and (b), respectively.

2.1 Basic Concepts

Given an HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its network schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{L}, \mathcal{Q})$, where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the link set, $\mathcal{L}$ is the node type set, $\mathcal{Q}$ is the link type set. We propose a novel concept, named weighted meta-graph, to capture complex relationship between two HIN objects.

Definition 1 Weighted Meta-graph. A meta-graph $\mathcal{D}$ is a directed acyclic graph with a single source node $\mathcal{N}_s$ (i.e., with in-degree 0) and a single target node $\mathcal{N}_t$ (i.e., with out-degree 0). Formally, $\mathcal{D} = (\mathcal{N}, \mathcal{M}, \mathcal{N}_s, \mathcal{N}_t)$, where $\mathcal{N} \subseteq \mathcal{V}$ is a set of nodes and $\mathcal{M} \subseteq \mathcal{E}$ is a set of links. For any node $x \in \mathcal{N}$, $x \in \mathcal{L}$; for any link $(x, y) \in \mathcal{M}, (x, y) \in \mathcal{Q}$. If any link (x, y) has an attribute value c on it, where $c \in (0, \mathcal{C})$, $\mathcal{C} \in \mathbb{R}^+$, the meta-graph is called a *weighted meta-graph*; if any link (x, y) has no attribute value on it, the meta-graph is called an *unweighted meta-graph.*

Figure 2(b) illustrates a meta-graph $\mathcal{D}_3$, which depicts that two users provide rating scores to the same app, and have mentioned the same topic at the same

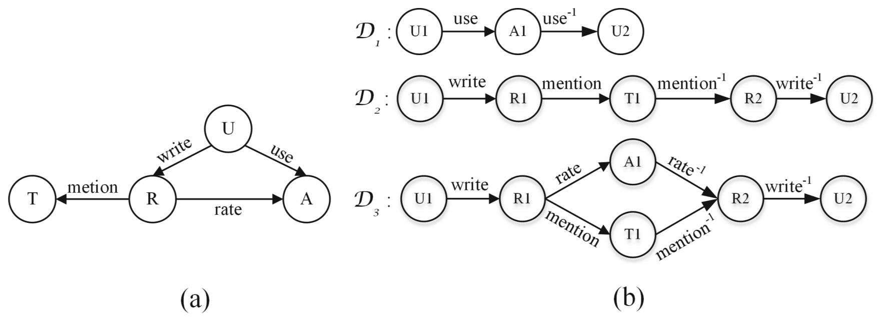

U
write
use
T
metion
R
rate
A
(a)
$\mathcal{D}_1$: U1 use A1 use^{-1} U2
$\mathcal{D}_2$: U1 write R1 mention T1 mention^{-1} R2 write^{-1} U2
$\mathcal{D}_3$: U1 write R1 rate A1 rate^{-1} mention T1 mention^{-1} R2 write^{-1} U2
(b)

Fig. 2. Example of HIN network schema and meta-graphs used for mobile app dataset. T: topics extracted from reviews; R: reviews; U: users; A: apps.

time. However, a meta-path fails to capture such complex relationship. A meta-graph is convenient to express this kind of relationship. A meta-path (e.g., $\mathcal{D}_1$ or $\mathcal{D}_2$ in Fig. 2(b)) is a special case of a meta-graph. Therefore, we call it meta-graph uniformly in the following. In addition, an unweighted meta-graph cannot capture the subtle deference of users' preference due to lack of considering rating scores on links.

Definition 2 Atomic Meta-graph. If all attribute values in a weighted meta-graph take a specific value, the meta-graph is called an atomic meta-graph. A weighted meta-graph is a group of atomic meta-graphs which contains all atomic meta-graphs that satisfy the constraint $\mathcal{C}$.

The attribute value c in our datasets is an integer which is in the range of $[1, 5]$. A toy example is shown in Fig. 3, "$\text{U}\overset{1}{-}\text{A}\overset{1}{-}\text{U}$" and "$\text{U}\overset{4}{-}\text{A}\overset{4}{-}\text{U}$" are two different atomic meta-graphs. The weighted meta-graph $\text{U}\overset{i}{-}\text{A}\overset{j}{-}\text{U}|i = j$ is a group of five atomic meta-graphs (e.g., $\text{U}\overset{2}{-}\text{A}\overset{2}{-}\text{U}$, $\text{U}\overset{3}{-}\text{A}\overset{3}{-}\text{U}$ and $\text{U}\overset{5}{-}\text{A}\overset{5}{-}\text{U}$).

Definition 3 Commuting Matrix. A commuting matrix M for a meta-graph $\mathcal{D} = (\mathcal{L}_1\mathcal{L}_2...\mathcal{L}_l)$ is defined as $M = W_{\mathcal{L}_1\mathcal{L}_2} \circ W_{\mathcal{L}_2\mathcal{L}_3} \circ ... \circ W_{\mathcal{L}_{l-1}\mathcal{L}_l}$, where $W_{\mathcal{L}_i\mathcal{L}_j}$ is the adjacency matrix between type $\mathcal{L}_i$ and type $\mathcal{L}_j$. $\circ$ can be two operations (Hadamard product "$\odot$" and multiplication "$\cdot$").

To better comprehend the computation of commuting matrix, we take the meta-graphs in Fig. 2(b) for example. For meta-graph $\mathcal{D}_1$, the commuting matrix $M_{\mathcal{D}_1}$ is computed as $M_{\mathcal{D}_1} = W_{UA} \cdot W_{AU}$. For meta-graph $\mathcal{D}_3$, the computation of the commuting matrix $M_{\mathcal{D}_3}$ should be divided into four steps:

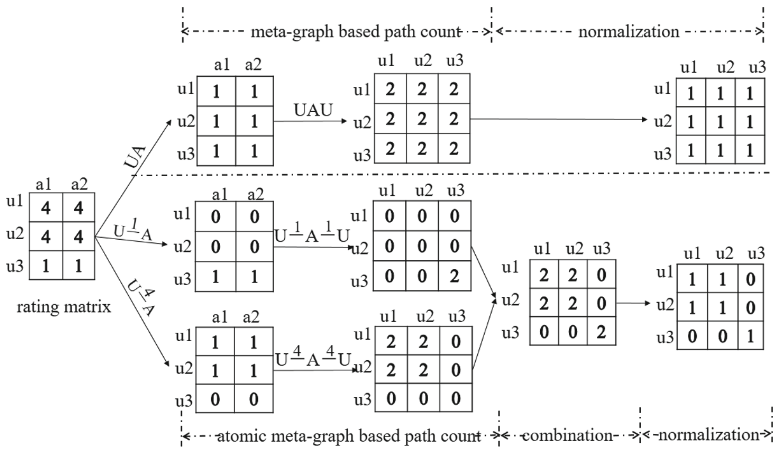
- Compute t_1: $t_1 = W_{RA} \cdot W_{AR}$,
- Compute t_2: $t_2 = W_{RT} \cdot W_{TR}$,
- Compute t_3: $t_3 = t_1 \odot t_2$,
- Compute $M_{\mathcal{D}_3}$: $M_{\mathcal{D}_3} = W_{UR} \cdot t_3 \cdot W_{RU}$.

2.2 Meta-graph Based Similarity

There are plenty of similarity measurements [16] to compute the similarity between two objects. Herein, we exploit the most widely used similarity measurement in HIN. Given a meta-graph $\mathcal{D}$, PathSim [14] between two objects i and j from the same type can be calculated as:

$$S_{i,j} = \frac{2M_{i,j}}{M_{i,i} + M_{j,j}} \tag{1}$$

where M is the commuting matrix for the meta-graph $\mathcal{D}$, $M_{i,i}$ and $M_{j,j}$ are the visibility for i and j in the network, namely, the number of meta-graphs between themselves.



meta-graph based path count
normalization
rating matrix
UA
UAU
$U\xrightarrow{1}A$
$U\xrightarrow{1}A\xrightarrow{1}U$
$U\xrightarrow{4}A$
$U\xrightarrow{4}A\xrightarrow{4}U$
atomic meta-graph based path count
combination
normalization

Fig. 3. A toy example for pathsim based similarity measurement on meta-graph

A toy example for PathSim based similarity measurement on unweighted meta-graph (the upper part) and weighted meta-graph (the lower part) is presented in Fig. 3. After the process of path count (the number of path instances between two objects), atomic meta-graph combination and normalization, the traditional unweighted meta-graph considers that $u1, u2, u3$ are all similar with each other. The weighted meta-graph draws a conclusion that only $u1$ and $u2$ are similar due to their same taste in apps. Obviously, the result found by weighted meta-graph is more accurate than that of unweighted meta-graph.

After obtaining the meta-graph based similarity of users, we can find the similar users of a target user under a given meta-graph. And then, the similarity between the target user and the app can be inferred according to his/her similar users.

$$\begin{gathered} H^{(l)}_{u,i,r} = \sum_{v} S^{(l)}_{u,v} \times B_{u,i,r}, \\ B_{u,i,r} = \{^{1 \quad \mathcal{R}_{u,i} = r}_{0 \; otherwise}, \end{gathered} \tag{2}$$

where $S_{u,v}^{(l)}$ is the PathSim based similarity between user u and user v under the meta-graph $\mathcal{D}_l$. $B_{u,i,r}$ is an indication function. If user u provides a rating score r to app i, $B_{u,i,r}$ is equal to 1; otherwise, it is equal to 0. $H_{u,i,r}^{(l)}$ is the intensity of user u rating app i with score r.

The similarity $\hat{\mathcal{R}}_{u,i}^{(l)}$ between user u and app i along a given meta-graph $\mathcal{D}_l$ can be computed as follows:

$$\hat{\mathcal{R}}_{u,i}^{(l)} = \sum_{r=1}^{N} r \times \frac{H_{u,i,r}^{(l)}}{\sum_{k=1}^{N} H_{u,i,k}^{(l)}} \tag{3}$$

2.3 Meta-graph Based Latent Features

By repeating the above similarity computation process to all meta-graphs, we can obtain L different kinds of similarity matrices. We denote these similarity matrices as $\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(L)}$. For each similarity matrix under a given meta-graph, a non-negative matrix factorization [7] technique is applied to get the low-rank matrix representation for users and apps. The idea of matrix factorization is that a matrix is approximately equal to the multiplication of two low-rank matrices. Mathematically,

$$\begin{aligned} \hat{\mathcal{R}} \approx &\quad \mathcal{U} \cdot \mathcal{A} \\ s.t. &\quad \mathcal{U} \geq 0, \mathcal{A} \geq 0 \end{aligned} \tag{4}$$

By minimizing the following equation, the low-rank representation of users and apps can be obtained:

$$\min_{\mathcal{U},\mathcal{A}} \frac{1}{2} \| I(\mathcal{R} - \hat{\mathcal{R}}) \|_F^2 + \frac{\lambda_1}{2} \| \mathcal{U} \|_F^2 + \frac{\lambda_2}{2} \| \mathcal{A} \|_F^2 \tag{5}$$

where I is the indicator matrix. If the rating of user i on app j is observed, I_{ij} is 1; otherwise, I_{ij} is 0. λ_1 and λ_2 are the hyper-parameters that control the influence of the regularization term to avoid overfitting. $\|\cdot\|_F$ is the Frobenius norm.

After factorizing all similarity matrices $\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(L)}$, we can obtain L types of low-rank representation matrix pairs $(\mathcal{U}^{(1)}, \mathcal{A}^{(1)}), \ldots, (\mathcal{U}^{(L)}, \mathcal{A}^{(L)})$. For each pair of low-rank representation matrix $\mathcal{U}^{(l)}$ and $\mathcal{A}^{(l)}$, we concatenate the ith row of $\mathcal{U}^{(l)}$ (e.g., $u_i^{(l)}$) and the jth row of $\mathcal{A}^{(l)}$ (e.g., $a_j^{(l)}$). Taking all meta-graphs into consideration, we can finally get a sample or a feature vector x^n like this:

$$x^n = \underbrace{u_i^{(1)}, \ldots, u_i^{(l)}, \ldots, u_i^{(L)}}_{L \times K}, \underbrace{a_j^{(1)}, \ldots, a_j^{(l)}, \ldots, a_j^{(L)}}_{L \times K} \tag{6}$$

where the value of i is from 1 to the number of users, the value of j is from 1 to the number of apps. K is the number of latent factors. It could be different for different matrices, but we keep it the same for simplicity.

2.4 WMGRec Model

After the above calculation, we can get the concatenated feature vectors of samples. These features vectors are fed into DeepFM model to learn the low- and high-order interactions between users and apps.

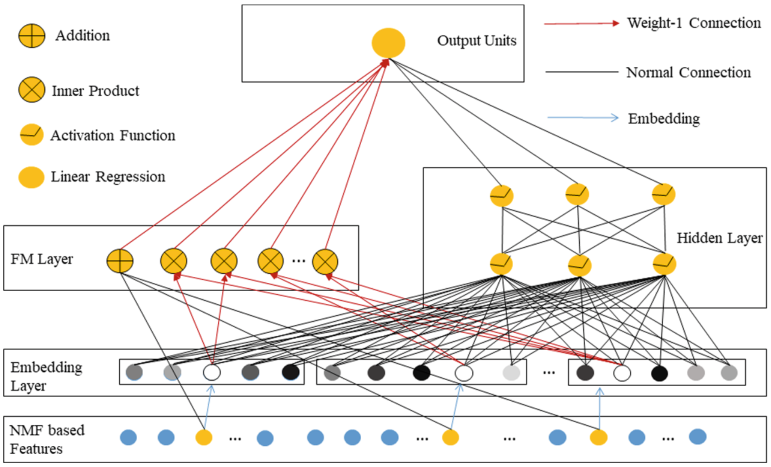


Fig. 4. The architecture of DeepFM

The architecture of DeepFM [4] is introduced in Fig. 4. The DeepFM model is divided into two components: FM (factorization machine) component and DNN (deep neural network) component. The two components share the same input feature vectors. The FM component is an improved factorization machine [9]. As aforementioned, we have obtained the feature vectors. These feature vectors are firstly fed into the embedding layer. The embedding techniques in neural network have the advantage of reducing the dimensionality of feature vectors. Secondly, the embedding vectors are put into the FM layer to learn the low-order interactions between users and apps. FM is a good way to deal with sparsity data and can be calculated in linear time. The output of FM component is formulated as follows:

$$\hat{y}(X) = w_0 + \sum_{i=1}^{f} w_i x_i + \sum_{i=1}^{f-1} \sum_{j=i+1}^{f} < V_i, V_j > x_i x_j \tag{7}$$

where, f is the length of the feature vector. w_0 is the global bias, w_i is the weight of the ith variable. A row V_i within $V \in \mathbb{R}^{f \times d}$ describes the ith variable with d factors. X is the feature vector, which is concatenated by user latent features and app latent features.

The DNN component is a traditional feed-forward neural network in which data flows from the input layer to the output layer without looping back. DNN can model complex non-linear relationships. For DNN component, embedding vectors are put into one or more hidden layer to learn the high-order interactions between users and apps. The output of the embedding layer can be formulated as:

$$z^{(0)} = [e_1, e_2, ..., e_m] \tag{8}$$

where e_i is the embedding of the ith field and m is the number of fields. Then, $z^{(0)}$ is fed into the DNN, and the forward process is:

$$z^{(q+1)} = \sigma(\mathcal{W}^{(q)} z^{(q)} + b^{(q)}) \tag{9}$$

where q is the layer depth and σ is an activation function. $z^{(q)}, \mathcal{W}^{(q)}, b^{(q)}$ are the output, model weight, and bias of the qth layer, respectively.

Through a series of hidden layers, the output of the final hidden layer is fed into the output units. An activation function is used to acquire the output of the DNN component.

$$y_{DNN} = \sigma(\mathcal{W}^{|Q|+1} z^{Q} + b^{|Q|+1}) \tag{10}$$

where $|Q|$ is the number of hidden layers.

By combing the output of FM component and DNN component, all parameters (e.g., including w_i, V_i, and the network parameters $\mathcal{W}^{(q)}, b^{(q)}$) can be trained jointly. The predicted rating score $\hat{y}$ is calculated as follows:

$$\hat{y} = h(y_{FM} + y_{DNN}) \tag{11}$$

where y_{FM} is the output of FM component, and y_{DNN} is the output of deep component, and h is a linear regression. Note that we modify the original DeepFM model (i.e., a classification task model) to fit our regression task (i.e., rating prediction) by leveraging the linear regression in the output units.

3 Empirical Study

In this section, we first describe the two mobile app datasets. Then, we introduce the evaluation metrics for performance comparison. Moreover, we compare our WMGRec model with other state-of-the-art models. Finally, we study impact of parameters on performance.

3.1 Dataset Description

We conduct experiments on two datasets. One dataset is crawled from a famous app market Apple App Store. This dataset includes 12688 users and 10556 apps with 335744 app ratings ranging from 1 to 5. The other dataset is crawled from Google Play, provided by [8]. This dataset includes 14379 users and 25515 apps with 330212 app ratings ranging from 1 to 5. The density $(\frac{\#Ratings}{\#Users \times \#Apps})$ of Apple App Store and Google Play dataset are 0.2507% and 0.09%, respectively. Both of the datasets also collect the reviews of users on apps. The detailed statistics of two datasets are given in Table 1.

Table 1. Statistics of the datasets

Dataset	Relation of (A-B)	Number of A	Number of B	Number of (A-B)
Apple App Store	User-app	12688	10556	335744
	User-review	12688	335744	335744
	Review-app	335744	10556	335744
	Review-topic	335744	4	973499
Google Play	User-app	14379	25515	330212
	User-review	14379	330212	330212
	Review-app	330212	25515	330212
	Review-topic	330212	4	288927

3.2 Evaluation Metrics

In the following experiments, we use Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to evaluate the errors between our predicted results and the reality outcomes [15]. MAE is given by:

$$MAE = \frac{\sum_{(i,j)\in\Gamma_{test}} |\hat{r}_{ij} - r_{ij}|}{|\Gamma_{test}|}, \tag{12}$$

and RMSE is denoted as:

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in\Gamma_{test}} (\hat{r}_{ij} - r_{ij})^2}{|\Gamma_{test}|}}. \tag{13}$$

where, Γ_{test} represents the set of all user-app pairs (i, j) in the testing set, $\hat{r}_{ij}$ represents the predicted rating score of user i on app j, while r_{ij} is the observed rating score of user i on app j in the testing set.

3.3 Performance Comparison

To validate the effectiveness of the proposed approach, we compare WMGRec model with the following models.

- NMF [7]. This approach employs matrix factorization to user-app rating matrix with a constraint that the factorized matrix is positive.
- FM [9]. This approach is the traditional factorization machine. It concatenates user id and app id as sparsity features, and learns the interactions between users and apps to complete the user-app rating matrix.
- SemRec [12]. This approach applies weighted meta-path based similarity, and designs different kinds of strategies to predict the rating scores of users on apps.
- FMG [19]. This approach utilizes unweighted meta-graph based similarity and standard matrix factorization to obtain user and app latent features. And then, it uses factorization machine to predict the rating scores of users on apps.

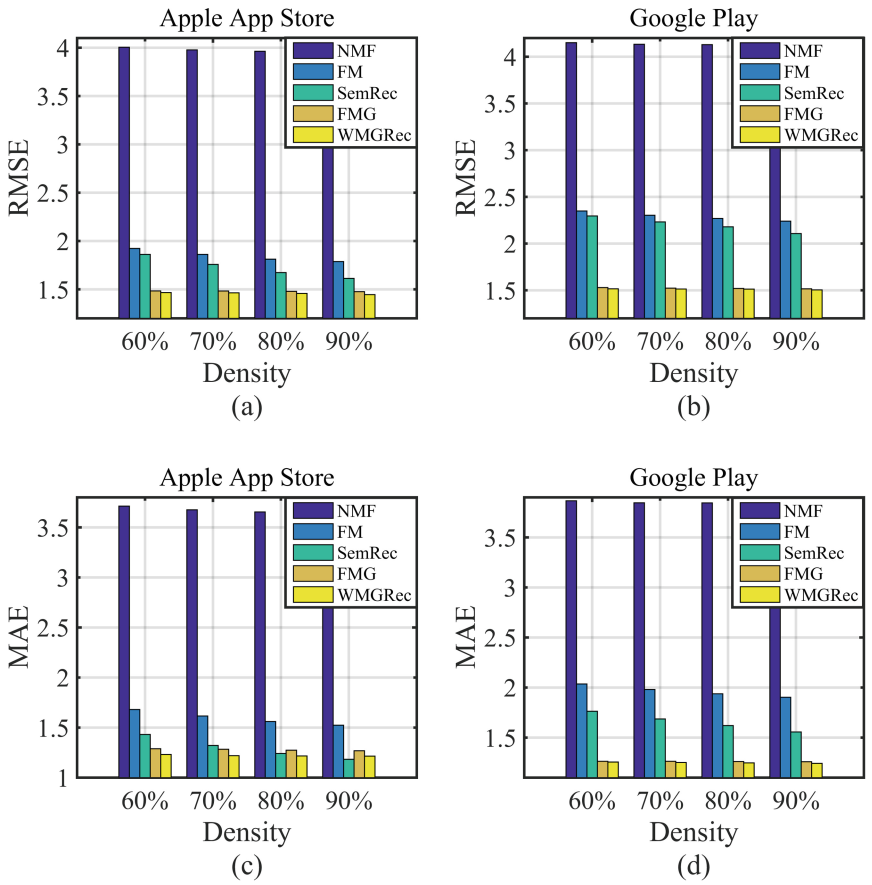


Fig. 5. Experimental results of all comparison methods

We employ the meta-graphs in Fig. 2(b) whose length are not longer than 4, since the longer meta paths are not meaningful and they fail to produce good similarity measure [14]. We repeat the experiments five times and use the average RMSE and MAE of five rounds as the final result. The topics are extracted from review texts by applying LDA (Latent Dirichlet Allocation) [1] model. The number of hidden layers is 2. The dropout in WMGRec is 0.5 and learning rate is set as 0.001. The latent factors and embedding size of all methods are fixed as 10 for fair. The other parameters in those comparison methods are set with the best performances.

Experiment results of all comparison methods are shown in Fig. 5. It can be found that the denser training data brings better performance. The reason lies in that with more observations in the training set, more information of the whole matrix can be obtained. This leads to more accurate predicted rating scores of users on apps. From Fig. 5, we can see that our proposed method beats the other state-of-the-art methods under all training data density (i.e., 60%, 70%, 80% and 90%) and evaluation metrics (i.e., RMSE and MAE). Specifically, the performance of FM is better than that of NMF due to capturing interactions between users and apps. HIN based approaches (e.g., SemRec, FMG, WMGRec) are better than FM due to taking advantage of rich structure and abundant semantics. The comparison between SemRec and FMG validates the effectiveness of meta-graph. By comparing WMGRec with FMG, it indicates considering the rating scores on links and applying DeepFM to learn high-order interactions are indeed helpful to improve the recommendation accuracy. Concretely, compared with FMG, the improvement of WMGRec on Apple App Store dataset is 1.15% to 2.05% for RMSE and 4.25% to 4.99% for MAE. In addition, the improvement of WMGRec on Google Play datset is 0.46% to 0.88% for RMSE and 0.62% to 1.31% for MAE. The enhancement of Google Play datset is smaller than that of Apple App Store dataset, because the density of the former is considerably sparse. Note that SemRec achieves the best MAE when the training data density is 90%. That's because it's not always possible to get the best performance of both evaluation metrics simultaneously, when optimizing the loss function. Moreover, we mainly optimize the RMSE evaluation metric in this paper.

3.4 Study on Parameter Impacts

In this section, we discuss the impact of some major parameters on Apple App Store dataset. In the following experiments, we fix 80% of the whole data as the training set and the remaining 20% as the testing set.

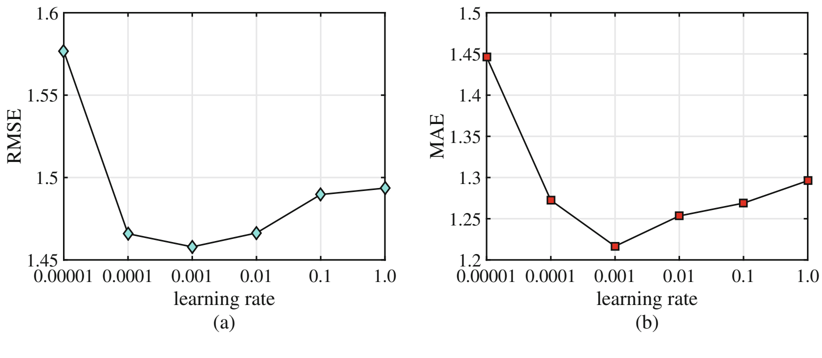


Fig. 6. Impact of learning rate

Impact of Learning Rate. Learning rate is an important factor in machine learning and deep learning. How to adjust the learning rate is one of the key steps to train a model. When solving the minimum problem with gradient descent, the gradient can be neither too large nor too small. If the learning rate is too large, it will hinder the convergence. That is, the loss shocks near the extreme point. If the learning rate is too small, it will lead to the inability to quickly find a good downward direction. Namely, when the number of iterations increases, the loss keeps unchanged. In this experiment, we set learning rate from 0.00001 to 1.0. As shown in Fig. 6, the RMSE and MAE firstly present a downward trend, and then an upward trend. The best performance achieves when the learning rate is set as 0.001. The result indicates that a relatively larger learning rate is helpful to improve the recommendation accuracy.

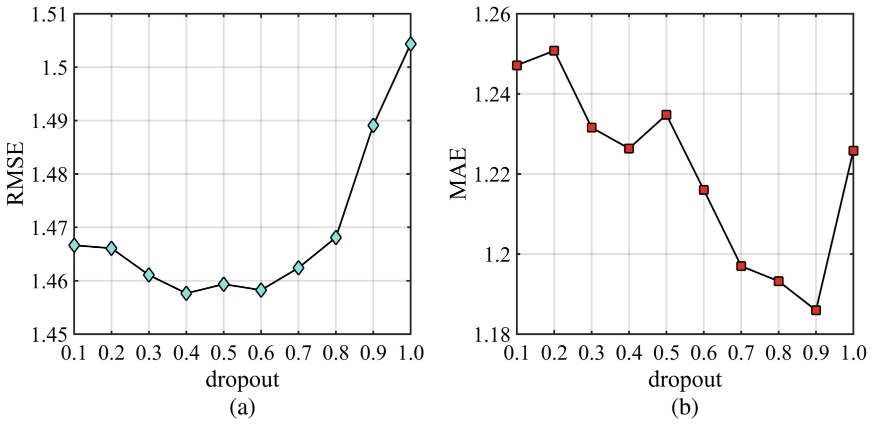


Fig. 7. Impact of dropout

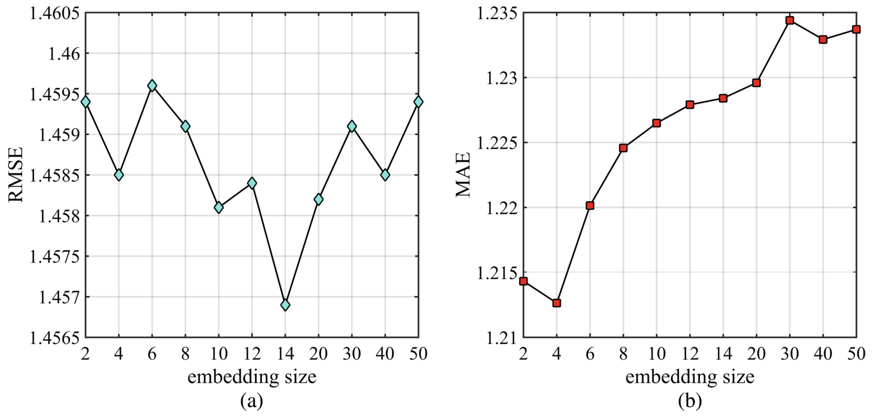


Fig. 8. Impact of embedding size

Impact of Dropout. Dropout [13] refers to a probability that each neural network unit may be discarded from the network in the process of DNN training.

Dropout is a regularization technique for reducing overfitting in neural networks. We set the dropout from 0.1 to 1.0 with a step value of 0.1. As shown in Fig. 7, the RMSE and MAE increase after the first decline. Our model can reach the best performance when the dropout is set as 0.4 for RMSE and 0.9 for MAE. The observation demonstrates that adding reasonable randomness to the model can enhance the effectiveness of the model.

Impact of Embedding Size. Embedding size d determines how many embedding features are extracted from feature vectors. Namely, the dimensionality of embedding vector. To study the impact of embedding size, we vary it from 2 to 50. It can be observed from Fig. 8, when increasing the embedding size from 2 to 50 the RMSE and MAE value decline firstly, and then grow up. The lowest RMSE can be obtained when the embedding size is 14. For MAE, the best performance can be acquired when the embedding size is 4. The result shows that a relatively smaller embedding size is better to achieve a good model. It is reasonable because larger embedding size means higher computational cost.

4 Related Work

To the best of our knowledge, this paper is the first work to exploit weighted meta-graph and HIN for mobile app recommendation. The work of this paper is mainly related to two aspects: mobile app recommendation and HIN for recommendation. Several studies referred to these two areas will be introduced in the following.

Mobile App Recommendation. Most existing investigations usually consider the context information, user's interest, privacy, security and other characteristics for mobile app recommendation. Liang et al. [8] proposed a broad learning approach for context-aware app recommendation with tensor analysis. Yin et al. [18] proposed a mobile sparse additive generative model to recommend apps by considering both user interests and category-aware user privacy preferences. Huang et al. [5] presented a skewness-based framework for mobile app permission recommendation and risk evaluation. Zhu et al. [21] proposed a novel location-based probabilistic factor analysis mechanism to help people get an appropriate mobile app. Cao et al. [2] proposed a novel version-sensitive mobile app recommendation framework by jointly exploring the version progression and dual-heterogeneous data.

HIN for Recommendation. HIN is widely applied to movie recommendation, research collaborator recommendation, product recommendation and social recommendation. Shi et al. [12] proposed a weighted HIN and weighted meta-path based personalized recommendation method to predict the rating scores of users on items. Shi et al. [11] proposed a matrix factorization based dual regularization framework to integrate different types of information. Yang et al. [17] generated high quality expert's profiles and proposed an approach based on the multiple heterogeneous network features. Zhao et al. [19] proposed a group lasso regularized FM to automatically learn from the observed ratings to effectively select useful meta-graph based features. Zheng et al. [20] proposed a new dual similarity regularization to impose the constraint on users and items with high and low similarities simultaneously. However, to the best of our knowledge, most of the existing studies do not explored weighted meta-graph and HIN in the scenario of mobile app recommendation. Weighted meta-graph takes into account the rating scores to capture the subtle semantics. HIN contains rich structure and semantic information, which can satisfy different kinds of requirements of users and generate better recommendation results.

5 Conclusion

This paper presents a weighted meta-graph based approach for mobile app recommendation. Firstly, we utilize weighted meta-graph, which considers the rating scores on links and captures more complex semantics, to measure the semantic similarity between users and apps. Then, we employ non-negative matrix factorization to obtain user latent features and app latent features. Finally, we exploit DeepFM model to predict the rating score of users on apps by leveraging the concatenated user and app latent features. Furthermore, we conduct a series of comprehensive experiments on two real-world datasets. First of all, we compare WMGRec model with other baseline approaches under different training data density, which indicates that our WMGRec model generates better recommendations and improves the recommendation accuracy. And then, we study how parameters (i.e., learning rate, dropout and embedding size) impact the recommendation results.

In the future, we attempt to collect more attribute information (e.g., permission and category) of apps and social relation information (e.g., friends relation and trust relation) of users to enrich features and semantics in the network.

Acknowledgments. The paper was supported by the National Key Research and Development Program (2017YFB0202200), the National Natural Science Foundation of China (61702568, U1711267), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (No.2017ZT07X355) and the Fundamental Research Funds for the Central Universities under Grant (17lgpy117).

References

1. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**(Jan), 993–1022 (2003)
2. Cao, D., et al.: Version-sensitive mobile app recommendation. Inf. Sci. **381**, 161–175 (2017)
3. Fang, Y., Lin, W., Zheng, V.W., Wu, M., Chang, K.C.C., Li, X.L.: Semantic proximity search on graphs with metagraph-based learning. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 277–288. IEEE (2016)

4. Guo, H., Tang, R., Ye, Y., Li, Z., He, X.: DeepFM: a factorization-machine based neural network for CTR prediction. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1725–1731 (2017)
5. Huang, K., Han, J., Chen, S., Feng, Z.: A skewness-based framework for mobile app permission recommendation and risk evaluation. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 252–266. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_16
6. Huang, Z., Zheng, Y., Cheng, R., Sun, Y., Mamoulis, N., Li, X.: Meta structure: computing relevance in large heterogeneous information networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 1595–1604. ACM (2016)
7. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: Conference on Neural Information Processing Systems (NIPS), pp. 556–562 (2001)
8. Liang, T., He, L., Lu, C.T., Chen, L., Yu, P.S., Wu, J.: A broad learning approach for context-aware mobile application recommendation. In: International Conference on Data Mining (ICDM), pp. 955–960. IEEE (2017)
9. Rendle, S.: Factorization machines. In: International Conference on Data Mining (ICDM), pp. 995–1000. IEEE (2010)
10. Shi, C., Li, Y., Zhang, J., Sun, Y., Philip, S.Y.: A survey of heterogeneous information network analysis. IEEE Trans. Knowl. Data Eng. **29**(1), 17–37 (2017)
11. Shi, C., Liu, J., Zhuang, F., Philip, S.Y., Wu, B.: Integrating heterogeneous information via flexible regularization framework for recommendation. Knowl. Inf. Syst. **49**(3), 835–859 (2016)
12. Shi, C., Zhang, Z., Luo, P., Yu, P.S., Yue, Y., Wu, B.: Semantic path based personalized recommendation on weighted heterogeneous information networks. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM), pp. 453–462. ACM (2015)
13. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
14. Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T.: PathSim: meta path-based top-k similarity search in heterogeneous information networks. Proc. VLDB Endow. **4**(11), 992–1003 (2011)
15. Willmott, C.J., Matsuura, K.: Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Clim. Res. **30**(1), 79–82 (2005)
16. Wu, J., Chen, L., Xie, Y., Zheng, Z.: Titan: a system for effective web service discovery. In: Proceedings of the 21st International Conference on World Wide Web, pp. 441–444 (2012)
17. Yang, C., Sun, J., Ma, J., Zhang, S., Wang, G., Hua, Z.: Scientific collaborator recommendation in heterogeneous bibliographic networks. In: Hawaii International Conference on System Sciences (HICSS), pp. 552–561. IEEE (2015)
18. Yin, H., Chen, L., Wang, W., Du, X., Nguyen, Q.V.H., Zhou, X.: Mobi-SAGE: a sparse additive generative model for mobile app recommendation. In: International Conference on Data Engineering (ICDE), pp. 75–78. IEEE (2017)
19. Zhao, H., Yao, Q., Li, J., Song, Y., Lee, D.L.: Meta-graph based recommendation fusion over heterogeneous information networks. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 635–644. ACM (2017)

20. Zheng, J., Liu, J., Shi, C., Zhuang, F., Li, J., Wu, B.: Dual similarity regularization for recommendation. In: Bailey, J., Khan, L., Washio, T., Dobbie, G., Huang, J.Z., Wang, R. (eds.) PAKDD 2016. LNCS (LNAI), vol. 9652, pp. 542–554. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31750-2_43
21. Zhu, K., Zhang, L., Pattavina, A.: Learning geographical and mobility factors for mobile application recommendation. IEEE Intell. Syst. **32**(3), 36–44 (2017)

Temporal-Sparsity Aware Service Recommendation Method via Hybrid Collaborative Filtering Techniques

Shunmei Meng[1,2](✉), Qianmu Li[1](✉), Shiping Chen[3], Shui Yu[4], Lianyong Qi[5], Wenmin Lin[6], Xiaolong Xu[7], and Wanchun Dou[2]

[1] Department of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
{mengshunmei, qianmu}@njust.edu.cn

[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
douwc@nju.edu.cn

[3] CSIRO Data61, Sydney, Australia
shiping.chen@data61.csiro.au

[4] School of Software, University of Technology Sydney, Ultimo, NSW, Australia
shui.yu@uts.edu.au

[5] School of Information Science and Engineering, Qufu Normal University, Jining, China
lianyongqi@gmail.com

[6] Department of Computer Science, Hangzhou Dianzi University, Hangzhou, China
linwenmin@hdu.edu.cn

[7] School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China
xlxu@nuist.edu.cn

Abstract. Temporal information has been proved to be an important factor to recommender systems. Both of user behaviors and QoS performance of services are time-sensitive, especially in dynamic cloud environment. Furthermore, due to the data sparsity problem, it is still difficult for existing recommendation methods to get the similarity relationships between services or users well. In view of these challenges, in this paper, we propose a temporal-sparsity aware service recommendation method based on hybrid collaborative filtering (CF) techniques. Specifically, temporal influence is considered into classical neighborhood-based CF model by distinguishing temporal QoS metrics from stable QoS metrics. To deal with the sparsity problem, a time-aware latent factor model based on a tensor decomposition model is applied to mine the temporal similarity between services. Finally, experiments are designed and conducted to validate the effectiveness of our proposal.

Keywords: Service recommendation · Temporal · Sparsity
Collaborative filtering · CP decomposition

1 Introduction

With the rapid growth of public cloud offerings, service-oriented systems are becoming large-scale and complex. It becomes more difficult for users to find the appropriate services. Recommendation has been a hot research topic with the rapid growth of information. There have been many efforts done both in industry and academia on developing effective recommender systems [1, 2]. Collaborative filtering has been proved to be one of the most successful recommendation methods to deal with information overload in real world [3]. However, most of existing CF recommendation models provide recommendations based on rating models directly without considering context information, such as temporal influence. Actually, because of business competition and temporal dynamics of cloud environment, most cloud services are time-sensitive.

Recently, there have been some studies [4, 5] that investigate the importance of temporal influence in recommender systems, which mainly focus on the temporal dynamics on user preferences. Nevertheless, few works pay attention to temporal influence to Quality of Service (QoS). Compared with traditional internet services, QoS of cloud services is more sensitive to time due to the dynamics of cloud environment. The correlation among cloud services and user behaviors over long time may be weakened because of temporal dynamics. Thus it is important to consider the temporal impact on QoS of cloud services. Besides, the data sparsity problem is always a serious shortcoming to affect the performance of recommendation methods. Due to the sparsity problem, existing similarity models in collaborative methods fail to capture the similarity relationships between users or services effectively.

Based on these observations, in this paper, a temporal-sparsity aware service recommendation method via hybrid CF techniques is proposed. Firstly, temporal influence is considered into neighborhood-based CF recommendation model by distinguishing temporal QoS metrics from the stable QoS metrics. To address the data sparsity problem and mine the similarity relationships between services over time adaptively, a time-aware latent factor model based on tensor decomposition is presented. Finally, experiments based on real-world service dataset are designed and conducted to validate the effectiveness of our proposal.

2 Problem Formulation

In this section, some important concepts and definitions are presented. Firstly, to mining the temporal features of QoS, the QoS metrics of services are divided into two parts, i.e., stable QoS metrics and temporal QoS metrics:

Stable QoS Metrics and Temporal QoS Metrics: Stable QoS metrics are the metrics evolving at a rather slow speed, which are regular features (such as availability). Temporal QoS metrics are the metrics that have clear temporal features and present dynamic trends over time, which are dynamic features (such as price and duration).

To analyze the temporal influence to recommendation performance, the history time period of rating dataset is divided into K time slots, i.e., $\{T_1, T_2, \ldots., T_K\}$. The temporal QoS metrics are distinguished from stable QoS metrics by measuring the fluctuation of the rating for each individual QoS metric over time. The fluctuation of QoS metric q_h is measured by the variance of QoS rating for q_h during the K time slots, which is defined in Eq. (1):

$$F(q_h) = \frac{1}{K \cdot |S|} \sum_{s=1}^{|S|} \sum_{k=1}^{K} (\overline{rq}_{hk}^{s} - \overline{rq}_{h}^{s})^2 \tag{1}$$

Where $|S|$ is the number of services in service set S, $\overline{rq}_{hk}^{s}$ is the average rating of metric q_h in time slot T_k of service s, and $\overline{rq}_{h}^{s}$ is the overall average rating of q_h in total time period of service s.

A fluctuation threshold δ_t is given to decide whether a QoS metric is stable or temporal. If $F(q_h) > \delta_t$, then q_h can be seen as a temporal QoS metric, otherwise q_h is considered as a stable metric. So the QoS metric vector $\boldsymbol{Q} = [q_1, q_2, \ldots, q_H]$ can be divided into two parts: stable QoS vector $\boldsymbol{SQ} = [sq_1, sq_2, \ldots, sq_a]$ and temporal QoS vector $\boldsymbol{TQ} = [tq_1, tq_2, \ldots, tq_b]$ $(a + b = H)$.

Global Nearest Neighbors: For a candidate service s, its global nearest neighbors are the services that have the most similar QoS ratings with service s at the whole time period of the history records.

Temporal Nearest Neighbors: The temporal nearest neighbors of service s are the services that have the most similar QoS ratings with service s at time slot T_k and its similar time slots, i.e., $T_{sim}(k)$. (The definition of $T_{sim}(k)$ is given in Sect. 3.1).

3 Temporal-Sparsity Aware Service Recommendation Method

In this paper, to make more appropriate recommendations from time-sensitive cloud services, a temporal-sparsity aware service recommendation method based on hybrid CF techniques is proposed. Our method can be designed as a three-phase process: (1) Time slot aggregation, (2) Similarity calculation and similarity prediction, (3) Rating prediction, which are described in detail in the following.

3.1 Time Slot Aggregation

To make more accurate recommendations, we make predictions based on the QoS ratings at the target time slot. However, the related rating dataset of the target user at the target time slot maybe very spare. We provide an aggregation strategy to merge the similar time slots for the target time slot.

A temporal similar coefficient $\varphi(T_i, T_j)$ is defined to measure the temporal closeness of the temporal QoS metrics between time slot T_i and T_j. As shown in Eq. (4), $\varphi(T_i, T_j) \in [0, 1]$ is defined based on Pearson Correlation Coefficient (PCC). The larger $\varphi(T_i, T_j)$ is, the closer the temporal features of candidate services between T_i and T_j is. Given a threshold θ to determine whether two time slots are similar.

If $\varphi(T_i, T_j) \geq \theta$, then time slots T_i and T_j can be considered to be similar. The similar time slots of T_i is defined as $T_{sim}(k)$ and $T_k \subset T_{sim}(k)$.

$$\varphi(T_i, T_j) = \frac{\sum_{s \in S(T_i) \cap S(T_j)} (\overline{\boldsymbol{RT}}_{is} - \overline{\boldsymbol{RT}}_i) \bullet (\overline{\boldsymbol{RT}}_{js} - \overline{\boldsymbol{RT}}_j)}{\sqrt{\sum_{s \in S(T_i) \cap S(T_j)} \|\overline{\boldsymbol{RT}}_{is} - \overline{\boldsymbol{RT}}_i\|^2} \cdot \sqrt{\sum_{s \in S(T_i) \cap S(T_j)} \|\overline{\boldsymbol{RT}}_{js} - \overline{\boldsymbol{RT}}_j\|^2}} \tag{2}$$

In Eq. (2), $S(T_i) \cap S(T_j)$ is the set of coinvoked services by users at time slot T_i and T_j, $\overline{\boldsymbol{RT}}_{is}$ is the average temporal QoS-rating vector of service s at T_i, $\overline{\boldsymbol{RT}}_i$ is the average temporal QoS-rating vector of all candidate services in $S(T_i) \cap S(T_j)$ at T_i.

3.2 Similarity Calculation and Similarity Prediction

As presented in Sect. 2, there are two kinds of nearest neighbors in our work, i.e., global nearest neighbors (denoted as S_{GNN}) and temporal nearest neighbors denoted as S_{TNN}). The global nearest neighbors of a service s can be decided by Eq. (3):

$$sim_{sv}^{GNN} = \frac{\sum_{u \in U(s) \cap U(v)} \left(\boldsymbol{RQ}_{us} - \overline{\boldsymbol{RQ}}_s\right) \bullet \left(\boldsymbol{RQ}_{uv} - \overline{\boldsymbol{RQ}}_v\right)}{\sqrt{\sum_{u \in U(u) \cap U(v)} \|\boldsymbol{RQ}_{us} - \overline{\boldsymbol{RQ}}_s\|^2 \times \|\boldsymbol{RQ}_{uv} - \overline{\boldsymbol{RQ}}_v\|^2}} \tag{3}$$

where $U(s) \cap U(v)$ is the set of users that rated both service s and service v in the whole time period. Here, we give a preset similarity threshold δ_{sim}, then the services that have similarity with service s no less than δ_{sim} can be considered as global nearest neighbors of service s.

Similarly, the temporal nearest neighbors of service s can be determined by Eq. (4). The temporal nearest neighbors of service s at time slot T_k are the services that have similarity with service s no less than δ_{sim} at time slot T_k and its similar time slots, i.e., $T_{sim}(k)$.

$$sim_{sv}^{TNN}(T_k) = \frac{\sum_{u \in U_{sv}(T_k)} \left(\boldsymbol{RQ}_{us} - \overline{\boldsymbol{RQ}}_s\right) \bullet \left(\boldsymbol{RQ}_{uv} - \overline{\boldsymbol{RQ}}_v\right)}{\sqrt{\sum_{u \in U_{sv}(T_k)} \|\boldsymbol{RQ}_{us} - \overline{\boldsymbol{RQ}}_s\|^2 \times \|\boldsymbol{RQ}_{uv} - \overline{\boldsymbol{RQ}}_v\|^2}} \tag{4}$$

where $U_{sv}(T_k) = \{u | u \in U(s) \cap U(v) \,\&\, t_{us} \in T_{sim}(k) \,\&\, t_{uv} \in T_{sim}(k)\}$, which is the set of users that rated both service s and service v at $T_{sim}(k)$.

To solve sparsity problem further, a time-aware latent factor model based on CANDECOMP/PARAFAC (CP) decomposition [6] is applied to predict temporal similarity between services. The triadic relations among services, neighbors and time features are formulated as a three-dimensional similarity tensor $\boldsymbol{Sim} \in \Re^{M \times M \times K}$. The element in tensor $\boldsymbol{Sim}$ is denoted as sim_{ijk}, which represents the temporal similarity of service i and service j at time slot T_k. The tensor $\boldsymbol{Sim}$ is symmetric as $sim_{ijk} = sim_{jik}$. Then, based on the CP decomposition model, the tensor $\boldsymbol{Sim} \in \Re^{M \times M \times K}$ can be decomposed into a sum of component rank-one tensors:

$$\boldsymbol{Sim} \approx \sum_{r=1}^{R} s_r \circ s_r \circ t_r \tag{5}$$

where R is actually the rank of tensor ***Sim***, which is defined as the smallest number of rank-one tensors. s_r and t_r represent the latent factor vectors associated with service and time, respectively.

Then the temporal similarity can be predicted by Eq. (6). The observed temporal similarity can be broken into two components: biases and service-neighbor-time interaction. The bias component contains the overall average similarity μ and time bias b_{tk}. In similarity prediction, we didn't consider service bias.

$$\hat{s}im_{ijk} = \mu + b_{tk} + \sum_{r=1}^{R} s_{ir} \circ s_{jr} \circ t_{kr} \tag{6}$$

To learn the involved parameter b_{tk} and the involved vectors, i.e., s_{ir}, s_{jr} and t_{kr}, we minimize the regularized squared error function:

$$\min_{b,s,t} \sum_{(i,j,k) \in Train} \left\| sim_{ijk} - \hat{s}im_{ijk} \right\|^2 + \lambda W \tag{7}$$

where $Train$ is the set of the (i, j, k) pairs for sim_{ijk}, which is known as the training set. sim_{ijk} is obtained by Eq. (4). $W = b_{tk}^2 + \|s_{ir}\|^2 + \left\|s_{jr}\right\|^2 + \|t_{kr}\|^2$, which is applied to regularize the learned parameters to avoid overfitting and the constant λ controls the extent of regularization. In this paper, we adopt stochastic gradient descent to solve Eq. (7) by looping through all similarity values in the training set.

3.3 Rating Prediction

The prediction of target user i on candidate service j at T_k (denoted as r_{ijk}) is defined in Eq. (8). The prediction consists of two parts, i.e., prediction based on the global nearest neighbors and prediction based on the temporal nearest neighbors, which are combined by a weight coefficient α. In our proposal, α is set as a/H, and $1 - \alpha = b/H$.

$$r_{ijk} = \alpha \cdot \left(\bar{r}_j + \frac{\sum_{s \in S_{GNN}(j)} (r_{is} - \bar{r}_s) \cdot sim_{is}^{GNN}}{\sum_{s \in S_{GNN}(j)} \left| sim_{is}^{GNN} \right|} \right) + (1 - \alpha) \cdot \left(\bar{r}_j^k + \frac{\sum_{s' \in S_{TNN}(j,T_K)} (r_{is'} - \bar{r}_{s'}) \cdot sim_{is'}^{TNN}(T_k)}{\sum_{s' \in S_{TNN}(j,T_K)} \left| sim_{is'}^{TNN}(T_k) \right|} \right) \tag{8}$$

Where $\bar{r}_j^k$ is the average rating of service j at the aggregated time slot $T_{sim}(k)$, $S_{GNN}(j)$ and $S_{TNN}(j, T_k)$ respectively represent service *j's* global and temporal nearest neighbor set where the services have been used by user i, and r_{is} represents the rating of user i on service s, $\bar{r}_s$ is the overall average rating of services in $S_{GNN}(j)$, and $\bar{r}_{s'}$ is the overall average rating of services in $S_{TNN}(j,\, T_k)$.

4 Experiment

4.1 Experimental Setup

Datasets: We employ a real-world service dataset to simulate the history QoS ratings of services in the cloud market. The dataset is collected from a well-known travel review site (www. tripadvisor.com). In our experiment, we use five-fold cross validation approach, and the dataset is split into 20% test data and 80% train data.

Comparative Approaches: To evaluate the effectiveness of our proposal, we compare our method with four other approaches: Item-based CF algorithm using PCC (IPCC) [7], regularized Singular Value Decomposition (RSVD) [8], a temporal QoS-aware web service recommendation method via tensor factorization (TWS) [9], and a time-aware hybrid collaborative recommendation method (THC) [10].

Performance Metrics: Four widely used metrics are applied to evaluate the statistical accuracy of recommendation approaches: mean absolute error (MAE), root-mean-square error (RMSE) [11], precision and recall [12].

4.2 Experimental Result

In our experiment, to evaluate the recommendation accuracy of our proposal, we compare our method (denoted as TSSRec) with four approaches. Figure 1 shows the best prediction performance (performance under the optimal parameter settings) of all methods in MAE and RMSE. We can see that both of MAE and RMSE of TSSRec are better than other four comparative approaches.

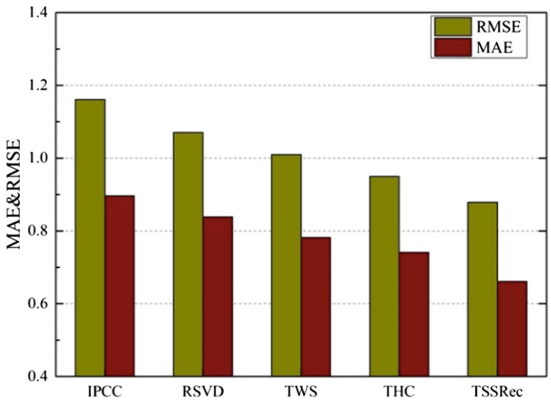


Fig. 1. Comparison of prediction performance in MAE and RMSE.

Figure 2 shows the performance of Top-N (N = 3, 5, 7) recommendations. Figure 2(a) and (b) present the precision@N and recall@N performance of all methods, respectively. It can be found that TSSRec also outperform other methods in Top-N (N = 3, 5, 7) recommendation accuracy.

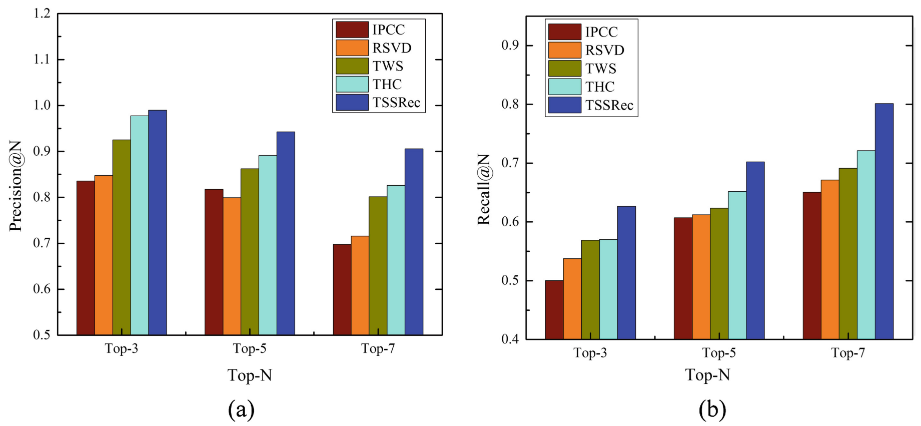


Fig. 2. Comparison of Top-N recommendation accuracy. (a) Comparison in precision@N. (b) Comparison in recall@N.

From the experimental results above, we can see that temporal influence is important to service recommendation in dynamic cloud environment. And our method achieves considerable improvement on recommendation accuracy compared to other comparative approaches.

5 Related Work

In the past, there has been some research work which integrates temporal influence into collaborative recommendation methods. Hu et al. [4] integrate temporal information into both similarity measurement and QoS prediction by considering the time gap between recommendation time and the occurring time of previous rating. The research [5] considers time information into recommendations based on probabilistic models. Wang et al. [13] adapt matrix factorization techniques to learn user-group affinity based on two different implicit engagement metrics. Recently, more and more influence factors, such as location influence, social influence, are considered into recommender systems. Lian et al. [14] propose a collaborative location recommendation framework to exploit the relations between users, activities and locations. The literature [15] focuses on the problem of joint modeling user check-in behaviors for real-time POI recommendation. Wang et al. [16] propose a spatial-temporal QoS prediction method where the temporal QoS prediction is formulated as a generic regression problem and a zero-mean Laplace prior distribution assumption is made on the residuals of QoS prediction.

6 Conclusion

In this paper, a temporal-sparsity aware service recommendation method based on hybrid CF techniques is proposed. Specifically, temporal influence is considered into classical neighborhood-based CF model by distinguishing temporal QoS metrics from stable QoS metrics. Accordingly, stable nearest neighbors and temporal nearest neighbors are defined. Then a time-aware latent factor model based on CP decomposition is integrated into neighborhood model to mine the temporal similarity relationships between services to address the data sparsity problem. Finally, experiments based on real-world service dataset are conducted to demonstrate the effectiveness of our proposal. In our future work, we will do further research in collaborative recommendation models based on multi-model integration and consider more context information in dynamic cloud environment.

Acknowledgment. This paper is partially supported by the National Science Youth Foundation of China under Grant No. 61702264, the Open Research Project of State Key Laboratory of Novel Software Technology (Nanjing University) under Grant No. KFKT2017B07.

References

1. Yang, J., Liu, C., Teng, M., Chen, J., Xiong, H.: A unified view of social and temporal modeling for B2B marketing campaign recommendation. IEEE Trans. Knowl. Data Eng. **30**(5), 810–823 (2018)
2. Zhang, Y., et al.: Personalized quality centric service recommendation. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 528–544. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_39
3. Meng, S., Dou, W., Zhang, X., Chen, J.: KASR: a keyword-aware service recommendation method on mapreduce for big data applications. IEEE Trans. Parallel Distrib. Syst. **25**(12), 3221–3231 (2014)
4. Hu, Y., Peng, Q., Hu, X., Yang, R.: Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering. IEEE Trans. Serv. Comput. **8**(5), 782–794 (2015)
5. Zhang, J.D., Chow, C.Y.: TICRec: a probabilistic framework to utilize temporal influence correlations for time-aware location recommendations. IEEE Trans. Serv. Comput. **9**(4), 633–646 (2016)
6. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
7. Linden, G., Smith, B., York, J.: Amazon. com recommendations: item-to-item collaborative filtering. IEEE Internet Comput. **7**(1), 76–80 (2003)
8. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 42–49 (2009)
9. Zhang, W., Sun, H., Liu, X., Guo, X.: Temporal QoS-aware web service recommendation via non-negative tensor factorization. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 585–596. ACM (2014)
10. Meng, S., et al.: A temporal-aware hybrid collaborative recommendation method for cloud service. In: 2016 IEEE International Conference of Web Services (ICWS), pp. 252–259. IEEE (2016)

11. Chai, T., Draxler, R.R.: Root mean square error (RMSE) or mean absolute error (MAE)?– arguments against avoiding RMSE in the literature. Geosci. Model. Dev. **7**(3), 1247–1250 (2014)
12. Carullo, G., Castiglione, A., De Santis, A., Palmieri, F.: A triadic closure and homophily-based recommendation system for online social networks. World Wide Web **18**(6), 1579–1601 (2015)
13. Wang, X., Donaldson, R., Nell, C., Gorniak, P., Ester, M., Bu, J.: Recommending groups to users using user-group engagement and time-dependent matrix factorization. In: AAAI, pp. 1331–1337 (2016)
14. Lian, D., et al.: Scalable content-aware collaborative filtering for location recommendation. IEEE Trans. Knowl. Data Eng. **30**(6), 1122–1135 (2018)
15. Yin, H., Cui, B., Zhou, X., Wang, W., Huang, Z., Sadiq, S.: Joint modeling of user check-in behaviors for real-time point-of-interest recommendation. ACM Trans. Inf. Syst. (TOIS) **35**(2), 11 (2016)
16. Wang, X., Zhu, J., Zheng, Z., Song, W., Shen, Y., Lyu, M.R.: A spatial-temporal QoS prediction approach for time-aware web service recommendation. ACM Trans. Web (TWEB) **10**(1), 7 (2016)

QoS-Aware Web Service Recommendation with Reinforced Collaborative Filtering

Guobing Zou[1,2], Ming Jiang[1,2], Sen Niu[3], Hao Wu[1], Shengye Pang[1], and Yanglan Gan[4](✉)

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, China
guobingzou@gmail.com, jimmyme520@gmail.com

[2] Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai, China

[3] College of Computer and Information, Shanghai Polytechnic University, Shanghai, China

[4] School of Computer Science and Technology, Donghua University, Shanghai, China
ylgan@dhu.edu.com

Abstract. With the overwhelming increase of web services on the Internet, how to accurately perform QoS prediction has played a key role in service recommendation. Recently, three kinds of approaches have been presented on service QoS prediction based on collaborative filtering (CF), including user-intensive, service-intensive and their combination. However, the deficiency of current approaches is that all of the services invoked by target user (or all of the users who invoked target service) are applied to calculate average QoS, without the reduction to those dissimilar with target service (or target user). In this paper, we propose a reinforced collaborative filtering approach, where both similar users and services are integrally considered into a singleton CF. The experiments are conducted on a large-scale dataset called WS-DREAM, involving 5,825 real-world Web services in 73 countries and 339 service users in 30 countries. The experimental results demonstrate that our approach for QoS prediction outperforms the competing approaches.

Keywords: Service-oriented computing · Service recommendation QoS · Collaborative filtering

1 Introduction

With the rapid development of network technology and increasing demands on service-oriented application integration, more and more software developers publish their softwares as web services on the Internet. It accelerates the interoperable machine-to-machine interaction and greatly promotes the advancements on service discovery, optimum selection, automated composition and recommendation. However, as the overwhelming explosion on the number of the registered

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 430–445, 2018.
https://doi.org/10.1007/978-3-030-03596-9_31

services, many of them merge with the same or similar functionality in a service repository. That tends to be a labor-intensive challenging task for service requesters to choose their desired services from a large-scale service repository. Quality of Service (QoS) as a non-functionality criterion has been widely applied as a key factor to differentiate those functionally equivalent web services. In many cases, however, a target user has not invoked a target service. That is, there are very few historical QoS invocations, leading to difficulty in recommending appropriate services to a target user. In this application scenario, how to designing an effective approach of unknown QoS prediction for service recommendation has become a critical research issue to be addressed.

Applying the techniques of collaborative filtering in recommender system, correlative research efforts [1–7,9,10,12–14] have been made on service QoS prediction. They can be grouped into model-based and memory-based approaches. Matrix factorization (MF) [4–6,12,14] as the typical model-based technique has been used for missing QoS prediction, where the original user-service QoS matrix with sparsity is converted into two low dimensional matrices. Memory-based collaborative filtering approaches for QoS prediction are usually classified into user-based [2], service-based [7], and their linear combination with a confidence weight [13]. They leverage similar users (or services) for a target user (a target service) to predict unknown QoS.

Although these existing approaches can assist and facilitate QoS prediction of web services, the deficiency is that they still cannot reach the accuracy satisfying service requesters' demands. More specifically, the traditional collaborative filtering algorithm consists of two major components, including average QoS calculation and deviation migration calculation. It is observed that the existing approaches either directly employ all the services that a target user has invoked, or all the users who have invoked a target service, when performing average QoS calculation. That is, they did not consider the discrepancies between all of the services invoked by a target user and a target service. Simultaneously, the differences between all of the users who invoked a target service and a target user have also not been considered. That kind of collaborative filtering algorithm decreases the purity on average QoS calculation and affects the accuracy of QoS prediction. To partially solve this issue, enhanced collaborative filtering algorithms have been proposed with the help of external context information, such as a user's geographical location. However, it is difficult to obtain context information in real-world applications. Therefore, how to design an effective collaborative filtering algorithm for QoS prediction without any external heuristic information support has become a challenging research issue.

To handle above issue, we proposed a novel reinforced collaborative filtering algorithm (RECF) for QoS prediction. When performing user-based RECF, the implicit context information from the service side is obtained by a ratio-based similarity calculation method [10] in order to eliminate those dissimilar services with the target one. It has been integrally taken into account for calculating average QoS. In this way, the advantage of our approach is that user-based and service-based similarity information can be integrated into a singleton collaborative filtering algorithm without additional parameters learning and estimation. Similarly, service-based RECF takes those dissimilar users with the target one as hidden factors to promote QoS prediction.

To test the performance of QoS prediction, extensive experiments are conducted on a large-scale real-world dataset called WS-DREAM, involving 5,825 real-world web services in 73 countries and 339 service users in 30 countries. We compare our approach with seven existing collaborative filtering-based methods on QoS prediction accuracy. The experimental results demonstrate that our approach can outperform those competing approaches.

The main contributions of this paper are summarized as follows.

- We propose a novel reinforced collaborative filtering framework for QoS prediction, where the similarity calculation from user-perspective and service-perspective are integrally integrated into a singleton CF method, instead of a linear combination by weighted parameters.
- We propose an approach for optimizing the average QoS calculation. By applying ratio-based similarity computation, implicit features among users or services can be discovered without any external context information.
- We design and implement a prototype system and conduct extensive experiments on a real-world dataset called WS-DREAM. The experimental results demonstrate that our approach of QoS prediction is superior to existing completing methods in terms of accuracy.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 elaborates our approach of reinforced collaborative filtering for service QoS prediction. Section 4 shows the experimental evaluation. Finally, Sect. 5 concludes the paper.

2 Related Work

According to [2], collaborative filtering algorithms for service QoS prediction can be mainly divided into two categories, including model-based and memory-based approaches. We review the advancement on collaborative filtering algorithms for QoS prediction that is highly related with our work.

In model-based collaborative filtering approaches, matrix factorization [4,5,12,14] is the typical technique for predicting missing QoS values of web services. It turns an original matrix into two low dimensional matrices to reveal pivot features that can be used to estimate an unknown QoS value, where a target user has not ever invoked a target service.

In memory-based collaborative filtering approaches, it consists of average QoS calculation step and deviation migration step. This kind of approach has three different variations, including user-based CF [2], service-based CF [7] and their linear combination with confidence weights [13]. Based on historical QoS invocation logs, neighborhood users or services can be chosen by similarity calculation in deviation migration step. Moreover, correlative research works concentrated on how to more accurately quantify the correlation between users or services. The authors in [9] investigated the QoS distribution characteristics and proposed a novel collaborative filtering approach for QoS prediction. It normalizes the QoS values to the same range and then unifies the similarity in different multi-dimensional vector spaces. The authors in [10] proposed a novel ratio-based similarity approach to measure neighborhood users and services. Compared with PCC similarity [8] and cosine similarity [7], it is more precise for predicting the unknown QoS of web services.

To further improve the prediction accuracy, several recent approaches have been proposed by adding context information as heuristic knowledge during the procedure of deviation migration. The authors in [4,12,14] took the geography location of the users or services, the provider of the services and the infrastructure information into account to extract more accurate neighborhood users or services. The authors in [1] found out a group of trusted users or services based on pareto dominance comparison. In addition, time series was used to predict the tendency information of missing QoS prediction of web services [3]. The authors in [11] proposed a hybrid approach of QoS prediction via combining the information from geography location and time series.

From the above investigation, we observe that the existing methods mainly focused on how to improve the accuracy of similarity calculation in deviation migration. However, they rarely made the reduction to those dissimilar services invoked by a target user (or dissimilar users who invoked a target service) in average QoS calculation. Although some of the recent research has borrowed external context information as heuristic knowledge to optimize the similar services for a target user (or similar services for a target user), they only apply them to deviation migration, instead of average QoS calculation. That affects the prediction accuracy of QoS prediction. The ideal way of overcoming the problem is to design an effective approach to eliminate those dissimilar services (or users) for more accurate QoS prediction of web services.

3 Reinforced Collaborative Filtering for QoS Prediction

In this section, we first formulates the problem of QoS prediction. Then, the framework of our approach is illustrated. Finally, we elaborate user-based and service-based reinforced collaborative filtering approach, respectively.

3.1 Problem Formulation

Definition 1 (Service Ecosystem). In a web service ecosystem, $M = <U, I, R>$, $U = \{u_1, u_2, \ldots\}$ is a set of users and $I = \{i_1, i_2, \ldots\}$ is a set of web services. $R = \{r_{u,i}\}_{m*n}$ is QoS matrix, where each entry $r_{u,i}$ represents the invocation QoS value when u invoked i.

For example, Table 1 illustrates a service ecosystem, where $U = \{u_1, u_2, \ldots, u_5\}$ and $I = \{i_1, i_2, \ldots, i_6\}$. $R = \{r_{u,i}\}_{5*6}$ is a QoS matrix and each entry represents the response time when a user u invokes a service i.

Table 1. Sample of user-service QoS matrix

U/I	i_1	i_2	i_3	i_4	i_5	i_6
u_1	0.0	0.228	0.237	0.0	0.0	0.0
u_2	0.453	0.0	0.0	0.649	0.0	0.0
u_3	0.0	0.0	0.0	5.782	0.239	0.25
u_4	0.249	0.0	5.294	0.366	0.0	0.285
u_5	0.0	0.288	0.0	0.0	0.328	5.93

Definition 2 (QoS Invocation Log). Given a service ecosystem $M = <U, I, R>$, a QoS invocation log is defined as 3-tuple $<u, i, r_{u,i}>$, where $u \in U$ is a user, $i \in I$ is a service, and $r_{u,i}$ is the QoS value when u invoked i.

Note that if an entry of a QoS invocation log is equal to 0, indicating that a user has not ever invoked a service. In such case, its QoS value need to be further predicted for use. The QoS prediction problem is defined as below.

Definition 3 (QoS Prediction Problem). Given a service ecosystem $M = <U, I, R>$, QoS prediction problem is defined as 3-tuple $Q = <M, u, i>$, where u is a target user, i is a target service and $r_{u,i}$ has no invocation log. The goal is to predict its QoS value $\widehat{r}_{u,i}$.

The solution to a QoS prediction problem is $<u, i, \widehat{r}_{u,i}>$. It indicates the predicted value when a target user invokes a target service. Based on a set of predicted QoS values, desired services can be recommended.

Definition 4 (Service Recommendation). Given a service ecosystem $M = <U, I, R>$, a target user u, and a set of functionally equivalent services I', it has no QoS invocation log from u to a service $i' \in I'$. Service recommendation problem is defined as 3-tuple $S = <M, u, I'>$ and the goal is to choose a subset of services from I' that the recommended services can be invoked with the best predicted QoS by u.

By predicting missing QoS values on each service in I', we have their predicted QoS values as

$$R_{u,I'} = \{<u, i'_1, \widehat{r}_{u,i'_1}>, <u, i'_2, \widehat{r}_{u,i'_2}>, \ldots\} \tag{1}$$

In terms of the ranking of predicted QoS values, a subset of web services can be recommended to a target user. Here, we mainly focus on predicting QoS value when a target user invokes a target service.

3.2 The Framework of Our Approach

Figure 1 illustrates the overall framework of our proposed approach. Given a target user and a target service, the procedure of task functionality goes through four stages, including finding similar users (or services), detecting neighbor services (or users), average QoS calculation, and deviation migration.

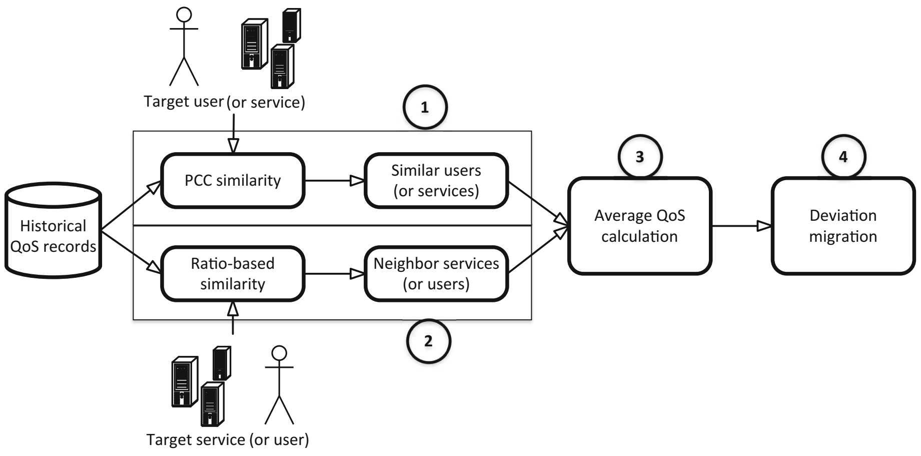


Fig. 1. The framework of our approach.

In the stage of finding similar users (or services), pearson correlation coefficient (PCC) is used to generate a group of similar users with the target user from historical QoS records (or similar services with the target service). In the stage of detecting neighbor services (or users), ratio-based similarity (RBS) is used to generate a group of neighbor services with the target service (or users with the target user). In the stage of average QoS calculation, taking all of the neighbor services (or neighbor users) as inputs, we calculate average QoS. In the stage of deviation migration, taking all of the similar users (or services) to calculate the deviation migration. Furthermore, the predicted comprehensive QoS for a target user invoking a target service can be finally calculated by integrally integrating average QoS value and deviation migration in a singleton CF.

3.3 User-Based Reinforced Collaborative Filtering

Given a target user and a target service, the procedure of user-based reinforced collaborative filtering approach for QoS prediction consists of finding similar users based on PCC, detecting neighbor services based on RBS, average QoS calculation and deviation migration.

(1) Finding Similar Users. Given a target user u, PCC [8] measures the correlation between a target user u and another user $v \in U$ in a service ecosystem.

$$Sim_{PCC}(u,v) = \frac{\sum_{i \in I_c}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_c}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i \in I_c}(r_{v,i} - \bar{r}_v)^2}} \tag{2}$$

Where $I_c = I_u \cap I_v$ is the intersection of web services that both u and v have invoked previously, and $r_{u,i}$ is a vector of QoS values of service i observed by u. $\bar{r}_u$ and $\bar{r}_v$ represent average QoS values of different services observed by u and v, respectively.

By the PCC similarity calculation, we choose the k similar users with the highest similarity degree.

$$U_{PCC} = topSimilar(u) \tag{3}$$

As a result, U_{PCC} contains a set of similar users with target user u from U. In the deviation migration step, U_{PCC} is used to aggregate deviated QoS values.

(2) Detecting Neighbor Services. Given a target user u, it corresponds to a set of services I_u that have been invoked by u. In this step, we aim to reduce these services and only remain a subset of services that hold similar historical QoS invocation values with the target service i. Here, we filter out those services dissimilar with the target service i by the ratio-based similarity (RBS) [10].

$$Sim_{RBS}(i,j) = \frac{\sum\limits_{u \in U_c} \frac{\min(r_{u,i}, r_{u,j})}{\max(r_{u,i}, r_{u,j})}}{|U_c|} \tag{4}$$

Where U_c contains a set of common users who have invoked both i and j. $|U_c|$ is the number of users in U_c. $r_{u,i}$ and $r_{u,j}$ represent the QoS values while u invoked i and j, respectively. $\min(r_{u,i}, r_{u,j})$ and $\max(r_{u,i}, r_{u,j})$ calculate the minimum and maximum QoS values between $r_{u,i}$ and $r_{u,j}$, respectively.

Note that if the ratio-based similarity between i and j tends to be 1, it reflects that almost all the users are apt to obtain the highly close QoS values when they invoked these two services. In other words, they share implicit characteristics and provide similar invocation experiences, such as deployed in close geographical locations with the same network setting or hosted by the same service provider. However, we do not require the precisely underlying context information, since the neighborhood relationship among services can be evaluated by analyzing the historical invocation QoS values.

By setting a neighbor similarity threshold θ, we detect a subset of services that have been invoked by the target user u and share highly close invocation QoS values with the target service i.

$$I'_u = \{i' \in I_u \,|Sim_{RBS}(i', i) \geq \theta\} \tag{5}$$

After filtering out all of those dissimilar services from I_u, detected neighbor services is used in average QoS calculation step.

(3) Average QoS Calculation. Given a target user u, we make a reduction to all of the services I_u invoked by u and obtain a subset of neighbor services I'_u, where each service $i' \in I'_u$ has a high similarity degree with the target service i. Obviously, the target user could obtain similar QoS values with neighbor services when invoking the target service. Naturally, we use neighbor services to initially estimate the QoS value. For each $i' \in I'_u$, we calculate an initially predicted QoS for the target user u.

$$\hat{r}^{i'}_{u,i} = \begin{cases} r_{u,i'} * Sim_{RBS}(i, i') & \text{if } \bar{i} \leq \overline{i'} \\ r_{u,i'} / Sim_{RBS}(i, i') & \text{otherwise} \end{cases} \tag{6}$$

Where $\overline{i'}$ and $\bar{i}$ represent the average QoS value of service itself, respectively. $Sim_{RBS}(i, i')$ is the ratio-based similarity of the target service i and a neighbor service i'. By calculating the predicted QoS on each neighbor service $i' \in I'_u$, we get a group of predicted QoS values if the target user u invokes the target service i.

$$AvgQoS_u = \{\hat{r}_{u,i}^{i'_1}, \hat{r}_{u,i}^{i'_2}, \cdots, \hat{r}_{u,i}^{|I'_u|}\} \tag{7}$$

Taking the ratio-based similarity between two services as weight, we calculate the average QoS value for the target user u invoking the target service i.

$$\overline{u}_{avg} = \frac{\sum_{k=1}^{|I'_u|} Sim_{RBS}(i, i'_k) * \hat{r}_{u,i}^{i'_k}}{\sum_{k=1}^{|I'_u|} Sim_{RBS}(i, i'_k)} \tag{8}$$

The predicted average QoS value $\overline{u}_{avg}$ is still a preliminary result, as we mainly rely on the QoS values from those neighbor services similar to the target service, while similar users with the target user are not taken into consideration at this point. They are integrally integrated into the QoS prediction in deviation migration step.

(4) Deviation Migration. It calculates the QoS deviation of each similar user between the QoS obtained from invoking target service and its mean value, and then accumulates and migrates these deviation values to the average QoS for a target user.

First, based on found $U_{PCC} = \{u_1, u_2, \ldots, u_k\}$ for a target user u, we use Eqs. (4)–(7) to obtain k groups of average QoS values.

$$\begin{cases} AvgQoS_{u_1} = \{\hat{r}_{u_1,i}^{i'_1}, \hat{r}_{u_1,i}^{i'_2}, \cdots, \hat{r}_{u_1,i}^{|I'_{u_1}|}\} \\ AvgQoS_{u_2} = \{\hat{r}_{u_2,i}^{i'_1}, \hat{r}_{u_2,i}^{i'_2}, \cdots, \hat{r}_{u_2,i}^{|I'_{u_2}|}\} \\ \qquad\vdots \\ AvgQoS_{u_k} = \{\hat{r}_{u_k,i}^{i'_1}, \hat{r}_{u_k,i}^{i'_2}, \cdots, \hat{r}_{u_k,i}^{|I'_{u_k}|}\} \end{cases} \tag{9}$$

After that, we use Eq. (8) to calculate and generate a set of average QoS values for each similar user in U_{PCC}.

$$AvgQoS_{U_{pcc}} = \{\overline{u}_1, \overline{u}_2, \cdots, \overline{u}_k\} \tag{10}$$

Finally, applying collaborative filtering algorithm with $\overline{u}_{avg}$ and $Avg\,QoS_{U_{PCC}}$, we make the final QoS prediction.

$$\widehat{r}_{u,i} = \overline{u}_{avg} + \frac{\sum_{m=1}^{k} Sim_{PCC}(u, u_m) * (r_{u_m,i} - \overline{u}_m)}{\sum_{m=1}^{k} Sim_{PCC}(u, u_m)} \tag{11}$$

Where $\widehat{r}_{u,i}$ is the final predicted QoS value for a target user u when invoking a target service i.

3.4 Service-Based Reinforced Collaborative Filtering

Similar to user-based reinforced collaborative filtering, the procedure of service-based RECF approach also includes the same four steps.

(1) Finding Similar Services. Given a target service i, PCC measures the correlation between a target service i and another service in a service ecosystem. we use PCC to evaluate the correlation between two services.

$$Sim_{PCC}(i,j) = \frac{\sum_{u \in U_c} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U_c} (r_{u,i} - \bar{r}_i)^2}\sqrt{\sum_{u \in U_c} (r_{u,j} - \bar{r}_j)^2}} \tag{12}$$

Where $U_c = U_i \cap U_j$ is the intersection of users who have invoked both i and j previously, and $r_{u,i}$ is a vector of QoS values of service i observed by u. $\bar{r}_i$ and $\bar{r}_j$ represent average QoS values of i and j observed by a set of common users in U_c, respectively.

By the PCC similarity calculation, we choose the k similar services with the highest similarity degree.

$$I_{PCC} = topSimilar(i) \tag{13}$$

I_{PCC} consists of a set of similar services with target service i from I. In the deviation migration step, I_{PCC} is used to aggregate deviated QoS values.

(2) Detecting Neighbor Users. Given a target service i, it corresponds to a set of users U_i who have invoked i. In this step, we aim to eliminate those dissimilar users with a target user u from U_i, where they have low similar historical QoS values on their commonly invocated services. Here, we still apply the ratio-based similarity to measuring the neighborhood similarity degree.

$$Sim_{RBS}(u,v) = \frac{\sum\limits_{i \in I_c} \frac{\min(r_{u,i}, r_{v,i})}{\max(r_{u,i}, r_{v,i})}}{|I_c|} \tag{14}$$

It is observed that if the similarity degree $Sim_{RBS}(u,v) = 1$, it reflects that two users nearly received the same quality of QoS values when they invoked their commonly requested services in I_c. The implicit possibility is that these two users live in the same city and use the same network environment.

With a similarity threshold θ, we detect a subset of users who have invoked the target service i and obtain highly close invocation QoS values with the target user u.

Then, using a threshold to find out those users who are more similar to the target user u.

$$U_i' = \{u' \in U_i \,|Sim_{RBS}(u', u) \geq \theta\} \tag{15}$$

By the elimination of the dissimilar users from U_i, detected neighbor users is used in average QoS calculation step.

(3) Average QoS Calculation. Given a target service i, we make a reduction to all of the users U_i who invoked i and obtain a subset of neighbor users U_i', where each user $u' \in U_i'$ has a high similarity degree with the target user u. Similar to user-based reinforced collaborative filtering, we use neighbor users to initially estimate the QoS value. For each $u' \in U_i'$, we calculate an initially predicted QoS for the target service i.

$$\hat{r}_{u,i}^{u'} = \begin{cases} r_{u',i} * Sim_{RBS}(u,u') & \text{if } \overline{u} \leq \overline{u'} \\ r_{u',i}/Sim_{RBS}(u,u') & \text{otherwise} \end{cases} \tag{16}$$

Here, $\overline{u'}$ and $\overline{u}$ represent the average QoS value of user itself, respectively. $Sim_{RBS}(u,u')$ is the ratio-based similarity of the target user u and a neighbor user u'. By calculating the predicted QoS on each neighbor user $u' \in U_i'$, we get a group of predicted QoS values.

$$AvgQoS_i = \{\hat{r}_{u,i}^{u_1'}, \hat{r}_{u,i}^{u_2'}, \cdots, \hat{r}_{u,i}^{|U_i'|}\} \tag{17}$$

Taking the ratio-based similarity between two users as weight, we calculate the average QoS value for the target service i to be invoked the target user u.

$$\overline{i}_{avg} = \frac{\sum_{k=1}^{|U_i'|} Sim_{RBS}(u,u_k') * \hat{r}_{u,i}^{u_k'}}{\sum_{k=1}^{|U_i'|} Sim_{RBS}(u,u_k')} \tag{18}$$

In the same way, $\overline{i}_{avg}$ is still a preliminary result, as we mainly rely on the QoS values from those neighbor users similar to the target user, while similar services with the target service are not considered at this point. They are integrally integrated into the QoS prediction in deviation migration step.

(4) Deviation Migration. Based on found $I_{PCC} = \{i_1, i_2, \cdots, i_k\}$ for a target service i, we use Eqs. (14)–(18) to calculate and generate a set of average QoS values for each similar service in I_{PCC}.

$$AvgQoS_{I_{PCC}} = \{\overline{i}_1, \overline{i}_2, \cdots, \overline{i}_k\} \tag{19}$$

Finally, applying collaborative filtering algorithm with $\overline{i}_a vg$ and $AvgQo S_{I_{PCC}}$, we make the final QoS prediction.

$$\widehat{r}_{u,i} = \overline{i}_{avg} + \frac{\sum_{m=1}^{k} Sim_{PCC}(i,i_m) * (r_{u,i_m} - \overline{i}_m)}{\sum_{m=1}^{k} Sim_{PCC}(i,i_m)} \tag{20}$$

Where $\widehat{r}_{u,i}$ is the final predicted QoS value for a target user u when invoking a target service i.

4 Experiments

4.1 Experimental Setup and Dataset

The experiments are conducted on a large-scale real-world dataset called WS-DREAM [15], involving 5,825 real-world Web services in 73 countries and 339 service users in 30 countries. This dataset consists of two QoS invocation matrices, one for response time and the other for throughput.

To validate the performance of our approach, we use the response time matrix to perform our experiments. We extract the matrix into 1,873,838 QoS invocation logs, after removing those invocations where a target user failed to access a target service. All the QoS invocation logs are partitioned into two parts, one for the training set and the other for the test set. During the experiments, the proportion of the number of QoS invocation logs in training set among the whole dataset is called density. The user-service QoS invocation matrix always keeps sparse in real-world applications. Thus, we conduct a series of experiments with the density varying from 0.04 to 0.32 with a step of 0.02. In order to fairly perform reliable experimental evaluation, we repeat each experiment 5 times for each density and calculate their average results.

4.2 Competing Methods

In order to show the feasibility and effectiveness of our approach, we compared with seven competing approaches, including UMEAN, IMEAN, UPCC [2], IPCC [7], WSRec [13], NRCF [9] and RACF [10].

– UMEAN. It is a user-based QoS prediction method. It averages the QoS values that the target user invoked all of the services as the predicted result.
– IMEAN. It is a service-based QoS prediction method. It averages the QoS values that all of the users invoked the target service as the prediction result.
– UPCC. It is a user-based QoS prediction method. It is required to find a set of similar users to the target user. The prediction result combines the average QoS value by UMEAN and the deviation migration based on the found similar users.
– IPCC. It is a service-based QoS prediction method. It selects the most similar services to the target service. The prediction result is composed of the average QoS value by IMEAN and the deviation migration based on the found similar services.
– WSRec. It is a QoS prediction approach by the combination of UPCC and IPCC, which utilizes a parameter to respectively weigh the importance of UPCC and IPCC.
– NRCF. It improves the accuracy of traditional collaborative filtering algorithm for QoS prediction by novel similarity computation, where it normalizes the QoS values of web services to the same range and unifies the similarity in different multi-dimensional vector spaces.
– RACF. It a QoS prediction approach based on a novel similarity computation method called ratio-based similarity (RBS). The prediction results can be calculated by the similar users or services.

4.3 Experimental Results on Accuracy of QoS Prediction

In the experiments, mean absolute error (MAE) is used as the evaluation metric that measures the average absolute deviation of the predicted QoS values to the ground truth ones. Thus, the smaller value it is, The better performance the approach has. MAE is defined as below.

$$MAE = \frac{\sum_{i=1}^{N} |r_{u,i} - \widehat{r}_{u,i}|}{N} \tag{21}$$

Where $r_{u,i}$ and $\widehat{r}_{u,i}$ represent the ground truth QoS and predicted QoS of the target user u invoking a target service i. N is the number of QoS invocation logs for test. Under different densities of QoS invocation matrix, we compare our proposed approach with the existing seven approaches on QoS prediction accuracy. The experimental results are shown in Table 2.

Table 2. MAE within the different QoS matrix densities among competing methods

Density	UNEAN	IMEAN	UPCC	IPCC	WSRec	NRCF	RACF	RECF
0.04	0.8822	0.7138	0.7107	0.7339	0.6362	0.6528	0.5838	**0.5655**
0.06	0.8766	0.7004	0.6794	0.7177	0.6395	0.5892	0.5617	**0.4837**
0.08	0.8742	0.6890	0.6294	0.7048	0.6450	0.5514	0.5157	**0.4453**
0.10	0.8752	0.6887	0.6063	0.7000	0.6394	0.5312	0.4937	**0.4332**
0.12	0.8763	0.6880	0.5869	0.6535	0.6042	0.5104	0.4674	**0.4197**
0.14	0.8747	0.6832	0.5662	0.6145	0.5689	0.4910	0.4499	**0.4099**
0.16	0.8753	0.6844	0.5485	0.5764	0.5381	0.4770	0.4370	**0.4027**
0.18	0.8755	0.6821	0.5459	0.5516	0.5173	0.4677	0.4281	**0.4028**
0.20	0.8737	0.6819	0.5379	0.5351	0.5024	0.4607	0.4208	**0.3946**
0.22	0.8737	0.6792	0.5293	0.5185	0.4887	0.4483	0.4149	**0.3905**
0.24	0.8738	0.6791	0.5181	0.5072	0.4767	0.4405	0.4104	**0.3843**
0.26	0.8751	0.6801	0.5177	0.4973	0.4725	0.4396	0.4060	**0.3859**
0.28	0.8755	0.6787	0.5096	0.4855	0.4613	0.4339	0.4032	**0.3766**
0.30	0.8752	0.6778	0.5084	0.4783	0.4571	0.4296	0.3997	**0.3789**
0.32	0.8749	0.6779	0.4980	0.4672	0.4459	0.4249	0.3991	**0.3692**

From the experimental results in Table 2, it is observed that the prediction accuracy on MAE among all of the competing methods decreases along with the increase of density in QoS invocation matrix. The reason is that the similarity degree can be more accurately calculated to improve the prediction result, as the density becomes larger and sufficient QoS invocation logs can be provided. Given a specific density, our proposed approach receives lower MAE than that of existing ones, indicating that RECF is superior to those state-of-the-art methods in terms of QoS prediction accuracy. The main reason is that we strategically eliminate all of the dissimilar services (or users) that boost the noisy of average QoS calculation in traditional collaborative filtering methods.

In order to further test the QoS prediction accuracy, we count the QoS invocation logs of test samples within multiple deviation intervals and analyze the performance among different approaches. QoS deviation represents the gap between the predicted QoS value and the true real QoS value, which is shown in the form of absolute value interval. In our experiments, the QoS deviation interval is set to 0.04. In other words, the test samples' QoS deviation intervals are divided as $[0.0, 0.04), [0.04, 0.08), [0.08, 0.12), \ldots, [1.96, 2.0)$. We calculate the number of samples in each QoS deviation interval as a percentage of the total test samples. Under the setting of QoS matrix density as 0.08 and 0.12, two groups of experiments have been performed and the results are illustrated in Figs. 2 and 3, respectively.

For each subgraph in Figs. 2 and 3, the value of each point on the polyline in lower part indicates the proportion of the number of test samples in that QoS deviation interval to the total number of test samples. The polyline in upper part is the cumulative value of the polyline in lower part till the point. It is observed that from the Figs. 2 and 3 more test samples are distributed in lower QoS deviation intervals when using our approach to predict QoS value, compared with other existing seven approaches. More specifically, making statistics on lower part in Fig. 2, we can find that 30% of test samples are distributed in $[0.0, 0.04)$ QoS deviation interval by our approach RECF, while UMEAN is 3%, IMEAN is 14%, UPCC is 13%, IPCC is 6%, WSRec is 7%, NRCF is 14% and RACF is 20%.

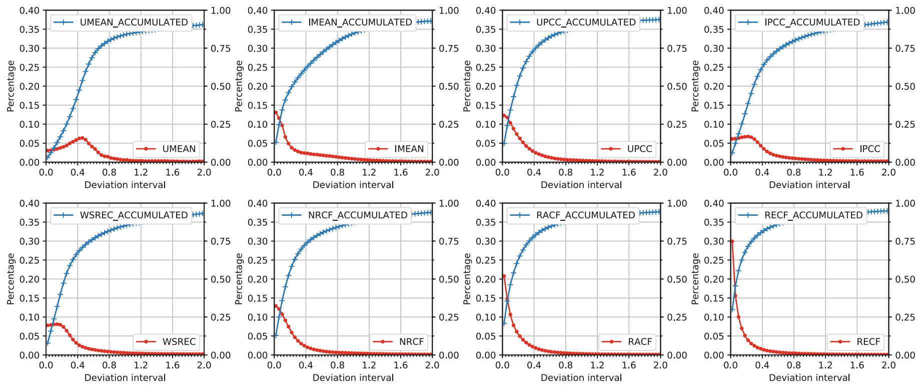


Fig. 2. The experimental results on distributions of QoS prediction among different deviation intervals (density: 0.08)

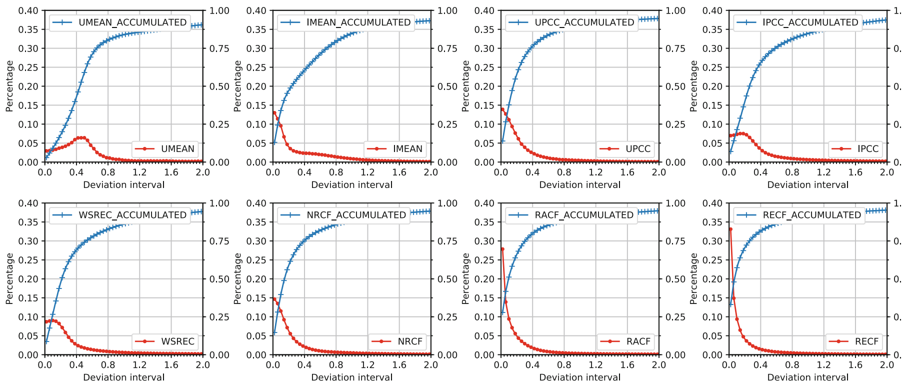


Fig. 3. The experimental results on distributions of QoS prediction among different deviation intervals (density: 0.12)

From the above experiments, we conclude that our proposed approach outperforms the existing competing ones for QoS prediction.

4.4 Impact of Parameter Tuning

In our proposed approach, there are two main parameters that affect the QoS prediction accuracy. They are the top k number of similar users (or services) in PCC similarity and neighbor similarity threshold θ in ratio-based similarity. In order to analyze the trends of QoS prediction accuracy and find out the optimal parameter value, we conduct two groups of experiments. Figure 4 illustrates that when θ keeps constant, the MAE fluctuation of our RECF is measured along with the changes of k at different densities. On the contrary, Fig. 5 shows that when k remains unchanged, the MAE variation of our RECF is measured along with the changes of θ at different densities.

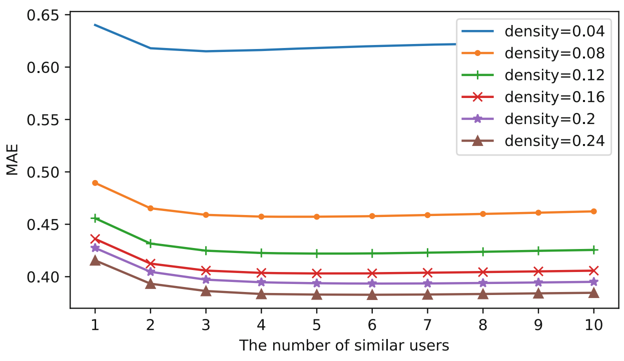


Fig. 4. The experimental results of MAE affected by the parameter (k)

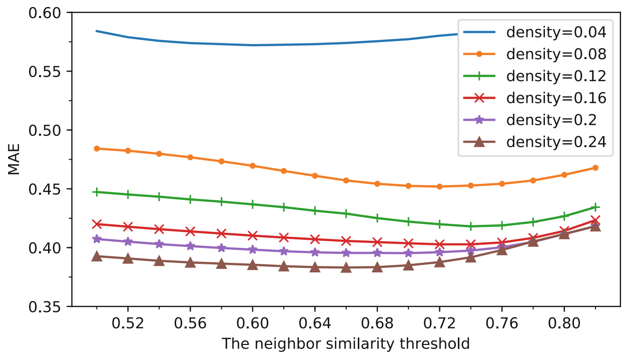


Fig. 5. The experimental results of MAE affected by the parameter (θ)

We can observe from Fig. 4, the QoS prediction accuracy achieves the best on MAE within different QoS matrix densities, when the number of similar users is set to 4. The QoS prediction accuracy is almost unchanged, as the number of similar users increases. However, the computational complexity grows with the number of increasing similar users.

From the experimental results in Fig. 5, we can find that the QoS prediction accuracy achieves the best on MAE within different QoS matrix densities when the neighbor similarity threshold in ratio-based similarity is set as 0.72.

5 Conclusion

To effectively predict missing QoS of web servies, we proposed a reinforced collaborative filtering approach that eliminates dissimilar services (or users) for improving QoS prediction accuracy. It goes through four steps: finding similar services, detecting neighbor users, average QoS calculation and deviation migration. Extensive experiments are conducted on a large-scale real-world web service QoS dataset. The results demonstrate its effectiveness competing with the existing methods. In the future work, we will apply our approach to the real-world recommender systems for modern microservices. Moreover, since collaborative filtering algorithm has high computational complexity, we will further optimize our approach making it more efficient by incremental learning.

Acknowledgements. This work was supported by National Natural Science Foundation of China (61772128), Shanghai Natural Science Foundation (18ZR1414400, 17ZR1400200) and Fundamental Research Funds for the Central Universities (16D111208).

References

1. Azadjalal, M.M., Moradi, P., Abdollahpouri, A., Jalili, M.: A trust-aware recommendation method based on Pareto dominance and confidence concepts. Knowl.-Based Syst. **116**, 130–143 (2017)
2. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 43–52. Morgan Kaufmann Publishers Inc. (1998)
3. Ding, S., Li, Y., Wu, D., Zhang, Y., Yang, S.: Time-aware cloud service recommendation using similarity-enhanced collaborative filtering and ARIMA model. Decis. Support Syst. **107**, 103–115 (2018)
4. He, P., Zhu, J., Zheng, Z., Xu, J., Lyu, M.R.: Location-based hierarchical matrix factorization for web service recommendation. In: IEEE International Conference on Web Services (ICWS), pp. 297–304. IEEE (2014)
5. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
6. Rennie, J.D., Srebro, N.: Fast maximum margin matrix factorization for collaborative prediction. In: International Conference on Machine Learning (ICML), pp. 713–719. ACM (2005)
7. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: International World Wide Web Conference (WWW), pp. 285–295. ACM (2001)
8. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating "word of mouth". In: ACM CHI Conference on Human Factors in Computing Systems, pp. 210–217. ACM (1995)
9. Sun, H., Zheng, Z., Chen, J., Lyu, M.R.: Personalized web service recommendation via normal recovery collaborative filtering. IEEE Trans. Serv. Comput. **6**(4), 573–579 (2013)
10. Wu, X., Cheng, B., Chen, J.: Collaborative filtering service recommendation based on a novel similarity computation method. IEEE Trans. Serv. Comput. **10**(3), 352–365 (2017)
11. Xiong, W., Wu, Z., Li, B., Gu, Q.: A learning approach to QoS prediction via multi-dimensional context. In: IEEE International Conference on Web Services (ICWS), pp. 164–171. IEEE (2017)
12. Xu, Y., Yin, J., Deng, S., Xiong, N.N., Huang, J.: Context-aware QoS prediction for web service recommendation and selection. Expert Syst. Appl. **53**, 75–86 (2016)
13. Zheng, Z., Ma, H., Lyu, M.R., King, I.: QoS-aware web service recommendation by collaborative filtering. IEEE Trans. Serv. Comput. **4**(2), 140–152 (2011)
14. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Collaborative web service QoS prediction via neighborhood integrated matrix factorization. IEEE Trans. Serv. Comput. **6**(3), 289–299 (2013)
15. Zheng, Z., Zhang, Y., Lyu, M.R.: Distributed QoS evaluation for real-world web services. In: IEEE International Conference on Web Services (ICWS), pp. 83–90. IEEE (2010)

Unit of Work Supporting Generative Scientific Workflow Recommendation

Jia Zhang[1](✉), Maryam Pourreza[1], Seungwon Lee[2], Ramakrishna Nemani[3], and Tsengdar J. Lee[4]

[1] Carnegie Mellon University, Mountain View, USA
jia.zhang@sv.cmu.edu
[2] NASA Jet Propulsion Laboratory, Pasadena, USA
[3] NASA Ames Research Center, Mountain View, USA
[4] Science Mission Directorate, NASA Headquarters, Washington, D.C., USA

Abstract. Service discovery and recommendation is playing increasingly important role, as more and more reusable web services are published onto the Internet. Existing methods typically recommend either individual services, or multiple services without their interconnections. In contrast, this research aims to mine service usage history and extract units of work (UoWs) comprising a collection of services chained together through intermediate components. A novel technique is proposed in this paper to study how services collaborated, or could collaborate, in the form of reusable UoWs to serve various workflows (i.e., mashups), based on an evolving service social network. Upon receiving a scientific workflow request, a recommend-as-you-go algorithm simulates how human minds work and relies on a sliding aggressiveness gauge to incrementally recommend context-aware UoWs. In this way, we hope to move one step further toward automatic service composition. Extensive experiments on the real-world datasets demonstrate the effectiveness and efficiency of the UoW-oriented service recommendation approach.

Keywords: Unit of work · Service composition
Recommend-as-you-go

1 Introduction

As service-oriented software engineering (SOSE) becomes mainstream, increasingly more software components have been published onto the Internet as reusable services (or so-called APIs as lightweight services). People can leverage and compose existing services as components to build new functionality (so-called mashup or workflow) faster than before [8]. Thus, it is becoming increasingly important to build techniques of service discovery and recommendation to help people find and compose suitable services from a sea of available candidates [12].

In science domain, our previous work [19] reveals that one major obstacle stopping scientists from reusing services (i.e., algorithms) developed by peer scientists is how to transform data types to feed in the (usually comprehensive) input of the algorithms. In Earth science, for example, a data analytics service may require more than a dozen of parameters [19]. Unless one knows exactly what such parameters mean and how to feed in data accordingly, she will be reluctant to reuse the service. When multiple external services are used in a scientific workflow, how to chain them remains a significant challenge. As indicated by our earlier work, the past linkage among services may be useful in service recommendation [20] and will help automatic service composition [7].

Take Fig. 1 as a highly simplified scenario. Assume that existing semantics-based methods (e.g., [10]) have recommended three services (s_1, s_2, and s_3) to be included in a workflow request. How to link them together, however, remains unresolved. As shown in Fig. 1, it is found that in a past workflow wf_1, s_1 and s_2 were used together with an indirect path linking between them: $s_1 \longmapsto s_2$. This means that the output of service s_1 is transformed through some intermediate steps and becomes the input of service s_2. Similarly, in another past workflow wf_2, services s_2 and s_3 were indirectly linked together: $s_2 \longmapsto s_3$. Such provenance means that one can reuse the past linkages to chain between s_1 and s_2, as well as s_2 and s_3. Note that although the three services were never used together in the past, by integrating the two chains $s_1 \longmapsto s_2$ and $s_2 \longmapsto s_3$, we will harvest an unprecedented service chain $s_1 \longmapsto s_2 \longmapsto s_3$. This example illustrates that by mining service usage history, the three services can be automatically chained together without worrying how to transform data among them.

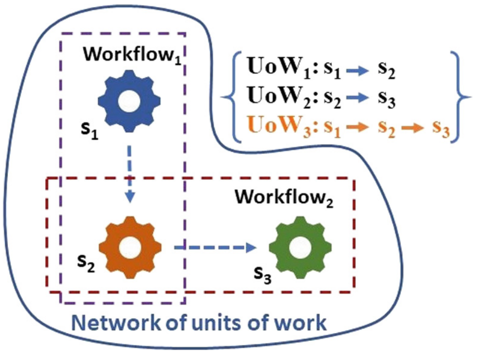

Workflow$_1$
s_1
s_2
Workflow$_2$
s_3
$UoW_1: s_1 \rightarrow s_2$
$UoW_2: s_2 \rightarrow s_3$
$UoW_3: s_1 \rightarrow s_2 \rightarrow s_3$
Network of units of work

Fig. 1. Mining service usage history to facilitate service composition.

The example shown in Fig. 1 directly motivates this research, aiming to systematically study how to mine service usage history and identify reusable, and maybe unprecedented, service chain snippets to facilitate automatic service composition. In concurrent database operations, a transaction is considered a unit of work (UoW) meaning that all operations encompassed in a UoW either all complete and become persistent, or have to rewind as if nothing happened. In this project, we borrow this term **UoW to represent a collection of services chained together, maybe through some software components as glues,**

to fulfill some business goals. In other words, this research aims to mine workflow provenance and extract service units of work (UoWs) to facilitate new workflow design and development.

The contributions of this work is three-fold. First, we coined the concept of service unit of work and have developed a service social network to support the recording and retrieval of service UoWs. Second, we have developed novel algorithms to mine reusable service UoWs as a recommend-as-you-go service serving context-aware workflow queries. Third, our experiments over the real-life datasets have demonstrated the effectiveness and efficiency of our approach.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces the UoW-driven workflow recommendation framework. Section 4 describes our service UoW mining and recommendation algorithms. Section 5 discusses and analyzes our experiments. Section 6 draws a conclusion.

2 Related Work

Service composition remains a fundamental topic in the field of service oriented computing [12]. Paik et al. [11] divide service composition into four phases: planning, discovery, selection, and execution. Receiving a user functional query, the four stages generate abstract workflow tasks, map abstract tasks to service instances, select a combination of service instances based on constraints, and run service instances, in a sequential order. According to their categorization, our work mainly falls in the third phase: service selection phase, where we aim to decide on concrete service instances for abstract workflow tasks. However, a typical service composition process carries two fundamental assumptions. First is that a planning phase has resulted in a structured workflow plan, i.e., a reference model [4]. Second is that identified service components will be linked to each other directly [2], meaning that the output of a service component has to be plugged into another service component as input. In contrast, our research raises the bar and aims to fill the gap by automatically chaining service components based on their past collaboration history. Therefore, the starting point of our work is a collection of abstract tasks identified, however their relationships remaining to be decided by our work. As a result, our work does not require a typical full service planning phase to generate a structure of abstract workflow tasks interconnected.

A typical service selection phase focuses on finding a combination of service instances mapping to the identified collection of abstract tasks, based on certain constrains typically as QoS requirements or semantic compliances. Researchers have applied various types of optimization methods to tackle the multiple QoS-constrained service combination problem, including evolutionary algorithms [15] and Integer Programming [17]. In contrast, our research is complementary to the QoS-based service selection, in the sense that we focus on automatically finding suitable connections among service components. In recent years, Rodríguez-Mier et al. [12] optimize global QoS of the composition based on a service match graph containing all possible semantically matched edges between service candidates. Different from their work where service connections are possible links based on their semantic input/output descriptions, service connections in our work come from historical successful service connections happened in past workflows. For scientific researchers, such service connections are more trustworthy.

Semantic web technologies introduce rich and machine understandable representations of service functions and properties, which facilitate automatic service composition [10]. For example, Li et al. [9] analyze user requirements and service descriptions and recommend services based on their semantic compatibility. In recent years, researchers further leverage natural language processing and machine learning to enhance service discovery and composition. For example, Xia et al. [16] develop a category-aware service clustering and recommending method for automatic workflow composition. In this work, we leverage their techniques [9,16] to measure the distance toward the goal during service composition, to dynamically compute the greediness of our algorithm.

IBM's MatchUp project [6] recommends workflow fragments (mashlets with glue patterns) based on user context. Similarly, Roy Chowdhury et al. [3] recommend composition patterns (data components and connectors) based on partial mashup. Our work differentiates from theirs in three significant ways. First, their work only consider service linkage patterns (glue patterns or connectors) in individual workflows. In contrast, our work considers service linkage patterns contributed by multiple workflows together, since we aggregate all past workflows into a service social network. Second, compared with their work aiming to link identified services instances, our work aims to link identified conceptual services and result in (partially) linked service instances. Third, while their work only recommends linkage patterns between services and programmers still have to develop actual glue code, our method will automatically reuse previous linkage code to chain services together.

In addition to automatically linking service components, our work also, as other workflow recommender systems do, recommends related services based on the partial workflow placed by users. Based on a partial workflow, VisComplete [7] examines comprised paths and recommends path extensions based on a repository of workflows represented as graphs. Similarly, Zhang et al. [18] and Deng et al. [4] extract workflow patterns (nodes and their upstream subgraphs) from past workflow provenance. Such patterns are used as templates to match partially finished workflow to recommend extending nodes. In addition, IBM's MashupAdvisor [5] recommends related services based on conditional co-occurrence probability and semantic matching. In contrast to their works focusing on finding what may co-occur based on structural comparison, our method focuses on finding services toward fulfilling the user intent.

The Business Process Management (BPM) community has contributed a number of approaches that recommend patterns as reusable components based on label/text similarity (such as [14]) or control flow patterns (e.g., [1]). In contrast, our project focuses on finding glues to link service candidates together based on the past workflows carrying the usage history of the services. Our aim is to eliminate human efforts on linking services together. Furthermore, by constructing a service social network, our work may extract unprecedented service usage paths that may be useful for scientific experiments (i.e., workflow) design.

3 UoW-Driven Workflow Recommendation Framework

In this research, we propose a novel technique to facilitate workflow design and development. At each step during a workflow design, we recommend a context-aware runnable group (i.e., Unit of Work) comprising multiple services, based on a service social network constructed from service usage history. Meanwhile, a sliding-window-like gauge is developed to predict the aggressiveness of a user on selecting UoWs during the workflow design process.

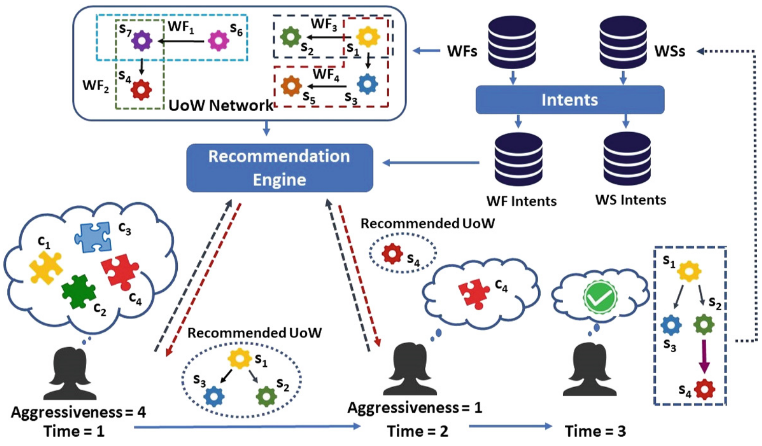


Fig. 2. UoW-driven workflow recommendation framework

Figure 2 illustrates the overall blueprint of our UoW-driven workflow recommendation framework using a highly simplified scenario. On the top left corner, a UoW network is constructed offline based on existing workflows and services. Meanwhile, the descriptions of the workflows and services are analyzed and their intents are stored in databases. At time 1, upon receipt of a user query with overall goal and conceptual services identified (c_1–c_4), the Recommendation Engine will decide the aggressiveness of the user (i.e., 4). After consulting with the UoW network and the workflow and service intent databases, a connected UoW with three service instances (s_1–s_3) will be recommended, covering three conceptual services (c_1–c_3). At time 2 after the user accepts the recommendation, her aggressiveness drops to 1. Based on the partial workflow (i.e., UoW) at hand, the Recommendation Engine will recommend service instance s_4. Since s_4 is not connected to the UoW, the user has to build the connection herself. This scenario shows that our technique can not only help to find concrete service instances, but also significantly save the user efforts on linking the service instances together. Note that a finished workflow at time 3 will be added into the UoW network and the workflow/service intent databases on the fly, as shown in Fig. 2. In the next sections, we will explain the technique in details.

3.1 UoW Network Construction

In the first step, we describe how to construct a UoW network.

Definition 1 (Unit of Work - UoW). *A unit of work is a connected directed graph* $uow = <S', E'>$ *extracted from a directed graph of a workflow* $w = <S, E>$: $uow \subseteq w$, *iff:*

1. $S' \subseteq S$
2. $E' \subseteq E$
3. $\forall s_i, s_j \in S', \exists$ *path* $s_i \longmapsto s_j$ *and/or* $\exists$ *path* $s_j \longmapsto s_i$

where S *is the set of services (vertices) used by the workflow, and* E *is the set of edges linking between the services labeled with workflow identifier.*

A uow might consist of a single service with no edge or it might be the entire workflow. The number of uows depends on the edges existing between various services. If we consider the set of all UoWs extracted from a workflow w_i with k edges among services to be $\{uow_{1,i}, uow_{2,i}, \ldots, uow_{m,i}\}$, then the size of this set is $m \leq 2^k$. The set of all units of work extracted from a collection of workflows can form a network of units of work as defined below.

Definition 2 (Network of Units of Work). *A network of units of work over* M *workflows is defined as* $N_{uow} = <S'', E''>$ *where* $S'' = \bigcup_{j=1}^{M} S_j$ *is the set of all services included in all workflows, and* $E'' = \bigcup_{j=1}^{M} E_j$ *is the set of all the edges in the network each labeling with a workflow identifier.*

The total number of uows that can be extracted from the whole network would be $\geq |S''|$ and $\leq 2^{|E''|}$. A UoW network is constructed by creating a directed graph with vertices to be the set of all services contained in all workflows, and each labeled edge represents a path between a pair of services in a workflow.

3.2 Search Query Analysis

Based on our previous work [16] on service discovery driven by topic modeling, assume that the network N_{uow} implies a collection of latent topics that can be automatically extracted from all comprising workflow descriptions. Such latent topics form an Intent space of T. Each service or workflow serves some functionality, meaning that each of them can be represented by a distribution of topics over T. Meanwhile, we use term *context* to represent the scenarios where a service can serve, also in the form of a distribution of topics over $|T|$ based on its past contribution in workflows in the network.

Definition 3 (Service Intent and Context). *A service $s \in S''$ is defined as a tuple $s = <\phi_s, \psi_s>$. ϕ_s denotes the intent of service s as a distribution of topics over the Intent space of the network N_{uow}, where its $|T|$-dimensional vector of probabilities sum to 1: $\sum_{i=1}^{|T|} p_{i,s} = 1$. ψ_s denotes the context of service s as a union of the intent of workflows in the network N_{uow}, where service s is a component: $\psi_s = \bigcup_{j=1}^{|M|} \phi_{w_j}$. ϕ_w denotes the intent of workflow w as a distribution of topics over N_{uow}, where the $|T|$-dimensional vector of probabilities sum to 1: $\sum_{k=1}^{|T|} p_{k,w} = 1$.*

Note that through vector arithmetic, the context of a service also represents a distribution of topics over the Intent space of the network N_{uow}. Consequently, the intent of a uow can be viewed as a union of the intent of services included.

Definition 4 (Intent of Unit of Work). *Intent of a unit of work u is defined as $\phi_u = \{\phi_{1,u}, \phi_{2,u}, \ldots, \phi_{|T|,u}\}$, where the intent value can be calculated using a SoftMax function σ such that:*

$$\phi_{j,u} = \sigma(\sum_{s \in u} \phi_{j,s}) = \frac{e^{\sum_{s \in u} \phi_{j,s}}}{\sum_{j=1}^{|T|} e^{\sum_{s \in u} \phi_{j,s}}} \tag{1}$$

where $\phi_{j,s}$ denotes the j^{th} intent value of service $s \in u$ and $\phi_{j,u}$ denotes the j^{th} intent value of uow u.

Before a user starts to design a workflow, a user query would be provided containing information about the user's goal of her desired workflow, which means intents can also be extracted from the query to make it machine understandable.

Definition 5 (Intent of user query). *Intent of a user query q, can be represented by ϕ_q as a distribution of topics in query q over the Intent space of the network N_{uow}.*

With the introduction of the intent and context of service/workflow/query, a search query at a given time point can be defined as follows:

Definition 6 (Search Query). *A search query at time t is a triple $q_t = <G_q, W_t, A_t>$, where $G_q = <\phi_q, Pr_q>$ is the final goal of the user which contains ϕ_q as the user's intent and Pr_q as the list of user's desired conceptual services identified, respectively. W_t represents the current partial workflow, and A_t is the user's current aggressiveness which will be defined shortly.*

In the beginning, users may intuitively like to obtain UoW recommendation that covers as many as possible conceptual services intended, even though some irrelevant services (i.e., noises) are included. During a workflow design process, however, the aggressiveness of a user choosing larger-sized units of work would reduce as the workflow-in-progress grows towards the goal. Thus, a user's aggressiveness can be defined as follows:

Definition 7 (User's Aggressiveness). *The aggressiveness of user u, with query q at time t, is a function of the number of user's desired conceptual services G_q and what she has achieved so far by her current workflow W_t:*

$$\begin{aligned}A_t = \alpha * (1 - coverage(W_t, G_q.Pr_q)) + \beta * similarity(\phi_{W_t}, \phi_q) \\ - \gamma * \frac{noise(W_t, G_q.Pr_q)}{|W_t|}\end{aligned} \tag{2}$$

where coverage is a function that computes the overlap rate of the current partial workflow over the expected goals, similarity is a function that measures the semantic (i.e., intent) similarity between the partial workflow and the user query, and noise is a function that computes unwanted services introduced through the UoWs, normalized. α, β, and γ are coefficients such that $\alpha + \beta + \gamma = 1$.

3.3 Basic UoW Extraction and Recommendation

After the UoW network is built, Algorithm 1 shows how to extract all candidate UoWs from the network based on a user search query q_t. This algorithm first finds the desired conceptual intent from the user's goals that have not been satisfied yet by the user's current workflow (line 3). Afterwards, for each of the conceptual services, we find all the services in the network that have similar intents with the similarity threshold of λ. In this way, we obtain candidate services that are clustered based on the conceptual services to which they are similar (line 4). In line 5, we fetch all the combinations from these services for satisfying the user's remaining desired conceptual services.

For finding the candidate UoWs, we try to find UoWs from the network with connection to the user's current workflow (lines 6–15). We first consider service s_i from the current workflow to be our connection point to the candidate UoWs. Then, we remove all other services in the current workflow from the network and get all the weakly connected subgraphs from the remaining network. Next, we try to focus only on the subgraph which contains our connection point. Finally, for every combination of candidate services from different clusters, we will add all the connected subgraphs attached to the connection point to the UoW recommendation list.

Some candidate UoWs might not have any path in history to any of the services in the user's current workflow. However, they might actually help the user towards her final goal and the user can decide how to attach them to her current workflow. Thus, for finding such UoWs (lines 16–24), we first remove all the current services from the network and then get all the weakly connected subgraphs in this remaining network. Then in the next step, for every combination of service candidates and every subgraph $subgraph_j$, we first find the services S' from this combination that exist in the selected subgraph. Finally, we add all the candidate UoWs containing S' in the $subgraph_j$ to the UoW candidate list.

In the very last line of the Algorithm 1 (line 25), we add all the candidate services to the recommendation list as individual UoWs. This way, if there is

Algorithm 1. Extract all candidate UoWs from network based on user query

```
1: procedure ExtractCandidateUoWs($N_{uow}$, $q_t$, $W_t$)
2:     $uowSet_{candidate} \leftarrow \emptyset$
3:     $Pr'_q \leftarrow findRemainingIntent(q_t, W_t)$
4:     $candidateServiceClusters \leftarrow findCandidateServiceClusters(N_{uow}, Pr'_q)$
5:     $candidateCompositions \leftarrow selectServiceCompositions(candidateServiceClusters)$
6:     for $s_i \in W_t$ do
7:         $subnetwork \leftarrow N_{uow} - W_t \rightarrow Nodes + s_i$
8:         $connectedGraphs \leftarrow findAllConnectedGraphs(subnetwork)$
9:         $subgraph_{s_i} \leftarrow findGraphContainingService(s_i)$
10:        for $S_k \in candidateCompositions$ do
11:            $S' \leftarrow$ find subset of $S_k$ which is in $subgraph_{s_i} + s_i$
12:            $hypothesisGraph \leftarrow$ a graph containing $S'$ as nodes and no edges
13:            $addUoWs(uowSet_{candidate}, hypothesisGraph, S', subgraph_{s_i}, s_i)$
14:        end for
15:    end for
16:    $subnetwork \leftarrow N_{uow} - W_t$
17:    $connectedCompositions \leftarrow findAllConnectedGraphs(subnetwork)$
18:    for $S_k \in candidateCompositions$ do
19:        for $subgraph_j \in connectedGraphs$ do
20:            $S' \leftarrow$ find subset of $S_k$ which is in $subgraph_j$
21:            $hypothesisGraph \leftarrow$ a graph containing $S'$ as nodes and no edges
22:            $addUoWs(uowSet_{candidate}, hypothesisGraph, S', subgraph_j)$
23:        end for
24:    end for
25:    add candidate services as single UoWs to the $uowSet_{candidate}$
26:    return $uowSet_{candidate}$                    ▷ List of UoW candidates
27: end procedure
```

no path between them in the history network, we can still recommend single services to the user.

The function *addUoWs* used on lines 13 and 22 of Algorithm 1 is presented in Algorithm 2. This algorithm is used for recursively finding candidate UoWs in a specific component, containing a number of desired services. The base case in this algorithm (lines 2–6) is when the $uow_{hypothesis}$ is a weakly connected graph and this is when we can add this UoW to the recommendation list. Here, if we want the resulting UoWs to be connected to a connection point (which is, as described before, a service from the user's current workflow), then it should also be among the services in the candidate hypothesis so that we can add it to the candidate list (lines 3–5). However, if the hypothesis is still not connected (lines 7–17), for every pair of services, we find the shortest path between them. If found, we add the path to the hypothesis and try to find the candidate UoWs with the remaining services and the extended $uow_{hypothesis}$.

4 UoW Recommendation As-You-Go

With the preliminaries defined, the UoW recommendation problem can be formalized as to search for a largest-sized unit of work from the network of UoWs, which is the closest to the goal of the user query with the minimum number of irrelevant services. A comprehensive user query may take multiple steps during some time period. At a given time, recommendation should take into consideration of the partial workflow placed by users.

Algorithm 2. Find candidate UoWs containing a given set of services in a given connected subgraph recursively

```
 1: procedure ADDUoWs($uowSet_{candidate}$, $uow_{hypothesis}$, $S$, $component$, $s$)
 2:     if $uow_{hypothesis}$ is connected then
 3:         if $\nexists$ $s$ OR $s \in uow_{hypothesis}$ then
 4:             $uowSet_{candidate} \leftarrow uowSet_{candidate} + uow_{hypothesis}$
 5:         end if
 6:     else
 7:         if $S! = \emptyset$ then
 8:             for $s_i, s_j \in S$ do
 9:                 if $\exists s_i \longmapsto s_j \in component$ then
10:                     $uow_{hypothesis} \leftarrow uow_{hypothesis}$ + shortestPath($s_i \longmapsto s_j$)
11:                     $S \leftarrow S - Nodes_{si \longmapsto s_j}$
12:                     $addUoWInComponent(uowList_{candidate}, uow_{hypothesis}, S, component)$
13:                 end if
14:             end for
15:         end if
16:     end if
17: end procedure
```

Definition 8 (UoW Recommendation Problem). *Given a search query q, the UoW recommendation problem at time t is defined as finding a unit of work (uow) in the network of UoWs (N_{uow}), such that under aggressiveness A_t:*

1. $max(coverage(uow, G_q.Pr_q))$
2. $max(similarity(\phi_{uow}, \phi_q))$
3. $min(noise(uow, G_q.Pr_q))$

Based on the initial UoWs found in the last section, we develop an approach which can answer the UoW recommendation problem defined above. Algorithm 3 searches the network of UoWs along with the user's query and recommends a set of UoWs.

In this algorithm, we first fetch a set of candidate UoWs by using Algorithm 1 and then for each of these candidates, we make some examinations. We check if the context is similar enough to the query's context, and if the candidate has enough useful services with respect to the user's aggressiveness and the noise in the whole candidate (lines 4–8). If the candidate UoW meets the criteria, it is added to the candidate list. It should be noted that we consider threshold δ for the context similarity. For checking the usefulness, we use the equation below:

$$\frac{N_{usefulServices_i}}{N_{V_i'} - N_{usefulServices_i}} \leq A_t \tag{3}$$

Based on the above equation, we want the number of useful services in the uow_i with regard to the number of noise services (irrelevant services in uow_i) to be no more than the aggressiveness of the user at the time. In case the number of noise is equal to zero, we only consider the number of useful services.

Finally in line 4, we sort the candidate list based on the $\frac{N_{usefulServices_i}}{N_{V_i'} - N_{usefulServices_i}}$ in descending order and return the list to the user.

Algorithm 3. Recommend UoWs for the existing workflow

```
1: procedure RecommendUoWs($N_{uow}, q_t, W_t$)
2:     $recommendedUoWs \leftarrow \emptyset$
3:     $uowList_{candidates} \leftarrow extractCandidateUoWs(N_{uow}, q_t, W_t)$
4:     for $uow_i \in uowList_{candidates}$ do
5:         if $similarContexts(uow_i, q)$ and $isUsefulEnough(\phi_{uow_i}, \phi_q, Ag_q)$ then
6:             $recommendedUoWs \leftarrow recommendedUoW \cup uow_i$
7:         end if
8:     end for
9:     $recommendedUoW \leftarrow Sort(recommendedUoWs)$
10:    return $recommendedUoWs$
11: end procedure
```

5 Experiments and Analysis

We have designed and conducted a collection of experiments to evaluate the effectiveness and efficiency of our method over a real-world dataset.

5.1 Experimental Setup

We chose myExperiment.org as our testbed since it is the world largest service-oriented scientific workflow repository. Its metadata was used to analyze workflow intent, and source code to analyze UoWs. Since the majority of workflows were developed using Taverna tool, we crawled all the publicly available Taverna workflows up to May 2018. This resulted in 3,277 workflows in different versions from 2,030 unique workflows for our experiments. For each workflow, we scrutinized all the services contained and extracted 513 unique services coming in various types including: WSDL, SoapLab, BioMoby, and SADI services along with the REST calls. Among all the initial list of workflows, we obtained 1,719 workflows which invoke at least one of those services. Totally, from these services, 1,248 unique operations are used in all the workflows. We thus create our UoW network as summarized in Table 1.

Table 1. Summary of experiments

# Unique workflows	2,030
# Workflow versions	3,277
# Unique services	513
# Unique service operations	1,248
# Workflows with at least one service	1,719
# Workflows with at least two services	511

Over the dataset, all workflows were sorted based on their creation dates. We designed two types of experiments to study how our technique would have worked in the history, starting from the oldest workflows toward the most recent ones.

The difference between them is the way to create the UoW network. In the first type of experiment, for each workflow under study, the UoW network contains all *in prior* workflows. In the second type of experiment, for every workflow, the UoW network remains almost the same - including all workflows in the testbed except the one under study and its subsequent versions. The rationale is that the UoW network would become very rich after many workflows join.

For each type of experiment, chronologically, each workflow served as a search query, including its description and conceptual components (i.e., the processor names defined in all Taverna workflows). For each query, we ran our recommendation algorithm, used the top recommended UoW and continued the recommend-as-you-go process, until either the goal of the workflow was reached or the algorithm could not find any more UoWs.

Our baseline methods are the recommendation methods based on keywords, semantics (i.e., the method in [12]) and patterns (i.e., the methods in [4,18]). All baseline approaches were applied to our two types of experiments for comparison.

We used $\lambda = 0.2$ and $\delta = 0.01$ for the experiments. The Dijkstra algorithm [13] was used for finding the shortest paths in the UoW network. The experiments were run on a Windows 10 64-bit machine with 64 GB memory and Intel i7-7700 CPU @ 3.60 GHz*8. The code was implemented in Java 8 and JGraphT library was used for building the graphs and algorithms on top of them.

5.2 Evaluation Metrics

In order to evaluate our method, we adopted four evaluation metrics. Note that the measurements are only applied to the final step of our method, because our method operates a multi-step procedure to identify UoWs and services, controlled by dynamically changing aggressiveness.

1. **Average Precision:** The average number of correctly predicted services at the final step of the recommend-as-you-go process, for the top recommended UoWs over the total number of services predicted $\frac{N_{correctly-predicted}}{N_{total-recommended-services}}$.
2. **Average Recall:** The average number of correctly predicted services at the final step of the recommend-as-you-go process, for the top recommended UoWs over the total number of services in the test workflow $\frac{N_{correctly-predicted}}{N_{total-WF-services}}$.
3. **Accumulated Saved Steps:** The accumulated number of steps saved for reaching the last result, by using our approach.
4. **Accumulated Saved Links:** The accumulated number of decisions about the connections among services saved for the user by our approach.

5.3 Experimental Results and Analysis

Recommendation Effectiveness: Since the starting point of the experiments are just intents (i.e., descriptions) and a collection of conceptual components (i.e., services candidates), without structural design (i.e., reference model), the baseline methods based on patterns [4,18] will not work in the beginning.

We then studied how the pattern-based baseline methods could work after our method identified the first-round of UoWs, meaning that when a partial structure exists. For type 1 experiment, out of the 1,719 workflows in the testbed, 191 workflow designs require multiple steps in our method. For those workflows, the baseline methods recommended services for 7 workflows. For type 2 experiment, the baseline methods recommended 16 services out of 291 workflows that require multiple steps in our method. The reason is that most of the services contained in a workflow were not used together in earlier workflows, thus their co-occurrence was not recorded in the pattern table/repository in the baseline methods. This situation is common in scientific research, because scientific workflows typically imply unprecedented experiment design.

We also studied whether the semantics-based baseline approach [12] could work on the testbed. It does not due to a fact that scientific services (i.e., algorithms) typically cannot be directly chained together without data transformation [19]. Thus, semantic input/output match-making among services did not work well for the testbed.

In contrast, our approach successfully recommended services components for all 1,719 workflows. Therefore, we will only discuss in detail the recommendation quality of our approach under the two types of experiments.

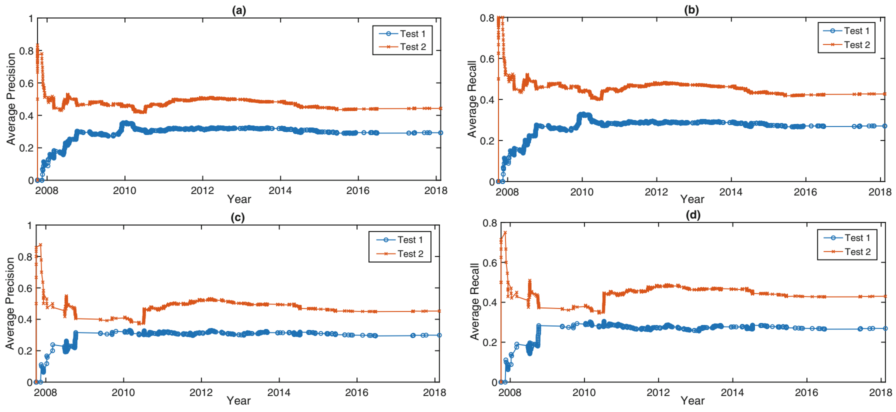


Fig. 3. (a) Average precisions comparison (b) Average recalls comparison (c) Average precisions comparison for workflows with more than one service (d) Average recalls comparison for workflows with more than one service

Recommendation Quality: Figure 3 compares the precision and recall of our algorithm in the two types of experiments aforementioned. For each diagram, the x axis represents the timeline, and the y axis represents the average precision/recall for each day if our recommendation algorithm is applied on the workflows published on the date. Figure 3(a) and (b) compare the recommendation precision and recall for all workflows in the testbed. For each figure, Test

1 and Test 2 curves represent the precision/recall of our algorithm applied on type 1 scenario (where N_{uow} only comprises all workflows published prior to the date) and type 2 scenario (where N_{uow} contains all workflows except the one under examination and its subsequent versions), respectively.

As it can be seen, type 2 experiment with a rich UoW network has demonstrated more effective results, especially in the early times. For type 1 experiment, the UoW network for those workflows in early experiments contains insufficient services and links to be used for the recommendation. However, as times goes by, both precision and recall improve by having a richer UoW network.

From the 1,719 workflows, we also removed the ones with only one service in order to investigate the results for more complex scenarios. Figure 3(c) and (d) present the comparison results of the algorithms for the remaining 511 workflows.

Over the entire testbed, the average precision/recall of our proposed method are 44/43% for type 2 experiments, comparing to 29/31% for type 1 experiments. For workflows containing at least two external services, the average precision/recall of our method are 52/43% for type 2 experiments, comparing to 34/25% for type 1 experiment.

Development Efforts Saved: Figure 4 shows that accumulated development efforts would have been saved by our approach, from the inception of myExperiment.org to May 2018. The baseline method is a keyword-based method, finding one service at a time. For each diagram, the x axis represents the timeline. Figure 4(a) examines the development steps saved. The y axis represents the development steps saved for each day, if our algorithm was applied on the workflows published on the date. For each workflow comprising N conceptual services, the baseline method will require N steps. In contrast, our approach recommends units of work each may comprising multiple services. Therefore, development steps may be saved. Assume our algorithm recommends K UoWs, each containing $|uow_i|$ services. We will have: $\sum_{i=1}^{K} |uow_i| = N$. If each uow is a single service, $K = N$; otherwise, $K < N$. For each uow_i, $|uow_i| - 1$ steps will be saved. Therefore, for the entire process with K UoWs, a total of $\sum_{i=1}^{K}(|uow_i| - 1) = N - K$ steps will be saved. For the testbed, we identified UoWs for 107 workflows under type 1 experiments, and 183 workflows under type 2 experiments. As shown in Fig. 4(a), our algorithm accumulatively would have saved 303 or 166 steps comparing to the baseline methods, in two types of experiments respectively.

Figure 4(b) studies the service linkages saving using our approach. Unlike the baseline method recommending only services and scientists having to chain services manually, our approach recommends units of work including the linkages among services. Consider again the same example above, where a workflow with N conceptual services and our algorithm recommending K UoWs. Using the baseline method, one has to make decision on C_N^2 connections. Using our method, a total of $\sum_{i=1}^{K} C_{|uow_i|}^2$ link consideration could have been saved. As shown in Fig. 4(b), a rich UoW network significantly facilitates linkage saving, from an accumulative 460 linkages comparing to about 242 linkages. As discussed earlier,

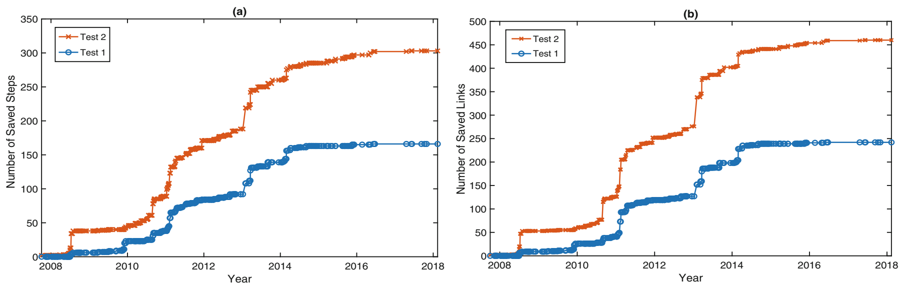


Fig. 4. Development efforts saved comparison (a) Steps saved (b) Links saved

chaining among data services may be a hard problem. As a result, our approach may significantly save scientists' time and let them focus on science.

Time Complexity: We also studied the performance of applying our recommendation algorithm online. Using the same example where a workflow with N conceptual service and a UoW network with S'' services and E'' edges, the time complexity would become $O(S''^{N}log(S''^{N}))$. First, finding candidate services will cost $O(S''N)$. Second, finding all their combinations for different clusters will cost $O(S''^{N})$. Third, finding all service instances in the network will cost $O(|S''|+|E''|)$. Fourth, finding the shortest path between services will cost $O(|S''|^2)$. Fifth, sorting candidates will cost $O(S''^{(}N+1)log(S''^{(}N+1)))$.

6 Conclusions

In this paper, we have presented a novel technique to facilitate workflow design and development. At each step, we recommend a runnable group comprising multiple services, based on a service social network constructed from service usage history. Such a technique stands out for four significant reasons. First, recommending multiple services instead of a single service shall shorten the overall design phase. Second, the recommended service group is derived from past workflow usage history with analogous context, so the recommendation becomes more trustworthy. Third, some unprecedented service collaboration patterns may be extracted through the cross-workflow mining over the service social network. Fourth, the recommended services are chained thus to save users efforts to interconnect them, which may be not only time consuming but also difficult.

We plan to continue our research in the following two directions. First, we will further study the scalability of our technique and tune performance. Second, we will study how to integrate our approach with existing methods focusing on other phases during service composition process, toward building an end-to-end service composition recommendation methodology and a tailored platform.

Acknowledgement. This work is partially supported by National Aeronautics and Space Administration under grant NNX16AB22G, and National Science Foundation under grant ACI-1443069.

References

1. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**, 5–51 (2003)
2. Bevilacqua, L., Furno, A., di Carlo, V.S., Zimeo, E.: A tool for automatic generation of WS-BPEL compositions from OWL-S described services. In: Proceedings of IEEE International Conference on Software, Knowledge Information, Industrial Management and Applications, Benevento, Italy, pp. 1–8 (2011)
3. Roy Chowdhury, S., Daniel, F., Casati, F.: Efficient, interactive recommendation of mashup composition knowledge. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 374–388. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25535-9_25
4. Deng, S., et al.: A recommendation system to facilitate business process modeling. IEEE Trans. Cybern. **47**(6), 1380–1394 (2017)
5. Elmeleegy, H., Ivan, A., Akkiraju, R., Goodwin, R.: MashupAdvisor: a recommendation tool for mashup development. In: Proceedings of IEEE International Conference on Web Services, pp. 337–344. IEEE, Beijing (2008)
6. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for mashups. Proc. VLDB Endow. **2**(1), 538–549 (2009)
7. Koop, D., Scheidegger, C.E., Callahan, S.P., Freire, J., Silva, C.T.: VisComplete: automating suggestions for visualization pipelines. IEEE Trans. Vis. Comput. Graph. **14**, 1691–1698 (2008)
8. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Comput. Surv. **48**(3), Article no. 33 (2016)
9. Li, C., Zhang, R., Huai, J., Guo, X., Sun, H.: A probabilistic approach for web service discovery. In: Proceedings of IEEE International Conference of Services Computing, pp. 49–56. IEEE, Santa Clara (2013)
10. McIlraith, S.I., Son, T.C., Zeng, H.: Semantic web services. IEEE Intell. Syst. **16**(2), 46–53 (2001)
11. Paik, I., Chen, W., Huhns, M.N.: A scalable architecture for automatic service composition. IEEE Trans. Serv. Comput. **7**(1), 82–95 (2014)
12. Rodríguez-Mier, P., Mucientes, M., Lama, M.: Hybrid optimization algorithm for large-scale QoS-aware service composition. IEEE Trans. Serv. Comput. **10**(4), 547–559 (2017)
13. Skiena, S.: Dijkstra's Algorithm. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, pp. 225–227. Addison-Wesley, Reading (1990)
14. Smirnov, S., Weidlich, M., Mendling, J., Weske, M.: Action patterns in business process models. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC/ServiceWave-2009. LNCS, vol. 5900, pp. 115–129. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10383-4_8
15. Wada, H., Suzuki, J., Yamano, Y., Oba, K.: E3: a multiobjective optimization framework for SLA-aware service composition. IEEE Trans. Serv. Comput. **5**(3), 358–372 (2012)
16. Xia, B., Fan, F., Tan, W., Huang, K., Zhang, J., Wu, C.: Category-aware API clustering and distributed recommendation for automatic mashup creation. IEEE Trans. Serv. Comput. **8**(5), 674–687 (2015)
17. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30**(5), 311–327 (2004)

18. Zhang, J., Liu, Q., Xu, K.: FlowRecommender: a workow recommendation technique for process provenance. In: Proceedings of the Australasian Data Mining Conference, pp. 55–61. Australian Computer Society, Inc. (2009)
19. Zhang, J., et al.: Climate analytics workflow recommendation as a service - provenance-driven automatic workflow mashup. In: Proceedings of IEEE International Conference on Web Services, pp. 89–97. IEEE, New York (2015)
20. Zhang, J., Tan, W., Alexander, J., Foster, I., Madduri, R.: Recommend-as-you-go: a novel approach supporting services-oriented scientific workflow reuse. In: Proceedings of IEEE International Conference on Services Computing, pp. 48–55. IEEE, Washington DC (2011)

Mobile Crowdsourced Sensors Selection for Journey Services

Ahmed Ben Said[1](✉), Abdelkarim Erradi[1], Azadeh Gharia Neiat[2], and Athman Bouguettaya[2]

[1] Department of Computer Science and Engineering, College of Engineering, Qatar University, Doha, Qatar
{abensaid,erradi}@qu.edu.qa
[2] School of Information Technologies, University of Sydney, Sydney, Australia
{azadeh.gharineiat,athman.bouguettaya}@sydney.edu.au

Abstract. We propose a mobile crowdsourced sensors selection approach to improve the journey planning service especially in areas where no wireless or vehicular sensors are available. We develop a location estimation model of journey services based on an unsupervised learning model to select and cluster the right mobile crowdsourced sensors that are accurately mapped to the right journey service. In our model, the mobile crowdsourced sensors trajectories are clustered based on common features such as speed and direction. Experimental results demonstrate that the proposed framework is efficient in selecting the right crowdsourced sensors.

Keywords: IoT · Travel planning service · Spatiotemporal data Crowdsourcing · Sensors selection · Unsupervised learning

1 Introduction

With the increasing use of mobile devices such as smartphones and wearable devices mobile crowdsourced sensing is emerging as a new sensing paradigm for obtaining required sensor data and services by soliciting contribution from the crowd. Storing, processing and managing continuous streams of crowdsensed data pose key challenges. The cloud offers a new paradigm, called sensor cloud [5,6] to efficiently handle these challenges.

In our previous works [7,8,10], we explore a new area in spatio-temporal travel planning by abstracting the problem using the service paradigm. We assume that we have a map consisting of spatial routes which in turn consist of segments. Each sensor cloud segment service is served by a journey vehicle (e.g., buses, trams or trains) and has a number of attributes and associated quality of service. Th functional attributes of a line segment service include GPS coordinates of the source and destination points and the mode of transportation (e.g. train service). Quality of service parameters include times of arrival and departure, accuracy, cost and so on. Therefore, a journey would consist of composing a set of line segment services on the map according to a set of functional and non-functional requirements. In [7,8,10], we made the assumption that the sensor cloud services are modeled by fixed sensors, i.e. sensors embedded on the journey vehicle e.g. bus, tram.

In this paper, we consider the scenario illustrated in Fig. 1 in which the crowd is the source of information. Indeed, we solely rely on commuters providing real-time geolocation data collected through their mobile devices (e.g. smartphone) instead of fixed sensors, called Mobile Crowdsourced Sensors (MCS). These MCS are abstracted on the cloud and can be used by the journey planning service to serve commuters' requests to find optimal journey plans. This is an interesting development since MCS represent an alternate source of information in absence of sensory infrastructure and eliminate the need to deploy costly sensory equipment. Commuters representing the source of crowdsourced sensors are willing to participate if they are convinced and well-incentivized, i.e. they are provided with a reward either as a service compensation or money [9]. For example, participants can benefit from enhanced journey planning and real-time transport network update using the collected data. In addition, they can get credit compensation according to the level of participation. In this scenario, we suppose that commuters are well-incentivized to participate in sharing their real-time locations.

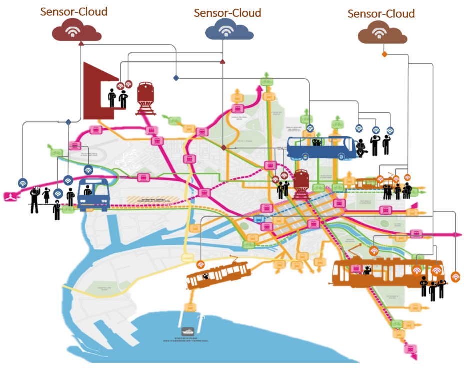


Fig. 1. Overview of public transport scenario powered by MCS

The key challenge of leveraging MCS is to identify and access the "right" crowdsourced sensors that are applicable to a particular journey planning request. This is important as MCS will move from one mode of transportation to another, i.e., they would serve different sets of sensor cloud services at different locations and times. As a result, there is a need to develop an efficient selection technique to accurately identify MCS that will be mapped to the right sensor cloud service.

This paper focuses on developing a MCS selection technique to enable an optimal journey planning powered by MCS. Specifically, we propose an unsupervised learning approach to select and cluster the right mobile crowdsourced sensors based on common patterns in their trajectories.

The main contributions of this paper are: (1) A new formal model of mobile crowdsourced sensors allowing access to the right MCS in space and time to enable identifying and tracking the location of journey vehicles. (2) A new unsupervised learning approach for clustering MCS. (3) Novel quality measures based on moving characteristic of MCS to assess the homogeneity of members of identified crowdsourced sensor clusters.

The rest of the paper is organized as follows: in Sect. 2, we review relevant related works. In Sect. 3, we present the problem formulation and the proposed model. In Sect. 4, we discuss the details of the proposed crowdsourced sensors selection and clustering algorithm that allows the identification of journey vehicles. In Sect. 5, we present and analyze the results of the experimental evaluation. Finally, conclusions and future work are presented in the last section.

2 Related Work

Several research proposals have focused on mobile crowdsourcing for journey planning service. Yu et al. [1] proposed a MCS-based travel package recommendation system. A profile is constructed for each user to leverage spatio-temporal features of check-in in points of interests (POI) which are hierarchically classified. Each POI is characterized by its periodic popularity. These information are then used in real-time to recommend personalized travel packages while taking into account user preferences, POI characteristics, and spatio-temporal constraints such as travel time and initial location. Ye et al. [2] used MCS to build the pattern map of the metro line, which can then be used for localization. The system consists of two phases: in the first one, patterns from user traces are extracted, and mined to identify the ones which are linked to specific tunnels. This allows to construct the graph of the metro line. In the second phase, the pattern map is made available on the cloud for users to download. When user travels using metro line, barometric pressure and magnetic fields data are logged along with stop and running events. Therefore, the train and user locations are known. Shin et al. [3] proposed a MCS-based approach for classification of transport mode. Authors collected information including date and time, x-, y-, and z-acceleration values, latitude and longitude. By analyzing these records, the walking pattern is characterized and used to segment the overall activities. To determine the travel mode of a vehicle-ride activity, the acceleration profile for each mode is estimated and used to classify particular acceleration behavior into one of the modes. In [4], authors proposed TrafficInfo, a participatory sensing based live public transport information service. Instead of relying costly sensing infrastructure, the proposed service relies on contribution from the crowd to visualize the actual position of the journey vehicle.

The aforementioned works consider the mapping between MCS and journey vehicle (tram, metro, etc...) as a prerequisite or assume that the crowd are fully cooperative and handle this mapping even though they can move from one mode of transportation to another or share erroneous information. However, this assumption does not always hold. Indeed, such task requires an effective incentive mechanism to motivate the participants. Furthermore, the crowd can be indifferent to handle this task or introduce erroneous information. Therefore it is essential to develop techniques for automatic selection of the right MCS that enable the estimation of the journey vehicle location in real time.

The availability of data enjoying spatio-temporal properties has elicited new data analysis paradigm to explore and extract new spatio-temporal patterns. Spatiotemporal clustering is the process of grouping data objects based on space and time relationships. Spatiotemporal clustering methods determine to which cluster a given object belongs based on different features such as the speed, the direction and the similarity in the trajectory origin and destination. Since trajectory is a sequence of time-stamped location points of a moving object through space, grouping moving trajectories is complex due to their continuous movement. Thus, more efforts are needed to discover the interaction and change in the spatiotemporal trajectory movements in order to achieve more accurate partitioning [11]. Recently, researchers are proposing modifications of existing clustering algorithms to make them more suitable for spatiotemporal data. Birant et al. [12] proposed a spatio temporal algorithm called ST-DBSCAN, an extension of the well known DBSCAN algorithm to the spatiotemporal domain. Avni et al. [13] also extended the Ordering Points to Identify the Clustering Structure (OPTIC) algorithm to cluster spatiotemporal data for taxi recommendation system. On the other hand, spatiotemporal pattern mining focuses on discovering hidden movement patterns from the trajectories of moving objects. Multiple methods were proposed to mine several types of movement patterns for groups of objects that move together in a near space and time. These patterns include periodic or repetitive pattern that concerns regular movement e.g. bird migration [14,15], flock [16], convoy [17], swarm [14], leadership [18] and chasing [19].

The evaluation of spatiotemporal clustering approaches remains an open and challenging issue. While the traditional clustering approaches require computation in single Euclidean space, the spatio-temporal clustering approaches need computation in multiple spaces [20]. In addition, grouping spatio-temporal data is affected by the large data size which leads to a trade-off between accurate clustering results and computational cost [21]. Clustering is also affected by noise and outliers. Additionally, the presence of clusters of different shapes, e.g. ellipsoid, and of unbalanced sizes may result in the inaccurate data partition. Indeed, some clustering algorithms, e.g. K-means, form clusters with a circular shape which leads to misleading results.

3 System Model

Our objective is to identify journey vehicles and track their location by relying solely on crowdsourced sensors. Therefore, it is important to select the right subset of crowdsourced sensors. In this context, a group of MCS associated to a journey vehicle are very likely to have similar spatiotemporal features. Our strategy consists of grouping or clustering the set of MCS that have common patterns.

In the following, we first define preliminary concepts and then present the problem statement. In the remainder, we refer to sensor cloud service as a journey service. A journey network is a spatial representation of journey services (i.e., bus, train etc) available in a given area. It consists of journey service routes composed of route segments (see Fig. 2). We formally define a route segment and journey service as follows:

Definition 1: Route segment. A route segment (rs_i) connects two nodes representing the source and destination points. It is identified by a tuple $<n_d, n_a, dist, speed, att>$ where:

- n_d: GPS coordinates of the departure node.
- n_a: GPS coordinates of the arrival node.
- *dist*: Distance between n_d and n_a.
- *speed*: Average travel speed along the route segment.
- *att*: Average travel time duration to traverse the route segment.

Definition 2: Journey Service. A Journey Service (JS) is modeled as a composition of route segments. JS is described by a tuple $<JS_{id}, RS, schedule>$ where:

- JS_{id}: A unique identifier of the service. We consider each inbound or outbound direction as a separate service (e.g., Bus-30-Manhattan-JFK-Express is a service).
- *RS*: A list of route segments $\{rs_1, rs_2, \ldots rs_n\}$ comprise the service route.
- *schedule*: Scheduled JS trips per day. It can also be represented by an average headway which is defined as the time difference between any two successive vehicles.

A journey service could be served by one or more Journey Vehicles (JV) such as buses.

Definition 3: Journey Vehicle. A JV is identified by a tuple $<v_s, dt, rs, loc, t_d, t_a>$ where:

- v_s: Journey service Id (e.g., bus line 100) whose journey vehicle is currently serving.
- *dt*: Departure time from the start of the journey.
- *rs*: Current route segment being traversed by the vehicle.
- *loc*: Current vehicle location.
- t_d: Departure time for the current route segment.
- t_a: Estimated arrival time to the next stop which is the arrival node of the current route segment.

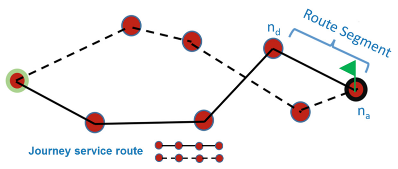


Fig. 2. A journey network consists of journey service routes which in turn consist of route segments

Crowdsourced sensors are constantly sending their location information to a cloud-hosted journey service. These spatio-temporal records can be modeled as MCS trajectory. Given a set of crowdsourced sensors represented by their trajectories $\mathcal{S} = \{Tr_1, Tr_2, \ldots, Tr_N\}$, where N is the number of trajectories, the proposed algorithm discovers clusters of crowdsourced sensors $\mathcal{C} = \{C_1, C_2, \ldots, C_M\}$, where M is the number of cluster centers.

Definition 4: Crowdsourced sensor trajectory. A crowdsourced sensor trajectory Tr_i is a set of sequential timestamped geolocations:

$Tr_i = [(p_1, t_1), (p_2, t_2) \ldots (p_L, t_L)]$. A geolocation p_i is a pair of latitude and longitude sent by the sensor at time t_i. L is the trajectory length. It can be different from one trajectory to another. We assume that trajectories are defined for the same time intervals. A trajectory segment is a pair of consecutive timestamped geolocations: $ts = [(p_j, t_j), (p_k, t_k)], t_j < t_k$. A trajectory of length L is composed of $L - 1$ trajectory segments. It is also characterized by its associated direction and speed. A cluster of crowdsourced sensors is a group of sub-trajectories as illustrated in Fig. 3.

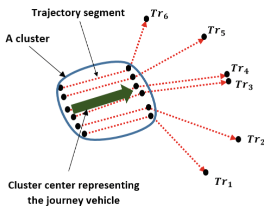


Fig. 3. An example illustrating a journey vehicle identification by clustering the crowdsourced sensor trajectories

A cluster center is an imaginary trajectory segment with specific characteristics i.e. start and end point as well as start and end time. This particular sub-trajectory is the representation of the journey vehicle.

Definition 5: Distance Function. A clustering algorithm, whether density based, partitional or hierarchical, is formulated using a distance measure. To take the particularity of our trajectory data structure into account, we propose to use the distance measure proposed by Lee et al. [22] illustrated in Fig. 4. Specifically, the distance between trajectory segments ts_1 and ts_2 is a linear combination of three distances: perpendicular distance $d_\perp$, parallel distance $d_{||}$ and angle distance d_θ:

$$d(ts_1, ts_2) = d_\perp(ts_1, ts_2) + d_{||}(ts_1, ts_2) + d_\theta(ts_1, ts_2) \tag{1}$$

where:

$$d_\perp(ts_1, ts_2) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \tag{2}$$

$$d_{||}(ts_1, ts_2) = \min(l_{||1}, l_{||2}) \tag{3}$$

$$d_\theta(ts_1, ts_2) = length(ts_2) \times sin(\theta) \tag{4}$$

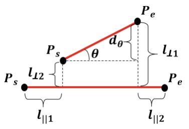


Fig. 4. The distance measure between two trajectory segments

For more accurate distance measure between two geolocations, the Haversine distance or Vincenty distance [23] can be used. However, for simplicity, we use the classic Euclidean distance to calculate $l_{||1}$, $l_{||2}$, $l_{\perp 1}$ and $l_{\perp 2}$ which is suitable for the small area of study used for evaluation.

Definition 6: ϵ-Neighborhood. The neighborhood of a trajectory segment ts_i with respect to $\epsilon > 0$, denoted $N_\epsilon(ts_i)$, is a subset of trajectory segments and defined as:

$$N_\epsilon(ts_i) = \{ts_j \mid d(ts_i, ts_j) \leq \epsilon\} \tag{5}$$

It is the set of trajectory segments whose distance to ts_i is less than a threshold ϵ.

Definition 7: Core Trajectory Segment. A trajectory segment ts_i is a core trajectory segment defined with respect to ϵ and $MinS > 0$ iff:

$$|N_\epsilon(ts_i)| \geq MinS \tag{6}$$

Where $|N_\epsilon(ts_i)|$ is the cardinality of $N_\epsilon(ts_i)$. A core trajectory segment highlights the presence of dense region. $MinS$ is a minimum number of neighbor trajectory segments required to form a dense region around ts_i.

Definition 8: Following Degree (FD). The following degree between two trajectory segments is established given the three possibilities illustrated in Fig. 5. It takes into account whether the trajectory segments are converging, diverging or parallel. It is defined as follows:

$$FD = \begin{cases} 1, & if \ \ d_1 = d_2 \\ -1, & if \ \ d_1 < d_2 \\ \frac{d_2}{d_1}, & if \ \ d_1 > d_2 \end{cases} \tag{7}$$

If the two trajectory segments originate from the same geolocation i.e. $d_1 = 0$, the trajectory segments are diverging and $FD = -1$. For the special case where $d_1 = d_2 = 0$, FD is equal to 1.

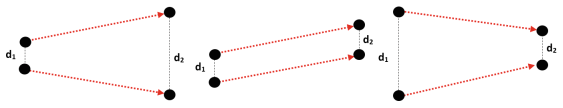


Fig. 5. Three following scenarios

Definition 9: Trajectory Segment Direction (DR). The trajectory segment direction is defined as the counterclockwise angle of the trajectory segment with respect to the reference line of Equator. The direction can be also derived from the accelerometer and the geomagnetic field sensor embedded on the smartphone.

Definition 10: Trajectory Segment Speed (SP). The trajectory segment is characterized by its speed. It is the distance between the departure node n_d and the arrival node n_a over the difference in time: $t_k - t_j$.

4 Mobile Crowdsourced Sensors Selection Algorithm

In this section, we propose our spatio-temporal crowdsourced sensors clustering algorithm for crowdsourced sensors selection. We first present the details of the algorithm. Then, we define the homogeneity score which is used to form clusters.

4.1 Spatio-Temporal Crowdsourced Sensors Clustering Algorithm

In classic clustering task, a high performance algorithm achieves a partition of objects where members of each cluster are as homogeneous as possible with respect to certain criteria. For example, a cluster should be as dense as possible. The density can be quantified using a discrepancy measure such as the variance.

Our strategy for trajectory segment clustering originates from the following intuitive idea: MCS contributing in identifying a journey vehicle associated to a journey service should (1) be as dense as possible and (2) share common spatio-temporal patterns such as speed, following degree and direction. Consequently, our algorithm seeks to identify dense regions with respect to predefined parameters: ϵ and $MinS$. These regions indicate the presence of potential clusters. However, it is important to identify clusters with the highest homogeneity among each member. Therefore, we define a homogeneity score. It captures the spatio-temporal dynamism of trajectory segments such as speed, direction and following degree and then identifies the cluster with the highest homogeneity.

Our algorithm, detailed in Algorithm 1, first identifies the list of core trajectory segments (line 3–6) at each timestamp. Then, it considers every core trajectory segment ts_i and its ϵ-neighborhood as a potential cluster. It seeks also to form clusters as homogeneous as possible with respect to a particular score called the Homogeneity Score (HS). A potential cluster is added to the set of clusters if it fulfills one of the following requirements:

– Its associated core trajectory segment has no other core trajectory segment in the list of its ϵ-neighborhood (line 10–12)
– Its associated core trajectory segment has the lowest HS among the neighbor core trajectory segments (line 14–17).

After a cluster of trajectory segments is formed, its cluster center can be used to identify the associated journey service. In this regard, we consider the vectorized version of the trajectory segments $\overrightarrow{ts_1}, \overrightarrow{ts_2}, \ldots, \overrightarrow{ts_N}$ where:

$$\overrightarrow{ts_i} = \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \\ t_{j+1} - t_j \end{pmatrix} \tag{8}$$

The average trajectory segment of N trajectory segments is defined as:

$$\vec{cc} = \frac{\overrightarrow{ts_1} + \overrightarrow{ts_2} + \ldots + \overrightarrow{ts_N}}{N} \tag{9}$$

The journey vehicle is identified by the trajectory segment associated with $\vec{cc}$. Therefore, the spatio-temporal properties of the cluster centers correspond to the journey vehicle properties. By gradually capturing the set of cluster center and therefore the journey vehicle, we continuously update the journey service and route segment attributes such as the speed and arrival time.

4.2 The Homogeneity Score

The Homogeneity Score HS is of paramount importance since it identifies the clusters and therefore the journey vehicles. HS is a combination of three scores: the following, the speed and the direction scores. It captures the spatio-temporal properties of MCS. Indeed, members of a cluster are supposed to follow each other with relatively the same speed and direction. These scores are defined as follows:

Algorithm 1. Crowdsourced Sensors Clustering

Input: Trajectory set $\mathcal{S}$, ϵ, $MinS$
Output: Identified journey vehicles

```
 1: List_{CC} ← ∅    # list of identified vehicles
    # Identify core trajectory segments
 2: for every time slot $\Delta t$ do
 3:    for $ts_i \in \mathcal{S}$ do
 4:       if $ts_i$ is a core trajectory segment (Definitions 5 and 6) then
 5:          Insert $ts_i$ in $List_{CS}$
 6:          Calculate $HS(ts_i)$ using (13)
 7:       end if
 8:    end for
 9: end for
    # Cluster trajectory segments
10: for every $ts_i \in List_{CS}$ do
11:    if $\forall\, ts_j \in N_\epsilon(ts_i)$; $ts_j \notin List_{CS}$ then
12:       Calculate the cluster center $cc$
13:       Insert $\big(ts_j, N_\epsilon(ts_j), cc\big)$ in $List_{CC}$
14:    else
15:       for each $ts_j \in \big\{N_\epsilon(ts_i) \cup ts_i\big\}$ & $ts_j \in List_{CS}$ do
16:          Identify $ts_j$ with the lowest $HS$
17:          Calculate the cluster center $cc$
18:          Insert $\big(ts_j, N_\epsilon(ts_j), cc\big)$ in $List_{CC}$
19:       end for
20:    end if
21: end for
```

Definition 11: Following Score (FS). The following score FS of a core trajectory segment ts_i is defined as:

$$FS = |N_\epsilon(ts_i)| - \sum_{j=1, j \neq i}^{|N_\epsilon(ts_i)|} FD(ts_i, ts_j) \tag{10}$$

It evaluates how different the following score of the core trajectory segment to the average following score of the cluster. A low FS score indicates better homogeneity in terms of following.

Definition 12: Speed Score (SS). It measures the homogeneity of the cluster in terms of speed. Indeed, a group of crowdsourced sensors should move with the homogeneous velocity. Given a potential cluster defined with respect to core trajectory segment ts_i, SS is expressed as:

$$SS = \frac{\left|\ SP(ts_i) - \frac{1}{|N_\epsilon(ts_i)|} \sum_{j}^{|N_\epsilon(ts_i)|} SP(ts_j)\ \right|}{\max_j \big\{SP(ts_j)\big\} - \min_j \big\{SP(ts_j)\big\}} \tag{11}$$

SS evaluates the difference between the core trajectory segment speed and the average speed of the clusters. A low SS score indicates better homogeneity in term of speed.

Definition 13: Direction Score (DS). It measures the homogeneity of the cluster in terms of direction. A group of crowdsourced sensors should move in the same direction. Given a potential cluster defined with respect to core trajectory segment ts_i, DS is expressed as:

$$DS = \frac{\left| \; DR(ts_i) - \frac{1}{|N_\epsilon(ts_i)|} \sum_{j}^{|N_\epsilon(ts_i)|} DR(ts_j) \; \right|}{\max\limits_j \left\{DR(ts_j)\right\} - \min\limits_j \left\{DR(ts_j)\right\}} \tag{12}$$

DS evaluates the difference between the core trajectory segment direction and the average direction of the clusters. A low DS score indicates better homogeneity in term of direction.

Definition 14: The Homogeneity Score (DS). HS is the linear combination of the three aforementioned scores. It is expressed as follows:

$$HS = \omega_1 \cdot FS + \omega_2 \cdot SS + \omega_3 \cdot DS \tag{13}$$

Where ω_1, ω_2 and ω_3 are tunable weights to adjust the contribution of each score. The optimal cluster achieves the lowest HS.

5 Experimental Results

We conduct a set of experiments to show the effectiveness of our approach in terms of Sum of Squared Error and accuracy.

5.1 Experimental Setup

For our experiments, we use the public bus transport dataset of New York City[1]. The dataset tracks 90 bus services across New York city for a full day. The buses arrival and departure times are recorded along with the geolocation of each station. Each station has a unique id. Between two consecutive stations, we randomly generate 40 geolocations with unique ids to simulate trajectories of MCS.

The simulated sensors may be widespread and therefore can be out of the route segments of interest. To deal with this issue, we used the following heuristic to pre-filter the irrelevant sensors. Given the historic journeys of vehicles, we establish the full regular route of every journey service such as a bus service. We consider the complete geolocations of every bus stop as illustrated in Fig. 6 to derive the static service route. This data is static (e.g. station geolocations) and

[1] web.mta.info/developers/MTA-Bus-Time-historical-data.html.

represent the journey service route as advertised by the service provider. Since our objective is to spatiotemporally identify vehicles through MCS, we only consider the crowdsourced sensors within a buffer area with radius R from the path line. Indeed, the objective is to filter out irrelevant MCS, i.e. the ones that do not contribute to identifying the journey vehicle and therefore the journey service.

To assess the effectiveness of the algorithms, we use the Sum of Squared Error (SSE). It reflects the overall compactness of the obtained data partition by calculating the distance between the center of each cluster and its associated objects. Therefore, the best clustering yields to the minimum SSE value which is computed as follows:

$$SSE = \sum\nolimits_{i=1}^{num\_cluster} \left(\frac{1}{2|C_i|} \sum_{ts_k \in C_i} \sum_{ts_l \in C_i} d(ts_k, ts_l) \right) \tag{14}$$

We also propose a new index to evaluate the trajectory clustering results. Inspired by the Xie-Beni cluster validity index [24], our proposed index named Tra-Xie-Beni (Tra-XB), takes into consideration the intra-cluster homogeneity as well as the inter-cluster separation. It is expressed as follows:

$$Tra\text{-}XB = \frac{\sum\limits_{i=1}^{num\_cluster} \sum\limits_{ts_j \in C_i} d(ts_j, cc_i)}{\min\limits_{i,j} d(cc_i, cc_j)} \tag{15}$$

The nominator term calculates the distance between every cluster cc_i and $ts_j \in C_i$. This quantifies the compactness of every cluster. The denominator evaluates the separation between clusters which is the minimum distance between all cluster centers. Therefore, the best partition corresponds to the minimum value of Tra-XB. In addition, we evaluate the spatial and temporal accuracy of both approaches by: 1 - Calculating the spatial distance $d(n_a, \hat{n}_a)$ between the true destination node of the journey vehicle n_a and the estimated destination point $\hat{n}_a$ computed by each algorithm. 2 - Calculating the temporal error $att - \hat{att}$ between the actual average travel time att of the journey vehicle and the average arrival time $\hat{att}$ computed by each algorithm. The estimated parameters ($\hat{n_a}$, $\hat{att}$) are identified by determining the closest cluster center end point and its associated timestamp.

We provide a quantitative analysis for the first 30 timestamps of the schedule, although our findings can be reproduced for any desired period. For the simulation settings, we set $\epsilon = 0.002$ and $MinS = 17$ for the proposed algorithm while we choose for Traclus the optimal values of $MinS$ and ϵ i.e. the ones achieving the best performance. Similarly, we report the best performance achieved by ST-DBSCAN. We also set $\omega_1 = \omega_2 = \omega_3 = 1$ and the preprocessing radius $R = 10\,\text{m}$.

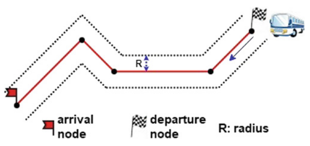
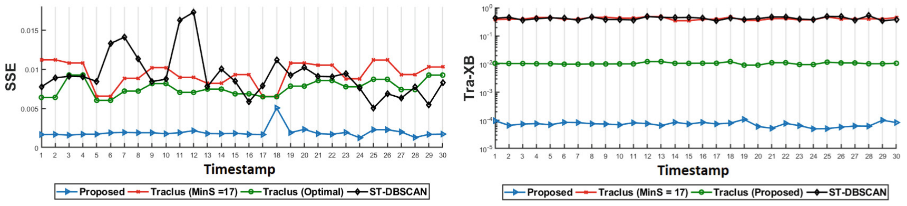
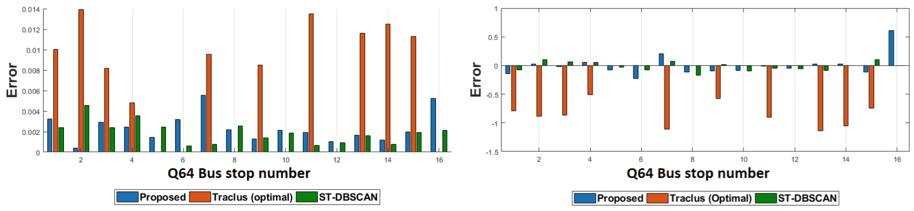


Fig. 6. Full bus route from the departure station to the arrival station



(a) SSE index for the first 30 timestamps (b) Tra-XB index for the next 30 timestamps

Fig. 7. Spatiotemporal accuracy



(a) Q64: spatial error (b) Q64: temporal error

Fig. 8. Spatiotemporal error

5.2 Discussion of Evaluation Results

Figure 7(a) depicts the SSE results of the algorithms for the first 30 timestamps of the schedule. We notice that the proposed clustering approach achieves better performance in terms of cluster compactness compared to ST-DBSCAN and Traclus with two settings: Optimal $MinS$ value and $MinS = 17$. This performance is confirmed by the Tra-XB index results illustrated in Fig. 7(b). Therefore, we can conclude that the proposed algorithm achieves the best clustering performance with the best separation between clusters.

Figure 8(a) and (b) represent the spatiotemporal evaluation of Q64 bus service starting from 164 Street/Jewel Avenue station at midnight to 108 Street/Queens Boulevard arriving at 9 min past midnight. The whole journey consists of 16 route segments. First, we notice that Traclus algorithm was not able to identify a cluster (5^{th} and 6^{th} Q64 bus instance) and therefore failed to identify the journey vehicle. On the other hand, the proposed algorithm and ST-DBSCAN exhibit better spatiotemporal accuracy. For example, for the 11^{th} Q64 bus instance, the spatial error achieved by the proposed algorithm is 0.002, 15 times less than the error achieved by Traclus. ST-DBSCAN achieves slightly better performance with spatial error = 0.0007. Furthermore, the proposed approach and ST-DBSCAN compute t_e with almost 100% accuracy (error = −0.08 s and −0.0475 s) unlike Traclus which achieved a temporal error of around 1 s.

We conclude that the proposed approach achieves better performance in accurately identifying vehicles and estimating their location and their arrival time to the next stop. It also achieves competitive performance compared to ST-DBSCAN. This performance can be explained by the capacity of the proposed approach to better capture the dynamism of the objects to be clustered since it takes into account several features such as speed, direction and following degree unlike density-based algorithm such as Traclus.

6 Conclusion

This paper proposed an approach to integrate real-time sensory data collected from multiple mobile crowdsourced sensors (MCS) to find a better journey service. We proposed and evaluated a clustering algorithm to select the right crowdsourced sensors, which enables the identification of the journey vehicles. This also helps to estimate journey services' location and arrival time to the next stop. The proposed algorithm takes into account the following degree, speed and direction of the crowdsourced sensors to build clusters of moving objects. This cluster allows the identification of journey vehicles. Experimental results demonstrate the effectiveness of the proposed algorithm in achieving better performance in terms of cluster compactness compared to the existing approaches. In future work, we will analyze the computation complexity of our approach and develop an enhanced algorithm to detect clusters. Devising incentive mechanisms to encourage commuters to participate and contribute as a sensor is another interesting future work direction.

Acknowledgment. This research was made possible by NPRP 9-224-1-049 grant from the Qatar National Research Fund (a member of The Qatar Foundation) and DP160100149 and LE180100158 grants from Australian Research Council. The statements made herein are solely the responsibility of the authors.

References

1. Yu, Z., Feng, Y., Xu, H., Zhou, X.: Recommending travel packages based on mobile crowdsourced data. IEEE Commun. Mag. **52**, 56–62 (2014)
2. Ye, H., Gu, T., Tao, X., Lu, J.: Crowdsourced smartphone sensing for localization in metro trains. In: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1–9 (2014)
3. Shin, D., et al.: Urban sensing: using smartphones for transportation mode classification. Comput., Environ. Urban Syst. **53**, 76–86 (2015)
4. Farkas, K., Nagy, A.Z., Tomaás, T., Szábo, R.: Participatory sensing based real-time public transport information service. IEEE International Conference on Pervasive Computing and Communications Demonstrations, pp. 141–144 (2014)
5. Ahmed, K., Gregory, M.: Integrating wireless sensor networks with cloud computing. In: Seventh International Conference on Mobile Ad-Hoc and Sensor Networks, pp. 364–366 (2011)

6. Neiat, A.G., Bouguettaya, A.: Crowdsourcing of Sensor Cloud Services. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91536-4
7. Ghari Neiat, A., Bouguettaya, A., Sellis, T.: Spatio-temporal composition of crowdsourced services. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 373–382. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_26
8. Neiat, A.G., Bouguettaya, A., Sellis, T., Dong, H.: Failure-proof spatio-temporal composition of sensor cloud services. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 368–377. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_26
9. Zhang, X., et al.: Incentives for mobile crowd sensing: a survey. IEEE Commun. Surv. Tutor. **18**, 54–67 (2016)
10. Neiat, A.G., Bouguettaya, A., Sellis, T., Mistry, S.: Crowdsourced coverage as a service: two-level composition of sensor cloud services. IEEE Trans. Knowl. Data Eng. **29**, 1384–1397 (2017)
11. Huang, Y., Chen, C., Dong, P.: Modeling herds and their evolvements from trajectory data. In: Cova, T.J., Miller, H.J., Beard, K., Frank, A.U., Goodchild, M.F. (eds.) GIScience 2008. LNCS, vol. 5266, pp. 90–105. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87473-7_6
12. Birant, D., Kut, A.: ST-DBSCAN: an algorithm for clustering spatial-temporal data. Data Knowl. Eng. **60**, 208–221 (2007)
13. Avni, M., Viswanath, G., Vinaya, N., ST-OPTICS: a spatial-temporal clustering algorithm with time recommendations for taxi services, Ph.D. thesis (2017)
14. Li, Z., Ding, B., Han, J., Kays, R.: Swarm: mining relaxed temporal moving object clusters. Proc. VLDB Endow. **3**, 723–734 (2010)
15. Li, Z., Ding, B., Han, J., Kays, R., Nye, P.: Mining periodic behaviors for moving objects. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1099–1108 (2010)
16. Wachowicz, M., Ong, R., Renso, C., Nanni, M.: Finding moving flock patterns among pedestrians through collective coherence. Int. J. Geogr. Inf. Sci. **25**, 1849–1864 (2011)
17. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. Proc. VLDB Endow. **1**, 1068–1080 (2008)
18. Andersson, M., Gudmundsson, J., Laube, P., Wolle, T.: Reporting leadership patterns among trajectories. In: Proceedings of the 2007 ACM Symposium on Applied Computing, pp. 3–7 (2007)
19. de Lucca Siqueira, F., Bogorny, V.: Discovering chasing behavior in moving object trajectories. Trans. GIS **15**, 667–688 (2011)
20. Shao, W., Salim, F.D., Song, A., Bouguettaya, A.: Clustering big spatiotemporal-interval data. IEEE Trans. Big Data **2**, 190–203 (2016)
21. Jiang, Z., Shekhar, S.: Spatial and spatiotemporal big data science. In: Jiang, Z., Shekhar, S. (eds.) Spatial Big Data Science, pp. 15–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60195-3_2
22. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD, pp. 593–604 (2007)
23. Mahmoud, H., Akkari, N.: Shortest path calculation: a comparative study for location-based recommender system. In: 2016 World Symposium on Computer Applications and Research (WSCAR), pp. 1–5 (2016)
24. Xie, X.L.: A validity measure for fuzzy clustering. IEEE Trans. Pattern Anal. Mach. Intell. **13**, 841–847 (1991)

RLRecommender: A Representation-Learning-Based Recommendation Method for Business Process Modeling

Huaqing Wang, Lijie Wen(✉), Li Lin, and Jianmin Wang

School of Software, Tsinghua University, Beijing, China
whq16@mails.tsinghua.edu.cn, veralin1994@gmail.com,
{wenlj,jimwang}@tsinghua.edu.cn

Abstract. Most traditional business process recommendation methods cannot deal with complex structures such as interacting loops, and they cannot handle large complex datasets with a great quantity of processes and activities. To address these issues, RLRecommender, a method based on representation learning, is proposed. RLRecommender extracts three kinds of relation sets from the models, both activities and relations between them are projected into a continuous low-dimensional space, and proper activity nodes are recommended by comparing the distances in the space. The experimental results show that our method not only outperforms other baselines on small dataset, but also performs effectively on large dataset.

Keywords: Business process modeling · Ordering relations
Representation learning · Recommendation

1 Introduction

Business process models are constructed to describe the prescribed behavior of business processes. One method to improve process modeling is process mining. Nonetheless, most process mining algorithms rely on completely purified event logs and real-world event logs are often noisy in most cases. Hence, process modeling is usually conducted by analysts manually. Nevertheless, modeling a business process from scratch is also highly complex, error-prone and time-consuming. Thus, an accurate and efficient business process recommendation method is required.

Business process recommendation has already been used in both academic research and industry applications, and there are several existing works [1,2]. These traditional methods recommend the next nodes of an uncompleted model by iterating all parts of all processes, and they have two main limitations:

1. These methods cannot deal with complex structures such as interacting loops in Fig. 1 (i.e., B and D, and C and D).

2. Existing methods cannot deal with large complex datasets with abundant processes, whose activities complicatedly relate to each other.

To address these problems, business process repositories should be considered as a whole. We find that business process repositories have several properties. First, quite a few activities appear frequently through most processes. Second, the flows in process models relate activities with each other. Thus, regarding activities and relations between them as nodes and edges, process repositories can be transformed into multi-relational data.

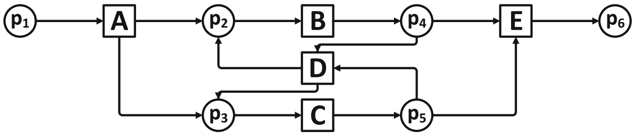


Fig. 1. An example for interacting loops

Recently, machine learning became a hot topic and there are a variety of representation learning models to handle multi-relational data [3]. In representation learning models, multi-relational data is a directed graph whose nodes are entities and edges are relations, and connections are represented as triples (*head entity, relation, tail entity*). Existing representation learning models embed both entities and relations into a continuous low-dimensional vector space [3], and have been proven simple and effective in tasks such as entity prediction.

Motivated by this idea, RLRecommender, a representation-learning-based recommendation method for process modeling, is proposed. To the best of our knowledge, this is the first work that resolves process recommendation problem using representation learning.

The remainder is organized as follows. In Sect. 2, we introduce RLRecommender in detail. Experimental results are given in Sect. 3. Section 4 concludes the paper and sketches future work.

2 The Representation-Learning-Based Method

Our method RLRecommender is composed of three phases: preprocessing, training and recommending as shown in Fig. 2. In this section, these three phases will be presented one by one.

2.1 Preprocessing

Our first phase focuses on extracting relations between activities from Petri nets. The definition of Petri nets can be found in [4]. Since the generated relation sets can be saved at back-end, this part is conducted only once.

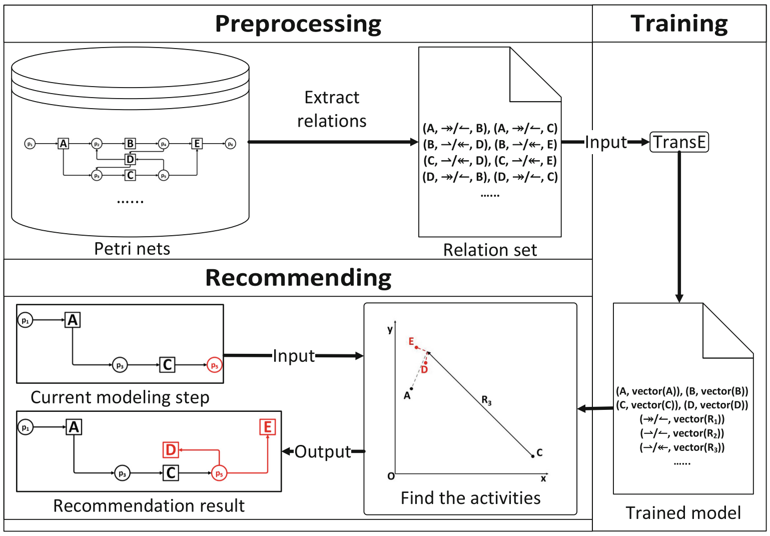


Fig. 2. Overview of our method RLRecommender

"Direct After" Relations. For relation extraction, our first strategy (i.e. RLafter) is to use only "direct after" relations, which is defined as follow.

Definition 1 *(Direct after). In a Petri net $N = (P, T, F)$, two transitions $x, y \in T$ are in "direct after" relation (denoted as $x \rightarrow y$) $\iff \exists p \in x\bullet$, s.t. $y \in p\bullet$.*

The "direct after" relations can correctly depict the connections of transitions. However, it cannot tell the differences between AND-splits and XOR-splits.

"Direct Causal" Relations. A combination of "direct causal" and "direct inverse causal" is used as the relations as the second strategy (i.e. RLcausal). We first define the two "direct causal" relations.

Definition 2 *(Always/sometimes direct causal). In a Petri net $N = (P, T, F)$:*

- *two transitions $x, y \in T$ are in "always direct causal" relation (denoted as $x \twoheadrightarrow y$) $\iff \exists p \in x\bullet$, s.t. $p\bullet = \{y\}$,*
- *two transitions $x, y \in T$ are in "sometimes direct causal" relation (denoted as $x \rightharpoonup y$) $\iff x \rightarrow y \wedge x \not\twoheadrightarrow y$.*

Two "direct inverse causal" relations are defined in a similar way (i.e. "always inverse direct causal" relation (denoted as $x \twoheadleftarrow y$) and "sometimes inverse direct causal" relation (denoted as $x \leftharpoonup y$)).

The two "direct causal" relations describe the possible orders of transitions, but they fail to distinguish transition groups $\{A, B\}$ and $\{C, D\}$ of the model in Fig. 3.

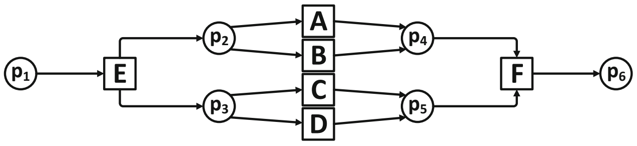


Fig. 3. An example that two "direct causal" relations cannot handle

"Direct Concurrent" Relations. We introduce "direct concurrent" relation in the following. A combination of two "direct causal" relations and "direct concurrent" relation is used as the third strategy (i.e. RLconcurrent).

Definition 3 *(Always/never/sometimes direct concurrent). In a Petri net $N = (P, T, F)$:*

- *two transitions $x, y \in T$ are in "never direct concurrent" relation (denoted as $x \nparallel y$) $\iff \exists p \in P$, s.t. $x, y \in p\bullet$,*
- *two transitions $x, y \in T$ are in "sometimes direct concurrent" relation (denoted as $x \Vdash y$) $\iff \exists p_1, p_2 \in t\bullet$, $t \in T, p_1 \neq p_2$, s.t. $(|p_1\bullet| > 1 \vee |p_2\bullet| > 1) \wedge x \in p_1\bullet \wedge y \in p_2\bullet \wedge \neg(x \nparallel y)$,*
- *two transitions $x, y \in T$ are in "always direct concurrent" relation (denoted as $x \parallel y$) $\iff \exists p_1, p_2 \in t\bullet$, $t \in T, p_1 \neq p_2$, s.t. $p_1\bullet = \{x\} \wedge p_2\bullet = \{y\} \wedge \neg(x \nparallel y) \wedge \neg(x \Vdash y)$.*

When generating relation set, each "direct concurrent" relation would only be recorded once between any pair of activities in the specific order such as dictionary order (e.g. $B \parallel C$ is recorded while $C \parallel B$ not).

The relation sets extracted from the model in Fig. 1 are shown in Table 1.

Table 1. The relation sets (direct after|direct causal/direct inverse causal|direct concurrent) extracted from the process model in Fig. 1

	A	B	C	D	E
A		$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$	$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$		
B			$\mid \quad \mid \quad \parallel$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$
C				$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$
D		$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$	$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$		$\mid \quad \mid \quad \nparallel$
E					

2.2 Training a Representation Learning Model

The second phase of our method is to learn a representation learning model through generated relation sets. This part can be done at back-end and the trained model can be saved and used for recommendation. The training model we use is based on TransE [3].

For the training relation set T_r, each triple (*precursor activity, relation, successor activity*) (denoted as $(p,\ r,\ s)$) is composed of two activities $p,\ s \in \mathscr{A}$ (the set of activities) and one relation $r \in \mathscr{R}$ (the set of relations). The embeddings take values in $\mathbb{R}^k$ ($\mathbb{R}^k$ means continuous k-dimensional vector space where k is a model hyperparameter) and are denoted with the same letters, in boldface characters (i.e. $\boldsymbol{p},\ \boldsymbol{r},\ \boldsymbol{s} \in \mathbb{R}^k$). We want $\boldsymbol{p} + \boldsymbol{r} \approx \boldsymbol{s}$ when $(p,\ r,\ s)$ holds and $\boldsymbol{p} + \boldsymbol{r}$ is far away from $\boldsymbol{s}$ otherwise. We use dissimilarity measure function d as the distance measurement, and in this paper d stands for L1-norm (i.e. Manhattan distance) while there are other choices such as L2-norm (i.e. Euclidean metric).

We minimize the margin-based loss function as the objective for training:

$$L = \sum_{(p,r,s)\in T_r} \sum_{(p^{'},r,s^{'})\in T_r^{'}} max(\gamma + d(\boldsymbol{p}+\boldsymbol{r},\boldsymbol{s}) - d(\boldsymbol{p}^{'}+\boldsymbol{r},\boldsymbol{s}^{'}),0) \tag{1}$$

$$T_r^{'} = \{(p^{'},r,s)|p^{'} \in \mathscr{A}\} \cup \{(p,r,s^{'})|s^{'} \in \mathscr{A}\} - T_r \tag{2}$$

where γ is a margin hyperparameter, and $T_r^{'}$ is a negative sampling set of training set T_r. The triples in $T_r^{'}$ are generated by replacing p or s in a triple in T_r by another activity. Note that a triple already in T_r will not be generated in $T_r^{'}$. The loss function L can reach minimum value only when $d(\boldsymbol{p}+\boldsymbol{r},\boldsymbol{s}) < d(\boldsymbol{p}^{'}+\boldsymbol{r},\boldsymbol{s}^{'})$ for most $(p,\ r,\ s)$.

2.3 Recommending

Our final phase is to recommend the activities using the trained model. After training, the embedded vectors for activities and relations can be used for recommendation by predicting the successor activity in a triple. During process modeling, the most suitable activities are recommended by traversing all activity vectors and calculating the distance with the sum of the vector of the previous activity and the vector of the proper relation. Notice that the proper relation can be either specified by user or chosen by the recommendation system.

More specifically, during the construction of one process, suppose the previous activity is $p_i \in \mathscr{A}$. For each relation set, we first check if each $r_i \in \mathscr{R}$ is valid after current position. Then we calculate the vector distance $d(\boldsymbol{p}_i + \boldsymbol{r}_i, \boldsymbol{s}_i)$ for each $s_i \in \mathscr{A}$. We then recommend K activities whose $d(\boldsymbol{p}_i + \boldsymbol{r}_i, \boldsymbol{s}_i)$ are the smallest. Here K is a parameter determined by user.

3 Experimental Evaluation

In this section, we first introduce the datasets and experiment settings. Then the results on two different datasets are presented. The code, datasets and results of our experiment are publicly accessible from GitHub[1].

[1] https://github.com/THUBPM/RLRecommender/.

3.1 Datasets and Experiment Settings

Datasets. In this paper, two datasets are used to evaluate our method, including large real dataset (LRD) [5] and small real dataset (SRD) [2]. The statistics of them are listed in Table 2.

Notice that for LRD, we only use BPMN 1.1, BPMN 2.0 [6] and Petri net models. All models are lowercased and only letters are retained, and non-English models and the ones without activity names are deleted from the dataset. BPMN models are transformed to Petri nets using the method in [7].

Table 2. Statistics of the two datasets used in the experiments

	LRD	SRD
#Processes	23576	221
#Distinct activities	36801	52
#Activities per process	2-380	5-15
AND/XOR-split/join	True	True
Cycle	True	True
Other complex structures	True	False

Benchmarks. We use *HitRate* and *F1 score* to estimate the performance of each method. For each position in a model, a list of K activity nodes are recommended, denoted by A_r. The set of activities that the real process actually has at the current position is denoted by A_t.

HitRate is the percentage of hits, which is defined as follow:

$$HitRate = \#hit/\#rec \tag{3}$$

where $\#hit$ is the number of times when $A_r \cap A_t \neq \varnothing$, and $\#rec$ is the number of recommendations.

F1 score is the harmonic mean of *precision* and *recall*, where *precision* and *recall* are the percentage of correctly recommended activities in recommended activity set and ground-truth activity set respectively. *F1 score* is defined as follow:

$$F1 = (2 \times Precision \times Recall) \,/\, (Precision + Recall) \tag{4}$$

$$Precision = (\Sigma|A_r \cap A_t|) \,/\, (\Sigma|A_r|) \tag{5}$$

$$Recall = (\Sigma|A_r \cap A_t|) \,/\, (\Sigma|A_t|) \tag{6}$$

Experiment Settings. Since the small dataset, benchmarks and experiment protocol we use are the same as those in [2], we use the settings determined in this paper for other state-of-the-art approaches.

For our method, we set the embedding dimension k to 200, the margin value γ in the loss function to 1.0, and the learning rate to 0.001. For the small and large datasets, we train our model for 5000 and 1000 rounds, respectively.

The user could set K to a proper value due to the size of the dataset. We conduct our experiment when recommendation number K is 1 to 5, we also show the result for $K = 10$ on LRD.

Experiments are conducted on an Intel Xeon E5-2620 v4@2.10 GHz CPU computer with 128 G RAM, and three GeForce GTX 1080Ti GPU are used.

Experiment Protocol. We perform all experiments on SRD using five-fold cross-validation. For LRD, we partition the 23576 models into 3041 models for the testing set, 1759 models for the validation set and the rest for the training set. For each model in the testing set, we conduct the experiment step by step for each position.

For SRD, our method takes 2–3 ms per recommendation. For LRD, our method takes 0.1–0.3 s per recommendation.

3.2 Evaluation Based on Large Complex Real-Life Dataset

The other recommendation system failed to deal with LRD while our RLRecommender is naturally capable of this situation. So we only conduct our methods on this dataset. Figure 4 shows the experimental results. We can conclude the following:

1. All three methods achieve good accuracy performance. Since this dataset has 36801 distinct activities, we also test the situation when $K = 10$, and the *HitRate* is up to 89.2%, which is highly acceptable in real life.
2. Our methods cannot handle the situation when $K = 1$, because the large dataset contains too many activities and all models are complicated. Nevertheless, our methods perform well when $K \geq 2$.

3.3 Evaluation Based on Small Real-Life Dataset

We compared with three methods MCSD, xGED and xSED in [2] on SRD. The results are shown in Fig. 5, from which we can observe that:

1. The RLRecommender based on "causal" and "concurrent" relations outperforms other methods on *HitRate* as recommendation number K increases.
2. For *F1 score*, our methods outperform the others most of the time. Especially, when $K = 5$, the improvement of *F1 score* is up to 14.4%.
3. The methods based on "causal" and "concurrent" outperform "after" on *HitRate*, while the latter performs better on *F1 score* when K increases.

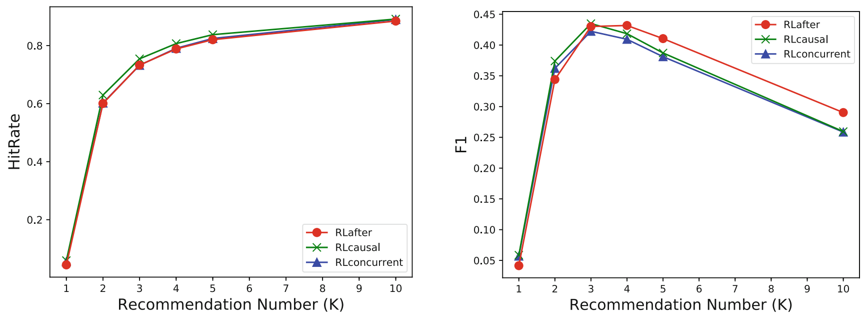


Fig. 4. Experimental results on LRD

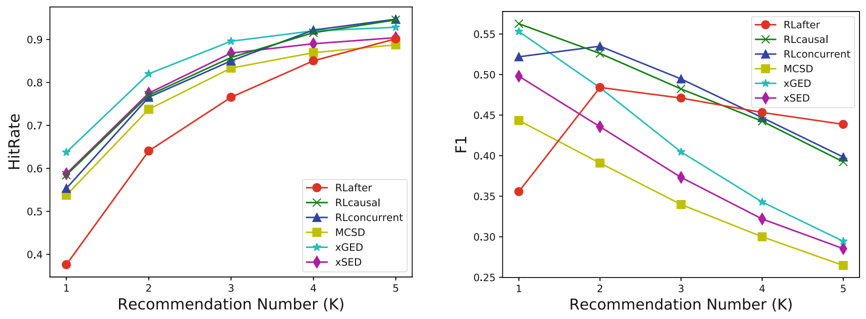


Fig. 5. Experimental results on SRD

4. When K is small, such as $K = 1$, traditional methods have higher *HitRate* since they iterate all structures of the models and this dataset is small and do not contain complex structures. Nevertheless, our methods still achieve better performance on *F1 score*.

4 Conclusion and Future Work

A representation-learning-based recommendation method named RLRecommender is proposed and it is the first recommendation method for process modeling using representation learning. It first extracts the relations between activities with three strategies. Relation sets are then used to train the representation learning model. The trained model is used in recommendation by finding the nearest vector for each modeling step. Experiments show that RLRecommender not only outperforms the state-of-the-art recommendation algorithms on small real dataset, but also performs uniquely well on large real dataset.

Our future work would mainly focus on improving the accuracy by modifying the current representation learning model. The current model only considers one previous activity during recommendation, while several activities can be used during one recommendation. We can also generate other kinds of relation set which may have positive effect on performance. Preliminary clustering can also be conducted on activity labels in order to make recommendations based on activities that relate to one another.

Acknowledgements. The work was supported by the National Key Research and Development Program of China (No. 2016YFB1001101), the National Nature Science Foundation of China (No. 61472207, No. 71690231), and BNRist.

References

1. Zhang, J., Liu, Q., Xu, K.: FlowRecommender: a workflow recommendation technique for process provenance. In: Proceedings of the Eighth Australasian Data Mining Conference-Volume 101, pp. 55–61. Australian Computer Society, Inc. (2009)
2. Deng, S., et al.: A recommendation system to facilitate business process modeling. IEEE Trans. Cybern. **47**(6), 1380–1394 (2017)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems, pp. 2787–2795 (2013)
4. Murata, T.: Petri nets: properties, analysis and applications. Proceed. IEEE **77**(4), 541–580 (1989)
5. BPM Academic Initiative: BPMAI process model collection. http://bpmai.org/
6. OMG: Business Process Modeling Notation (BPMN) Version 2.0. (2011)
7. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inform. Softw. Technol. **50**(12), 1281–1294 (2008)

Service Analytics

Domain Knowledge Driven Key Term Extraction for IT Services

Prateeti Mohapatra[1(✉)], Yu Deng[2], Abhirut Gupta[1], Gargi Dasgupta[1], Amit Paradkar[2], Ruchi Mahindru[2], Daniela Rosu[2], Shu Tao[2], and Pooja Aggarwal[1]

[1] IBM Research, Bangalore, India
pramoh01@in.ibm.com
[2] IBM Research, Yorktown Heights, NY, USA

Abstract. IT service support agents are trained on knowledge sources with large volumes of domain-specific documents, including product manuals and troubleshooting contents. Self-assist applications, such as search and support chat-bots must integrate such knowledge in order to conduct effective user interactions. In particular, the very large volume of domain-specific terms referenced in training documents must be accurately identified and qualified for relevance to specific context of support actions. We propose a weakly-supervised approach for extraction of key terms from IT support documents. The approach integrates domain knowledge to refine the extraction results. Our approach obviates the need for extensive expert work creating manual annotation and dictionary collection, as typically required in traditional supervised solutions, as well as the limited accuracy obtained in unsupervised methods. Results show that domain knowledge based refinement helps improve the overall accuracy of mined key terms by 25–30%.

Keywords: Key term extraction · Domain knowledge · IT support

1 Introduction

In IT services, the goal is to help customers resolve defects or usage issues for products or services they acquire. Agent-assist and self-assist applications are industry staples that help address the content complexity challenge [2]. These applications leverage AI technology to understand user requests and provide solutions or recommendations for next steps. A key part of understanding user requests involves identification of the relevant key terms in the domain.

The business of IT services is highly challenging, as customer satisfaction is directly influenced by speed and quality of support resolution provided. Most challenges stem from the large volume of products, components, their configuration context and documentation that have to be at the fingertips of technical support agents. AI technology based components are trained at large on product documents, but a considerable performance boost is achieved by explicitly utilizing domain-specific terms, such as names and actions related to products and

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 489–504, 2018.
https://doi.org/10.1007/978-3-030-03596-9_35

components. These terms become the key terms or features that help evolve the functionality for request and response understanding in those components, as they provide discriminative topical information about the content [22]. Furthermore, key terms in a document are building blocks for constructing a knowledge graph, which provides context information and inference capabilities for search and chat-bots.

It is a challenge to identify all relevant domain-specific key terms across staggering volumes of installation guides, user manuals and troubleshooting documents. For instance, there are over 100K relevant articles for IBM Server Power Platform[1], and over 60k documents for the storage brand of Storwize[2]. While the manual approach of identifying key terms is the most accurate, it is not feasible to do so for technical support content. Automation and accuracy of key term extraction are critical solution requirements. In this paper, we address the problem of automatically extracting key terms in IT services content, and propose and evaluate the accuracy of several methods. Namely, domain-specific key term extraction is to identify lexical expressions that represent important concepts in a given domain. These key terms are building blocks of entity recognition which affects the quality of cognitive applications on top of search, chat-bot/dialog systems, and knowledge graph creation etc.

Key term extraction in the literature has focused on detecting mentions of person, place, and organization [15,20]. Large open knowledge bases like Yago, DBPedia and Wordnet include general concepts, with limited coverage for the technical domain. Figure 1(a) shows a troubleshooting article and its important domain specific key terms (highlighted using green boxes). The article also shows (marked in red boxes) generic English terms which have minimal relevance in the context of the domain. If the article needs to be returned by a search application then the important key terms in green boxes should be extracted, and not the generic terms. Table 1 shows a sample of key terms from the page in Fig. 1(a), their domain-importance, and whether they can be extracted using standard noun phrase extractions, Yago, or through DBPedia concepts. These repositories contain a large amount of generic words and concepts and the idea is to validate if they can capture domain specific key terms indigenously. Table 1 shows that these approaches miss important terms like `drive fault type 3` while reporting non-useful terms like `user response` and `following steps`, indicating that these approaches are noisy and not useful for building technical support specific domain applications.

Prior art has addressed the problem of automated extraction of domain-specific key terms in different settings of domain knowledge. Methods are supervised - using large set of manually created key term annotations ([19], weakly-supervised - using only a few seed key terms [22], and unsupervised - using no annotation at all [9,11]. Weakly supervised and unsupervised methods use various statistical filters to reduce the large share of false positives. However, these methods do not address specific features of technical support content. Technical

[1] https://www.ibm.com/it-infrastructure/power.
[2] https://www.ibm.com/it-infrastructure/storage/storwize.

Table 1. Key terms extracted from relevant methods

Sample terms from page	Domain imp.	Noun phrase	Found in Yago	DBPedia concept
Drive fault type 3	Y	Y	N	N
Enclosure	Y	Y	Y	N
Canister	Y	Y	Y	Y
User response	N	Y	Y	Y
Following steps	N	Y	Y	Y

documents are loaded with acronyms, error codes, their explanations, executable commands and their outputs. Key terms often also tend to be subjective. The dense information in the pages, different vocabularies used, subjectivity across documents and less proportion of English dictionary words necessitate research effort to address them.

These tangible points for improved techniques to extract technical key terms have motivated us to design and develop a system for domain specific key term mining. Our intuition is that from a generic collection of key terms, domain specificity implemented as a set of noise removal methods can help us arrive at a good set of relevant key terms. We propose a novel weakly-supervised technique that uses various linguistic and non-linguistic features to output a relative ranking of key terms for any technical document. The algorithm begins with a generic extraction of candidate terms and refines them with domain glossaries. Domain annotations based on linguistic features like action-orientation or symptom-description along with position and frequency of the terms within the documents are used as indicators of document theme. We show that the proposed mining improves key term mining by as much as 30%.

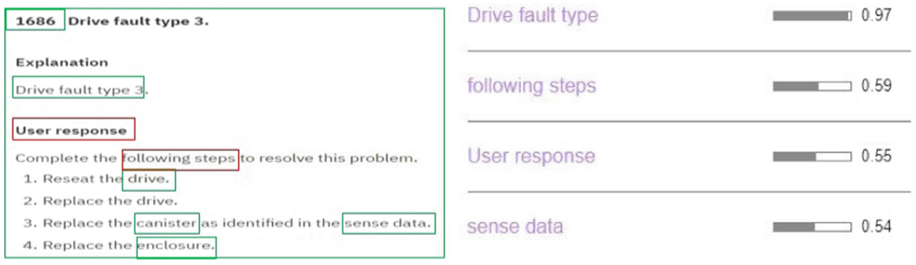
2 Motivating Example

A big motivator for mining accurate key terms is related to automated knowledge graph creation. A Knowledge Graph (KG) consist of entities and relations connecting these entities. KG based approaches have been used extensively for question answering in the open domain. For example, with entities and relations in a KG, system can do a better job on query disambiguation and understanding a user's intent, which leads to more accurate answers. KG based organization of content has also been used for efficient information retrieval and conversation applications in technical support.

2.1 Knowledge Graph Creation

The construction of Knowledge Graph relies heavily on extraction and linking of correct entities and relations. Identifying domain-specific key terms therefore becomes a crucial first step toward this endeavor. For example, as explained in [4], before extracting relations to populate a knowledge base from a large corpus of text documents, the system needs to run Entity Detection and Linking to identify entity mentions, which are essentially terms that people use to refer to those entities.

In Sect. 2.2, we will show an example to illustrate the domain specific terms important for IT Support, which are crucial for forming entities to construct a knowledge graph for this domain.



1686 **Drive fault type 3.**

Explanation

Drive fault type 3.

User response

Complete the following steps to resolve this problem.

1. Reseat the drive.
2. Replace the drive.
3. Replace the canister as identified in the sense data.
4. Replace the enclosure.

(a) Text Snippet

(b) Key Terms Extracted Using Watson NLU

Fig. 1. Error code 1686 & its resolution for IBM Storwize 5500 product

2.2 Example on Term Extraction

A typical technical-support troubleshooting article describes issues that a user may encounter (called *symptoms*) along with the suggested fixes thereof (called *resolution procedures*). Figure 1(a) illustrates a troubleshooting document for a particular *error code* (`1686`) related to the IBM Storwize product[3]. This article mentions the error code `1686` and the accompanying error message `Drive fault type 3` in its title. Following the title is a detailed description of the symptom (in this case it is vacuous repetition of the error message). The `User Response` section details the *resolution procedure*, which are the steps a user can take to address the *symptom*.

There are several domain-specific key terms that we need to automatically identify in this text for generation of nodes and relations in a knowledge graph. First, the error code, `1686` is an important key term. Second, the entire error message (`Drive fault type 3`) is a key term that appears in text analysis as a noun term (i.e., sequence of common nouns and cardinal numbers). For a different error message *symptom*, such as, `The enclosure identity cannot be read`, the subject noun term `enclosure identity` is a relevant keyterm.

Other terms of interest are mentioned in the resolution procedure. These terms are typically objects of the actions described in the resolution. For example, in the example above, `Reseat the drive`, the object `drive`, and the action

[3] https://www.ibm.com/support/knowledgecenter/en/STHGUJ_7.7.0/com.ibm.storwize.tb5.770.doc/code_1686.html.

`reseat`, are important keyterms in the technical support domain. Other objects like `canister` and `enclosure` are also domain-specific key terms. Similarly, the term `sense data` is an important term in the storage domain as well. The example article also contains terms, such as `Explanation` and `User Response`, which are not relevant for the technical support domain directly.

We submit that it is necessary to leverage available domain-specific knowledge and linguistic analysis to refine relevant terms identified by *domain agnostic* state-of-art language analysis services, such as Watson Natural Language Understanding (NLU)[4]. Figure 1(b) contains keywords extracted using Watson NLU service with confidence scores above our set threshold of 0.5. While some of the identified terms, like `Drive fault type` or `sense data` are extremely relevant for the technical support domain, other terms, such as `following steps` and `User response`, are not relevant. Furthermore, some of the relevant terms cannot be identified (e.g., error codes), or have low confidence scores.

3 Domain Knowledge Driven Term Extraction

There are two possible approaches to extracting domain-specific key terms from text: (1) domain knowledge in the form of glossaries is used as a seed for generating the candidates followed by an item set expansion approach using word embeddings for creating the final set of domain key terms [1], and (2) candidates are created by generic methods (noun phrases and standard tools like NLU and then domain glossaries along with word embeddings are used to filter out noise to create the final set of domain key terms. To evaluate the relative merit of the two approaches, we have annotated relevant key terms for twenty documents related to a hardware product and obtained the precision/recall/F1 scores in both cases. The noise removal methods had almost twice the precision, slightly less recall, and overall a 60% better F1 score over the item set expansion method. Hence, for the rest of the paper, we have started with generic terms and refined them using domain knowledge, in order to mine the most relevant key terms.

Our method, as illustrated in Fig. 2, comprises the following steps:

1. Candidate key terms are extracted using generic techniques
2. Noise filtering techniques are used to discard key terms not relevant to the domain of technical support service. Our proposal includes the following noise filter criteria:
   - Document-level relevance metric;
   - IT product domain-specific glossaries and word embeddings;
   - IT-support-specific annotations, referring to concepts such as symptom, problem, and resolution.

Filters are applied sequentially, and key terms ranked beyond a threshold are discarded. In the remainder of this section, we discuss each of these steps.

[4] https://natural-language-understanding-demo.ng.bluemix.net.

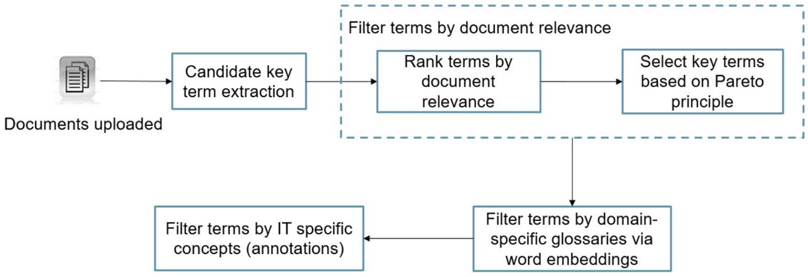

Documents uploaded
Candidate key term extraction
Filter terms by document relevance
Rank terms by document relevance
Select key terms based on Pareto principle
Filter terms by domain-specific glossaries via word embeddings
Filter terms by IT specific concepts (annotations)

Fig. 2. Major components for key term extraction

3.1 Select Candidate Generic Key Terms

The set of generic candidate key terms comprises of noun phrases and adjective phrases [5,8,13] that are identified using part-of-speech analysis and pattern matching. The pattern is defined by the regular expression `(adjective)*(noun)+`. Chunking, lemmatization, and POS tagging are done using the Spacy toolkit [5]. Candidate contiguous words are concatenated together to form candidate phrases. In addition, document analysis with generic key term extraction tools, such as Watson NLU [21], are included to generate candidate key terms.

3.2 Filter by Document Relevance

Relevant key terms in a document demonstrate a strong correlation with where and how many times they appear in a document. Building on the work proposed by [3], we use an unsupervised method for key term extraction and relevance scoring by incorporating words' sentence position information, and their frequency in the document.

Given a document, sentence tokenization and part-of-speech (POS) filter (using Spacy) are applied to generate key term features. The relevance score for ranking the terms is based on the insight that a key term is highly likely to be important if it occurs frequently and more recently in the current document. Recency of a word is related to its sentential positioning in the document. Each candidate key term is weighted with its inverse sentence position in the document. The position score is multiplied with its frequency score to get the final score of a term. For example, if a term is found on the following positions: 2^{nd}, 5^{th} and 10^{th}, its score is $\frac{1}{2} \times 1 + \frac{1}{5} \times 2 + \frac{1}{10} \times 3 = 1.2$. The score of each term is divided by the total number of sentences to get a normalized score.

Mathematically, for a document p, let $\phi(c)$ denote the set of candidate terms, and $freq(c, i)$ be the frequency of c in the i^{th} position. The score $S(.)$ of a key term c in p is

[5] https://spacy.io/.

$$S(c,p) = \frac{\sum_i \frac{1}{pos_i} \times freq(c,i)}{n_{sents}}$$

pos_i is the ith sentence where c appears, n_{sents} is the number of sentences in the document.

The well-known Pareto principle [10] is used to select the set of key terms for domain-specific documents. The procedure for Pareto analysis requires

1. Sorting the key terms in decreasing order of scores,
2. Key term-wise cumulating the scores,
3. Normalizing the cumulative scores so that the maximum cumulative normalized score equals 1.

Key terms are filtered out if they account for more than 98% of cumulative normalized score. Key terms are considered important and relevant if they account for less than 40% of cumulative normalized score. For the remaining key terms, domain-specific knowledge are used to further remove noisy key terms.

3.3 Filter Based on Domain Glossaries

Presence of key terms in domain glossaries and dictionaries is a good indicator of whether the word is relevant for the domain or not. Glossaries contain important words in the domain and their meanings. If available, they can serve as good noise removal filters. However, removing all extractions which do not match glossary terms *exactly* leads to sparsity. To overcome this limitation, we propose using embeddings to retain words that are "close to" glossary words. Word vector embeddings are used to measure if an extraction is "close" to any glossary term To create word vectors for potentially multi-word glossary terms, we used the `word2term` script from [17], running it twice over a large corpus of unlabeled documents. `word2vec` is applied after annotating common terms of upto three words as a single token. However the algorithm doesn't create vectors for all bigrams or trigrams, but only ones that frequently co-occur. Also, some extracted key terms or glossary terms could be longer than three words. The algorithm described in Algorithm 1 is used to create vectors for multi-word terms by averaging over all possible segmentations of the term, and adding vectors of segments inside a segmentation.

To compare the distance of an extracted key-term to a glossary term, both L2 distance (the euclidean or straight-line distance between two points or vectors) and cosine distance between their vectors are considered with different values of distance thresholds.

3.4 Filter Based on Domain Knowledge Annotations

Domains are often expressed by an ontology of important "aspects". Technical support involves various well understood attributes like a detailed description of the problem (symptom), error codes, and steps (procedures) that should be taken to fix the problem. Consider the snapshot of a DB2 support document in Fig. 3. Here, the document mentions the symptoms (marked within red boxes), the error code SQL3004N, and procedures (marked within the blue box). Annotations representing these attributes can be good indicators of key term importance. For example, when a key term occurs in the problem description it is highly likely that its domain relevance is high. key terms are ranked based on the document annotations and those below a rank threshold are discarded.

A Semantic Parser [7] is used to extract these attributes from technical support documents using rules built on the grammatical structure of sentences.

ALGORITHM 1. Algorithm to create vectors for multi-word terms

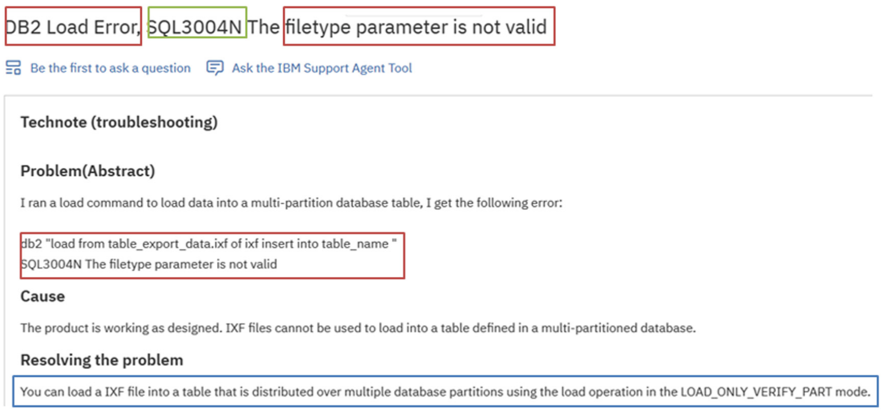
```
Input: Multi-word term ($term$), Dictionary of term vectors from word2vec
        ($term\_vectors$)
Output: A vector for the input $term$, ($term\_vector$)
if $term$ in $term\_vectors$
then
    return $term\_vectors[term]$
else
    // Tokenize the input term into words
    $term\_tokens = tokenize(term)$

    // Get all possible segmentations of these tokens
    $segmentations = get\_all\_segmentations(term\_tokens)$

    $term\_vector = []$
    foreach $segmentation$ in $segmentations$
    do
        $segmentation\_vector = []$
        $segments\_found = 0$
        foreach $segment$ in $segmentation$
        do
            // Check if segment is present in $term\_vectors$
            if $segment$ in $term\_vectors$
            then
                $segmentation\_vector$ += $term\_vectors[segment]$
                $segments\_found$ += 1
            end
        end
        // Weigh the segmentation vector by the number of segments found
        $segmentation\_vector$ *= $segments\_found/len(segmentation)$

        $term\_vector$ += $segmentation\_vector$
    end
    // Average the overall vector over all the segmentation vectors
    $term\_vector$ /= $len(segmentations)$

    return $term\_vector$
end
```


DB2 Load Error, SQL3004N The filetype parameter is not valid

Be the first to ask a question
Ask the IBM Support Agent Tool

Technote (troubleshooting)

Problem(Abstract)

I ran a load command to load data into a multi-partition database table, I get the following error:

```
db2 "load from table_export_data.ixf of ixf insert into table_name "
SQL3004N The filetype parameter is not valid
```

Cause

The product is working as designed. IXF files cannot be used to load into a table defined in a multi-partitioned database.

Resolving the problem

You can load a IXF file into a table that is distributed over multiple database partitions using the load operation in the LOAD_ONLY_VERIFY_PART mode.

Fig. 3. Illustration of Annotation of attributes on a Support document

Two deep parsing components are used: English Slot Grammar (ESG) followed by Predicate Argument Structure (PAS) for linguistic analysis of text [16].

The method for re-ranking key terms based on domain annotations is presented below, where $\mathbb{1}$ is an indicator function, $\bar{S}$ denotes the average of scores (S) for all key terms in a document. DA is the set of all terms in domain annotations for the document, and G is the set of all glossary terms. The adjusted score, $\hat{S}$, is increased if the key term is present in domain annotations or glossary, by the amount $\bar{S}$. $\hat{S}$ is then used for re-ranking key terms.

$$\hat{S}(c,p) = \mathbb{1}[\![c \in DA]\!]\bar{S} + \mathbb{1}[\![c \in G]\!]\bar{S} + S(c,p)$$

4 Evaluation

The experiments in this section demonstrate that using document relevance and domain knowledge helps noise removal effectively. Two domain datasets are considered, which are very different with respect to type/structure of IT support content, and to the technology and terms. The first dataset comes from a hardware domain of hybrid storage solutions, referred to as Storwize. The second dataset is a popular middleware product, referred to as DB2. Ground truth knowledge of important key terms for each of the two domains is collected from human annotators, across 50 documents. The metrics F1 and MRR (Mean Reciprocal Rank) are used to evaluate the efficacy of the noise removal techniques on the above datasets. In addition, a dedicated evaluation framework is used to evaluate the extraction of key terms in a larger set of documents without ground truth.

4.1 Data

The documents collected for DB2 and Storwize are from IBM's Knowledge Center sites[6]. We use a set of 172 DB2 documents and 50 Storwize documents, which are identified as answers of certain questions from end users in IT services. Ground truth knowledge of important key terms for each of the two domains is collected from human annotators, across 50 documents. Glossaries for Storwize and DB2, with 5K and 2K items respectively, are collected from the same site. Two question-answer datasets are created, comprising of those questions from end users along with accepted answer documents. A subset of these question-answer pairs were annotated with important key terms by domain experts. Table 2 summarizes both the datasets, where *DB2WithGT* and *StorwizeWithGT* datasets contain ground truth-labeled key terms, *#_Docs* is the number of documents in each dataset, *#_Keyterm* is the number of ground truth key terms or encompassing terms in each dataset, and *Avg_Keyterm* is the average number of ground truth key terms per document.

For running our evaluations, we used the ground truth key terms where available, and used key terms from problem titles otherwise.

Word embeddings are built only for the Storwize domain using 50,000 documents. Details are described in Sect. 4.3.

Table 2. Datasets used in our experiments

Dataset	#Docs	#Keyterm	Avg_Keyterm
DB2WithGT	22	421	16.84
StorwizeWithGT	25	614	24.56
DB2 (All)	172	722	4.19
Storwize (All)	50	127	2.54

4.2 Evaluation Metrics

The following evaluation metrics are used to compare performance of the different experimental setups [14]:

1. Precision (P), Recall (R), and F1 measure (F1) are computed for each experimental setup. Precision is the percentage of correctly extracted key terms by the total extracted key terms, Recall is the percentage of correctly extracted key terms by the total ground truth key terms, and F1 is the harmonic mean of precision and recall.

[6] https://www.ibm.com/support/knowledgecenter/.

2. Mean Reciprocal Rank (MRR) [12] between the ground truth key terms and the final relevance ranking of the key term.

$$MRR = \frac{1}{|D|} \sum_{d \in D} \sum_{i} \frac{1}{rank_i}$$

where, D is the set of documents, and $rank_i$ is the rank of the key term in document d.

4.3 Results and Discussions

The focus of the evaluation presented in this section is to compare the various noise removal methods.

First, we address the efficacy of the individual stages in the domain specific noise removal pipeline, i.e. generic key terms extraction, key term extraction with document relevance, glossaries and word embeddings. Generic candidate key terms are identified using tools like Spacy and Watson NLU as discussed in Sect. 3. Spacy's lemmatizer is used to lemmatize the extracted and assigned key terms. Precision, recall and F1 are computed based on the ground truth key terms for all the setups.

Table 3 reports the comparative analysis. Here, domain knowledge refers to using both glossaries and domain knowledge annotations. The results show that adding domain knowledge increases precision thus helping in noise removal. However, negative filtering using domain knowledge is likely to reduce the importance of a term to the extent that a key term is removed from the "gold" set. This could lead to a reduction in the recall measure. In terms of the F1 score, use of document relevance with domain knowledge outperforms the other approaches by 28% for the two datasets. In Storwize, additionally using word embeddings improved the F1 score over baseline by 30%.

The results for embeddings in Table 3 for Storwize are reported for the best `word2vec` [17] model. We next show the results of experiments with the two variations of the `word2vec` model: CBOW and Skipgram along with some relevant window sizes. Having trained a word2vec model on 50,000 Storwize documents, Table 4 shows the performance of the eight best embedding approaches on ground truth key terms. These are the models with different methods for training word vectors and different distance metrics used for pruning. Here, CBOW-NEG⟨X⟩-L2 are the word vectors trained with CBOW model with Negative Sampling and window size of X, with L2 distance used for pruning. Similarly, SKIPGRAM-NEG⟨X⟩-COSINE are the word vectors trained with Skipgram model and negative sampling and window size of X, with Cosine distance used for pruning. We show here only the embedding results for Storwize and not for DB2 because the low number of documents in the latter domain are insufficient to learn embeddings.

Next we evaluate the ranking scheme discussed in Sect. 3.4 where key terms are re-ranked based on domain knowledge. In Table 5, *KT* refers to generic key terms, and *DK* refers to domain knowledge. Precision, recall and F1 scores are

Table 3. Performance comparison: with ground truth

Dataset	Method	Precision	Recall	F1
DB2	Generic key terms	0.242	0.879	0.38
	Generic key terms + domain knowledge	0.278	0.846	0.419
	Generic key terms + document relevance + domain knowledge	0.356	0.779	0.489
Storwize	Generic key terms	0.284	0.844	0.425
	Generic key terms + domain knowledge	0.33	0.793	0.466
	Generic key terms + document relevance + domain knowledge	0.398	0.728	0.515
	Generic key terms + document relevance + domain knowledge embeddings	0.534	0.585	0.559

Table 4. Performance comparison (storwize): with ground truth

Model	Generic key terms			Generic key terms + document relevance		
	Precision	Recall	F1	Precision	Recall	F1
CBOW-NEG3-L2	0.627	0.191	0.293	0.752	0.299	0.428
CBOW-NEG3-Cosine	0.554	0.553	0.553	0.534	0.585	0.559
CBOW-NEG5-L2	0.63	0.192	0.295	0.752	0.299	0.428
CBOW-NEG5-Cosine	0.457	0.659	0.539	0.445	0.676	0.537
SkipGram-NEG3-L2	0.623	0.2	0.303	0.744	0.302	0.43
SkipGram-NEG3-Cosine	0.609	0.455	0.52	0.596	0.519	0.555
SkipGram-NEG5-L2	0.623	0.192	0.294	0.747	0.302	0.429
SkipGram-NEG5-Cosine	0.402	0.689	0.507	0.399	0.702	0.509

shown in comparison for the top@N (N = 2,4,6,8) on the two datasets. The use of domain knowledge increases F1 scores as N gets larger - beyond N larger than 2, there is a 3–5% increase in F1. The table also reports the MRR scores for the different combinations on these datasets with similar gains.

Table 5. Ranking performance comparison: precision, recall, F1

Dataset	Method	Top2				Top4				Top6			
		P	R	F1	MRR	P	R	F1	MRR	P	R	F1	MRR
DB2	KT	0.82	0.13	0.23	0.097	0.705	0.234	0.35	0.128	0.598	0.288	0.389	0.138
	KT + DK	0.818	0.133	0.228	0.0967	0.705	0.239	0.357	0.13	0.629	0.3	0.405	0.14
Storwize	KT	0.76	0.2	0.317	0.147	0.63	0.285	0.392	0.17	0.547	0.336	0.416	0.122
	KT + DK	0.8	0.204	0.326	0.149	0.69	0.303	0.42	0.176	0.6	0.37	0.459	0.189

Because of the lack of ground truth for a majority of domain-specific technical support documents, we designed a new evaluation framework to evaluate the ranking consisting of these steps: (i) First, we shortlisted a set of questions from end users for the two product with their corresponding answer document(s). We shortlisted 350 questions for Storwize and 400 questions for DB2. (ii) Subject matter experts were provided with these questions and were asked to annotate possible important key terms in them. Five experts in total participated in this task. (iii) We assumed these key terms extracted from the questions to be the "pseudo" ground truth. (iv) We calculated the precision, recall and F1 measure for each experimental setup.

Table 6 presents the comparative analysis of the different setups. We present the result of the best threshold value of domain knowledge embeddings to remove noisy extractions. As shown in the table, the precision is relatively low. This is because, as shown in Table 2, the number of key terms annotated from questions is very small, since in most cases the questions are very short in length. Compared to questions, the documents are much longer and the average number of key terms extracted from documents is much bigger.

Table 6. Performance comparison: with pseudo ground truth

Dataset	Method	Precision	Recall	F1
DB2	Generic key terms	0.06	0.727	0.113
	Generic key terms + domain knowledge	0.073	0.718	0.133
	Generic key terms + document relevance + domain knowledge	0.094	0.687	0.166
Storwize	Generic key terms	0.0384	0.696	0.0729
	Generic key terms + domain knowledge	0.048	0.696	0.09
	Generic key terms + document relevance + domain knowledge	0.06	0.685	0.112
	Generic key terms + document relevance + domain knowledge embeddings	0.196	0.286	0.233

Based on the results shown above, we can clearly see the benefit of using domain knowledge to extract and rank relevant key terms for domain-specific documents, thus helping us to remove noisy key terms.

5 Related Work

Domain-specific terms are essential to many knowledge management applications, and the limitations of manual identification have motivated many research efforts.

Supervised methods, suggested by [19], start from a significantly large volume of content labeled with domain-specific terms, and provide classification models that decide if a given term is relevant for the domain. The approach provides the best performance, but, the overhead of generating labeled content has redirected research to focus on unsupervised and weakly supervised methods.

One group of related works is interested to discover domain specific terms given a domain corpus. These words address the problem, similar to the one that we have addressed in the paper. Wang et al. [22] described an approach for using deep learning model together with a weakly supervised bootstrapping paradigm to automatically extract domain specific terms. It is an approach to boost the performance of deep learning models with very few training examples. This approach does not leverage any prior knowledge of a domain explicitly, as we do. Riloff et al. [18] proposed a mutual bootstrapping method to both the semantic lexicon and extraction patterns simultaneously, starting from unannotated training texts and a handful of "seed words". However, this method does not address the issue of extracting domain-specific terms.

In a different context, domain-specific key terms are identified in order to distinguish across domains [9,11] and then used for feature extraction in broader text analysis tasks. More specifically, given a corpus that spans multiple domains, domain-specific statistics, such as term frequency and inverse document or domain frequency [9], and entropy impurity [11] are used to determine the set of key terms most representative for a domain. In [23] domain-specific terms are extracted using an iterative bootstrapping method to learn term components from seed terms and then a maximum forward matching method and domain frequency is used to extract additional components. Our solution uses similar term frequency methods but applies them along with other domain-specific linguistic features in order to reduce the amount of false positives.

Frakes et al. [6] have emphasized the need for accurate automatic domain vocabulary selection. The objective of their work is to evaluate various automatic vocabulary extraction metrics (normalized and non-normalized term frequency metrics), domain analysis against the domain vocabulary provided by subject matter experts. Their analysis of various metrics confirms that term frequency is one of most important factors along with stop word list (removal of common English words) and stemming (grouping the related terms in a common term by removing suffixes and prefixes). Our solution starts with curated list of terms as input, hence need for stop word removal is not necessary. In addition to term frequency, our algorithm takes the position of the term into consideration to boost the term score.

6 Conclusion and Future Work

In this paper, we present a weakly-supervised approach to extract key terms from IT services documents, leveraging prior knowledge in the domain. Experimental results show that using domain knowledge effectively improves the quality of terms extracted from generic tools. Hence, having domain knowledge (in terms of glossaries and domain corpus) will definitely help in further extracting domain-specific key terms. In the future, we will further refine the key term extraction process by leveraging the cross document features. In addition, we will discover semantic relationships between the extracted terms (for the purpose of knowledge graph construction).

References

1. Contractor, D., Singla, P., Mausam: Entity-balanced Gaussian pLSA for automated comparison. In: HLT-NAACL (2016)
2. Deng, Y., et al.: Advanced search system for IT support services. IBM J. Res. Dev. **61**(1), 3:27–3:40 (2017). https://doi.org/10.1147/JRD.2016.2628658

3. Florescu, C., Caragea, C.: PositionRank: an unsupervised approach to keyphrase extraction from scholarly documents. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1105–1115. Association for Computational Linguistics (2017). https://doi.org/10.18653/v1/P17-1102
4. Glass, M., Gliozzo, A., Hassanzadeh, O., Mihindukulasooriya, N., Rossiello, G.: Inducing implicit relations from text using distantly supervised deep nets. In: Proceedings of the 17th International Semantic Web Conference (ISWC 2018) (2018)
5. Gollapalli, S.D., Caragea, C.: Extracting keyphrases from research papers using citation networks. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pp. 1629–1635 (2014)
6. Gupta, A., Akula, A., Dasgupta, G., Aggarwal, P., Mohapatra, P.: Desire: deep semantic understanding and retrieval for technical support services. In: Drira, K., et al. (eds.) ICSOC 2016. LNCS, vol. 10380, pp. 207–210. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68136-8_26
7. Gupta, A., Akula, A., Dasgupta, G., Aggarwal, P., Mohapatra, P.: Desire: deep semantic understanding and retrieval for technical support services. In: Drira, K. (ed.) ICSOC 2016. LNCS, vol. 10380, pp. 207–210. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68136-8_26
8. Hulth, A.: Improved automatic keyword extraction given more linguistic knowledge. In: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, EMNLP 2003, pp. 216–223, Stroudsburg, PA, USA (2003)
9. Kim, S.N., Baldwin, T., Kan, M.Y.: An unsupervised approach to domain-specific term extraction. In: Proceedings of the Australasian Language Technology Association Workshop 2009, pp. 94–98 (2009)
10. Koch, R.: The 80/20 Principle: The Secret to Achieving More with Less (1999)
11. Liu, T., long Wang, X., Yi, G., Xu, Z.M., Wang, Q.: Domain-specific term extraction and its application in text classification. In: Proceedings of 8th Joint Conference on Information Sciences, pp. 1481–1484 (2005)
12. Liu, Z., Huang, W., Zheng, Y., Sun, M.: Automatic keyphrase extraction via topic decomposition. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. pp. 366–376. EMNLP '10, Association for Computational Linguistics, Stroudsburg, PA, USA (2010). http://dl.acm.org/citation.cfm?id=1870658.1870694
13. Liu, Z., Li, P., Zheng, Y., Sun, M.: Clustering to find exemplar terms for keyphrase extraction. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1, EMNLP 2009, pp. 257–266, Stroudsburg, PA, USA (2009)
14. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York (2008)
15. McCallum, A., Li, W.: Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, pp. 188–191. Association for Computational Linguistics (2003)
16. McCord, M.C.: Slot grammar. In: Studer, R. (ed.) Natural Language and Logic. LNCS, vol. 459, pp. 118–145. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-53082-7_20
17. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26, pp. 3111–3119. Curran Associates, Inc. (2013)

18. Riloff, E., Jones, R.: Learning dictionaries for information extraction by multi-level bootstrapping. In: AAAI 1999. AAAI (1999)
19. da Silva Conrado, M., Salgueiro Pardo, T.A., Rezende, S.O.: A machine learning approach to automatic term extraction using a rich feature set. In: Proceedings of HLT-NAACL 2013, pp. 16–23 (2013). http://www.aclweb.org/anthology/N13-2003
20. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003-Volume 4, pp. 142–147. Association for Computational Linguistics (2003)
21. Vergara, S., El-Khouly, M., Tantawi, M.E., Marla, S., Sri, L.: Building Cognitive Applications with IBM Watson Services: Volume 7 Natural Language Understanding (2017)
22. Wang, R., Liu, W., McDonald, C.: Featureless domain-specific term extraction with minimal labelled data. In: Proceedings of the Australasian Language Technology Association Workshop 2016, pp. 103–112 (2016)
23. Zhang, C., Niu, Z., Jiang, P., Fu, H.: Domain-specific term extraction from free texts. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 1290–1293. IEEE (2012)

An Adaptive Semi-local Algorithm for Node Ranking in Large Complex Networks

Fanghua Ye[1,2], Chuan Chen[1,2(✉)], Jie Zhang[1,2], Jiajing Wu[1,2(✉)], and Zibin Zheng[1,2]

[1] School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
{yefh5,zhangj379}@mail2.sysu.edu.cn

[2] National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou, China
{chenchuan,wujiajing,zhzibin}@mail.sysu.edu.cn

Abstract. The issue of node ranking in complex networks is a classical problem that has obtained much attention over the past few decades, and a great variety of methods have consequently been developed. These proposed methods can be roughly categorized into the *global* and *local* methods. The global methods are usually time-consuming and the local methods may be inaccurate. In this paper, we propose a novel *semi-local* algorithm ASLA (Adaptive Semi-Local Algorithm) that seeks a tradeoff between the time efficiency and the ranking accuracy to overcome the limitations of the global and local methods. ASLA is able to adaptively determine the potential influence scope for each node. Then, the influence value of each node is calculated based on such a personalized influence scope. Finally, all the nodes are ranked according to their influence values. To evaluate the performance of ASLA, we have conducted extensive experiments on both synthetic networks and real-world networks, with the results demonstrating that ASLA is not only more efficient than the global methods but also more accurate than the local methods.

Keywords: Node ranking · Adaptive algorithm · Complex networks

1 Introduction

The complex network is a widely adopted scheme to represent many real-world complex systems, including social networks, information networks, service networks [8,20], and so on. Identifying influential nodes in these complex networks is of great significance to many practical applications, such as friend recommendation, viral marketing, and software system analysis [3,19]. Besides, identifying influential nodes plays an important role in understanding the structures and functions of complex networks. As a consequence, identifying influential nodes has attracted much attention in recent years [4,18].

The problem of identifying influential nodes is typically formulated as node ranking. Due to the great theoretical and practical significance of identifying influential nodes, various methods have been proposed to rank nodes in complex networks. These node ranking methods can be roughly categorized into the local methods and the global methods. Typical existing node ranking methods include but not limited to degree centrality [7], local centrality [2], coreness centrality [10], cluster centrality [1], closeness centrality [16], betweenness centrality [6], and Katz centrality [9]. Among them, degree centrality is the simplest yet widely used metric, but it fails to identify influential nodes in some cases, because it is based on a node's nearest neighbors that contain very limited information. Local centrality is an improved version of degree centrality by taking the fourth-order neighbors of each node into consideration. It has been pointed out that the location of a node is more significant than the number of its immediate neighbors in evaluating its spreading influence [10]. Therefore, coreness centrality has been proposed to determine the importance of nodes in a network more accurately [10]. Cluster centrality is developed based on the observation that the local clustering usually plays a negative role in the spreading process [14]. Unlike degree centrality, cluster centrality considers not only the number of the nearest neighbors but also the interactions among them. All these methods are neighborhood-based methods (i.e., local methods). That is, they are very time-efficient but lack performance guarantee. On the contrary, the closeness centrality, betweenness centrality, Katz centrality, etc. are path-based metrics (i.e., global methods). They take the global information of a network into consideration, thus can give better ranking results. However, these global methods are very time-consuming and consequently cannot be applied to large complex networks. Detailed descriptions of these methods can be found in [12].

In order to overcome the limitations of the existing node ranking methods, we propose a novel adaptive semi-local algorithm ASLA, which aims to seek a tradeoff between the time efficiency and the ranking accuracy. Specifically, to quantify the importance of nodes, ASLA first adaptively determines the potential influence scope for each node, and then calculates the influence value of each node based on such a personalized influence scope. Finally, the ranking list of all nodes is given by their influence values in a descending order. Extensive experimental studies on both synthetic and real-world networks have shown that ASLA can achieve excellent performance in terms of both effectiveness and efficiency.

2 Basic Concepts

Let $G = (V, E)$ be an undirected and unweighted simple network with $N = |V|$ nodes and $M = |E|$ edges, where V and E denote the node set and the edge set respectively. In the following, we provide some basic concepts that form the foundation of the proposed ASLA algorithm.

Definition 1 (*D*-Shell). *The d-shell of node u, denoted by S_u^d, consists of all the nodes v at depth d of node u, that is, the length of the shortest path between* *nodes u and v is d. The 0-shell of node u is defined as the node itself. The 1-shell of node u is exactly its immediate neighbors.*

Consider the toy network shown in Fig. 1. The 1-shell of node v_2 consists of all its immediate neighbors, i.e., nodes v_1, v_3, v_4, v_5, v_6, v_7, v_8, v_9. While the 2-shell of node v_2 only contains node v_{10}.

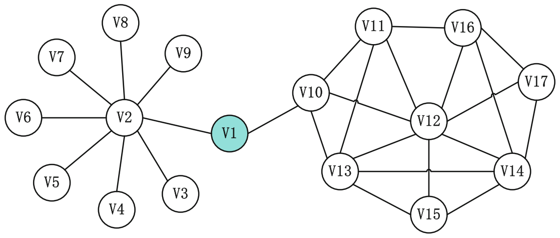


Fig. 1. A toy network as examples. Node v_1 plays the role of a bridge, thus, its influence value should be high, although its degree is low.

Definition 2 (Outer Degree). *The outer degree of node v in the d-shell of node u, denoted by $k_u^d(v)$, is defined as the number of edges that connect node v to nodes in the $(d+1)$-shell of node u. The total outer degree of the d-shell of node u, denoted by K_u^d, is defined as the sum of the outer degrees of all the nodes v in the d-shell of node u. Thus, $K_u^0 = k_u, K_u^d = \sum_{v \in S_u^d} k_u^d(v)$, where k_u denotes the degree of node u, i.e., the number of edges incident to node u.*

It is noted that the total outer degree of the d-shell is not necessarily the number of nodes in the $(d+1)$-shell. In fact, the total outer degree of the d-shell is the total number of edges connecting nodes in the d-shell to nodes in the $(d+1)$-shell. We define the total outer degree of the d-shell as the number of edges outward from this shell to the next shell instead of just merely the number of nodes in the next shell, the advantage of which is that this definition takes more structural information into account. Consider the network shown in Fig. 1 again. Assume that node v_{10} is the starting node, the total outer degree of the 1-shell is 8, obviously it is larger than the number of nodes (i.e., 5) in the 2-shell.

Definition 3 (Outer Density). *The outer density of the d-shell starting from node u, denoted by D_u^d, is defined as the total outer degree of this shell divided by the total number of nodes in this shell. That is, $D_u^d = K_u^d/|S_u^d|$, where $|S_u^d|$ denotes the total number of nodes in the d-shell of node u.*

Definition 4 (Influence Scope). *The influence scope of node u, denoted by $C(u)$, is composed of its nearest neighbors, the next nearest neighbors, and so on. $C(u)$ is formulated as $C(u) = \bigcup_{0 \leq d \leq \bar{d}} S_u^d$, where $\bar{d}$ satisfies the condition that the outer density of the d-shell is no larger than some threshold α, i.e., $D_u^{\bar{d}} \leq \alpha$. When more than one d-shell satisfies the condition, $\bar{d}$ is taken as the smallest d.*

Definition 5 (Influence Value). *The influence value of node u is defined as*

$$F(u, \alpha) = \sum_{0 \le d \le \bar{d}} f(d)(K_u^d + |S_u^d|), \tag{1}$$

where $f(d)$ is a nonincreasing function and the nonnegative parameter α is used to control the scale of node u's potential influence scope $C(u)$.

Consider two extreme cases: $\alpha = 0$ and $\alpha \ge k_{max}$, k_{max} is the largest degree of nodes in network G. When $\alpha = 0$, the d-shell will spread to much of the network, thus the influence scope tends to cover all the nodes in the entire network. When $\alpha \ge k_{max}$, the d-shell cannot spread outward at all, thus the influence scope just contains the starting node itself. $F(u, \alpha)$ takes node u's neighbors from 1-hop up to $\bar{d}$-hop into consideration, which demonstrates that $F(u, \alpha)$ is a semi-local measure for the importance of nodes in network G.

Now let us shed more light on $F(u, \alpha)$. Suppose that $f(d) = 1$ for all d values. According to Eq. (1), $F(u, \alpha)$ is composed of two parts: $\sum_{0 \le d \le \bar{d}} K_u^d$ and $\sum_{0 \le d \le \bar{d}} |S_u^d|$. The first part is the sum of the total outer degrees of all the d-shells contained in the influence scope $C(u)$, which represents the **propagation** ability of node u. The second part is the total number of nodes covered by $C(u)$, which characterizes the **aggregation** ability of node u. Therefore, $F(u, \alpha)$ captures the propagation ability and aggregation ability simultaneously. As a result, even one node has high aggregation ability, its influence value may be very small for its poor propagation ability. For example, as shown in Fig. 1, when $\alpha = 1.0$, the influence value of node v_{12} is 16, while the influence value of node v_1 is 32. As we can see, node v_1 plays the role of a bridge in the exemplary network. Therefore, even though node v_{12} has a larger degree, it is less influential than node v_1.

Problem Statement. Given an undirected and unweighted network $G = (V, E)$ and a parameter $\alpha \ge 0$, the problem is to compute the influence value $F(u, \alpha)$ for each node $u \in V$ (Note that the final definition of $F(u, \alpha)$ is given in Sect. 3).

3 The Adaptive Semi-local Algorithm

In this part, we propose a novel adaptive semi-local algorithm ASLA to combat the problem of node ranking in complex networks. ASLA aims to seek a tradeoff between the poor-quality local methods and the time-consuming global methods. Due to the introduction of the additional parameter α to adaptively adjust the influence scope of each node flexibly, ASLA is expected to be not only more efficient than the global methods but also more accurate than the local methods.

It is clear that the function $f(d)$ should be properly set to calculate the exact influence value $F(u, \alpha)$. To be better in line with actual occasions that the closer d-shells tend to be more significant than the farther ones, we take the form of $f(d)$ as $e^{-\frac{d}{k_u}}$, i.e., $f(d) = e^{-\frac{d}{k_u}}$. In general, nodes with larger degrees tend to have stronger ability to spread outward. Thus, we introduce the factor $\frac{1}{k_u}$ into the exponent. Besides, to better capture the influence scope of each node, the parameter α should be personalized. In this work, we define the parameter α_u of node u as $\frac{k_u}{k_{max}}\alpha$, i.e., $\alpha_u = \frac{k_u}{k_{max}}\alpha$. One may note that the propagation ability and the aggregation ability defined in Eq. (1) are highly possible to fall in distinct ranges, which may be caused by the sparsity of networks. To reduce the skew of this two abilities, we normalize both abilities into the range of $[0, 1]$. Based on these discussions, the influence value (i.e., the semi-local centrality) of node u in network G is finally defined as

$$F(u, \alpha_u) = \sum_{0 \le d \le \bar{d}} e^{-\frac{d}{k_u}} (\frac{K_u^d}{M} + \frac{|S_u^d|}{N}). \tag{2}$$

Algorithm 1. The Adaptive Semi-Local Algorithm: ASLA

```
Input: Network $G = (V, E)$, parameter $\alpha$;
Output: $F(u, \alpha_u)$ for each node $u \in V$.
 1: for each node $u \in V$ do
 2:     $J[u] \leftarrow -1$; $\alpha_u \leftarrow \frac{k_u}{k_{max}}\alpha$; $F(u, \alpha_u) \leftarrow 0$;
 3: for each node $u \in V$ do
 4:     Q $\leftarrow$ empty queue; Enqueue $u \rightarrow Q$; $J[u] = u$; $d \leftarrow 0$;
 5:     while (true) do
 6:         $|S_u^d| \leftarrow$ the current number of nodes in $Q$;
 7:         $K_u^d \leftarrow$ TotalOuterDegree$(u, d)$; $D_u^d \leftarrow K_u^d/|S_u^d|$;
 8:         $F(u, \alpha_u) \leftarrow F(u, \alpha_u) + e^{-\frac{d}{k_u}}(\frac{K_u^d}{M} + \frac{|S_u^d|}{N})$;
 9:         if $D_u^d \le \alpha_u$ then
10:             break;    $\triangleright\ d = \bar{d}$
11:         while $(|S_u^d| > 0)$ do
12:             Dequeue $v \leftarrow Q$; $|S_u^d| \leftarrow |S_u^d| - 1$;
13:             for all $w \in S_v^1$ and $J[w] \neq u$ do
14:                 Enqueue $w \rightarrow Q$; $J[w] \leftarrow u$;
15:         $d \leftarrow d + 1$;
16: return $F(u, \alpha_u)$ for each node $u \in V$.
```

Our new method ASLA will calculate the influence value of each node according to Eq. (2), and the corresponding pseudocode is outlined in Algorithm 1, where TotalOuterDegree(u, d) is a procedure to calculate the total outer degree of the d-shell of node u, and $J[w] = u$ is used to indicate that node w has been contained in the influence scope of node u, i.e., $w \in C(u)$. The inputs of Algorithm 1 include network G and parameter α. Then, it calculates the personalized parameter α_u for each node $u \in V$ (line 2). To determine the importance of all the nodes, Algorithm 1 is quite straightforward. It calculates the influence value of each node separately (line 3). More specifically, to calculate the influence value of each node $u \in V$, the key points are to determine all the d-shells contained in $C(u)$ and the corresponding total outer degree in each d-shell. To this end, Algorithm 1 resorts to a queue Q to store the nodes in each d-shell (line 4). In the d-th iteration, the nodes contained in Q just form the d-shell of node u (line 6). In order to find nodes in the next shell (i.e., the $(d+1)$-shell), Algorithm 1 searches all the neighbors of each node in the d-shell, then the nodes contained in the $(d+1)$-shell are added to Q (lines 11–14). Note that in each iteration, the calculation of the total outer degree of the d-shell can be finished by simply summing over the outer degrees of all nodes in this shell (line 7), thus we omit the details of the TotalOuterDegree(u, d) procedure. The processing of node u terminates when the outer density reaches its threshold α_u (lines 9–10).

Let N_{α_u} denote the number of nodes contained in u's influence scope $C(u)$, and k_{avg} denote the average degree of all nodes. Then the processing of node u will take $\mathcal{O}(N_{\alpha_u} k_{avg})$ time, for the reason that Algorithm 1 will traverse all the immediate neighbors of each node in $C(u)$. Therefore, Algorithm 1 will take $\mathcal{O}(NN_\alpha k_{avg})$ time to calculate the influence values for all the nodes. Here, we use N_α to denote the average size of the influence scopes of all nodes. It is obvious that N_α is closely related to α. When $\alpha = 0$, we have $N_\alpha \approx N$. However, when $\alpha \geq k_{max}$, we have $N_\alpha \approx 1$. Thus, the time complexity of Algorithm 1 can be controlled flexibly with a proper α. Moreover, Algorithm 1 is naturally suitable for parallel processing. Algorithm 1 also shows great advantages when one only cares about the importance of partial nodes.

To intuitively demonstrate the effectiveness of ASLA, we list the top-10 ranked nodes of different methods on the toy network (see Fig. 1) in Table 1. As can be seen, for betweenness centrality and closeness centrality, the top-3 ranked nodes are v_1, v_2, v_{10}. When $\alpha = 2.0$, ASLA can provide the same result. However, local centrality fails to identify these nodes. For example, it gives a very low rank to the bridge node v_1. Since degree centrality gives low ranks to nodes with small degrees, it also fails to identity the bridge node v_1.

Table 1. Comparison of the top-10 ranked nodes on the toy network as shown in Fig. 1. DC: Degree Centrality; LC: Local Centrality; BC: Betweenness Centrality; CC: Closeness Centrality; ASLA-1: ASLA with $\alpha = 1.0$; ASLA-2: ASLA with $\alpha = 2.0$.

Methods	1	2	3	4	5	6	7	8	9	10
DC	v_2	v_{12}	v_{13}	v_{14}	v_{10}	v_{11}	v_{16}	v_{15}	v_{17}	v_1
LC	v_{12}	v_{13}	v_{14}	v_{11}	v_{16}	v_{10}	v_{15}	v_{17}	v_1	v_2
BC	v_2	v_1	v_{10}	v_{12}	v_{13}	v_{11}	v_{14}	v_{16}	v_3	v_4
CC	v_1	v_2	v_{10}	v_{12}	v_{13}	v_{11}	v_3	v_4	v_5	v_6
ASLA-1	v_{10}	v_{13}	v_2	v_1	v_{11}	v_{14}	v_{16}	v_{12}	v_{15}	v_{17}
ASLA-2	v_{10}	v_2	v_1	v_{11}	v_{16}	v_{12}	v_{13}	v_{15}	v_{14}	v_{17}

4 Experiments

4.1 Experimental Settings

We conduct experiments on both synthetic and real-world networks. The synthetic networks are generated by SNAP [11], a general graph mining library. The generated synthetic networks include the Erdos-Renyi network ER (1K nodes, 5K edges) and the Power-Law network PL (5K nodes, 7K edges). The real-world networks are downloaded from Network Repository [15], including Epinions (27K nodes, 100K edges) and Douban (155K nodes, 327K edges).

The proposed method ASLA is compared with five representative node ranking methods including two global methods: betweenness centrality (BC) [6] and closeness centrality (CC) [16]; and three local methods: local centrality (LC) [2], k-core decomposition based coreness centrality (KC) [10], and hybrid centrality (HC) [17]. Since KC and the influence scope of ASLA are both relevant to the community structure [5] of networks, we also choose KC as a benchmark. HC can be treated as a combination of LC and KC. It considers the coreness and the neighborhood of a node simultaneously. Different from LC, HC just takes into account the information contained in the third-order neighbors of each node.

The brand-new robustness value metric (denoted by R) [13] is adopted to assess the quality of ranking results. Smaller R indicates better results. R is defined as $R = \frac{1}{N}\sum_{I=1}^{N}\delta(I)$, where $\delta(I)$ denotes the fraction of nodes in the largest connected component after removing I nodes from the original networks.

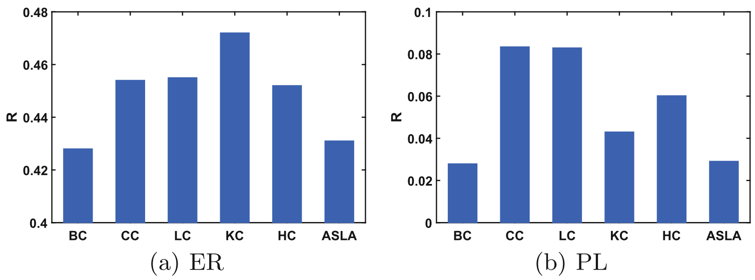


Fig. 2. Performance of different methods on synthetic networks.

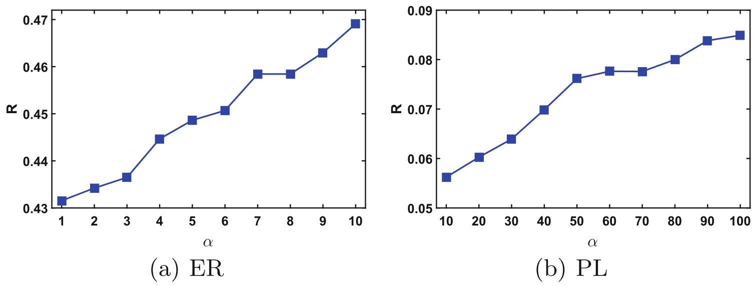


Fig. 3. Effects of parameter α on our method ASLA.

4.2 Results on Synthetic Networks

We first evaluate the performance of different ranking methods on the two synthetic networks. In this experiment, the parameter α is fixed at 1.0. Recall that Algorithm 1 only involves a single parameter α, as the personalized threshold α_u of node u is calculated as $\alpha_u = \frac{k_u}{k_{max}}\alpha$. The results are shown in Fig. 2. As can be seen, BC has the best performance. While the performance of CC, LC, KC and HC is very poor. Even though CC is a global method, its performance is not satisfactory. Our method ASLA obtains comparable performance with BC. The results verify our hypothesis that ASLA is capable of achieving better performance than the local methods.

We then evaluate the effects of parameter α on our method ASLA. For the ER network, we vary α from 1 to 10. For the PL network, we vary α from 10 to 100. The results are reported in Fig. 3. From Fig. 3, we can see that the performance of ASLA decreases as α grows larger. This is because larger α will lead to a smaller influence scope for each node, thus less information is considered when calculating the influence value. Therefore, when high ranking accuracy is required, α should be set to a small number.

4.3 Results on Real-World Networks

In this part, we first evaluate the performance of different ranking methods on the real-world networks. The results are illustrated in Fig. 4. As observed on the synthetic networks, ASLA and BC obtain much better performance than the other four ranking methods. Although ASLA is a semi-local method, it characterizes the propagation ability and the aggregation ability of each node simultaneously. Thus, ASLA can obtain comparable performance with the global method BC.

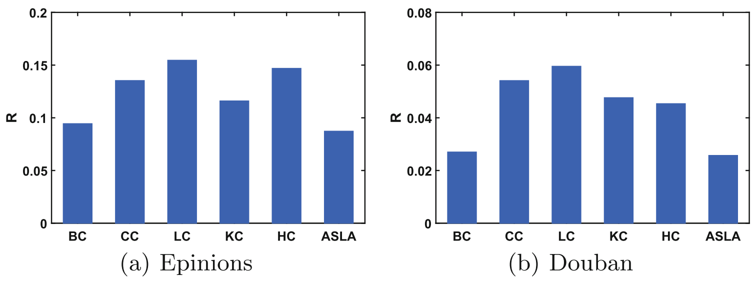
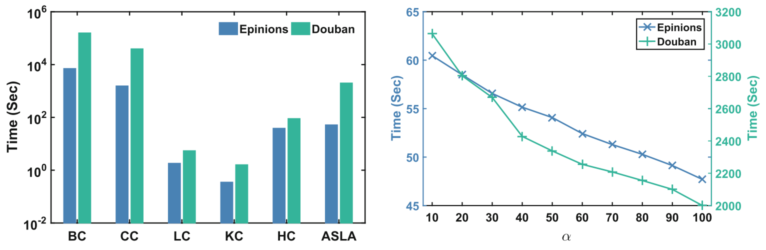


Fig. 4. Performance of different methods on real-world networks.

We further evaluate the efficiency of ASLA. We first report the time cost of different ranking methods in Fig. 5(a). Note that we have adopted logarithmic scale for the y-axis in this figure. As can be seen, the global methods BC and CC run very slow, the local methods LC, KC and HC run much faster. We can also see that ASLA is much faster than the global methods. For example, BC takes about two days on the Douban network, while ASLA takes less than one hour. In addition, with the aid of the parameter α, the time complexity of ASLA can be controlled flexibly. Next, we test the time overheads of ASLA via varying α from 10 to 100. The results are shown in Fig. 5(b), which is a double y-axes figure. It is observed that on both Epinions and Douban, the time cost of ASLA decreases rapidly as α grows larger. These results indicate that ASLA is capable of being applied to large-scale complex networks.



(a) ASLA with $\alpha = 1.0$. (b) ASLA with varying α.

Fig. 5. Comparison of time cost on real-world networks.

5 Conclusion

In this paper, we propose a novel adaptive semi-local method ASLA to identify influential nodes in complex networks. ASLA seeks a tradeoff between the time efficiency and the ranking accuracy. Therefore, ASLA can be applied to large complex networks. The main advantage of ASLA is that it can determine the potential influence scope for each node adaptively. Then the influence value of each node is calculated to capture the propagation ability and the aggregation ability simultaneously. Extensive experiments have been conducted and the results demonstrate the effectiveness and efficiency of ASLA. For future work, it is valuable to develop more advanced algorithms to further speed up ASLA. It is also valuable to design other functions to calculate the influence values of nodes.

Acknowledgement. This work was supported by the National Key Research and Development Plan (2018YFB1003800), the National Natural Science Foundation of China (61722214), the Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme 2016 and the Pearl River S&T Nova Program of Guangzhou (201710010046). Chuan Chen and Jiajing Wu are the co-corresponding authors.

References

1. Chen, D.B., Gao, H., Lü, L., Zhou, T.: Identifying influential nodes in large-scale directed networks: the role of clustering. PloS one **8**(10), e77455 (2013)
2. Chen, D., Lü, L., Shang, M.S., Zhang, Y.C., Zhou, T.: Identifying influential nodes in complex networks. Phys. A **391**(4), 1777–1787 (2012)
3. Domingos, P., Richardson, M.: Mining the network value of customers. In: SIGKDD, pp. 57–66. ACM (2001)
4. Dong, J., Ye, F., Chen, W., Wu, J.: Identifying influential nodes in complex networks via semi-local centrality. In: ISCAS, pp. 1–5. IEEE (2018)
5. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
6. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**(1), 35–41 (1977)
7. Freeman, L.C.: Centrality in social networks conceptual clarification. Soc. Netw. **1**(3), 215–239 (1978)
8. Gekas, J.: Web service ranking in service networks. In: ESWC (2006)
9. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
10. Kitsak, M., et al.: Identification of influential spreaders in complex networks. Nat. Phys. **6**(11), 888–893 (2010)
11. Leskovec, J., Sosič, R.: SNAP: a general-purpose network analysis and graph-mining library. ACM Trans. Intell. Syst. Technol. **8**(1), 1 (2016)
12. Lü, L., Chen, D., Ren, X.L., Zhang, Q.M., Zhang, Y.C., Zhou, T.: Vital nodes identification in complex networks. Phys. Rep. **650**, 1–63 (2016)
13. Morone, F., Makse, H.A.: Influence maximization in complex networks through optimal percolation. Nature **524**(7563), 65 (2015)
14. Petermann, T., De Los Rios, P.: Role of clustering and gridlike ordering in epidemic spreading. Phys. Rev. E **69**(6), 066116 (2004)
15. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015). http://networkrepository.com
16. Sabidussi, G.: The centrality index of a graph. Psychometrika **31**(4), 581–603 (1966)
17. Wang, Z., Du, C., Fan, J., Xing, Y.: Ranking influential nodes in social networks based on node position and neighborhood. Neurocomputing **260**, 466–477 (2017)
18. Ye, F., Liu, J., Chen, C., Ling, G., Zheng, Z., Zhou, Y.: Efficient influential individuals discovery on service-oriented social networks: a community-based approach. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 605–613. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_44
19. Zhang, J., Wu, J., Xia, Y., Ye, F.: Measuring cohesion of software systems using weighted directed complex networks. In: ISCAS, pp. 1–5. IEEE (2018)
20. Zheng, Z., Ye, F., Li, R.H., Ling, G., Jin, T.: Finding weighted k-truss communities in large networks. Inf. Sci. **417**, 344–360 (2017)

User Location Prediction in Mobile Crowdsourcing Services

Yun Jiang, Wei He(✉), Lizhen Cui, and Qian Yang

School of Software, Shandong University, Jinan, China
sdu_jy@163.com, {hewei,clz}@sdu.edu.cn, 791907346@qq.com

Abstract. In recent years, mobile crowdsourcing has been integrated into people's lives. A variety of mobile crowdsourcing services have emerged and been widely used, such as Gigwalk, Foursquare, and Uber. Due to the uncertainty of task distribution and workers' trajectory, as well as diverse worker interests and capabilities, it is crucial to effectively predict the mobile workers' trajectories such that they are willing to get to the location and perform their tasks with as little travel and time cost as possible. In this paper, we propose a context-sensitive prediction approach for workers' moving path in mobile crowdsourcing services. We predict the upcoming location of workers through movement rules, real-time perception of workers' moving path and contexts when assigning spatial tasks on a crowdsourcing platform, thereby pushing a task to the workers who will enter the region within the deadline of the task. Our location prediction method can avoid workers' extra cost such as time and charges in performing tasks. The analysis and simulation experiments based on real data sets show that this method can effectively predict the location of a worker and achieve better results in task assignment and completion.

Keywords: Mobile crowdsourcing · Context · Location prediction
Task assignment

1 Introduction

Jeff Howe's definition of "crowdsourcing" is: "A company or organization that outsources tasks that were performed by employees in the past to a non-specific public of the network in a voluntary form [3]. Mobile crowdsourcing services extend traditional crowdsourcing patterns to mobile space, and it does not require workers to perform tasks on a fixed web platform, but increases the constraints of time and location. The core issue is task assignment. Mobile crowdsourcing task assignment aims at assigning spatial tasks (i.e., tasks related to location and time) to a worker set [4], and the workers will complete it in a separate or cooperative manners, while meeting the requirements for time, location and other constraints of the task [1,2,5].

With the rapid development of computer network and the number of mobile intelligent terminal increasing, mobile crowdsourcing service platform can receive the task by the task publisher or the request of the worker to perform task anytime and anywhere, which puts forward high requirements for whether the platform can allocate tasks in time and dynamic adaptability. Today, a majority of the task assignment strategies are simply assigning tasks directly to workers near the task without paying attention to the variation of the workers' own trajectories and locations. If the workers are going away from the location of the task, they will probably refuse to accept the task. Because the location of workers will be far away from the task's location, the platform needs to pay extra cost to encourage workers to perform. This will not only reduce the success rate of task assignment, but also increase the time cost of workers and the additional cost of the platform.

We know that diverse data from different sources and types hide much valuable information. This paper takes into account that the user's historical track data can also bring us a lot of useful information. By analyzing these historical data, we can get the information about users' behaviors, interests and preferences. The user's location region is predicted based on the current location and information of the user and then tasks in the region are allocated to him.

When predicting, this paper is not only concerned with the binary relation of "user-location", but also taking into consideration the context information of the user (such as time, task location, weather, etc.) to form a "context-user-location" relationship. It enables us to automatically discover and use context information when predicting, and to satisfy users' personalized needs changing with the change of contextual information. For example, users are more willing to go to sports and eat after work rather than go to dinner first. Compared with workdays, users are more willing to go to entertainment plaza on weekend. In this paper, time is divided into workday and weekend. It is regarded as a kind of temporal context information and is integrated into the user's location prediction process. The context information is appropriately fused into the location prediction algorithm. On the one hand, it accords with the practical significance of contextual information; on the other hand, it has substantial help in location prediction and improves the accuracy of prediction.

The main contributions of this paper are as follows:

(1) Based on the discrete historical data, we mine the context dependent user movement pattern.
(2) Based on the context dependent user movement pattern, we propose a location prediction algorithm for mobile crowdsourcing workers, which can provide support for spatial task assignment.
(3) Based on the experiments on real data sets, we verify the validity and accuracy of the proposed mobile user location prediction algorithm.

The rest of this paper is organized as follows. In Sect. 2, we mainly discuss the related work; in Sect. 3 we give the definition; in Sect. 4, we propose the method, describe the specific algorithms and examples; in the Sect. 5, we show the experimental results; and in Sect. 6, we draw conclusions of this paper.

2 Related Work

There are usually two ways of location prediction. The first is to predict the current location based on the last access point of the user, and to predict the location by calculating the transfer probability. The Markov and hidden Markov algorithms in paper [10,12] are used for location prediction, combined with the relationship of the user and the time matching, which is only related to the transfer probability of the previous location to the current location. Paper [7] uses the ramble algorithm and the Markov algorithm for the simultaneous prediction; the user access path and the time interval are also the influencing factors of the prediction. The paper [6] predicts its future cell based on the user's current cell. Although the user's current location information has the most important meaning to predict its future location, it can greatly improve the system's pretest performance if the location information of the user's previous period of location is also taken into consideration.

The second way is to collect historical location point information to predict the current location. Paper [13] model the location of historical activities, and take the moving trend of the user as an important factor in location prediction. SPM (Sampled Pattern Matching) algorithm [8], PPM (Prediction by Partial Matching) algorithm [9] also based on the Markov model for trajectory prediction. These algorithms are based on the trajectory prediction based on Markov expansion, and some improvements have been made in improving the prediction accuracy or optimizing the time space complexity. However, there are still some problems, such as the lack of historical information, which leads to the lack of prediction accuracy in simple Markov models.

Li et al. [14] referred to the historical trajectory of workers, and recommended a route that contains as many tasks as possible for workers. Although the paper overcomes the problem of dynamic programming path, it can update the route timely when new tasks arrive. However, the influence of contextual information on workers is not taken into consideration, and workers may refuse to accept the recommended route.

In this paper we improve the location prediction method proposed in paper [13], add the influence of contexts to the prediction of movement patterns, taking into account the differences in user movement patterns on weekends/holidays and workdays, and extract movement rules based on context-sensitive movement patterns to improve the accuracy and adaptability of location prediction.

3 Problem Definition

In order to enhance the understanding, this section introduces the relevant definitions of the methods in this paper (Table 1).

Definition 1. *Workers' Movement Patterns (WMPs). Context dependent movement pattern WMPs is a sequence composed of multiple region numbers, which indicates that workers have been visited one after another in a day,* *expressed as* $W^{mp}(w) = (< (r_1, t_1), (r_2, t_2), \ldots, (r_n, t_n) >$*,C,supp). Movement Patterns can describe the trajectory of workers in daily life. C is context information, this paper mainly considers the context of time, and the C is divided into workday and weekend; supp is support, which is used to measure the possibility of a route appearing in the user's historical trajectory, supp*$\geq$*0. We refer to the Apriori algorithm for calculation and threshold setting. In this paper, the threshold is set to 1.33.*

Definition 2. *Workers' Movement Rules(WMRs). A movement rule, WMRs, describes the transfer relationship between regions which workers arrived at, expressed as* $W^{mr}(w) = < (r_1, t_1), (r_2, t_2), \ldots, (r_{k-1}, t_{k-1}) > \rightarrow < (r_k, t_k) >$*.* $< (r_1, t_1), (r_2, t_2), \ldots, (r_{k-1}, t_{k-1}) >$ *is the rule head, which represents the worker's current trajectory, and the tail of the rule* $< r_k >$ *represents the region where the worker will arrive with the greatest probability. The movement rule is obtained on the basis of the movement pattern. The following table gives an example of the set of movement rules:*

Table 1. An example of movement rule set

$< (r_1, t_1) > \rightarrow < (r_2, t_2), \ldots, (r_k, t_k) >$
$< (r_1, t_1) >, < (r_2, t_2) > \rightarrow < (r_3, t_3), \ldots, (r_k, t_k) >$
$\ldots$
$< (r_1, t_1), (r_2, t_2), \ldots, (r_{k-1}, t_{k-1}) > \rightarrow < (r_k, t_k) >$

4 Location Prediction Based on the Mining of Movement Rules

The location prediction process is shown in the Fig.1.

4.1 Generate Regions

The method proposed in this paper is based on regional prediction, all discrete location points in the history log of the mobile crowdsourcing platform are first clustered into regions, so the transfer of locations in worker's historical trajectory is converted to the transfer of regions. Assume that locations of all tasks in this paper can also be included in these regions, the location points can be aggregated into regions by using K-Means algorithm, thus to realize the transfer from points to regions.

Since the location points are discrete and relatively sparse, we use the k-means algorithm. The algorithm is simple and efficient for large datasets, and has low time complexity and space complexity.

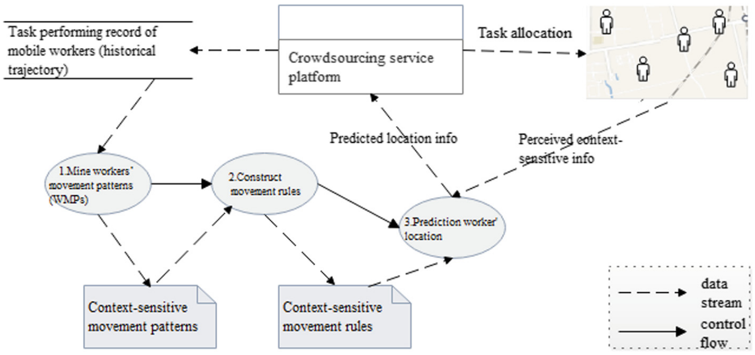

Task performing record of mobile workers (historical trajectory)
Crowdsourcing service platform
Task allocation
Predicted location info
Perceived context-sensitive info
1.Mine workers' movement patterns (WMPs)
2.Construct movement rules
3.Prediction worker' location
Context-sensitive movement patterns
Context-sensitive movement rules
data stream
control flow

Fig. 1. Location prediction process for mobile workers

Table 2. A set of workers' movement patterns

L					
Cand	C	supp	Cand	C	supp
$< 2 >$	PD/WD	3	$< 2, 3 >$	PD	2
$< 3 >$	PD/WD	3	$< 3, 8 >$	WD	2
$< 4 >$	PD/WD	3	$< 4, 3 >$	PD/WD	1.3
$< 6 >$	PD/WD	2	$< 6, 4 >$	WD	2
$< 6, 4, 2 >$	WD	1.5			

Table 3. Movement rule set

Movement rules		
Rule	Context	Confidence
$< 2 > \rightarrow < 3 >$	PD	66.7
$< 3 > \rightarrow < 8 >$	WD	66.7
$< 4 > \rightarrow < 3 >$	PD/WD	50
$< 6 > \rightarrow < 4 >$	WD	100
$< 6 > \rightarrow < 4, 2 >$	WD	75
$< 6, 4 > \rightarrow < 2 >$	WD	75

4.2 Mining Workers' Movement Patterns

In this section, we refer to Apriori algorithm, detailing how to mine workers' movement patterns. It is known that multiple workers' actual route, the first step is to determine the time context, then mining the workday and weekend movement pattern respectively. First we obtain a candidate pattern set C_1 of length 1, calculate the support and add into the movement pattern set L_1 of length 1 if the support is greater than the threshold 1.33 set in this paper. Observe which regions can be directly reached from the current region L_1, and add their region numbers to the set and form a candidate pattern set C_2 of length 2. Then calculate the support and add those greater than the threshold into the movement pattern set L_2 of length 2. According to this rule, continue to generate movement pattern sets until no one is left. Finally, combine the pattern together. Table 2 gives an example of worker context dependent movement pattern (WD represents workday, and PD represents weekend).

4.3 Generate Movement Rules

For a rule R:$< (r_1, t_1), (r_2, t_2), \ldots, (r_{k-1}, t_{k-1}) > \rightarrow < (r_k, t_k) >$, confidence is defined using the following formula:

$$confidence = \frac{< (r_1, t_1), (r_2, t_2), \ldots, (r_k, t_k) > .supp}{< (r_1, t_1), (r_2, t_2), \ldots, (r_{k-1}, t_{k-1}) > .supp} \tag{1}$$

If the confidence of a rule is higher than the pre-set confidence threshold ($coff_{min}$), it will be selected for the next regional prediction phase. Since the movement pattern is extracted based on different contexts (workday/weekend), each movement rule also needs a contextual label to indicate a specific context.

Assume that the confidence threshold is 50, then the set of movement rules is shown in Table 3 (WD represents workday, and PD represents weekend).

4.4 Predict Workers' Regions

Prediction of regions is the last stage, and the pseudocode of the algorithm is described below:

Algorithm 1. MobilityPrediction()

```
Input: The current trajectory of the worker, P=$< (r_1, t_1), (r_2, t_2), \ldots, (r_{i-1}, t_{i-1}) >$;
    Set of movement rules,$r$; The maximum number of location for each prediction,$m$
Output: set of predicted regions, $PRegions$
 1: $PRegion = \phi$ //Initially the set of predicted regions is empty
 2: k=1
 3: for rule r:$< a_1, a_2, \ldots, a_j > \rightarrow < a_{j+1}, \ldots, a_i > \in$ R //check all the rules in R do
 4:     if P.time = R.context and $< a_1, a_2, \ldots, a_j >$ is contained by P= $< (r_1, t_1), (r_2, t_2), \ldots, (r_{i-1}, t_{i-1}) >$ then
 5:         MatchingRules←MatchingRules ∪ r//Add it into the set of matching rules
 6:         TupleArray[k]=$(a_{j+1}, n, r.confidence)$//Add the $(a_{j+1}, n, r.confidence)$ tuple to the Tuples array
 7:         k = k+1
 8:     end if
 9: end for
10: TupleArray←sort(TupleArray)//descending order the Truples array according to the second element of truples, which is the length of the corresponding rule's head
11: TupleArray←sort(TupleArray)//descending order the Truples array according to the third element of truples,which is the confidence of the corresponding rule
12: index = 0
13: while index<m &&index<TupleArray.length do
14:     $PRegions \leftarrow PRegions \cup$ TupleArray[index]
15:     index = index +1
16: end while
17: return $PRegions$
```

When scanning, if the context information is inconsistent, skip the current rules and then scan the next rule which can improve the efficiency of the algorithm. After getting the matching rule set, they are first sorted according to the length of the header, and then sort according to the confidence. This ensures that the prediction based on the longest sequence is as much as possible, and the accuracy of the prediction is improved.

After predicting the location of workers, the tasks in the region are then recommended to workers. The purpose of task assignment is to achieve local optimization by allocating the maximum tasks within a period of time. We consider the change of the worker's movement trajectory, and then allocate the task to him to avoid the extra time and travel cost of the workers. It will increase the success rate of the task assignment, maximize the number of assignment tasks and reduce the cost of the platform.

5 Experiment

5.1 Experimental Design

In order to test the proposed method in real-world environment, we use the data set of Gowalla, a location-based social network, on which users can sign-in at different locations, including user time, latitude, longitude, and ID of location. More than 644 million data from 2009 to October 2010 are collected; we selected the top 1 million data with user number from 0 to 4806 as our data set, containing more than 4,000 users and 45,000 different locations.

In the experiment, the locations and users of the data set are used to represent the spatial crowdsourcing tasks and the locations of workers. As long as the worker arrives at the designated place to sign in, it is considered that the crowdsourcing task has been accepted and completed. Although the data set do not come directly from spatial crowdsourcing, it provides the distributions of workers and tasks. Since the algorithms studied in this paper rely on locations, we use this data set to draw some reasonable conclusions about their relative performance.

5.2 Experimental Result

As shown in Fig. 2, we can see that after dividing into workdays and weekends, the success rate of workdays is significantly higher than that we don't distinguish workdays. After our investigation and analysis, a worker has only two days in a week and has a weekend trajectory, the data volume is small, on the other hand, the choice of workers is too much, but the impact is small. The main reason is insufficient data. After dividing the workdays, not only the success rate is improved, but also the time complexity of the algorithm is lower. Owing to we can directly judge whether workdays are based on the context labels, reduce the time of scanning rules, and then efficiently predict the region and assign tasks.

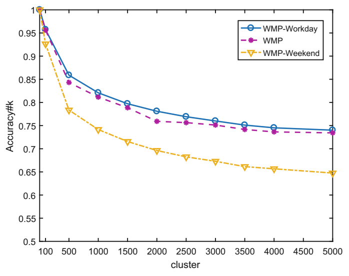


Fig. 2. Match of WMP prediction

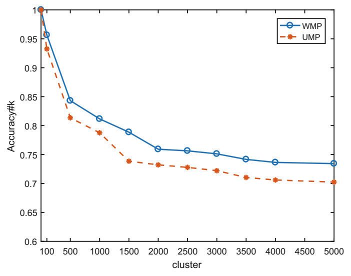


Fig. 3. Accuracy of WMP and UMP prediction

5.3 Experimental Evaluation

Next, we compare the WMP-methods in this paper with the UMP-methods proposed by Yavas et al. [13] from the perspective of accuracy of the region number that the test set predicted, as shown in Fig. 3.

The accuracy is defined as follows:

$$Accuracy\#k = \frac{|hit\#k|}{|Total|} \tag{2}$$

Accuracy#k represents the accuracy on condition that there are k-clusters; *hit#k* represents the number of data items predicted successfully; $|Total|$ represents the total number of the sign-in data items by k-clusters.

It is clear that the accuracy of this method is higher than the UMP-method in [13]. We consider the sequential of the historical track of the workers, the probability of a user going to the first 100 locations is 0.5 greater than that of the following locations, which indicates that there is some potential connection between locations, not only considering the last region when predicting [11]. We take into account the areas that workers have been visited in the history, which greatly improves the accuracy, and further increases the probability of success in task assignment.

6 Conclusion

In mobile crowdsourcing services, it is crucial to effectively predict the mobile workers' trajectories, so that they are willing to get to the location and perform their tasks with travel and time costs as little as possible. We propose a context-sensitive prediction approach for workers' moving path in mobile crowdsourcing services. Thereby, when assigning spatial tasks on a crowdsourcing service platform, a task can be pushed to the workers who will enter the region within the deadline of the task. Our approach can avoid workers' extra time and travel cost in performing the spatial tasks, and as a result, it is expected to increase the probability that a task is accepted and completed, and ultimately improve the success rate of task assignment.

Acknowledgements. This work is supported by Natural Science Foundation of Shandong Province under Grant No. ZR2018MF014 and No. ZR2017MF065.

References

1. Dang, H., Nguyen, T., To, H.: Maximum complex task assignment: towards tasks correlation in spatial crowdsourcing. In: ACM International Conference Proceeding Series, pp. 77–81 (2013)
2. Deng, D., Shahabi, C., Demiryurek, U.: Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In: ACM Sigspatial International Conference on Advances in Geographic Information Systems, pp. 324–333 (2013)
3. Howe, J.: The rise of crowdsourcing. Wired **14**(14), 1–5 (2011)
4. Kazemi, L., Shahabi, C.: GeoCrowd: enabling query answering with spatial crowdsourcing. In: International Conference on Advances in Geographic Information Systems, pp. 189–198 (2012)
5. Kazemi, L., Shahabi, C., Chen, L.: GeoTruCrowd: trustworthy query answering with spatial crowdsourcing. In: ACM Sigspatial International Conference on Advances in Geographic Information Systems, pp. 314–323 (2013)
6. Laasonen, K.: Clustering and prediction of mobile user routes from cellular data. In: Jorge, A.M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 569–576. Springer, Heidelberg (2005). https://doi.org/10.1007/11564126_59
7. Li, W., Xia, S.X., Liu, F., Zhang, L.: Hybrid Markov location prediction algorithm based on dynamic social ties. IEICE Trans. Inform. Syst. **E98.D**(8), 1456–1464 (2015)
8. Meng, W., Fang, R., Li, C., Yu, Q.: Soft network coding design in two-way relay channel. In: Global Communications Conference, pp. 4441–4446 (2013)
9. Pang, L., Zhang, Y., Li, J., Ma, Y., Wang, J.: Power allocation and relay selection for two-way relaying systems by exploiting physical-layer network coding. IEEE Trans. Veh. Technol. **63**(6), 2723–2730 (2014)
10. Robards, M.W., Sunehag, P.: Semi-Markov kmeans clustering and activity recognition from body-worn sensors. In: IEEE International Conference on Data Mining, pp. 438–446 (2009)
11. Wang, W., Yin, H., Sadiq, S., Chen, L., Xie, M., Zhou, X.: SPORE: a sequential personalized spatial item recommender system. In: IEEE International Conference on Data Engineering, pp. 954–965 (2016)
12. Yang, Y., Wang, Z., Zhang, Q., Yang, Y.: A time based Markov model for automatic position-dependent services in smart home. In: Chinese Control and Decision Conference, pp. 2771–2776 (2010)
13. Yavaş, G., Katsaros, D., Ulusoy, Ö., Manolopoulos, Y.: A data mining approach for location prediction in mobile environments $\star$. Data Knowl. Eng. **54**(2), 121–146 (2005)
14. Li, Y., Yiu, M.L., Xu, W.: Oriented online route recommendation for spatial crowdsourcing task workers. In: Claramunt, C., et al. (eds.) SSTD 2015. LNCS, vol. 9239, pp. 137–156. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22363-6_8

Leveraging Regression Algorithms for Process Performance Predictions

Karthikeyan Ponnalagu[1](✉), Aditya Ghose[2], and Hoa Khanh Dam[2]

[1] Robert Bosch, Bangalore, India
karthikeyan.ponnalagu@in.bosch.com
[2] University of Wollongong, Wollongong, Australia
aditya.ghose@gmail.com,hoa@uow.edu.au

Abstract. Industry-scale context-aware processes typically manifest a large number of variants during their execution. Being able to predict the performance of a partially executed process instance (in terms of cost, time or customer satisfaction) can be particularly useful. Such predictions can help in permitting interventions to improve matters for instances that appear likely to perform poorly. This paper proposes an approach for leveraging the process context, process state, and process goals to obtain such predictions.

Keywords: Variability · Contextual factor analysis
Business process mining

1 Introduction

Execution of complex business processes that are specifically knowledge driven, generally leads to significant amounts of event records corresponding to the execution of activities in the processes. Dealing with large scale process variations [9] is a significant challenge in process centric enterprise organizations. This is typically the case in Business Process Outsourcing (BPO) support organizations, as they support multiple business processes for different clients. Given the strictness and penalty aspects of violating service contracts governing such business processes, these organizations tend to improve the performance aspects of individual process executions Most of the current literature assumes that the performance of a process instance is entirely determined by what happens over the course of the execution of the process instance. We see limitations in such assumptions [7], when applied in knowledge intense process models, where the specific instance executions are dictated by other factors that are not part of process executions. Mining such factors and discovering correlations with process performance and validity of execution remains a significant challenge. Research works [8,11] in the field of process flexibility management, treats goal models as one of such factors that dictate execution of processes, in addition to contexts. Goal models [10] provides a natural underpinning for both validating and classifying process executions as different variants. In this paper, we argue that the correlation of semantic effect traces with goal alignments and associated context facilitates a comprehensive approach for predicting performances. In our earlier work [8], we discussed the notion of *process effect logs* as a series of time stamped ticket description entries, along with the semantic trace (effects from process execution), task details, performance time spent and the process instance identifier. The process performance time is defined as the time interval between the start and completion of execution of a process instance.

2 Motivating Example

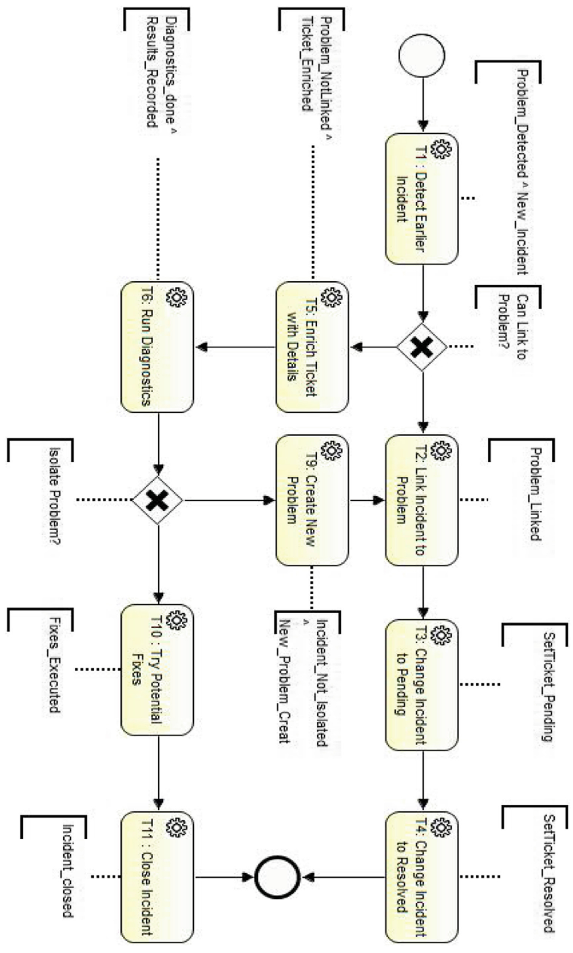

Problem_Detected ^ New_Incident
T1 : Detect Earlier Incident
Can Link to Problem?
T2: Link Incident to Problem
Problem_Linked
T3: Change Incident to Pending
SetTicket_Pending
T4: Change Incident to Resolved
SetTicket_Resolved
Problem_NotLinked ^ Ticket_Enriched
T5: Enrich Ticket with Details
T9: Create New Problem
Incident_Not_Isolated ^ New_Problem_Creat
Diagnostics_done ^ Results_Recorded
T6: Run Diagnostics
Isolate Problem?
T10 : Try Potential Fixes
Fixes_Executed
T11 : Close Incident
Incident_closed

Fig. 1. Annotated incident resolution process in VAGAI tool

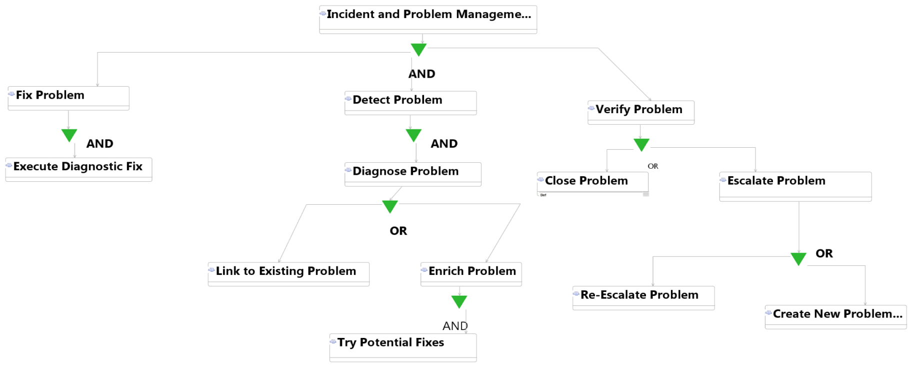


Fig. 2. Incident management goal model

In this paper, we consider an `Incident management` process design depicted in Fig. 1 as our running example. A process log[1] containing 1400 executed instances of this process design is considered for the evaluation of our proposed approach. A total of over 25000 task execution records is available as part of this process log. Each process instance in this log indicates how after receiving a complaint from a customer, an incident ticket is created, resolved and closed. We leverage annotated goal models with end effects as illustrated in Fig. 2. Such a goal model can be constructed through a goal refinement machinery as discussed in [2].

A variety of outcome predicting process monitoring techniques have been proposed in the literature [6]. In [4], the authors clearly establish the need for a general framework for mining and correlating business process characteristics from event logs. In [1], the authors discuss construction of a configurable process model as a family of process variants discovered from a collection of event logs. The existing works in the area of contextual correlation of business processes have addressed different challenges related to collaboration, contract conformance, process flexibility [5]. In comparison our work uses contextual factors and semantic effect traces on both partial and completed executions to correlate and predict execution deviation based on goal alignments. Works such as [3] focuses on generating performance predictions leveraging process simulation data. Works such as [12] focus on generating hybrid process model creation by leveraging event log clusters. In comparison, we focus on an orthogonal approach of discovering multiple process designs that are goal aligned variants of the original process design.

3 Identifying Process Context, Goals and Process State

Contextual information can be traced from process instances to a range of time-stamped information sources, such as statements being made on enterprise social

[1] https://www.scribd.com/document/333254045/IncidentLog.

media, financial market data, weather data and so on. Process log time-stamps can be correlated with time-stamps in these repositories of information to derive a wealth of information about the context within which a process instance was executed. In our proposed approach, we leverage this specific category of contextual information.

The performance indicators associated with process effect assertions are typically influenced with the entailment to specific OR-refinement sub goals (Email confirmation or Telephonic confirmation with customer) in the goal model. Given a state S and a set of effect assertions e obtained from events accruing from the execution of a task, the resulting partial state is given by $S \oplus e$, where $\oplus$ is a state update operator [8]. Similarly, given a normative state S_N and a set of effect assertions e_N obtained from events accruing from the execution of a task in a process, the resulting partial state is given by $S_N \oplus e_N$ where $\oplus$ is a state update operator. We also use a knowledge-base KB of domain constraints. If $S \cup e \cup KB$ is consistent, then $S \oplus e = S \cup e$. Otherwise, $S \oplus e = e \cup \{s \mid s \subseteq S, s \cup e \cup KB$ is consistent, and there does not exist any s' where $s \subset s' \subseteq S$ such that $s' \cup e \cup KB$ is consistent$\}$. We start with an initial partial state description (which may potentially be empty) and incrementally update it (using $\oplus$) until we reach the partial state immediately following the final task in the process instance. Towards achieving this, the proposed machinery leverage the OR-refinement goal correlations associated with each state transition from the process event log. For generating goal correlations based on the end effects (at the process or task levels), we have leveraged the `Process Instance Goal Alignment Model (PIGA)` discussed in our previous work [8]. Therefore, given a *goal-realizing effect group* S, finding correlation with a goal G in formal terms is simply finding the truth assignments in the CNF expression of G using the cumulative end effects of S. Towards generating PIGA, the list of state transitions and the goal decomposition model as input are considered. Then, for each event group in the process log, the truth assignments of all goals in the goal model are validated. This is repeated for all event groups in the process log to identify the "valid process instances". The representation of each process instance as a list of *maximally refined correlated goals* constitutes the completion of generating Process Instance Goal Alignment (PIGA).

The CCGM generated for our running example is illustrated in Table 1. For example as observed in row 3, 11 process instances are partially executed without a resolution to a reported incident due to a collection of contextual factors (CM3). To support predictions both at the process and individual task levels, we have leveraged two categories of effect log data sets: **Process Data Set (PD)**, where record in this data set is a tuple { Process Instance Identifier, a semantic trace, process execution time, context, aligned OR-refinement sub-goals } and **Task Data Set (TD)** : Each record in this data set is a tuple { Process Instance Identifier, Task Identifier, semantic trace from the execution of task, task execution time, total process execution time, context, task aligned goals, process aligned goals}.

Table 1. Context correlated goal models (CCGM)

No. of instances	Observed state effects	OR-refined goal entitlement	Context name (value)
62	T4: (Resolution_Suggested)	(Link to Existing Problem, Close Problem)	CM1 = Connection(‘Remote’, ‘NotAvailable’, ‘BehindFirewall’), CustomerExpertise(‘High’), CustomerPriority(‘Low’)
155	T3: (Resolution_Known)	(Link to Existing Problem, Close Problem)	CM2 = Solution(‘Known’, ‘AutoFix’, ‘BroadCast’), CustomerAffected(‘Group’)
11	T5: (Resolution_Cancelled)	(Close Problem)	CM3 = Agent(‘New’), ProblemOrigin(‘3rd Party’, ‘NotUnderContract’)
51	T5: (Ticket_NotEnriched)	(Escalate Problem)	CM4 = CustomerProvided (‘NoEventTrace’, ‘NotReproduced’)
10	T1: (Problem_NotCategorized), T9: (Problem_DetailIncomplete)	(Escalate Problem, Link to Existing Problem)	CM5 = Agent(‘New’), ProblemAutoCategory(‘Failed’)
5	T2: (Problem_SeverityWarning), T3: (Set_TicketPriorityHigh)	(Escalate Problem, Enrich Problem)	CM6 = Agent(‘Expert’) , ProblemAutoCategory(‘Complex’)
31	T4: (Customer_NotNotified)	(Escalate Problem, Enrich Problem)	CM7 = CustomerSupport (‘Rare’), CustomerProvided(‘NoEventTrace’, ‘NotReproduced’)

For our evaluation in this paper, we used Watson Analytics Engine’s Deep QA pipeline, to generate insights for some very interesting questions. The training data set belongs to two categories of process log data sets **PD** and **TD**. The questions that were asked using both these data sets are listed in Table 2.

4 Empirical Evaluation

In this section, we will evaluate our proposed approach using the event log data set, discussed in Sect. 2. Our evaluation is conducted in two phases : **Phase 1** : This is basically a pre-processing step that enables generation of effect logs, which are provided as input data to the Watson Analytics Engine (discussed in Phase 2). The `VAGAI tool` [8] annotates semantic traces from process logs with goal alignments to generate process effect logs (PD) and task effect logs (TD) respectively[2]. **Phase 2** : `Watson Analytics Engine` for generating performance and goal alignment predictions using the PD and TD data sets respectively as depicted in Table 2. For individual task level executions, the alignment predictions are at OR-refinement sub goal levels (providing alternate realization of its parent goal) for a given goal model. This is based on the accumulated effects at the completion of corresponding task execution.

[2] https://www.scribd.com/document/333254045/IncidentLog.

Table 2. Questions to Watson Analytic Engine

Question ID	Question text	Used data set	Question type
Q01	Given a performance limit - what are the most commonly occurring semantic effect traces?	TD	Exploratory
Q02	What are the context sets associated with processes taking high performance time?	TD	Exploratory
Q03	Given the effect sequence E1E2E3, what is the probability of the process being aligned for a given goal G?	PD	Predictive
Q04	Given the current effect sequence taking performance time N, what is the projected completion time of the process	PD	Predictive
Q05	Given the current context, and the current effect sequence, what is the remainder of the effect sequence for a successful (goal-aligned) execution	TD	Predictive
Q06	Given the current context, what will be the number of instances that are aligned with Goal G1?	PD	Predictive
Q07	Given the current context, what is the probability of this instance to conclude with a specific effect sequence?	PD	Predictive
Q08	Given the tickets with current effect sequence, what is the average total performance time of completion of these tickets?	TD	Predictive
Q09	Given the current context, how many executed instances will be valid ?	TD	Predictive
Q10	Given the current effect sequence, which process designs the completed instances will be aligned with?	TD	Predictive

The consolidated view of predictive insights as a visualization is depicted in Fig. 3. Here the performance prediction in terms of total process execution time is depicted for each observed effect at completion of a task. We started with questions of type **Q01, Q02** to generate the predictions of process performance time (in minutes) for each of the six contextual factors `DataIssues` + `AgentExplow`,`DataIssues` + `Highseverity`, `RemoteResolution` + `CustomerNew`, `RemoteResolution` + `AlertsComplete`, `SoftwareUpgrade`, `PasswordReset` + `AgentExplow`, `PasswordReset` + `Severity High` at specific

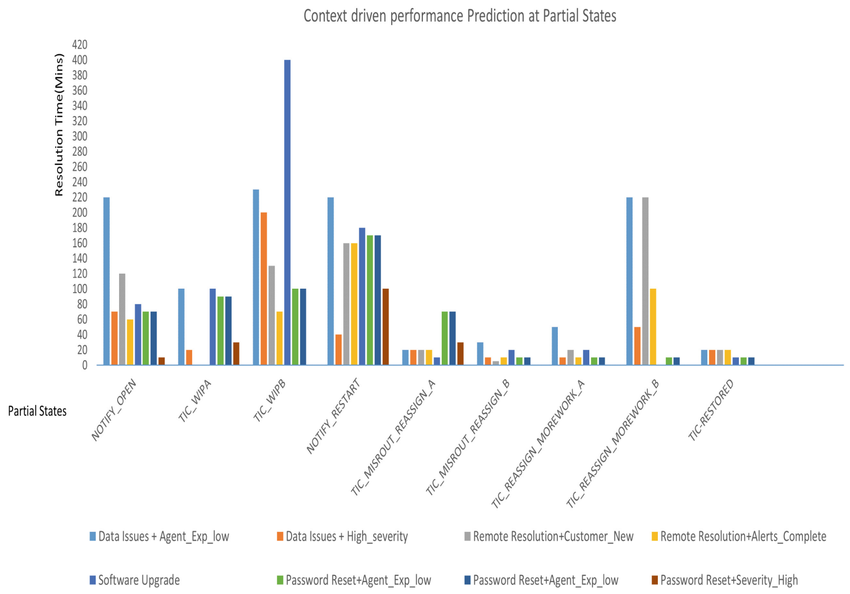


Fig. 3. Performance predictions at partial states

semantic traces in the execution of process instances. This consolidated representation generated using the *Watson Analytics Engine* helps in predicting performance at different partial states of an instance execution. This demonstrates the impact of contexts on the execution of otherwise similar process execution instances. Similarly using this prediction model represented in Fig. 3, we can make predictions of performances at multiple states of process execution. This eventually can lead the organization to evaluate their resource deployment strategies, shifting to a different process design variant.

5 Conclusion

Organizations increasingly tend to analyze the performance drifts in day to day execution of customer and context sensitive business processes. In our proposed approach, we leverage goal correlated process variations and contextual factors mined from process log and goal correlated state transitions mined from effect logs. In our future work, we will focus on correlating dynamic run-time variations in contextual factors with shifts in goal alignment.

References

1. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Mining configurable process models from collections of event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 33–48. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_5
2. Ghose, A.K., Narendra, N.C., Ponnalagu, K., Panda, A., Gohad, A.: Goal-driven business process derivation. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 467–476. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25535-9_31
3. Heinrich, R., Merkle, P., Henss, J., Paech, B.: Integrating business process simulation and information system simulation for performance prediction. Softw. Syst. Model. **16**(1), 1–21 (2015)
4. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general framework for correlating business process characteristics. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 250–266. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_16
5. Magdaleno, A.M., de Oliveira Barros, M., Werner, C.M.L., de Araujo, R.M., Batista, C.F.A.: Collaboration optimization in software process composition. J. Syst. Softw. **103**, 452–466 (2015)
6. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_31
7. Mrquez-Chamorro, A.E., Resinas, M., Ruiz-Corts, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv. Comput. **1**, 1 (2017)
8. Ponnalagu, K., Ghose, A., Narendra, N.C., Dam, H.K.: Goal-aligned categorization of instance variants in knowledge-intensive processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 350–364. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_24
9. Reichert, M., Hallerbach, A., Bauer, T.: Lifecycle management of business process variants. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 1. IHIS, pp. 251–278. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-642-45100-3_11
10. Ruiz, M., Costal, D., España, S., Franch, X., Pastor, Ó.: GoBIS: an integrated framework to analyse the goal and business process perspectives in information systems. Inf. Syst. **53**, 330–345 (2015)
11. Weske, M.: Business process management architectures. In: Business Process Management. Springer, Heidelberg (2012) https://doi.org/10.1007/978-3-642-28616-2_7
12. Yu, Y., et al.: Case analytics workbench: platform for hybrid process model creation and evolution. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 226–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_16

Using Machine Learning to Provide Differentiated Services in SDN-like Publish/Subscribe Systems for IoT

Yulong Shi[1,2](✉), Yang Zhang[1], Hans-Arno Jacobsen[2], Bo Han[1], Mengxi Wei[1], Runyuan Li[1], and Junliang Chen[1]

[1] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
{shiyulong2015,yangzhang,hanbo92,wmx,lirunyuan,chjl}@bupt.edu.cn

[2] Middleware Systems Research Group, University of Toronto, Toronto M5S 1A1, Canada
jacobsen@eecg.toronto.edu

Abstract. At present, most publish/subscribe systems assume that all participants have the same Quality of Service (QoS) requirements. However, in many real-world IoT service scenarios, different users may have different delay requirements. How to provide differentiated services has become an urgent problem. The rise of Software Defined Networking (SDN) provides endless possibilities for meeting customized services due to greater programmability. In this paper, we first propose two new methods to predict the queuing delay of switches. One is an improvement of the traditional Random Early Detection (RED) algorithm; the other is a machine learning method using the eXtreme Gradient Boosting (XGBoost) model. Then we describe an SDN-like publish/subscribe system architecture and priority queues supported by OpenFlow switches to realize differentiated services. In order to guarantee QoS, we present a two-layer queue management mechanism based on user requirements. In the end, we compare our delay prediction methods with the RED method and verify the effectiveness of the two-layer queue management mechanism. Experimental results show that our solution is effective.

Keywords: Publish/Subscribe · Software Defined Networking
Quality of Service · Queue management · Machine learning

1 Introduction

Internet of Things (IoT) is the third wave of the world's information industry revolution following computers and the Internet. Especially in recent years, with the widespread popularity of smart phones and the development of sensing technology, such as Radio Frequency Identification (RFID), barcodes, and Quick Response (QR) codes, IoT devices and services have increased explosively. Different devices, end users and application scenarios have different Quality of

Service (QoS) requirements. However, how to meet these requirements is a huge challenge. It is a good way to solve these problems by providing QoS-ware differentiated services.

Middleware systems play an important role between the network layer and application layer of IoT. The publish/subscribe (pub/sub) system is an event-driven middleware system which provides distributed, asynchronous, loosely coupled communication between message producers (publishers) and consumers (subscribers). Publishers publish events, subscribers receive events which they express their interests in. The full decoupling provided by a pub/sub paradigm in time, space and synchronization between publishers and subscribers makes it particularly suitable for large-scale distributed IoT service deployments.

Software Defined Networking (SDN) is an emerging networking paradigm in which the control plane is separated from the forwarding plane. In this way, SDN simplifies the design, management of networks and also makes the network have more programmability. SDN-like is a new pub/sub model [5] that further extends pub/sub decoupling. We can make full use of the programmability of SDN-like to provide differentiated services for customized user requirements.

Most pub/sub systems consider that all subscribers have the same QoS requirements [3]. However, in real-world scenarios, different users may have different delay requirements. Many delay-sensitive IoT applications need real-time response to anomalies which should be dealt with high priority to prevent any danger. Wang et al. [4] tried to use the Random Early Detection (RED) algorithm to predict the queuing delay of switches. However, it is inaccurate because we cannot get the enqueued and dequeued data at the same time, there is often a significant difference compared to the real delay. In this paper, we propose an improved RED algorithm and a machine learning method to predict the delay. In order to guarantee QoS, we present a two-layer queue management mechanism. The evaluations demonstrate the effectiveness of our solution.

The major contributions of this paper are as follows:

– To the best of our knowledge, we are the first to predict the queuing delay of switches using the eXtreme Gradient Boosting (XGBoost) model of machine learning. We also propose the Incremental Difference Method (IDM), an improvement of the RED algorithm, and compare the performance of them.
– We describe an SDN-like pub/sub system architecture and how to use priority queues to provide differentiated services for subscribers.
– We present a two-layer queue management mechanism based on user requirements from two different perspectives: (1) The local queue bandwidth adjustment algorithm for a single switch in the SDN controller; (2) The global QoS control strategy for all switches on the path from a publisher to a subscriber in the administrator of the system.

The remainder of the paper is organized as follows. Section 2 describes the preliminaries. Section 3 proposes an SDN-like pub/sub system architecture. Section 4 introduces the queuing delay prediction method. Section 5 presents the two-layer queue management mechanism. Section 6 provides the experimental results. Section 7 concludes this paper with an outlook on future research.

2 Preliminaries

XGBoost Model. XGBoost model is an effective machine learning method proposed by Tianqi Chen in 2016 [2], which can solve regression prediction problems. In [2], the objective function is proposed as shown in Eq. (1). We use the classic Root Mean Square Error (RMSE) loss function as the evaluation function, as shown in Eq. (2).

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \sum_{k=1}^{K} \Omega(f_k) + const \tag{1}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2} \tag{2}$$

Here, $l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i))$ is the loss function, $\Omega(f_k)$ is the regularization term. $RMSE$ is the square root of the mean of the squares of the errors between the prediction $\hat{y_i}$ and the target y_i.

RED Algorithm. The basic principle is that it monitors the average queue length to reflect the queue congestion. The RED formula is shown in Eq. (3).

$$avgQ = (1 - w) * avgQ + w * qLen \tag{3}$$

$$qLen = enQ - deQ \tag{4}$$

$$Delay = avgQ/Width \tag{5}$$

Where $avgQ$ is the average queue length, $qLen$ is the real-time queue length, w is the weight, enQ and deQ are the total number of bytes enqueued and dequeued, respectively. $Delay$ is the queuing delay. $Width$ is the queue bandwidth.

Incremental Difference Method. Equation (4) can be improved as follows:

$$qLen = \Delta enQ - \Delta deQ + qLen \tag{6}$$

Where ΔenQ and ΔdeQ are the increment of enQ and deQ, respectively. This method does not need to guarantee the simultaneity of getting the enqueued and dequeued data, avoiding the influence of measurement time difference on data.

3 SDN-like Pub/Sub System Architecture

SDN-like Pub/Sub System Architecture. The SDN-like pub/sub system architecture is shown in Fig. 1, which includes one administrator and several clusters. The administrator is responsible for the global network management and interacts with the controller of each cluster. A cluster contains a controller, several switches, publishers and subscribers. Border switches are used to interconnect clusters. Users communicate with the system by Web Service Notification (WSN). The system contains three layers: global management layer, control

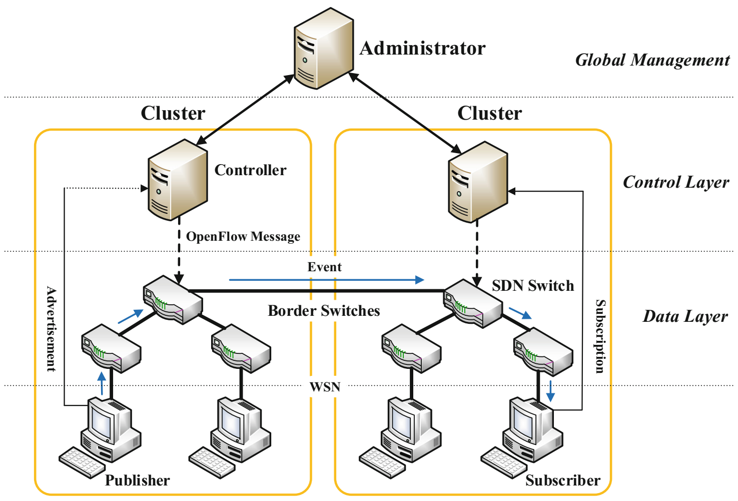


Fig. 1. SDN-like pub/sub system architecture

layer and data layer. Our SDN-like pub/sub system is mainly implemented in the control layer, namely, SDN controllers.

Two-Layer Queue Management Mechanism. The two-layer queue management mechanism is shown in Fig. 2. It is implemented in the control layer and the application layer. Specifically, the local queue bandwidth adjustment is implemented in the control layer; the global QoS control is implemented in the application layer. The priority queue is mainly implemented in SDN switches.

Topic Encoding. In our topic-based pub/sub system, topics are represented as a Lightweight Directory Access Protocol (LDAP) topic tree. The topic, event type and queue priority are encoded into binary strings of the 128 bits IPv6 multicast address in the header of packets. They are used to match flow tables directly when forwarding.

Priority Queue. Different priority queues have different bandwidths. The bandwidth size determines the forwarding capability of queues. In this way differentiation services are provided. OpenFlow switches can support up to 8 priority queues per port. These queues are numbered from 0 to 7, and the larger the queue number is, the higher the priority is. Messages are divided into three levels according to their emergency degrees: low, medium, high. The low priority messages enter queue 5, the medium enter queue 6, and the high enter queue 7, as shown in Fig. 2.

4 Queuing Delay Prediction Using XGBoost

Data Preprocessing. We collect a large amount of real data, such as bandwidth, package size by capturing data packets and logging once per monitoring

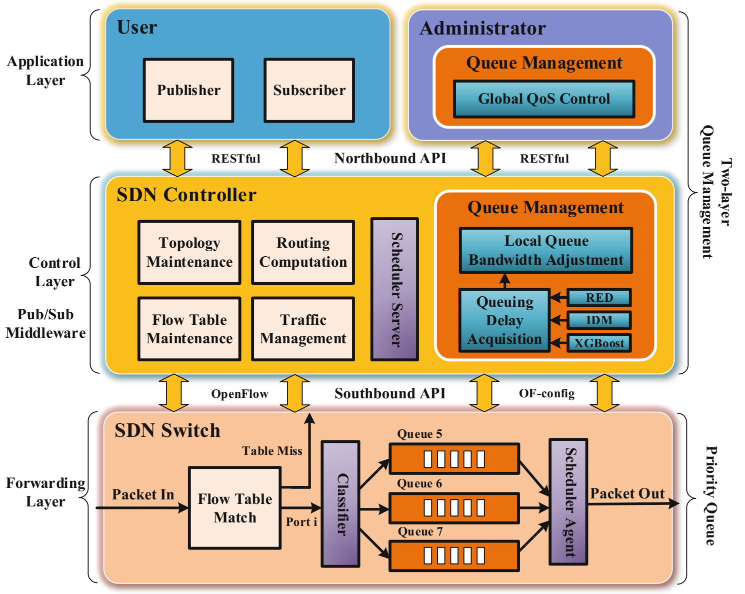


Fig. 2. Two-layer queue management mechanism

period (200 ms). Then we get the queue data after preprocessing by cleaning dummy data, filling missing values and calculation.

Feature Selection. Packets distribution means the distribution of packet transmission time intervals. The packets distribution shows periodicity, so we use the Autoregressive Integrated Moving Average (ARIMA) model [1] to obtain the cycle. We also perform a covariance test on the cycle between two adjacent packets. The correlation coefficient is 0.87293211. This higher value shows that there is little difference in their waveform distribution, and this method is reasonable. so we use packets distribution as a feature. In raw data, there are many features represented by string that the XGBoost model cannot receive, so we encode them into integer.

Model Training and Parameter Adjustment. We use the XGBoost model for training. The tree model is easily overfitting, so we divide training set by 20% as validation set and set it as watchlist to obtain the optimal number of iterations. We also use the

Table 1. Training results

XGBoost parameter	RMSE
min_child_weight=10; subsample=0.7; colsample_bytree=0.7; scale_pos_weight=0.8; max_depth=4; eta=0.1; early_stopping_rounds=30;	5.42028e+07
min_child_weight=10; subsample=0.7; colsample_bytree=0.7; scale_pos_weight=0.8; max_depth=6; eta=0.1; early_stopping_rounds=40;	5.59389e+07
min_child_weight=10; subsample=1; colsample_bytree=1; scale_pos_weight=1; max_depth=10; eta=0.1; early_stopping_rounds=50;	7.49699e+07

score of the verification set no longer declining for 10 generations consecutively as a criterion for early stop. The training results are shown in Table 1. The smaller the RMSE is, the closer the prediction is to the real value, so we choose the first row in Table 1.

5 Two-Layer Queue Management Mechanism Based on User Requirements

We achieve the two-layer queue management mechanism from two perspectives. One is the local bandwidth adjustment for a single switch, SDN controllers adjust the bandwidth according to the queue priority and the queuing delay. The other is the global control for all switches on the path. The administrator configures the delay constraint of each switch as the local bandwidth adjustment reference.

Local Queue Bandwidth Adjustment Algorithm. The bandwidth of each queue needs to be readjusted according to the delay and the queue priority. The constraints for queues are as follows:

$$w_q * t_q = AvgQ_q, q = 5, 6, 7 \tag{7}$$

$$t_q \leq T_q, q = 5, 6, 7 \tag{8}$$

$$\textstyle\sum_{q=5}^{7} w_q = Port \tag{9}$$

$$w_q > 0, q = 5, 6, 7 \tag{10}$$

$$\text{Minimize}(c_5 * t_5 + c_6 * t_6 + c_7 * t_7) \tag{11}$$

Where w_q is the bandwidth of queue q, t_q is the queuing delay, $AvgQ_q$ is the average queue length, T_q is the delay constraint, $Port$ is the total bandwidth of each switch port. Equation (11) is the adjustment goal, namely, minimizing the weighted delay of queues. c_q is queue weight (coefficient).

Algorithm 1. Local Queue Bandwidth Adjustment Algorithm

```
Input: $w_q$, $t_q$, $c_q$, $T_q$, $q = 5, 6, 7$ //$t_q$ is predicted by the XGBoost model, IDM or RED algorithm.
Output: $B_q$, $D_q$
1: Initialize $B_q = w_q$, $D_q = 0$, $Port = 100$
2: $AvgQ_q = w_q * t_q$
3: $Sum = \sum_{q=5}^{7}(AvgQ_q * c_q)^{\frac{1}{2}}$
4: if $t_q \leq T_q$ then //Lines 4∼6, use Eqs. (9)∼(11) to reason
5:     $B_q = Port * (AvgQ_q * c_q)^{\frac{1}{2}} / Sum$ //calculate the new bandwidth $B_q$
6:     $D_q = Sum * (AvgQ_q / c_q)^{\frac{1}{2}} / Port$ //calculate the new delay $D_q$
7: else
8:     notify the administrator to adjust $T_q$
9: end if
```

In Algorithm 1, B_q is sent to the switch, and D_q is fed back to the administrator. The complexity of Algorithm 1 is $O(1)$.

Global QoS Control Strategy. We use U_j to represent the delay requirements proposed by subscriber j. There is a lower delay limit t_i for hop (switch) i. We use f_i to represent the queuing delay fed back by the controller where switch i resides. For the whole path, the constraints are as follows:

$$\sum_{i=1}^{n} T_i \leq U_j, j = 1, 2, ..., m \tag{12}$$

$$T_i \geq t_i, 1 \leq i \leq n \tag{13}$$

$$\text{Minimize}(U_j - \sum_{i=1}^{n} T_i) \tag{14}$$

$$T_i' = \frac{f_i U_j}{\sum_{i=1}^{n} f_i} T_i \tag{15}$$

Where n is the number of hops, m is the number of subscribers of a topic. T_i' is the new delay constraint, T_i is the last one. We use the best adaptation principle to adjust the bandwidth, as shown in Eq. (14). If it has a solution, the administrator will take T_i as the T_q of node i, and send it to the controller. If no solution, the administrator will notify the subscriber by controller to resubmit a new one. The administrator recalculates T_i' of each switch according to Eq. (15).

The global QoS control strategy is shown in Algorithm 2. The complexity of Algorithm 2 is $O(n)$.

Algorithm 2. Dynamic Threshold Calculation Algorithm

```
Input: $CurrentDelay$, $LastDelayConstraint$, $UserDelay$, $Path$, $Priority$, $ConstraintTable$
Output: $Res$ //the new delay constraint
1: Initialize $Res = 0$, $temp = 0$
2: for $Switch$ in $Path$ do //calculate the sum of the last delay constraint on the path
3:     $Con = ConstraintTable.get(Switch).get(Priority)$
4:     $temp = temp + Con$
5: end for
6: $Res = CurrentDelay * UserDelay * temp/LastDelayConstraint$
```

This strategy makes full use of the administrator's characteristics which can control the global network. In this way, the delay of the entire path can satisfy the user needs, realizing the SDN-like topic-based differentiated services.

6 Performance Evaluation

Experiment Setup. We use three SDN-enabled physical switches and several PCs to setup the experiment topology as shown in Fig. 3. Each OpenDayLight controller, switches and some hosts form a cluster such as G1. The switch model is Pica8-p3290, the bandwidth of each switch port is 100 Mb/s.

Queuing Delay Prediction Methods Comparison. In this experiment, we set the bandwidths of queue 5, 6 and 7 are 10 Mb/s, 30 Mb/s and 60 Mb/s, respectively. For each queue we run three queuing delay prediction methods. The packet size is 1 KB. The experimental results about queue 5 are shown in Fig. 4. We can conclude that the two new methods are both better than the RED method, and XGBoost is better than IDM, so we choose the XGBoost method for the following experiments.

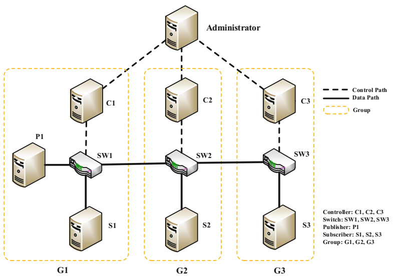


Fig. 3. Experiment topology

Local Queue Bandwidth Adjustment Algorithm Verification. For each combination of three priority queues, we compare the delay and packet loss rate under different frequencies of sending packets. The experimental results about one queue congestion are shown in Fig. 5. The bandwidth of queue 7 is 60 Mb/s before adjustment. When the frequency is between 5000 and 10000, queue 7 starts congestion, the delay remains at 125 milliseconds. After the adjustment, the queue starts becoming congested when the frequency is between 10000 and 20000, the delay remains at 83 milliseconds, the packet loss rate drops significantly and the bandwidth of this queue is 88 Mb/s. The data show that this algorithm is effective.

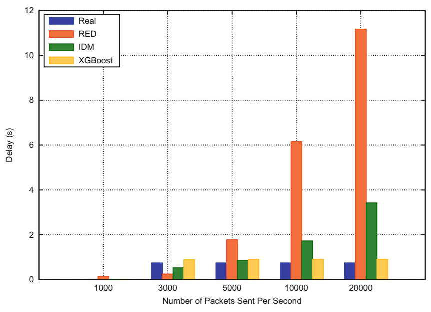


Fig. 4. Prediction methods comparison

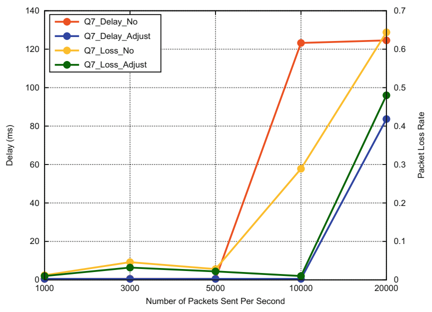


Fig. 5. Local queue bandwidth adjustment

7 Conclusion

We propose an effective machine learning method using the XGBoost model to predict the queuing delay of switches. Experiments show that it is better than IDM and the traditional RED method. We also present an SDN-like pub/sub system architecture and a two-layer queue management mechanism based on user requirements to provide differentiated IoT services. Experimental results show that our solution is effective. However, we only use three OpenFlow physical switches to do the experiments due to their high costs, therefore it is difficult to involve routing problems. In the future, we will try to solve these problems. On the other hand, we can combine the local queue bandwidth adjustment algorithm and routing algorithms to improve the QoS of pub/sub systems.

Acknowledgement. This research is supported by the National Key Research and Development Program of China (No. 2018YFB1003800), the State Scholarship Fund of China Scholarship Council (No. 201706470069), China Postdoctoral Science Foundation (No. 2017M620617). The authors would like to thank Geoffrey Elliott at the University of Toronto and the anonymous reviewers for reviewing this manuscript.

References

1. Bartholomew, D.: Time series analysis forecasting and control. J. Oper. Res. Soc. **22**(2), 199–201 (1971)
2. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. ACM (2016)
3. Tariq, M.A., Koldehofe, B., Koch, G.G., Khan, I., Rothermel, K.: Meeting subscriber-defined QoS constraints in publish/subscribe systems. Concurr. Comput.: Pract. Exp. **23**(17), 2140–2153 (2011)
4. Wang, Y., Zhang, Y., Chen, J.: Pursuing differentiated services in a SDN-based IoT-oriented pub/sub system. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 906–909. IEEE (2017)
5. Zhang, K., Jacobsen, H.A.: SDN-like: the next generation of pub/sub. arXiv preprint arXiv:1308.0056 (2013)

Quality of Service

Constraint-Based Model-Driven Testing of Web Services for Behavior Conformance

Chang-ai Sun[1](✉), Meng Li[1], Jingting Jia[1], and Jun Han[2]

[1] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
casun@ustb.edu.cn

[2] School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia

Abstract. In the current *Web Service Description Language* (*WSDL*), only the interface information of a web service is provided without any indication on its behavior logic. Naturally, it is difficult for the service user and developer to achieve a shared understanding of the service behavior through such a description. A particular challenge is how to make explicit the various behavior assumptions and restrictions of a service (for the user), and make sure that the service implementation conforms to them (for the developer). In order to improve the behavior conformance of services, in this paper we propose a constraint-based model-driven testing approach for web services. In our approach, constraints are introduced in an extended *WSDL*, called *CxWSDL*, to formally and explicitly express the implicit restrictions and assumptions on the behavior of web services, and then the predefined constraints are used to derive test cases in a model-driven manner to test the service implementation's conformance to these behavior constraints from the user's perspective. We have conducted an empirical study with three real-life web services as subject programs, and the experimental results have shown that our approach can effectively validate the service's conformance to the behavior constraints.

Keywords: Web services · Conformance testing
Model-driven testing · Test case generation

1 Introduction

In the context of Service Oriented Architecture, the implementation of web services is separated from their interface description. Service users invoke a web service only based on its interface description written in *WSDL*. Since *WSDL* provides only the signature information for web service invocations, such as types, messages, operations and bindings, a service description in *WSDL* cannot help consumers to understand the way in which the web service should be invoked because it does not indicate any restrictions or assumptions on the behavior of a service.

The behavior expected of a web service is the key to achieve the proper use of the service. A feasible way of avoiding the potential misuse of a service is to enhance the service description with the restrictions and assumptions that underlie its behavior as intended by the service developer. Furthermore, such behavior description can also be used to test the service implementation to ascertain the service's conformance to the expected behavior. However, this kind of behavior-related information is neither formally nor explicitly described in the *WSDL* description.

In this paper, we propose a constraint-based model-driven testing approach to improve the understanding and conformance of web service behavior. We leverage the description of behavior constraints to establish a bridge between service developers and service users. Constraints are used to formally and explicitly describe the implicit behavior restrictions and assumptions on service invocations and to validate the service implementation's conformance to them. The main contributions of this paper are as follows:

1. We summarize a range of common behavior constraints for web services that are useful for potential violation detection.
2. We design an extended *WSDL*, called *CxWSDL*, to incorporate the formal description of behavior constraints.
3. We develop a model-driven testing technique to validate the service implementation's conformance to the behavior constraints. The technique first derives a service behavior model from the constraint-enriched description of a service written in *CxWSDL*. Then, it uses three coverage criteria to generate test sequences from the behavior model, aimed at exercising the service implementation's support for the constraints. Test suites are consequently generated from the test sequences using a constraint solver, and used to test the web service from the user's perspective.
4. We evaluate the effectiveness of the proposed approach with three real-life web services.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. Section 3 summarizes the common behavior constraints and presents a formal description for them. Section 4 discusses the proposed constraint-based model-driven testing technique. Section 5 reports an empirical evaluation of the proposed approach. Section 6 discusses related work and the paper is concluded in Sect. 7.

2 Approach Overview

Our approach aims to achieve better understanding and conformance of service behavior and has two major aspects. First, we introduce behavior constraints to express the implicit restrictions and assumptions expected of a web service's implementation. In this regard, we extend *WSDL* to enable the explicit description of the various constraints on the invocation of a web service, resulting in *CxWSDL* (***W****eb* ***S****ervices* ***D****escription* ***L****anguage with* ***C****onstraints*). The service description written in *CxWSDL* provides the basis for a shared understanding of the service's behavior constraints between service users and service developers.

Second, we propose a model driven testing technique that first derives a $\boldsymbol{C}onstraint - based\,\boldsymbol{B}ehavior\,\boldsymbol{M}odel$ (CBM) of the service from its extended description in *CxWSDL*, then generates test cases from its *CBM*, and finally validates the service implementation's conformation to the constraints by executing the generated test cases. The proposed testing framework is shown in Fig. 1, which consists of five major components:

(1) *CxWSDL Parsing*, which parses the *CxWSDL* document provided by the web service developer to obtain the operations and constraints document for the web service, the *SOAP* message environment and the *XSD* (*XML Structure-definition Document*) for invoking these operations.
(2) *Behavior Model Construction,* which constructs the web service behavior model according to the operations and constraints document.
(3) *Test Path Generation*, which uses three coverage criteria and the web service behavior model to generate test sequences.
(4) *Test Case Generation*, which outputs an executable test suite by means of a constraint solver, taking as input a decision table provided by the web service developer and the previously generated test sequences.
(5) *Test Case Execution*, which simulates a client by executing the test cases, validates the conformance and violations to the constraints, and generates a test report according to the test results.

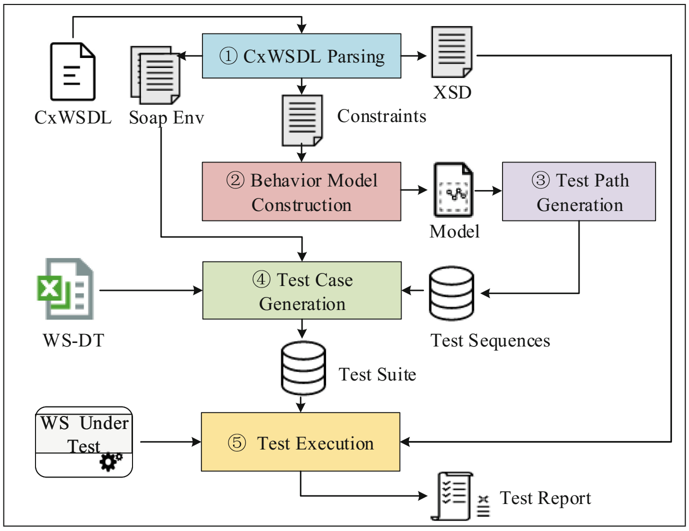


Fig. 1. Framework of model-driven testing of web services

A tool, called *MDGen*, has been developed to provide automated support for the above process. Its details cannot be included due to space limitation.

3 Constraints and Their Formal Description

This section summarizes the different types of behavior constraints and presents a formal description for them.

3.1 Types of Behavior Constraints

In our approach, constraints are used to explicitly express the behavioral assumptions and restrictions behind a service's implementation, which makes it possible to achieve a shared understanding between service developers and service users. That is, the service developer states the assumptions and restrictions in the description of a service via constraints, while a service user understands the service behavior via the stated constraints to achieve proper invocation of the service. From a literature review, we summarize the following common assumptions and restrictions, the misunderstanding of which may result in possible failures of service invocations.

- *Time Constraint* [16], which is necessary for restricting the service availability, especially when a service is being modified or in an inactive or maintenance state. If the access is outside its available period, a service invocation fault may happen due to the unavailability.
- *Region Constraint* [16], which restricts the valid range of *IP* addresses in case some operations of a service can only be accessed in a specific network.
- *Parameter Restriction Constraint* [7,18], which specifies the type and range of an input parameter of an operation.
- *Parameter Relation Constraint* [8,18], which states a relationship between the input parameters of different operations. Even if the input parameters conform to the *WSDL* type restrictions of the operations, the operation invocations may fail due to a violation of such a relationship constraint between the operations.
- *Sequence Constraint* [1,2,4], which can be a *Sequential Constraint* or a *Repeated Invocation Constraint*. The former specifies the order in which operations need to be performed or invoked for the service to function correctly, and the latter specifies whether an operation can be invoked repeatedly.
- *Invocation Constraint* [15], which identifies the other operations called by a given operation. An operation of a web service can involve another operation in performing its tasks, and thus it is important to trace and state such cascading relationships.

3.2 Formal Description of Constraints

We now consider the formal description of the above constraints. Due to the *XSD* type system can be used to define the types in a message and restriction defines the acceptable values for *XML* elements or attributes, the *XSD* restriction is well suited to describing *Parameter Restriction Constraint*. For description of other constraints, we introduce specific description constructs and their grammar is given in the *Extended Backus-Naur Form* (see Fig. 2).

```
<Constraint>       ::=  '{' (('"paraRelation":' <ValuePR> ',' '"ipRegion":' <ValueIR> ',' '"invokeOp":' <ValueIO> ','
                        '"preOp":' <ValuePO> ',' '"Iteration":' <ValueI>) | ('"eTime":' <eDate>)) '}'
<ValuePR>          ::=  '[' ']' | '['<ElementsPR>']'
<ElementsPR>       ::=  <Relationship> | <Relationship> ',' <ElementsPR>
<Relationship>     ::=  '"' <OpName> '.' <OpParameter> <RelationSymbol> <OpName> '.' <OpParameter> '"'
<RelationSymbol>   ::=  '=' | '>' | '<' | '>=' | '<=' | '!='
<ValueIR>          ::=  '"' <IpAdress> '-' <IpAdress> '"'
<IpAdress>         ::=  <IpField>'.' <IpField>'.' <IpField>'.' <IpField>
<IpField>          ::=  ('25'[0-5]|'2'[0-4][0-9]|(('1'[0-9][0-9])|([1-9]?[0-9])))
<ValueIO>          ::=  '[' '] ' | '['<ElementsIO>']'
<ElementsIO>       ::=  '"' <OpName>'"' | '"' <OpName>'"'',' <ElementsIO>
<OpName>           ::=  ([A-Z] | [a-z] | '_' | '$') (([A-Z] | [a-z] | [0-9] | '_' | '$'))*
<OpParameter>      ::=  ([A-Z] | [a-z] | '_' | '$') (([A-Z] | [a-z] | [0-9] | '_' | '$'))*
<ValuePO>          ::=  '"' <Exp>'"'
<Exp>              ::=  ('('<Exp>')*' | '('<Exp>')+' | '('<Exp>')|('<Exp>')' | '('<OpName>')('<OpName>'Response_succ)')*
<ValueI>           ::=  '"true"' | '"false"'
<eDate>            ::=  '"'<DateFormat>'"'
<DateFormat>       ::=  the value of Date Class whose Format is yyyy-MM-dd
```

Fig. 2. The grammar of *CxWSDL*

– *Constraint* identifies behavior constraints, expressed in the form of *JSON* attribute-value pairs, including *Parameter Relation Constraints* (i.e. *paraRelation*), *Region Constraints* (i.e. *ipRegion*), *Invocation Constraints* (i.e. *invokeOp*), *Sequential Constraints* (i.e. *preOp*), *Repeated Invocation Constraints* (i.e. *Iteration*), and *Time Constraints* (i.e. *eTime*).
– *ValuePR* states the *Parameter Relation Constraints*, which is a *JSON* array consisting of multiple relationships. A relationship between two parameters consisting of parameter names and relation operators. The relation operators include $=$, $>$, $<$, $>=$, $<=$, and $!=$.
– *ValueIR* states the *Region Constraints*, normally specifying the address range from which an operation can be accessed.
– *ValueIO* states the *Invocation Constraints*, which is a *JSON* array of multiple operation names that identify other operations called by an operation.
– *ValuePO* states the *Sequential Constraints*, which defines the sequential dependencies required for an operation being correctly executed, in a form of regular expressions, supporting repetition (*, +) and alternation (|).

- *ValueI* states whether an operation can be invoked repeatedly, indicated with true or false.
- *eDate* states the available time for a web service in the yyyy-mm-dd form.

To support the deployment of a service whose description is written in *CxWSDL*, we utilize the <documentation> element in *WSDL*, which is a container for human readable documentation. *Time Constraints* are added to the <*documentation*> element under the <*service*> element, and other types of constraints are added to the <*documentation*> element under each <*operation*> element. In this way, an existing container that supports *WSDL*-based web services can be directly used to deploy an extended service with behavior constraints in *CxWSDL* without any modifications.

4 Constraint-Based Model-Driven Testing of Web Services

This section presents our approach to detecting invocation violations to service behavior constraints, which improves the behavior conformance of services in a service-based system.

4.1 Constraint-Based Behavior Model Generation

In order to detect the improper invocations that violate the behavior constraints, we uses *Model Based Testing* (*MBT*) [12] for test case generation and violation detection. In particular, we propose the *Constraint-based Behavior Model* (*CBM*) of a web service based on event sequence graph.

Definition 1 (*CBM*). The *Constraint-based Behavior Model* is defined as a 4-tuple $CBM = <N_s, D, V, E>$, where

- N_s is the name of the model corresponding to a given web service,
- D is the available date of web service,
- V is a finite set of nodes in the CBM, representing the request events (operation invocations) or response events (responding to the request),
- E is a finite set of edges, representing a directed transfer from one node to another, i.e. $E \subset V \times V$.

Definition 2 (*Node*). Let V be the node set of a CBM. Each node v_i in V = $\{ v_0, \ldots, v_n \}$ is represented as a 6-tuple $v_i = <N_d, I_d, C, Pre, Suc, T>$, where

- N_d is the name of v_i,
- I_d is the unique identity of v_i,
- C is the set of constraints of v_i as defined in Sect. 3,
- Pre is the set of *Preceding Nodes* of v_i,
- Suc is the set of *Succeeding Nodes* of v_i,

- T is the type of v_i, where the different node types are: *Start* (i.e. the entry of the *CBM*), *Initial* (i.e. the initialization of a service invocation process), *End* (i.e. the end of the *CBM*), *Request* (i.e. a request event), and *Response* (i.e. a response event).

Definition 3 (*Preceding Node*). Let V be the node set of a CBM. We refer to v_i as a *Preceding Node* of v_j (denoted as $preNode(v_j)$), if and only if the following condition is true: $v_i \in V$, $v_j \in V$, and $(v_i, v_j) \in E$.

Definition 4 (*Succeeding Node*). Let V be the node set of a CBM. We refer to v_i as a *Succeeding Node* of v_j (denoted as $sucNode(v_j)$), if and only if the following condition is true: $v_i \in V$, $v_j \in V$, and $(v_j, v_i) \in E$.

Definition 5 (*Edge*). Let E be the edge set of a CBM. Each edge e_i in $E = \{ e_0, \ldots, e_m \}$ is defined as a 3-tuple $e_i = <N_e, FR, TO>$, where

- N_e is the name of the edge e_i,
- FR refers to the identify of the source node of e_i,
- TO refers to the identity of the target node of e_i.

We propose Algorithm 1 to construct a *CBM* from a *CxWSDL* document. It has the following major steps:

- *Initialization (lines 1–5)*: Initialize the *Behavior Model*, G, set its name property and *Time Constraints*, and add the *Start*, *Initial*, and *End* nodes to G.
- *Add nodes into the model (lines 6–15)*: Parse the *CxWSDL* document to identify the set of the operations of the web service under test. For each operation, add the *Request* and *Response* nodes to G and associate each node with the constraint properties.
- *Build sequence relation of nodes (lines 16–25)*: Set the sequence relation of nodes according to the sequence-related constraints.
- *Add edges (lines 26–30)*: For each node in the model, add an edge between the node and each of its *Succeeding Nodes* to the set of edges of G.

Note that the sequence-related constraints determine the behavior model's structure, which will be used to generate the test sequences (see Sect. 4.2). The non-sequence-related constraints are associated with the model's nodes, and will be used to generate test cases (see Sect. 4.3).

4.2 Test Sequence Generation

Test sequences can be generated from the service behavior model and they are classified into two types, namely *Constraint comPliant Sequences* (*CPSs* for short) and *Constraint conFlicting Sequences* (*CFSs* for short).

Definition 6 (*Constraint Compliant Sequence*). Let V and E be the node and edge sets of a CBM, respectively. A sequence of nodes $<v_0, \ldots, v_k>$ is called a *Constraint Compliant Sequence* (*CPS*), if $(v_i, v_i + 1) \in E$ for i = 0, … , $k - 1$, and v_0 is the CBM's *Start* node and v_k is the CBM's *End* node.

Algorithm 1. Behavior Model Construction

```
Input:
    $CxWSDL$ document
Output:
    $G$: $Constraint\text{-}based\ Behavior\ Model$;
1: Parse $CxWSDL$ to get $service\ name$ ($sn$), valid time ($vt$), and $operation$ set ($OpSet$);
2: Initialize $G$, set $G.V \leftarrow \varnothing$ and $G.E \leftarrow \varnothing$;
3: Let $G.S_n \leftarrow sn$ and $G.D \leftarrow vt$;
4: Add a $Start$ node $start$, an $Initial$ node $init$, and an $End$ node $end$ to $G.V$;
5: Let $start \leftarrow preNode(init)$ and $init \leftarrow sucNode(start)$;
6: for each operation $op$ in $OpSet$ do
7:    Add $Request$ node $req$, set its attributes and constraints ;
8:    Add all $Response$ nodes to $resSet$, set their attributes and constraints ;
9:    for each node $res$ in $resSet$ do
10:       $res \leftarrow preNode(req)$, $req \leftarrow sucNode(res)$;
11:       if $Iteration$ = true then
12:          $req \leftarrow preNode(res)$, $res \leftarrow sucNode(req)$;
13:       end if
14:    end for
15: end for
16: for each node $n$ in $G.V$ do
17:    if $n.T = Request$ then
18:       if $preOp$ = null then
19:          $init \leftarrow preNode(n)$, $n \leftarrow sucNode(init)$;
20:       else
21:          Parse the $preOp$ constraint;
22:          Set the preceding and succeeding correlation between nodes;
23:       end if
24:    end if
25: end for
26: for each node $n$ in $G.V$ do
27:    for each $fnode$ in $n.Suc$ do
28:       Add $< n, fnode >$ to $G.E$ and set its attributes;
29:    end for
30: end for
```

Definition 7 (*Constraint Conflicting Sequence*). Let V and E be the node and edge sets of a CBM, respectively. A sequence of nodes $<v_0, \ldots, v_k>$ is called a *Constraint Conflicting Sequence* (*CFS*), if there exists a $(v_i, v_i + 1) \notin E$, for i $= 0, \ldots, k-1$.

For a large-scale application, there may be many services that collaborate with each other and thus a large number of operations are included in such services. It is impractical or even impossible to test all the possible event sequences or paths. Thus, we define three coverage criteria to control the number of the generated test sequences.

- *Request Node Coverage*, which requires that all nodes whose type is *Request* be covered at least once.
- *Response Node Coverage*, which requires that all nodes whose type is *Response* be covered at least once.
- *Edge Coverage*, which requires that all edges should be covered at least once.

As to the generation of *CPSs*, we employ an open source testing tool, *GraphWalker* [11]. GraphWalker provides a general model traversal strategy supporting the generation of test sequences that execute each of the elements in a given model. As to the generation of *CFSs*, we first parse the *Sequential Constraint* and *Repeated Invocation Constraint*, and then generate the sequence that violates these *Sequence Constraints*. If there is a *Sequence Constraint* for an operation, a *CFS* test sequence is generated for this operation. If the *Repeated Invocation Constraint* is false, we generate a sequence that invokes an operation repeatedly. If an operation has a *Sequential Constraint*, we generate a sequence that invokes the operation without including any preceding operations.

We propose Algorithm 2 for generating test sequences from a service's *CBM*, which has five major steps:

- *Initialization (lines 1–2)*: Initialize the Constraint Compliant Sequence set (Tss), the Constraint Conflicting Sequence set ($cTss$), the initial test sequence set ($initTss$), and the set of elements ($eleCoverSet$), and set the coverage criterion.
- *Set Coverage Criterion (lines 3–12)*: Based on the selected coverage criterion, traverse G to obtain the set of elements ($eleCoverSet$) to be covered.
- *Generate CPSs (lines 13–16)*: For each element ele in $eleCoverSet$, use *GraphWalker* to generate initial *CPSs* test sequences ($initTss$).
- *Remove Redundant Sequences (lines 17–25)*: For $initTss$, delete the redundant sequences and obtain the final *CPS* test sequences tss.
- *Generate CFSs (lines 26–36)*: Generate the *CFS* test sequence set $cTss$.

4.3 Test Case Generation

We first use the constraint solver tool *Z3* to generate the combinations of input parameter values that satisfy the constraints involved in each test sequence, then incorporate such parameter values into corresponding *SOAP* messages, and finally generate the executable test cases.

The executable test cases on each test sequence are derived from the behavior model and the decision table for the operations of a web service. The Decision Table (DT) is a triple $DT = <C, E, R>$, where the *Conditions* part (C) specifies a set of constraints on the input parameters that can be evaluated to true or false, the *Events* part (E) contains a set of response events related to the *Response* type nodes, the *Rules* part (R) denotes a specific value of any combination of the conditions and their corresponding execution events. For a parking fee service *PFC*, for example, R_3 in Table 1 means that if the *login_License* input parameter to operation *login* satisfies a regular expression (i.e., MATCH (*login_License*,

Algorithm 2. Test Sequence Generation

```
Input:
    G: Constraint-based Behavior Model;
    gf: a graphml file;
Output:
    Tss: a CPS set;
    cTss: a CFS Set;
 1: Let $initTss \leftarrow \varnothing$, $Tss \leftarrow \varnothing$, $cTss \leftarrow \varnothing$, and $eleCoverSet \leftarrow \varnothing$;
 2: Set a Coverage Criterion cc;
 3: if cc = Request Node Coverage then
 4:     Parse G to get ReqNodeSet whose element is a Request node;
 5:     $eleCoverSet \leftarrow ReqNodeSet$;
 6: else if cc = Response Node Coverage then
 7:     Parse G to get ResNodeSet whose element is a Response node;
 8:     $eleCoverSet \leftarrow ResNodeSet$;
 9: else
10:     Parse G to get EdgeSet whose element is an Edge;
11:     $eleCoverSet \leftarrow EdgeSet$;
12: end if
13: for each element ele in eleCoverSet do
14:     Generate test sequence ts which covers ele;
15:     Add ts to initTss;
16: end for
17: while $eleCoverSet \mathrel{!=} \varnothing$ do
18:     Get the maximum length ts in initTss;
19:     Get the element set eleTCoverSet which ts covers;
20:     if $eleTCoverSet \mathrel{!=} \varnothing$ then
21:         $eleCoverSet \leftarrow eleCoverSet - eleTCoverSet$;
22:         $initTss \leftarrow initTss - ts$;
23:         $Tss \leftarrow Tss + ts$;
24:     end if
25: end while
26: Parse G to get ReqNodeSet whose element is a Request node;
27: for each node n in ReqNodeSet do
28:     if n.Iteration = false then
29:         Generate test sequence ts which repeated calls to n;
30:         Add ts to cTss;
31:     end if
32:     if n.preOp != null then
33:         Generate test sequence ts which directly calls to n;
34:         Add ts to cTss;
35:     end if
36: end for
```

[BJ][A-Y][0-9]{5}) and the *login_loginTime* input parameter is between 0 and 24, then the response event is *loginResponse_succ*. Thus, each rule of the *DT* defines a pre-condition of a *Response* node. Table 1 shows an example *DT* for the operation *login* of *PFC*, which has three rules, namely R_1, R_2, and R_3.

Table 1. Decision Table for a login operation

		Rules		
		R_1	R_2	R_3
Conditions	MATCH(*login_License*, [BJ][A-Y][0-9]{5}) == true	F	T	T
	$0 <= login\_loginTime <= 24$	T	F	T
Events	*loginResponse_succ*			$\surd$
	loginResponse_fail	$\surd$	$\surd$	

We traverse all the nodes of a test sequence to get their associated constraints. For the *Request* node, we obtain the related input parameter name and type from *CBM* and convert these constraints into variable definitions of *Z3*. For the *Response* node, we first parse the decision table for the node, select the appropriate rule where the event is the target node. Then, we convert those conditions to assert commands of *Z3*. For example, the *loginResponse_fail* node has two rules (i.e. R_1 and R_2 in Table 1). We then run the constraint solver script to get the solution (combinations of input parameter values) that satisfies the constraints mentioned above. Finally, we combine the parameter value combinations with the test sequences to form the executable test cases.

The above process only considers which element should be covered, without taking into account the state of the transferred data when the element is executed. Therefore, we propose a set of test suite generation strategies based on these coverage criteria with or without considering the node state in the test sequence, namely: *ReqN-S* and *ReqN-NS* representing *Request Node Coverage* with and without considering the state of node, respectively; *ResN-S* and *ResN-NS* representing *Response Node Coverage* with and without considering the state of node, respectively; *E-S* and *E-NS* representing *Edge Coverage* with and without considering the state of node, respectively.

4.4 Test Execution

We execute the service under test with the generated test cases wrapped in *SOAP* messages. In our experiment, these SOAP messages are coded into a client script. During the execution, we monitor the invocations of the service operations from the client script and determine whether an invocation violates the constraints. If a violation is detected, we record the type of constraint violated by the test case. Finally, we check whether such violations are as intended by the test cases.

5 Evaluation

5.1 Research Questions

In this study, we aim to answer the following research questions:

RQ1 Can *CxWSDL* effectively describe all the presented behavior constraints?
To answer this question, we examine possible underlying restrictions and assumption in the experimental web services and evaluate whether *CxWSDL* is able to describe them.

RQ2 Can the proposed behavior constraint-based testing technique validate the behavior conformance of web services from a user perspective?
To answer this question, we evaluate whether our approach can effectively detect service invocations that violate the behavior constraints during execution as intended by the test cases, by comparing the expected and actual invocation violations.

RQ3 What is the difference between the test suites generated using different coverage criteria in terms of violation detection effectiveness?
To answer this question, we evaluate whether the test suites generated using different coverage criteria show different detection effectiveness of invocation violations.

5.2 Subject Programs

We choose three web services to evaluate the effectiveness of our technique. *Parking Fee Calculation* (*PFC*) calculates the parking fee according to the vehicle type (e.g. *motorcycle, van, coupe*), the parking day (whether weekend or workday), the parking time, and whether using a discount coupon. *Expense Reimbursement System* (*EXP*) assists the sales director of a firm in determining the fee to be charged to each senior sales manager or sales manager for any excessive mileage in the use of the company car, and in processing reimbursement requests regarding various kinds of expenses such as airfare, hotel accommodation, meals, and phone calls. *PostalMethods* (*PostalWS*) provides the service of mailing documents such as letters, invoices, notices, and contracts.

PostalWS is a real-world web service provided by *PostalMethods.com* (http://www.postalmethods.com/), while *PFC* and *EXP* are two web services developed based on real-world business specifications. In order to illustrate the diversity of constraints, we derived another variant for each of *PFC* and *EXP*, denoted as PFC^2 and EXP^2, respectively. PFC^2 considers an additional *Time Constraint*, and EXP^2 excludes the *Region Constraint*.

5.3 Result and Analysis

Following the process of specifying behavior constraints in *CxWSDL*, deriving behavior model, generating and executing test cases using *MDGen*, we have tested each of the subject web services and collected experimental data relevant to the three research questions. Due to space limitation, further details of the experiments can not be included in the paper.

Table 2. Summary of violation detection effectiveness

Services	Coverage strategy											
	ReqN-S		*ReqN-NS*		*ResN-S*		*ResN-NS*		*E-S*		*E-NS*	
	V	*TS*	*V*	*TS*	*V*	*TS*	*V*	*TS*	*V*	*TS*	*V*	*TS*
PFC	73	109	3	4	82	118	5	6	82	118	5	6
PFC^2	109	109	4	4	118	118	6	6	118	118	6	6
EXP	21	21	3	3	41	41	6	6	209	209	6	6
EXP^2	1	21	0	3	21	41	3	6	25	209	3	6
PosatlWS	7	26	5	12	85	102	13	20	85	152	13	23

(1) **Expressive power of *CxWSDL* for behavior constraints description.** We have analyzed each operation for *PFC*, PFC^2, *EXP*, EXP^2 and *PostalWS*, obtained the behavioral constraints of these experimental services. The different types of behavior constraints for the different experimental services are shown in the second column in Table 3.
We can see that the subject services cover all the six types of behavioral constraints discussed in Sect. 3. Furthermore, all these constraints are described in *CxWSDL* documents, which can be deployed and accessed in the same way as a WSDL document. In summary, the result shows that *CxWSDL* can adequately express the service behavior constraints proposed in this paper.

(2) **Behavior conformance.** After generating the behavior model and test sequences for each subject service, we generate test cases using six test case generation strategies. Each strategy and the number of test cases in the associated test suites are shown in Table 2. The test suites contain both constraint-conforming and constraint-violation test cases, where *V* refers to the number of test cases that detected violations and *TS* refers to the total number of test cases. In our experiment, once the *CxWSDL* document is obtained, it is easy to generate the *CBM* and test suite automatically using *MDGen*. The results show that our approach can detect all the improper invocations and can correctly locate the violations as determined by the types of service constraints being violated, as shown in Table 3.

(3) **Effectiveness of different coverage criteria.** Table 3 shows the violation detection effectiveness of the different coverage strategies. The number of related test cases generated using different coverage criteria are given in the third to eighth columns. The results show that the *Response Node* and *Edge* coverage criteria can cover more types of behavior constraints than the *Request Node* coverage criterion.

Table 3. Distribution of detected violation by different test case generation strategies

Services	Constraints	Coverage strategy					
		ReqN-S	*ReqN-NS*	*ResN-S*	*ResN-NS*	*E-S*	*E-NS*
PFC	*paraRestriction*	0	0	7	1	7	1
	preOp	36	1	36	1	36	1
	Iteration	37	2	37	2	37	2
	paraRelation	0	0	2	1	2	1
PFC ^{2}	*eTime*	108	3	110	4	110	4
	paraRestriction	0	0	7	1	7	1
	Iteration	1	1	1	1	1	1
EXP	*paraRestriction*	0	0	20	3	20	3
	invokeOp	12	1	3	1	9	1
	ipRegion	9	2	18	2	180	2
EXP ^{2}	*paraRestriction*	0	0	20	3	20	3
	invokeOp	1	0	1	0	5	0
PosatlWS	*paraRestriction*	0	0	52	7	52	7
	preOp	7	4	7	4	7	4
	Iteration	0	1	2	1	2	1
	paraRelation	0	0	24	1	24	1

6 Related Work

Many research efforts have been made to address the challenging issues of web services testing. We describe closely related work from the perspective of extensions to *WSDL* and model-based testing techniques.

6.1 Extensions to *WSDL*

A service description contains basic information as well as additional information, such as exceptions, operational semantics, and contractual conditions. Researchers have proposed extensions to *WSDL* with various purposes, such as testing and behavioral modeling.

For testing web services, Tsai et al. [15] proposed four types of extensions to *WSDL* (input-output dependency, invocation sequence, hierarchical functional description, and concurrent sequence specification) to support the description of dependencies. Similarly, Sneed et al. [14] extended *WSDL* with the pre-condition assertions, and Jiang et al. [8] extended *WSDL* using *Design-by-Contract* for precisely locating faults when the web service does not meet its requirements.

For modeling service behaviors, Sheng et al. [13] extended *WSDL* with *Semantic Markup for Web Service* (*OWL-S*) and *Web Service Semantics* (*WSDL-S*) to support the description of service behaviors. Bertolino et al. [3]

extended *WSDL* with *Protocol State Machine* to describe the prescribed ordering of operation invocations. Heckel et al. [6] extended *WSDL* with graph transformation rules to support the modeling of both the service's behavior and the client's requirements.

In this work, we have extended *WSDL* with constraints to support the description of restrictions or assumptions of service behaviors, and a formal language is provided for expressing common constraints. Such an extension provides the basis for testing the conformance of web services to their behavior constraints from a user perspective.

6.2 Model-Driven Testing of Web Services

Various models have been proposed for testing web services or their composites, such as *Finite State Machine* (*FSM*) [5,9,10], *Event Sequence Graph* (*ESG*) [1,4], and *Unified Modeling Language* (*UML*) [17,19].

Keum et al. [9] proposed to model web service behaviors with *Extended Finite State Machine* (*EFSM*) and generate test cases from the *EFSM* model to achieve a better test coverage. Endo et al. [5] proposed a model-based testing process for service-oriented applications, and *FSM* was used to model and support test case generation. Similarly, Kiran et al. [10] proposed an *FSM* model-based approach to testing composite services, which focuses on the test coverage required for testing the component services individually and their compositions.

Endo et al. [4] proposed an integrated testing strategy for web services, which first used *ESG* to model web services under test, then generated test cases from the *ESG* model, finally conducted a coverage analysis after the test case execution. Belli et al. [1] proposed a model-based approach to testing composite services, in which message exchanges in a web service were viewed as events modeled using *ESG*. These techniques mainly focus on structural testing of web services or their compositions without considering internal constraints on the invoked services.

Wu et al. [17] proposed a combination of *EFSM* and *UML* sequence diagram, called *EFSM-SeTM*, from which various coverage criteria are defined to test all possible scenarios. Similarly, Zhang et al. [19] proposed an extended *UML* activity diagram to model the behavior of *BPEL* service compositions, and defined coverage criteria on the model. These techniques focus on coverage testing of composite services, while ignoring behavior conformance of component web services.

In this work, we have proposed a model-driven approach to testing web services' conformance via behavior constraints from a user's perspective. The service behavior is modeled using *ESG* derived from constraints expressed in *CxWSDL*, and test cases are generated from the behavior model with respect to coverage criteria. Unlike the existing model-based testing approaches that mainly focus on test coverage of web services or their compositions, our approach focuses on the behavior conformance of web services, and connects the description of behavior constraints to service executions with executable test cases.

7 Conclusion

In this paper, we have proposed a constraint-based model-driven testing approach for testing the behavior conformance of web services. Our approach leverages constraints to provide more accurate descriptions of the behavior logic of web services and consequently enhances the testing of services through such behavior-based test case generation and execution. Experimental results have shown that our approach can effectively generate test cases and detect the service invocations that violate the service behavior constraints.

In future work, we plan to consider further types of constraints and carry out evaluations with more complex real-life web services.

Acknowledgment. This work is supported by the Beijing Natural Science Foundation of China under Grant No. 4162040, the National Natural Science Foundation of China under Grant No. 61872039, the Aeronautical Science Foundation of China under Grant No. 2016ZD74004, the Fundamental Research Funds for the Central Universities under Grant No. FRF-GF-17-B29, and China Postdoctoral Science Foundation under Grant No. 2017M620617.

References

1. Belli, F., Endo, A.T., Linschulte, M., Simao, A.: A holistic approach to model-based testing of web service compositions. Softw.: Pract. Exp. **44**(2), 201–234 (2014)
2. Belli, F., Linschulte, M.: Event-driven modeling and testing of web services. In: Proceedings of the 32nd IEEE International Computer Software and Applications Conference, pp. 1168–1173. IEEE CS (2008)
3. Bertolino, A., Polini, A.: The audition framework for testing web services interoperability. In: Proceedings of the 31st International Conference on Software Engineering and Advanced Applications, pp. 134–142. IEEE CS (2005)
4. Endo, A.T., Linschulte, M., Simao, A.D.S., Souza, S.R.S.: Event-and coverage-based testing of web services. In: Proceedings of the 4th International Conference on Secure Software Integration and Reliability Improvement Companion, pp. 62–69. IEEE CS (2010)
5. Endo, A.T., Simao, A.: Model-based testing of service-oriented applications via state models. In: Proceedings of the 8th IEEE International Conference on Services Computing, pp. 432–439. IEEE CS (2011)
6. Heckel, R., Mariani, L.: Automatic conformance testing of web services. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 34–48. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31984-9_4
7. Hou, K.J., Bai, X.Y., Lu, H., Li, S.F., Zhou, L.Z.: Web service test data generation using interface semantic contract. J. Softw. **24**(9), 2020–2041 (2013). (in Chinese)
8. Jiang, Y., Xin, G.M., Shan, J.H., Xie, B.: Research on a testing technology based on design-by-contract. J. Softw. **15**, 130–137 (2004). (in Chinese)
9. Keum, C.S., Kang, S., Ko, I.-Y., Baik, J., Choi, Y.-I.: Generating test cases for web services using extended finite state machine. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 103–117. Springer, Heidelberg (2006). https://doi.org/10.1007/11754008_7

10. Kiran, M., Simons, A.J.H.: Model-based testing for composite web services in cloud brokerage scenarios. In: Ortiz, G., Tran, C. (eds.) ESOCC 2014. CCIS, vol. 508, pp. 190–205. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-14886-1_18
11. Kristian, K.: Graphwalker (2017). http://graphwalker.github.io
12. Micskei, Z.: Model-based testing (MBT) (2017). http://mit.bme.hu/~micskeiz/pages/mbt.html
13. Sheng, Q.Z., Maamar, Z., Yao, L., Szabo, C., Bourne, S.: Behavior modeling and automated verification of web services. Inf. Sci. **258**(3), 416–433 (2014)
14. Sneed, H.M., Huang, S.: WSDLTest - a tool for testing web services. In: Proceedings of the 8th IEEE International Workshop on Web Site Evolution, pp. 14–21. IEEE CS (2006)
15. Tsai, W.T., Paul, R., Wang, Y., Fan, C., Wang, D.: Extending WSDL to facilitate web services testing. In: Proceedings of the 7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002), pp. 171–172. IEEE CS (2002)
16. Wang, P.W., Ding, Z.J., Jiang, C.J., Zhou, M.C.: Constraint-aware approach to web service composition. IEEE Trans. Syst. Man Cybern. Syst. **44**(6), 770–784 (2017)
17. Wu, C.S., Huang, C.H.: The web services composition testing based on extended finite state machine and UML model. In: Proceedings of the 5th International Conference on Service Science and Innovation, pp. 215–222. IEEE CS (2013)
18. Xu, L., Chen, L., Xu, B.W.: Testing web services based on user requirements. J. Softw. **36**(6), 1029–1040 (2011)
19. Zhang, G., Mei, R., Zhang, J.: A business process of web services testing method based on UML 2.0 activity diagram. In: Proceedings of the Workshop on Intelligent Information Technology Application, pp. 59–65. IEEE CS (2007)

QoS Optimization of Service Clouds Serving Pleasingly Parallel Jobs

Xiulin Li[1], Li Pan[1(✉)], Shijun Liu[1(✉)], Yuliang Shi[1,2(✉)], and Xiangxu Meng[1]

[1] School of Software, Shandong University, Jinan 250101, China
lixiulind@163.com, {panli,lsj,shiyuliang,mxx}@sdu.edu.cn
[2] Dareway Software Co., Ltd., Jinan 250101, China

Abstract. A service cloud could improve its QoS (Quality of Service) by partitioning jobs into multiple tasks and processing those tasks in parallel. In contrast to processing all jobs with the same degree of parallelism (DOP), dividing jobs into different groups and processing them with varying DOPs may achieve better performance results, especially focusing on those jobs which have a greater impact on performance of service clouds. In this paper, we describe a novel differentiated DOP policy, which divides jobs into several groups identified by jobs' service time and sets proper DOPs for different groups of jobs. Then, we propose a parallel multi-queue and multi-station analytical model for service clouds with our differentiated DOP policy, to predict important performance metrics. Thus this model can guide cloud providers to determine optimal DOPs and resource allocation schemes for different groups to improve the total QoS of a service cloud. We also present a new metric, called Optimized Performance of Groups (OPG), to quantify the level of performance optimization of every group. The objective is to maximize the minimum OPG to ensure OPG within a certain range, thereby enforcing a fair trade-off between all groups. Through extensive experiments, we validate the effectiveness of the proposed differentiated DOP policy and analytical model.

Keywords: QoS optimization · Service cloud
Pleasingly parallel jobs · Degree of parallelism

1 Introduction

Because of economies of scale and other factors, service clouds usually operate a large number of resources to provide a wide range of professional services to users at low prices [5,15]. Thus more and more users are beginning to migrate their jobs to service clouds. In general, in order to meet requirements of users, service cloud providers need to provide effective job execution strategies and resource allocation schemes to accelerate jobs' execution and thus to improve QoS (Quality of Service), which is a measurement of the overall performance of a service cloud and includes response time, waiting time, probability of immediate

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 560–575, 2018.
https://doi.org/10.1007/978-3-030-03596-9_41

service, and so on [3,9]. Taking a rendering service cloud as an example, after accepting a batch of rendering jobs submitted by users, it could divide those jobs into different groups and process them with proper resources and degrees of parallelism (DOPs) to speed up the processing of the jobs.

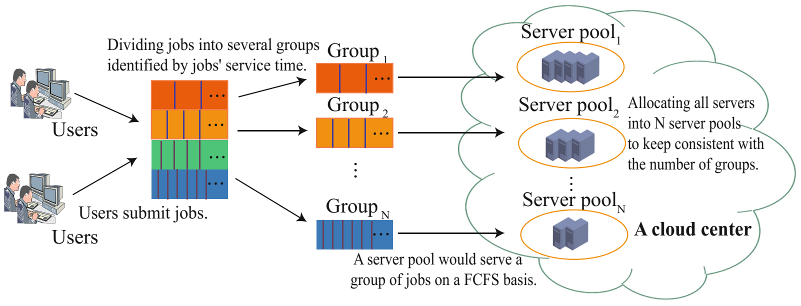

Dividing jobs into several groups identified by jobs' service time.
Users
Users submit jobs.
Users
Group 1
Group 2
Group N
A server pool would serve a group of jobs on a FCFS basis.
Server pool1
Server pool2
Server poolN
Allocating all servers into N server pools to keep consistent with the number of groups.
A cloud center

Fig. 1. A motivating example for a service cloud with differentiated DOP policy

A special kind of service cloud which serves pleasingly parallel jobs is considered in this paper, such as rendering service cloud [17], translating service cloud [16], designing and manufacturing service cloud [4], and so on [13]. A main characteristic of a pleasingly parallel job is that it can be partitioned into multiple tasks and these tasks are independent of one another and thus can be served separately by different servers in parallel without any overhead [6]. In contrast to processing all jobs with the same DOP, dividing jobs into different groups and processing them with varying DOPs may achieve better QoS, especially focusing on jobs that have a greater impact on performance of service clouds. However, for a service cloud provider, determining proper DOPs and resource allocation schemes for different groups of jobs to achieve optimal QoS is particulary complicated. This is mainly because of the following three reasons:

- The DOP of jobs can make a big difference in a service cloud's performance. A higher DOP could accelerate the processing of jobs, and thus improve the performance of a service cloud. However, it could cause more servers to be occupied which leads newly arrived tasks to be blocked.
- When the number of servers is fixed, allocating servers to serve different jobs can also make a big difference in a service cloud's performance. For example, if we allocate more servers for one group of jobs, there may not be enough severs available for other jobs. It could cause more jobs to wait in the buffer and increase mean response time of the jobs so that the number of completed jobs may drop in duration.
- When the optimal performance metrics are pursued, such as mean response time or mean waiting time of service clouds, the level of performance optimization of some groups of jobs may become ignored. This will lead to an unfair trade-off between different groups.

For achieving optimal QoS of service clouds, performance analytical models are usually established for service clouds, which could guide cloud providers to determine proper DOPs and resource allocation schemes. For dealing with the problem of QoS optimization for parallel jobs in service clouds, several studies have used queuing theories to establish analytical models [7,14,19]. These models could predict the important performance metrics of service clouds to help cloud providers improve the QoS of parallel jobs with the same DOP, but they cannot interpret more complicated situations where jobs have varying DOPs.

In this paper, for dealing with the problems of QoS optimization of service clouds serving jobs with varying DOPs, based on queuing theory, we propose a parallel multi-queue and multi-station analytical model to evaluate service clouds' performance. Based on homogeneous Markov chain model, we solve our analytical model to obtain an approximate estimation of important performance metrics such as mean service response time, mean waiting time, and so on. As different sizes of jobs have diverse impacts on performance of service clouds, we propose a differentiated DOP policy which divides jobs into several groups identified by jobs' service time and sets proper DOPs for different groups of jobs, as shown in Fig. 1. Then, we divide all servers into multiple server pools to remain consistent with the number of groups and one server pool serves one group of jobs on a FCFS (First Come First Serve) basis. Thus, based on our analytical model, we can determine optimal DOPs and resource allocation schemes for different groups of jobs, and thus improve the QoS of service clouds. We also propose a metric, called Optimized Performance of Groups (OPG), to quantify the level of performance optimization of each group, which is the ratio between the response time a group of jobs has spent in a service cloud without differentiated DOP policy and the response time this group would have spent in that service cloud with differentiated DOP policy. The objective is to maximize the minimum OPG to ensure OPG within a certain range, thereby enforcing a fair trade-off between all groups. Through extensive simulations based on synthetic data, we validate the effectiveness of our analytical model and differentiated DOP policy.

The rest of the paper is organized as follows. Section 2 describes our differentiated DOP policy. Section 3 describes our parallel multi-queue and multi-station model and solves our model to obtain performance metrics for a service cloud. Section 4 describes our proposed performance metric. Section 5 presents and discusses analytical results as well as simulation results. We discuss the related works in Sect. 6. Finally, we state concluding remarks and future work in Sect. 7.

2 Differentiated DOP Policy

Dividing jobs into different groups and processing them with varying DOPs may achieve better QoS, especially focusing on jobs that have a greater impact on performance of service clouds. In this paper, we describe a differentiated DOP policy in order to set proper DOPs for different jobs. Since different sizes of jobs have a diverse impact on performance, it could improve the QoS of service clouds obviously if we set a higher DOP for jobs which require longer service time. In this work, we divide jobs into N groups which are $\{group_1, group_2, \ldots, group_N\}$ identified by jobs' service time. There are $N+1$ points of service time: $\{t_0, t_1, \ldots, t_N\}$ and $t_0 < t_1, \ldots, < t_N$. If the service time of a job is within the range of t_{i-1} to t_i, this job will be divided into $group_i$, so that service time of jobs in $group_i$ is within the range of t_{i-1} to t_i, for $1 \le i \le N$. Then, we could set a proper DOP for each group of jobs and use d_i to denote the DOP of $group_i$.

We only consider the case of identical servers and allocate c identical servers contained in a service cloud into N server pools to remain consistent with the number of groups, which are $\{pool_1, pool_2, \ldots, pool_N\}$, as shown in Fig. 1. Servers in a server pool would serve a group of jobs on a FCFS basis, e.g., $pool_i$ contains c_i servers that serve jobs from $group_i$. Therefore, we have:

$$\sum_{i=1}^{N} c_i = c \tag{1}$$

In order to achieve specific QoS and save costs, we need to determine optimal settings, such as resource allocation schemes or DOP settings. Thus, a proper analytical model is needed to predict the performance metrics with different resource allocation schemes and DOP settings.

3 Analytical Model Formulation

In this work, based on queuing theory, we propose a parallel multi-queue and multi-station model to evaluate the performance of service clouds serving jobs with varying DOPs, as shown in Fig. 2. N queues can be constructed for a service cloud with differentiated DOP policy, which are $\{queue_1, queue_2, \ldots, queue_N\}$. In $queue_i$ $(1 \le i \le N)$, c_i servers in server $pool_i$ serve jobs from $group_i$.

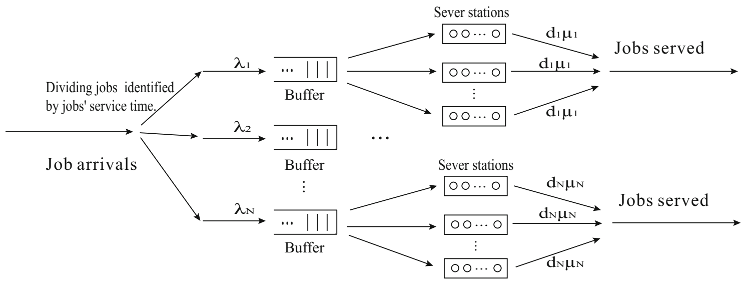


Fig. 2. A parallel multi-queue and multi-station model for jobs with varying DOPs

In order to obtain performance metrics of a service cloud, we need to predict performance metrics of all queues in that service cloud. For $queue_i$ that is any one of all queues, before jobs are partitioned, we assume that job arrivals follow an approximate Poisson process with a rate of λ_i and service time is an approximate exponentially distributed with a rate of μ_i, as is widely assumed in previous works [20]. This is because if job arrivals follow a Poisson process and service time is exponentially distributed, analytical results of these jobs can supply approximate lower bounds in performance prediction for other distributions of job arrivals, such as random distributions or normal distributions [14].

As discussed above, before jobs are partitioned, it can be concluded that $queue_i$ can be modeled as an $M/M/c_i$ queuing system. However, after jobs are partitioned into tasks, the actual probability distribution of service time would not follow the original distribution. As tasks will occupy more servers, the number of busy servers is different from the number of jobs in service clouds. Thus, we need to carefully establish a parallel multi-station analytical model for $queue_i$ serving the tasks partitioned from the original jobs, which is an important part of the parallel multi-queue and multi-station model.

3.1 Multi-station Model for $Queue_i$

Based on $M/M/c_i$ queuing model of $queue_i$, we describe a parallel multi-station model for $queue_i$, as shown in Fig. 2. In this work, a partition method is designed to help us establish a multi-station model for $queue_i$. All c_i servers contained in $queue_i$ are divided into d_i stations of servers, which are $\{C_{i1}, C_{i2}, \ldots, C_{id}\}$. We use c_{ij} $(j \in \{1, \ldots, d_i\})$ to denote the number of servers contained in the station C_{ij}. Here when dividing servers into multiple stations, a main principle we use is to *equally* divide all servers into d_i stations. Therefore, each server station consists of the same number of servers, which means $c_{i1} = c_{i2} = \ldots = c_{id} = \lfloor c_i/d_i \rfloor$. At each station, tasks will be served on a FCFS basis by one of the servers in that station. If at least one server is idle in each server station at the time when a job arrives, tasks of this job will get into service immediately. If all servers are busy at the time when a job arrives, the job will wait in the buffer. We assume that the buffer size of a service cloud is unlimited, so that jobs won't be lost. Based on our partition method, we describe how to obtain the relevant parameters of tasks in every station.

If a system could be modeled as an $M/M/c$ queuing system and jobs in this system could be equally partitioned into k $(k > 0)$ tasks, as the property of exponential distribution, the service time of tasks is also exponentially distributed [2]. However, its base has changed, and the mean service time of the tasks is $1/k$ of the mean job service time. In our analytical model, before entering into servers, each job is *equally* partitioned into d_i tasks which have the same service time. Then by dividing all servers equally into d_i stations, after an arriving job is equally split into d_i tasks, each task could be assigned to one of the d_i stations individually and no two stations could accept a same task. Thus all the stations have the same workload to serve and since they have the same number of servers with the same processing capability, they would achieve the same performance metrics in processing these tasks. Based on the above description, we can conclude that the service time in each server station is an independent identically distributed (IID) random variable having negative exponential distributions, with mean service time of $1/(d_i\mu_i)$. The arrival of each job is the same as its tasks. So task arrivals follow a Poisson process, which means the task interarrival time is exponentially distributed with a rate of λ_i, the same as the primitive job. The traffic intensity ρ_i denotes the average proportion of time for which each of the servers is occupied. Thus, the traffic intensity of one server station could be defined as $\rho_i = \lambda_i/(c_{i1}d_i\mu_i)$, where for practical reasons, it is known that $\rho_i < 1$.

As discussed above, we can establish an $M/M/c_{i1}$ queuing system for each station. All d_i stations have the same processing capability and the same workload, so that the performance metric of one station can represent the performance of other stations as well as $queue_i$. Thus we can analyze the server station C_{i1} to obtain its performance metrics through the proposed analytical model and then conclude performance metrics of the whole $queue_i$ from that.

In order to construct the analytical functions to calculate the important performance metrics, we need to calculate the steady-state probabilities of the number of tasks in the $M/M/c_{i1}$ queuing system of server station C_{i1}, which can be figured out by using a homogeneous Markov chain.

Underlying Markov Model. In this work, we establish a homogeneous Markov chain for server station C_{i1} to calculate the important performance metrics. We model the number of tasks in server station C_{i1} (both those in service and those queued but not yet served) at the moments immediately before a new task arrival as a Markov point. Then by enumerating these instances as $\{0, 1, 2, \ldots, n-1, n\}$, we can obtain a homogeneous Markov chain. Therefore, we can calculate the steady-state probabilities of the number of tasks in the queuing system to obtain the performance metrics of this server station.

When the steady-state system stays at state n, we use P_n $(0 \leq n)$ to denote the probability of having n tasks in the system immediately before a new task arrives. The balance equations can be established as:

$$d_i\mu_i P_1 = \lambda_i P_0 \tag{2}$$

$$(n+1)d_i\mu_i P_{n+1} + \lambda_i P_{n-1} = (\lambda_i + nd_i\mu_i)P_n \quad (1 \leq n \leq c_{i1}) \tag{3}$$

$$c_{i1}d_i\mu_i P_{n+1} + \lambda_i P_{n-1} = (\lambda_i + c_{i1}d_i\mu_i)P_n \quad (c_{i1} < n) \tag{4}$$

The normalization equation of the transition probability matrix P is:

$$\sum_{j=0}^{\infty} P_j = 1 \tag{5}$$

As we have defined above, the traffic intensity for C_{i1} server station is:

$$\rho_i = \lambda_i/(c_{i1}d_i\mu_i) \qquad (\rho_i < 1) \tag{6}$$

Using recurrence relations to solve differential equation (2) (3) (4) via (5) (6), in which P_n can be deduced from P_{n-1} that we have known, we can obtain the steady-state probability as:

$$P_0 = [\sum_{k=0}^{c_{i1}-1} \frac{1}{k!}(\frac{\lambda_i}{d_i\mu_i})^k + \frac{1}{c_{i1}! \times (1-\rho_i)}(\frac{\lambda_i}{d_i\mu_i})^{c_{i1}}]^{-1} \tag{7}$$

$$P_n = \begin{cases} \frac{1}{n!}(\frac{\lambda_i}{d_i\mu_i})^n P_0 & (n \leq c_{i1}) \\ \frac{1}{c_{i1}! \times {c_{i1}}^{n-c_{i1}}}(\frac{\lambda_i}{d_i\mu_i})^n P_0 & (c_{i1} < n) \end{cases} \tag{8}$$

Calculating Performance Metrics for *Queue$_i$*. Once we have obtained the steady-state probabilities, which are the basis of steady-state queuing service system, we can use them to construct analytical functions to calculate the important performance metrics concerning C_{i1} server station. As previously discussed, the performance metrics of one server station can be taken as the performance metrics of $queue_i$. Now we illustrate how to calculate the following five important performance metrics, which are the mean number of tasks queued in the buffer, the mean number of tasks in queuing system, mean response time, mean waiting time, and the probability of immediate service.

We use Lq_i to denote the expectations of the mean number of jobs queued in the buffer of $queue_i$, which is the waiting queue length. Therefore:

$$Lq_i = \sum_{n=c_{i1}+1}^{\infty} (n - c_{i1})P_n \tag{9}$$

We use Ls_i to denote the expectations of the mean number of all jobs currently in $queue_i$ (including both jobs being served in servers and jobs waiting in the buffer). Therefore:

$$Ls_i = \frac{\lambda_i}{d_i\mu_i} + Lq_i \tag{10}$$

We use Ws_i to denote mean response time of $queue_i$, which is the sum of jobs' service time and jobs' waiting time. We have discussed that the average number of all jobs in the system is Ls_i, thus according to Little's Law [8]:

$$Ws_i = \frac{Ls_i}{\lambda_i} \tag{11}$$

As is known, when a job arrives, if all servers are busy, the job will wait in the buffer until it can be served by an idle server. Thus, according to Little's Law, the average waiting time can be calculated as:

$$Wq_i = \frac{Lq_i}{\lambda_i} \tag{12}$$

At the time when a new job arrives, if there is no job waiting in the buffer and at least one server out of a station is idle, the new job will be served immediately.

Thus the probability of immediate service PI_i can be calculated as:

$$PI_i = \sum_{n=0}^{c_{i1}-1} P_n \tag{13}$$

3.2 Calculating the Performance Metrics for a Service Cloud

Once the performance metrics of $queue_i$ are obtained, we can use them to calculate the performance metrics concerning a service cloud, which are the mean of all jobs' performance value. Using Ws to denote mean response time of a service cloud and Q_i $(i \in \{1, \ldots, N\})$ to denote the number of jobs contained in $queue_i$. Mean response time of a service cloud can be calculated as:

$$Ws = \frac{\sum_{i=1}^{N} Q_i \cdot Ws_i}{\sum_{i=1}^{N} Q_i} \tag{14}$$

Using Wq to denote mean waiting time of a service cloud and Wq_i to denote mean waiting time of $queue_i$, mean waiting time of a service cloud can be calculated as:

$$Wq = \frac{\sum_{i=1}^{N} Q_i \cdot Wq_i}{\sum_{i=1}^{N} Q_i} \tag{15}$$

Using PI to denote the probability of immediate service of a service cloud and PI_i to denote the probability of immediate service of $queue_i$, the probability of immediate service of a service cloud can be calculated as:

$$PI = \frac{\sum_{i=1}^{N} Q_i \cdot PI_i}{\sum_{i=1}^{N} Q_i} \tag{16}$$

Based on the analytical results, we can determine proper DOP settings and resource allocation schemes to achieve the optimal mean response time, mean waiting time and the probability of immediate service, and thus to improve the QoS of service clouds.

4 The Proposed Performance Metric

However, when optimal performance metrics are pursued, such as mean response time or mean waiting time of service clouds, the level of performance optimization of some groups may be ignored. This will lead to an unfair trade-off between different groups. Therefore, we propose a new metric, called Optimized Performance of Groups (OPG), which is the ratio between the response time a group of jobs has spent in a service cloud without differentiated DOP policy and the response time this group would have spent in that service cloud with differentiated DOP policy. Using Wsp_i to denote the response time that $group_i$ has spent in a service cloud without differentiated DOP policy and Wsd_i to denote the response time that $group_i$ of the jobs would have spent in a service cloud with differentiated DOP policy. Hence, OPG_i can be formulated as follows:

$$OPG_i = \frac{Wsp_i}{Wsd_i} \tag{17}$$

The metric OPG_i is used to evaluate the level of performance optimization of $group_i$. OPG_i is within the range of $(0, +\infty)$, and the lower it is, the worse level of performance optimization the $group_i$ has. If the minimum OPG of all groups is less than a certain value which is a lower bound of the level of performance optimization, we need to improve the OPG of the group which has the minimum OPG of all groups by determining a higher DOP or allocating more servers to this group. Our final objective is to maximize the minimum OPG_i to ensure OPG_i within a certain range, thereby enforcing a fair trade-off between all groups when mean response time of all jobs are optimized.

5 Experimental Evaluation

In order to evaluate the effectiveness of our proposed differentiated DOP policy and analytical model, we have built a discrete event simulator of the cloud server system and conducted extensive experiments based on synthetic data extracted from real-world rendering jobs.

5.1 Simulation Methodology and Parameter Settings

There are four main purposes of our experimental simulations. First, the effectiveness of our proposed differentiated DOP policy is validated by comparing it to the performance of a service cloud without the policy. Second, the effectiveness of our proposed analytical model is validated by investigating whether the performance metrics predicted by the analytical model are close to the simulation results. Third, we examine how the parameter settings, such as different combinations of DOP settings or resource allocation schemes, can improve the QoS of a service cloud. Finally, we investigate whether the minimum OPG of all experiments is higher than the lower bound of the level of performance optimization and if not, we had better maximize the minimum OPG_i of that experiment by increasing the DOP or allocating more servers to $group_i$ to enforce a fair trade-off between all groups. For these aims, we built a simulator, in which we can change the number of servers, resource allocation schemes and DOPs.

In all of the simulations, the number of jobs used in each simulation is above 10000, and the testing time is more than 168 h (7 days). The service time of all jobs is collected from our working rendering service platform, which is within the range of [70 min, 130 min). Jobs are assigned into three groups identified by jobs' service time. Jobs' service time of $group_1$ is within the range of [70 min, 90 min) and mean service time is 80 min. Jobs' service time of $group_2$ is within the range of [90 min, 110 min) and mean service time is 100 min. Jobs' service time of $group_3$ is within the range of [110 min, 130 min] and mean service time is 120 min.

The jobs' service time of all groups is approximate exponentially distributed. Considering the characteristics of real-world rendering job arrivals, we assume that job arrivals of all groups follow an approximate Poisson process and the mean job interarrival time of $group_1$, $group_2$ and $group_3$ are 1.5 min, 1.4 min and 1.35 min respectively. Then, we could set a DOP for each group. Besides, we assume that the lower bound of the level of performance optimization is 0.9. Jobs in service clouds won't be lost as buffer size without limit. These settings are in reasonable sizes considering the real-world rendering service cloud and they are kept the same for all of the simulations. For other parameters concerning each simulation, we will explain them in the respective subsections.

5.2 Effect of Resource Allocation on Performance

Table 1. Resource allocation schemes

Experiment number	NoS in $group_1$	NoS in $group_2$	NoS in $group_3$	All servers
Experiment 1	58	76	106	240
Experiment 2	58	84	98	240
Experiment 3	62	80	98	240
Experiment 4	62	84	94	240
Experiment 5	66	76	98	240
Experiment 6	66	80	94	240

In this subsection, we investigate the influence of different resource allocation schemes on a service cloud's performance. Cloud providers need to determine how to allocate servers to serve different groups to achieve an optimal performance of a service cloud. For this aim, we randomly select six experiments, which have different resource allocation schemes for three groups, and investigate which scheme has better mean response time, mean waiting time or OPG. Experiment settings are introduced in Table 1. Besides, we use 'NoS' to denote the number of servers.

Response time, waiting time and the minimum OPG are shown in Table 2. It shows that the resulting mean response time and mean waiting time of all jobs are different with the change of resource allocation schemes. A service cloud would get an optimal response time and waiting time when 62 servers are allocated to serve $group_1$, 80 servers are allocated to serve $group_2$ and 98 servers are allocated to serve $group_3$. We find that compared with other experiments, groups in experiment 3 don't have higher traffic intensity. We could observe that allocating fewer servers to serve the group in which jobs require shorter service time would lead to a poor result, as the change of NoS has a greater influence in traffic intensity of that group.

Table 2 also shows that the simulation results are less than the analytical results when the traffic intensity is bigger. This is mainly because the following

Table 2. Effect of resource allocation on performance

Experiments	Method	Response time	Waiting time	The minimum OPG
1	Simulation	55.41	5.25	0.87
	Analytical	57.36	7.35	
2	Simulation	53.54	3.37	0.87
	Analytical	55.50	5.50	
3	Simulation	52.50	2.32	0.96
	Analytical	54.05	4.05	
4	Simulation	54.35	4.18	0.91
	Analytical	56.29	6.29	
5	Simulation	53.22	3.06	0.91
	Analytical	55.91	5.91	
6	Simulation	53.30	3.14	0.91
	Analytical	56.45	6.45	

two reasons. First, queuing theory has assumed that the jobs' service time is within the range of $[0, +\infty)$. In reality, the jobs' service time of one group is within the certain range, e.g., jobs' service time of $group_1$ is within the range of [70 min, 90 min). Obviously, the jobs in each group don't contain the jobs with a service time that exceeds the upper limit or that is under the lower limit, which has a significant impact on performance. Second, as the service time of all jobs is collected from real-world working rendering service platform, jobs' service time in each group is not completely following exponential distribution. So analytical results and simulation results show little difference, which is less than 10 min. More importantly, our analytical model can accurately indicate the trend of performance change and its results can supply approximate lower bounds in all situations. Thus, we can conclude that the analytical model could help us to determine proper resource allocation schemes to achieve optimal QoS of service clouds.

The resulting minimum OPG of all experiments is also detailed in Table 2. OPG of experiments 1 and 2 are less than 0.9. In this situation, we had better allocate more servers to the group which has the minimum OPG to ensure a fair trade-off between all groups.

5.3 Effect of DOP on Performance

In this subsection, we investigate the influence of different combinations of DOP settings on response time and the immediate service rate. We also investigate the minimum OPG of every experiment to ensure a fair trade-off between all groups. 240 servers contained in a service cloud are allocated to three groups. Considering the same traffic intensity, we allocate 60 servers to serve $group_1$, 80 servers to serve $group_2$ and 100 servers to serve $group_3$. In all cases, the traffic intensity is reasonably set as $\rho = 0.88$. We then select 8 experiments, that contain different combinations of DOP settings. Experiment settings are introduced in Table 3. Experiment 1 and 5 are comparative experiments which won't use our differentiated DOP policy. Then we compare the performance metrics of experiment 2, 3, and 4 with experiment 1. We also compare the performance metrics of experiment 6, 7, and 8 with experiment 5.

Table 3. Experiment settings with varying DOPs and their minimum OPG

Experiments	DOP of $group_1$	DOP of $group_2$	DOP of $group_3$	The minimum OPG
1	2	2	2	
2	2	2	5	0.93
3	2	5	2	0.93
4	5	2	2	0.97
5	4	4	4	
6	2	4	10	0.47
7	4	4	10	0.85
8	4	5	10	0.85

The minimum OPG of all experiments is shown in Table 3. The OPG of experiment 6 is 0.47, which means at least one group' performance is seriously worse than before. Thus, even if mean response time of all jobs is better than before, this parameter setting is not recommended. In this situation, we had better set a higher DOP to the group, which has the minimum OPG in all groups, to ensure a fair trade-off between all groups.

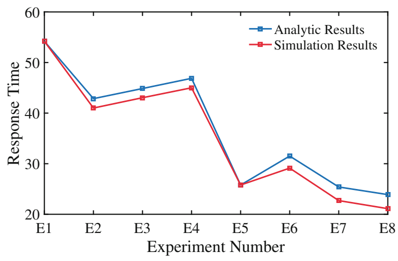


Fig. 3. Mean response time

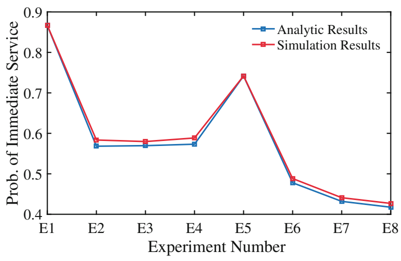


Fig. 4. Immediate service rate

The resulting response time is shown in Fig. 3. It can be noted that in contrast to processing all jobs with the same DOP, dividing jobs into different groups and processing them with varying DOPs has diverse service response time. As different groups of jobs have different influences on performance, we find that setting a higher DOP to a group that requires longer service time could achieve optimal performance, which proves the validity of our differentiated DOP policy.

Comparing with experiment 5, experiment 6 has poor performance results which include mean response time and the minimum OPG. Thus, we need to set a higher DOP for the group that has the minimum OPG of all groups to optimize mean response time, as shown in experiment 7 or 8. Therefore, analytical model could be helpful in setting DOPs for different groups to achieve optimal performance.

The probability of immediate service is shown in Fig. 4. In contrast to treating all jobs with the same DOP, treating jobs with varying DOPs produces an inferior immediate service rate. With the increasing of DOP, immediate service rate of all jobs decrease rapidly. Thus, when pursuing optimal immediate service rate, DOP should be limited.

Figures 3 and 4 also show that analysis results given by our proposed analytical model are close to simulation results and the results can supply approximate lower bounds in all situations. Thus, the analytical functions are valid for estimating response time and the probability of immediate service. Finally, we can conclude that the analytical model could help us to determine proper DOPs for different group to achieve optimal QoS of service clouds.

6 Related Work

With the rapid development of cloud computing technologies, there are more and more researchers focusing on optimizing QoS of a cloud center by establishing analytical models. Performance analysis generally focuses on evaluating a system's key performance metrics such as response time, throughput, and so on [18].

In [9,20], the cloud center is modeled as a queuing system with single task arrival and finite buffer capacity. For evaluating the performance of cloud computing, these analytical models obtain accurate estimations of the complete probability distribution of response time and other important performance indicators. Based on these accurate estimations, cloud providers could determine proper buffer space for different classes of tasks to avoid sudden long delay.

[11] and [10] propose analytical performance models that address the complexity of cloud centers. It has been assumed that a cloud center has a number of servers and each server has been configured as a number of virtual machines (VMs). They obtain a detailed assessment of cloud center performance. Several performance metrics are defined and evaluated to analyze the behavior of a cloud data center: utilization, availability, waiting time, and responsiveness [1]. A resiliency analysis is also provided to take into account load bursts. However, these analytical models cannot work well with parallel jobs.

In [12], the analysis model has been extended to a system where jobs consist of multiple tasks that have general independent service time distributions. The model could account for the deterioration of performance due to the workload at each node. In [2], the authors propose a mathematical model to predict the performance of parallel jobs based on a queuing system. This model has a good accuracy in predicting execution time and waiting time. In [19], the researchers focuses on the jobs that may consist of multiple tasks with each task requiring a VM for its execution. It has derived job blocking probabilities and distribution of the utilization of resources as a function of the traffic load under various scenarios for systems with both homogenous and heterogeneous VMs. In a previous study, we have proposed an approximate analytical model for cloud computing centers serving parallelizable jobs using M/M/c/r queuing systems [14]. This model has a good accuracy in predicting metrics of parallelizable jobs with the same DOP. The above models concern parallel jobs, but they didn't consider the problem of differentiated processing of jobs.

As discussed above, current performance analysis models cannot deal with the situations of service clouds serving parallel jobs with varying DOPs. Our approach can establish an analytical model for jobs with varying DOPs. Using this model, we can obtain an accurate estimation of the complete probability distribution of response time, waiting time and probability of immediate service, which could guide cloud providers to determine optimal DOPs and resource allocation schemes to achieve specific QoS of service clouds.

7 Conclusions and Future Work

Dividing jobs into different groups and processing them with varying DOPs could achieve better performance results, especially focusing on jobs that have a greater impact on performance of service clouds. In this paper, for a service cloud that serves pleasingly parallel jobs, we propose a differentiated DOP policy, which divides jobs into several groups identified by jobs' service time and sets proper DOPs for different groups of jobs. Then, we propose a parallel multi-queue and multi-station analytical model for a service cloud with our proposed policy to predict performance metrics. This model could help us make optimized decisions in determining DOPs and resource allocation schemes for different groups of jobs, and thus to improve the QoS of a service cloud. Besides, we present a metric named OPG, which quantifies the level of performance optimization of every group. The objective is to maximize the minimum OPG, thereby enforcing a fair trade-off between all groups. Through extensive experiments based on synthetic data extracted from real-world rendering jobs, we validate the effectiveness of our differentiated DOP policy and analytical model.

For the future work, we plan to extend our research to QoS optimization of service clouds which contain composite parallelizable services. Besides, we also plan to further validate our approach by collecting workload traces from more real-world cloud systems.

Acknowledgments. The authors would like to acknowledge the support provided by the National Key Research and Development Program of China (2017YF A0700601, 2018YFB1003800), the Key Research and Development Program of Shandong Province (2017CXGC0605, 2017CXGC0604, 2018GGX101019, 2016GGX106001, 2016GGX101008, 2016ZDJS01A09), the Natural Science Foundation of Shandong Province for Major Basic Research Projects (No. ZR2017ZB0419), the Young Scholars Program of Shandong University, and the TaiShan Industrial Experts Program of Shandong Province (tscy20150305).

References

1. Bruneo, D.: A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems. IEEE Trans. Parallel Distrib. Syst. (TPDS 2014) **25**(3), 560–569 (2014)
2. Chao, S., et al.: Predicting the performance of parallel computing models using queuing system. In: Proceedings of International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015), pp. 757–760 (2015)
3. Cordeiro, T.D., et al.: Open source cloud computing platforms. In: Proceedings of International Conference on Grid and Cooperative Computing (GCC 2010), Nanjing, China, pp. 366–371 (2010)
4. Dazhong, W., et al.: Cloud-based design and manufacturing: a new paradigm in digital manufacturing and design innovation. Comput.-Aided Des. **59**, 1–4 (2015)
5. Feng, G., Buyya, R.: Maximum revenue-oriented resource allocation in cloud. Int. J. Grid Util. Comput. **7**(1), 12–21 (2016)
6. Gunarathne, T., et al.: Cloud computing paradigms for pleasingly parallel biomedical applications. In: Proceedings of ACM International Symposium on High Performance Distributed Computing (HPDC 2010), Chicago, Illinois, pp. 460–469 (2010)
7. Haghighi, A.M., Mishev, D.P.: A parallel priority queueing system with finite buffers. J. Parallel Distrib. Comput. **66**(3), 379–392 (2006)
8. Keilson, J., Servi, L.D.: A distributional form of little's law. Oper. Res. Lett. **7**(5), 223–227 (1988)
9. Khazaei, H., et al.: Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. IEEE Trans. Parallel Distrib. Syst. (TPDS 2012) **23**(5), 936–943 (2012)
10. Khazaei, H., et al.: Analysis of a pool management scheme for cloud computing centers. IEEE Trans. Parallel Distrib. Syst. (TPDS 2013) **24**(5), 849–861 (2013)
11. Khazaei, H., et al.: A fine-grained performance model of cloud computing centers. IEEE Trans. Parallel Distrib. Syst. (TPDS 2013) **24**(11), 2138–2147 (2013)
12. Khazaei, H., et al.: Performance of cloud centers with high degree of virtualization under batch task arrivals. IEEE Trans. Parallel Distrib. Syst. (TPDS 2013) **24**(12), 2429–2438 (2013)
13. Li, S., et al.: Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud. IEEE Trans. Parallel Distrib. Syst. (TPDS 2017) **28**(6), 1607–1620 (2017)
14. Li, X., et al.: Performance analysis of cloud computing centers serving parallelizable rendering jobs using m/m/c/r queuing systems. In: Proceedings of International Conference on Distributed Computing Systems (ICDCS 2017), Atlanta, GA, USA, pp. 1378–1388 (2017)
15. Liu, C., Shang, Y., Duan, L., Chen, S., Liu, C., Chen, J.: Optimizing workload category for adaptive workload prediction in service clouds. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 87–104. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_6

16. Mai, X.T., et al.: Policy-aware optimization of parallel execution of composite services. In: Proceedings of International Conference on Services Computing (SCC 2015), New York, USA, pp. 106–113 (2015)
17. Pan, L., An, B., et al.: Nash equilibrium and decentralized pricing for qos aware service composition in cloud computing environments. In: Proceedings of International Conference on Web Services (ICWS 2017), Honolulu, HI, USA, pp. 154–163 (2017)
18. Seneviratne, S., Levy, D., Buyya, R.: A taxonomy of performance prediction systems in the parallel and distributed computing grids. Eprint Arxiv arXiv:1307.2380 (2013)
19. Vakilinia, S.: Modeling of the resource allocation in cloud computing centers. Int. J. Comput. Telecommun. Netw. **91**, 453–470 (2015)
20. Yang, B., Tan, F., Dai, Y.-S., Guo, S.: Performance evaluation of cloud service considering fault recovery. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 571–576. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10665-1_54

Estimating the Performance of Cloud-Based Systems Using Benchmarking and Simulation in a Complementary Manner

Haan Johng[1](✉), Doohwan Kim[1], Tom Hill[2], and Lawrence Chung[1]

[1] University of Texas at Dallas, Richardson, TX 75080, USA
{haanmo.johng,doohwan.kim,chung}@utdallas.edu
[2] Fellows Consulting Group, LLC, Dallas, TX, USA
tom@fellowsconsultinggroup.com

Abstract. Estimating future runtime performance and cost is an essential task for Chief Information Officers in deciding whether to adopt a Cloud-based system. Benchmarking and simulation are two techniques that have long been practiced towards reliable estimation. Benchmarking involves (potentially) high cost and time consumption, but oftentimes yields more reliable estimates than simulation, while the simulation is much cheaper and faster than benchmarking, but less reliable. In order to deal with this dichotomy, we propose a complementary approach to estimating the performance of Cloud-based systems, whereby performance estimates can be obtained in a fast, inexpensive, and also reliable way. In this approach, the ontological concepts of a benchmark model, whose benchmark results have already been obtained, are mapped into those of a simulation model, while the mismatches and similarities between the two models are taken care of, through measures of similarity between the two. This ontology-driven construction of simulation models is intended not only to yield more reliable simulation results but also to help better explain why the simulation results may, or may not, be reliable. To validate our complementary approach, simulation models are constructed using CloudSim, and the simulation results are compared against the corresponding benchmark results, by using our prototype tool, collected from Amazon Web Service (AWS) and Google Compute Engine (GCE) by using the Yahoo! Cloud Serving Benchmark (YCSB) tool. These experiments show that the simulation results show about 90% accuracy with respect to the benchmark results, and additionally we feel we could better explain why this happens.

Keywords: Cloud computing · Benchmark · Simulation
Goal-orientation · Complementary approach

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 576–591, 2018.
https://doi.org/10.1007/978-3-030-03596-9_42

1 Introduction

Cloud computing is being increasingly adopted in various real-life software systems, thanks to its potential benefits, such as cost reduction, improved performance, elasticity, and scalability - the so-called non-functional characteristics. In deciding whether to adopt Cloud computing in the early design phase, estimating such non-functional characteristics in a fast, inexpensive, and reliable manner is an essential task for Chief Information Officers (CIOs) with limited time constraints and budgets, especially when initial requirements are ambiguous, incomplete, inconsistent and changing [1–5].

Cloud benchmarking and simulation are two techniques long practiced for performance estimation. Although Cloud benchmarking typically yields reliable estimates, it is potentially expensive and time-consuming to use Cloud resources to process benchmark transactions. The nature of requirements in the early design phase (e.g., the unclear and changing number of concurrent users) increases the time and cost of benchmarking various architectural designs in Clouds [6–9]. On the other hands, Cloud simulation, which is faster and cheaper than benchmarking, has a difficulty of ensuring reliability and understandability of the simulation results, because of the reduced fidelity of the simulation model. For example, when representing the CPU clock speed, Cloud simulations tend to use Million Instructions Per Second (MIPS), while real-world Clouds use Gigahertz (GHz) [10–12].

To deal with this dichotomy, we propose a complementary approach, using both benchmarking and simulation, to estimating the performance of Cloud-based systems, whereby performance estimates can be obtained in a fast, inexpensive, and also reliable manner. Our approach is complementary in the sense that benchmarks enhance reliability and understandability of the simulation model, and simulation results reduce the cost and time for benchmarking Clouds by narrowing down the architecture design space. In our approach, the ontological concepts of a benchmark model, whose benchmark results have already been collected, are mapped into those of a simulation model, while similarities between the two models are considered and highlighted to help better explain *why* the simulation results may, or may not, be reliable. This work draws on our previous work [9,13,14], which used benchmarking *neither* for guiding the development *nor* estimating the reliability of simulation models, but for simply comparing the results of simulation against its corresponding benchmarking results, even this only if available.

For validating our complementary approach, simulation models are constructed by using CloudSim, and the simulation results are compared against the corresponding benchmark results obtained from benchmarking Amazon Web Service (AWS) and Google Compute Engine (GCE), by using the Yahoo! Cloud Serving Benchmark (YCSB) tool and a Cassandra NoSQL Database. In our experiments, the simulation results show about 90% accuracy regarding the benchmark results. Additionally, we also feel we could better explain why or why not our simulation results are reliable, and our approach helps evaluate and validate Cloud design alternatives in the early design stages.

In the rest of this paper, Sect. 2 provides related work, and Sect. 3 presents our proposed approach. Section 4 describes our experiments, followed by observations and discussions in Sect. 5. In the end, a summary of contributions and future directions are described.

2 Related Work

The key distinctive of our work lies in the use of a complementary approach, using benchmarks and simulations together, in estimating the performance of Cloud-based systems in a fast, inexpensive, and also reliable manner. More specifically, our work proposes a five-step process for capturing the ontologies of estimation, benchmark, and simulation, and mapping the benchmark ontologies to simulation ontologies, while considering the similarities, and mismatches, between the two. Our work also provides a prototype tool for our semi-automated approach.

While there seems to be little or no other work that proposes a complementary approach to estimating the performance of Cloud-based systems in a fast, inexpensive, and also reliable manner, some parts of our work share some similarity with some previous work, although they deal with only one, without regard to the other.

Concerning benchmarking, Cloud benchmark studies show the benchmark-based approach is expensive and time-consuming, leading to unsatisfaction for the CIOs with low budgets and restrictive time constraints [6–8]. Moreover, a case study [15] described difficulties in finding similar Cloud benchmarks to the design alternatives. Although these studies addressed the problems in the benchmark-based approach, there seems to be a lack of studies on solutions alleviating the issues - this further motivated our work in this paper.

Concerning simulation, simulation-based approaches to cloud computing have challenges, when ensuring that simulation results are reliable and understandably so. There seems to be a lack of papers describing how to construct cloud simulation models that understandably ensure the reliability of the model. For example, the MIPS values used in the simulation-based approaches widely ranged from as small as 250 to as high as 20,000 [10–12, 16].

3 Our Framework for a Complementary Approach

Our approach, using benchmarks and simulations in a complementary manner, consists of a five-step process and a prototype tool. The five-step process starts with capturing the ontology of applications, benchmarks, and simulations and ends with a design that outputs reliable and understandable performance estimates. Additionally, the similarity and mapping are taken care of among the ontologies; simulation models are constructed by using the ontological relationships and are used to confirm and reconfirm the quality of the design alternatives. The tool supports capturing and mapping the ontologies, measuring similarities, and running simulation models.

The Gane-Sarson DFD of the five steps is shown in Fig. 1. Each step is envisioned for use in an interactive, iterative, and interleaving manner rather than in a strictly sequential manner and enhance the outcomes of each step by utilizing new information obtained from the later steps.

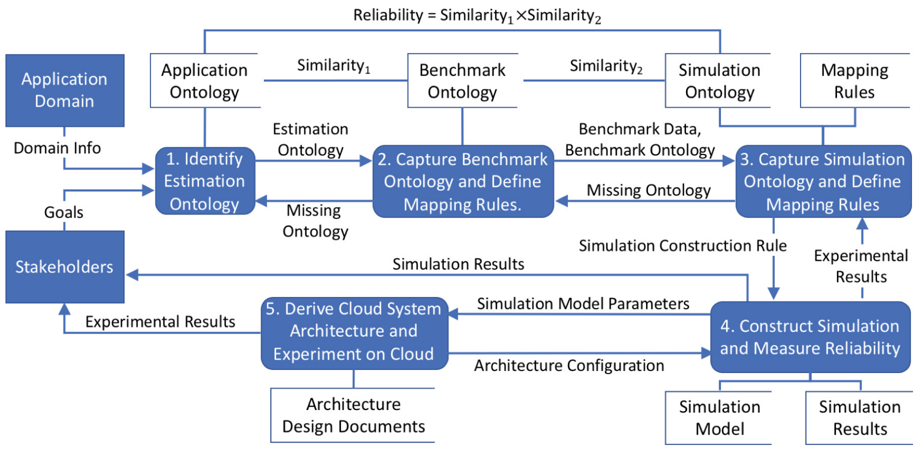

Reliability = Similarity$_1$ ×Similarity$_2$
Application Domain
Application Ontology
Similarity$_1$
Benchmark Ontology
Similarity$_2$
Simulation Ontology
Mapping Rules
Domain Info
Estimation Ontology
Benchmark Data, Benchmark Ontology
1. Identify Estimation Ontology
2. Capture Benchmark Ontology and Define Mapping Rules.
3. Capture Simulation Ontology and Define Mapping Rules
Missing Ontology
Missing Ontology
Goals
Stakeholders
Simulation Construction Rule
Experimental Results
Simulation Results
Simulation Model Parameters
5. Derive Cloud System Architecture and Experiment on Cloud
4. Construct Simulation and Measure Reliability
Experimental Results
Architecture Configuration
Architecture Design Documents
Simulation Model
Simulation Results

Fig. 1. A 5-step process for estimating the performance of Cloud-based system

3.1 Step 1: Identifying Estimation Ontology

Ontology, which is a set of essential concepts and interrelationships among them, plays a key role in successfully designing a domain by explicitly illustrating whether essential concepts are involved in the design or omitted. In our context, we identify the ontology of estimating the performance of Cloud-based systems, which means capturing essential system concepts for the estimations. Besides the ontology of the estimation, our approach takes account of softgoals of the estimation, such as cheap, fast, and reliable estimation; for example, the ontology of reliable estimation, the ontology of fast estimation, and the ontology of inexpensive estimation.

Estimation goals and ontology are extracted from our previous work and other literature [9,14,17]. Later, they are represented in a goal model using Softgoal Interdependency Graph (SIG). The SIG provides a useful framework for various domains to carry out the reasoning for deriving the goals, ontology, and interdependency among them [1,18,19]. The aim of the goal model is not meant to construct the completely generalized goal model from first principles. Rather, as initial research of its kind, the goal model is intended to act as an example to describe how goal models work in our approach and potentially as a reference goal model. If a reference goal model needs to be brought up in future works, it is required to be refined and customized because every domain has different stakeholders, ontologies, goals, and priorities.

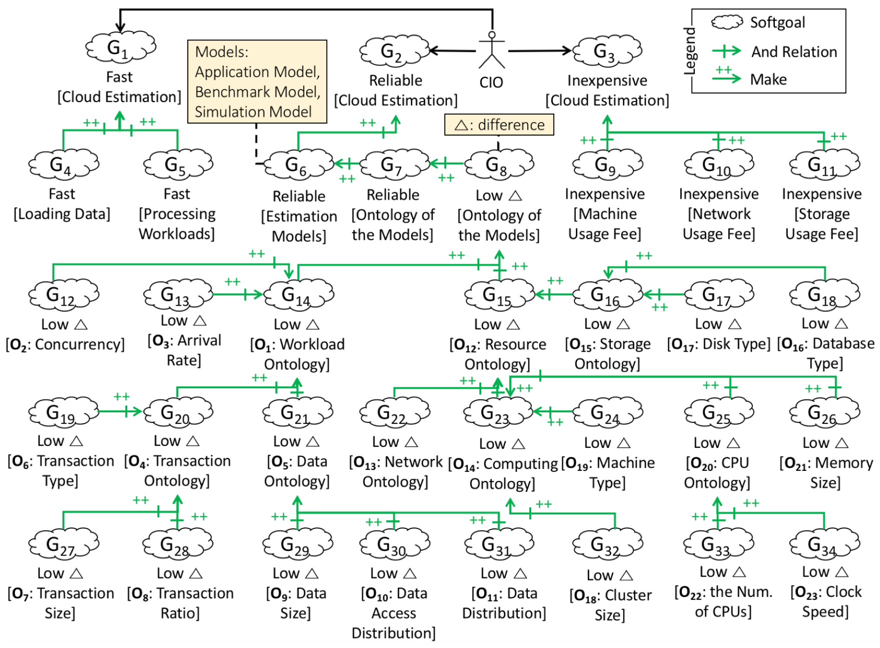

G1 Fast [Cloud Estimation]
Models: Application Model, Benchmark Model, Simulation Model
G2 Reliable [Cloud Estimation]
CIO
G3 Inexpensive [Cloud Estimation]
Legend
Softgoal
And Relation
Make
△: difference
G4 Fast [Loading Data]
G5 Fast [Processing Workloads]
G6 Reliable [Estimation Models]
G7 Reliable [Ontology of the Models]
G8 Low △ [Ontology of the Models]
G9 Inexpensive [Machine Usage Fee]
G10 Inexpensive [Network Usage Fee]
G11 Inexpensive [Storage Usage Fee]
G12 Low △ [O2: Concurrency]
G13 Low △ [O3: Arrival Rate]
G14 Low △ [O1: Workload Ontology]
G15 Low △ [O12: Resource Ontology]
G16 Low △ [O15: Storage Ontology]
G17 Low △ [O17: Disk Type]
G18 Low △ [O16: Database Type]
G19 Low △ [O6: Transaction Type]
G20 Low △ [O4: Transaction Ontology]
G21 Low △ [O5: Data Ontology]
G22 Low △ [O13: Network Ontology]
G23 Low △ [O14: Computing Ontology]
G24 Low △ [O19: Machine Type]
G25 Low △ [O20: CPU Ontology]
G26 Low △ [O21: Memory Size]
G27 Low △ [O7: Transaction Size]
G28 Low △ [O8: Transaction Ratio]
G29 Low △ [O9: Data Size]
G30 Low △ [O10: Data Access Distribution]
G31 Low △ [O11: Data Distribution]
G32 Low △ [O18: Cluster Size]
G33 Low △ [O22: the Num. of CPUs]
G34 Low △ [O23: Clock Speed]

Fig. 2. Estimating ontology using softgoal interdependency graph

Figure 2 illustrates a goal model using SIG. The SIG represents a softgoal in the form of *type[topic]*. The type part represents the softgoal type, and the topic part describes the ontology of the softgoal, such as inexpensive[Cloud Estimation]. Each softgoal is labeled as $\{G_i | 1 \le i \le 34\}$. To satisfy a softgoal, we decompose the softgoal into sub-softgoals. For example, inexpensive[Cloud Estimation] (G_3) is decomposed into inexpensive[Machine Usage Fee] (G_9), inexpensive[Network Usage Fee] (G_{10}), and inexpensive[Storage Usage Fee] (G_{11}). If the three sub-softgoals are satisfied, then the inexpensive[Cloud Estimation] (G_3) can be satisfied. Likewise, to satisfy fast[Cloud Estimation] (G_1), Fast[Data Loading] (G_4) and Fast[Workload Processing] (G_6) need to be satisfied.

In order for Cloud estimation to be reliable (G_2), the models of estimation techniques, such as the estimation model, benchmark model, and simulation model, should be reliable (G_6). To ensure the reliability of the estimation models, the ontologies of each model must be reliable (G_7), which means each model should sufficiently include essential system concepts. Furthermore, because our approach utilizes the three complementary models, the ontological differences between them need to be low (G_8). For example, the benchmark ontologies need to cover the estimation ontologies. Specifically, as the ontologies of the models can be classified into workload ontology and resource ontology [17], there should be low differences in workload ontology (G_{14}) and resource ontology (G_{15}). For example, a benchmark should have similar workload ontology to an estimation workload ontology. The ontology of estimation model is labeled as

$\{O_i|1 \leq i \leq 23\}$. For example, workload ontology is labeled as O_1, and resource ontology is shown as O_{12}. The workload ontology O_1 is represented as $O_1 = \{O_2,O_3,\{O_4\},\{O_5\}\}$, where $O_4 = \{O_6,O_7,O_8\}$ and $O_5 = \{O_9,O_{10},O_{11}\}$. Likewise, the resource ontology O_{12} is decomposed as $O_{12} = \{O_{13},\{O_{14}\},\{O_{15}\}\}$, where $O_{15} = \{O_{16},O_{17}\}$, $O_{14} = \{O_{18},O_{19},\{O_{20}\},O_{21}\}$, and $O_{20} = \{O_{22},O_{23}\}$.

3.2 Step 2: Capturing Benchmark Ontology and Defining Mapping Rules

The goal model with estimation ontology, constructed at the previous step, is used in an interactive, iterative and interleaving manner as a guide to provide insights into what benchmark ontologies to capture, which are depicted in Fig. 3 using the Class diagram. In the backward direction, while capturing the benchmark ontologies, the ontologies unrecognized in the previous step can be captured, which results in an enhanced estimation ontology in the goal model.

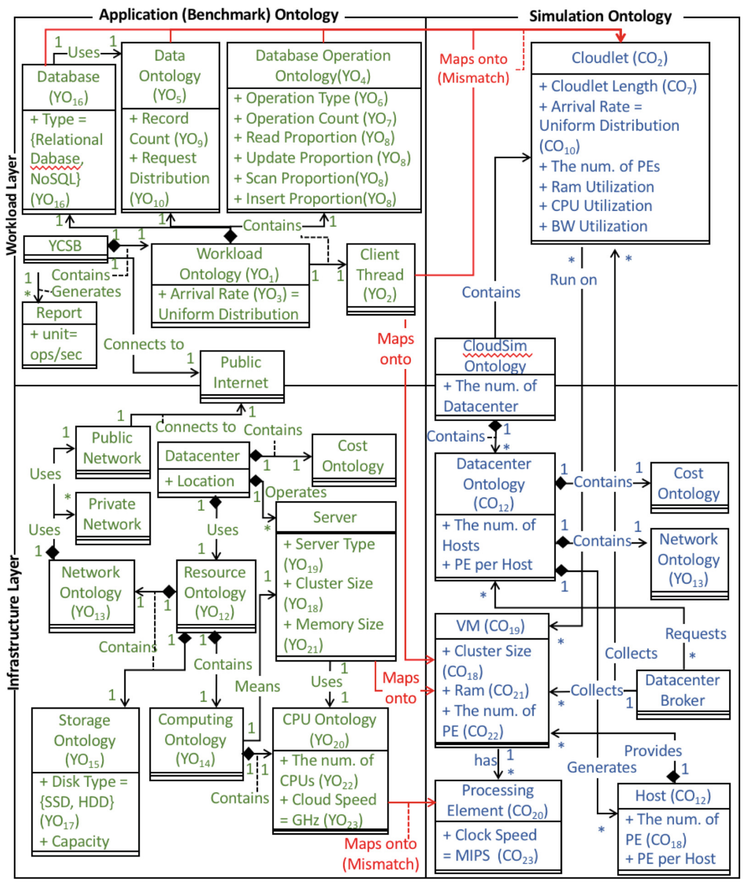


Fig. 3. Benchmark and simulation ontology (Color figure online)

For example, while capturing the benchmark ontologies that corresponds to the estimation ontology from [15,20], data size, data access distribution, and data distribution were captured as critical factors that impact the performance; these were not explicitly identified in the estimation ontology extracted from previous studies [14,17,21]. Then, the goal model is enhanced with the new ontology (i.e., the low difference[O_5: Data Ontology]) and the sub-ontologies $O_{9,10,11}$.

This study captures the Yahoo! Cloud Serving Benchmark (YCSB) ontology and the CloudSim ontology because the YCSB benchmark data collected from benchmarking Amazon Web Service (AWS) and Google Compute Engine (GCE) are utilized to construct the simulation models using Cloudsim. The ontologies are clustered into workload ontology and resource ontology, and the interrelationships among ontologies are depicted in the left part of Fig. 3.

The estimation ontologies are heuristically mapped onto the those of YCSB with the same semantics, and the mapping is expressed as below. The O_{11} is not mapped onto the YCSB ontology as YCSB does not provide the data distribution functionality.

- O_i: entities of the estimation ontology O, with index $\{1 \leq i \leq 23\}$.
- YO_i: entities of the YCSB ontology YO.
- $smap_1$:$O_i \rightarrow YO_i$, a semantic mapping function, entity index $i \in \mathbb{N}$. The estimation ontology O_i is mapped onto YO_i if the entities are semantically identical.
- $smap_1(O_i) = YO_i$, where $\{O_i, YO_i | 1 \leq i \leq 23 \; and \; i \neq 11\}$.

3.3 Step 3: Capturing Simulation Ontology and Defining Mapping Rules

Since the simulation ontologies have distance from the real-world Cloud ontologies, the Cloud benchmark ontologies, captured in the previous step, are mapped onto the simulation ontologies to complement the semantics of the simulation ontologies, which results in enhancing the reliability and understandability in the simulation models. We have captured the CloudSim ontology by adopting and refining the simulation model in our previous work [14], and the simulation ontology is illustrated on the right side of Fig. 3 in blue.

The technical difficulty in the mapping is ontological mismatches between the benchmark and simulation. The semantics of the three simulation ontologies do not directly match with the benchmark ontologies: the number of Cloudlets, the length of each Cloudlet, and the MIPS value of the CPUs. According to [16], Cloudlets are Cloud-based application services, where Cloudlet length is a pre-defined instruction length and data transfer overhead. However, it is difficult to precisely define what "Cloud-based application service" is and what "pre-defined instruction length" means regarding Cloud benchmarks.

For mapping ontologies, we define a mapping rule that maps semantics of benchmark ontologies and the values thereof onto simulation ontologies. The goal model in Fig. 2 acts as a guide along the same lines as the step 2. The ontologies that are semantically identical (e.g., memory size (YO_{21})) are directly mapped onto simulation ontologies (e.g., ram size(CO_{21})).

For the three mismatched ontologies, we define the number of Cloudlets as the number of concurrent users (O_2) and the Cloudlet length as the number of transactions (O_7) that each user generates. Since we defined the rules, $smap_1(O_2) = YO_2$ and $smap_1(O_7) = YO_7$, the number of client threads in each benchmark data is assigned to the number of Cloudlets, and the operation count that each client thread generates is assigned to the Cloudlet length. This is because the benchmark settings for the number of client threads and the operation count are known values, but the instruction length is not. Converting benchmark transactions to the instruction length is a non-trivial task because of difficulties in obtaining the internal information for the conversion from the Cloud providers, such as the compiler version, hypervisor version, optimization options, or VM deployment policies.

The mapping is illustrated in the Fig. 3 with red lines and can be expressed as below.

- CO_i: entities of the CloudSim ontology CO
- $smap_2$:$YO_i \rightarrow CO_i$, a semantic mapping function, entity index $i \in \mathbb{N}$. The estimation ontology YO_i is mapped onto CO_i if the entities are semantically identical
- $vmap$:$YO_i \rightarrow CO_i$, a value mapping function, entity index $i \in \mathbb{N}$. A value of YO_i is mapped onto a configuration CO_i
- $smap_2(YO_i) = CO_i$, where, $\{YO_i, CO_i | i = \{12, 13, 14, 18, 19, 21, 22, 23\}\}$
- $smap_2(YO_7) = CO_7$
- $smap_2(YO_i) = CO_2$, where $\{YO_i | 1 \leq i \leq 10 \text{ and } i = \{16, 20\}\}$
- $vmap(YO_2) = CO_2$
- $vmap(YO_i) = CO_i$ $\{YO_i | i = 7, 18, 21, 22, 23\}$.

The mapping rules aim to show an example for describing how to map benchmark ontologies to simulation ontologies for building reliable and understandable simulation models. The more benchmark data and mapping rules are developed, the more reliable simulation models for diverse domains can be constructed. Each mapping rule can have a weight, representing how much the mapping is satisfiable, and the maximum weight value is configurable in a fuzzy way. We assumed that every mapping rule has the maximum weight of five.

3.4 Step 4: Constructing Simulations and Measuring Reliability

After assigning the semantics and values of benchmark ontologies to simulation ontologies, except for MIPS, we drive the MIPS value to run the simulations, by utilizing benchmark data, as described in Algorithm 1. Algorithm 1 takes the inputs, such as machine type (*type_in*), the number of concurrent users (*threads_in*), the number of servers (*servers_in*), and the number of operations (*operations_in*), and returns a simulation throughput (t). One assumption is that n number of benchmark data is collected. The variables, which are *type*[*i*], *threads*[*i*], *servers*[*i*], and *operations*[*i*], contain the machine type, the number of client threads, the number of servers, and the number of operations of i^{th} benchmark data respectively. The throughput of the i^{th} benchmark is saved in $th[i]$.

Algorithm 1. Construct a simulation model and run a simulation

```
input : $type\_in, users\_in, severs\_in, operations\_in$: machine type and the
        number of concurrent users, servers, and operations
output: $t$: simulated throughput
1  for $i = 0;\ i < n;\ i++$ do                  // $n$: the number of benchmark data
2      $t$=0, $mips$=random_initialization() ;          // $t$: simulated throughput
3      $cl, len, vms$ = mapping($type[i], threads[i], servers[i], operations[i]$);
4      while $|t - th[i]| \geq threshold$ do  // $th[i]$: the $i^{th}$ benchmark throughput
5          if $t < th[i]$ then
6              $mips = mips + delta$;
7          else
8              $mips = mips - delta$;
9          $t$=run_simulation($mips, cl, len, vms$);
10     $M[i] = mips$ ;                          // $M$: a set of the mips values
11 $equ$ = derive_mips_equation($M, type, threads, servers, operations$);
12 $mips\_est$ = derive_mips_value($equ, type_in, user_in, servers_in, operations_in$);
13 $len, cl, vms$ = mapping($type\_in, users\_in, servers\_in, operations\_in$);
14 return run_simulation($mips\_est, cl, len, vms$);
```

Lines 1–10 are a forward and backward process to find MIPS values that yields simulation throughputs similar to the benchmark throughputs. The *mips* variable is randomly initialized. The function *mapping*() abides by the mapping rule defined at Step 3 to map i^{th} benchmark settings to the number of Cloudlets (*cl*), the length of each Cloudlet (*len*), and the number of servers (*vms*) that are needed to run CloudSim simulations (*run_simulation*()). If the difference between simulation throughput t and i^{th} benchmark throughput $th[i]$ is less than the *threshold*, the *mips* value is kept in $M[i]$. Otherwise, the *mips* value is adjusted by *delta* and the simulation is rerun. The variables *threshold* and *delta* are user-definable variables.

The function *derive_mips_equation*() in Line 11 is to derive a MIPS equation (*equ*) using the set of MIPS values (M) and simulation settings mapped from the benchmark settings. Our work uses non-linear regression to derive the MIPS equation because the size of the collected benchmark data is limited. However, other techniques could yield better equations. For example, if more benchmark data is collected, we can consider applying machine learning techniques. In that case, the simulation settings, obtained from the *mapping*() function, can act as features, and the (M) can be used as labels.

Lines 12–14 run a simulation using the input benchmark settings that must be simulated. The value of the *mips* variable is derived in Line 12 using the

MIPS equation, and the other simulation parameters are obtained in Line 13. Line 14 runs the simulation and returns the simulation throughput.

To yield reliable performance estimates by using benchmarks and simulation, the benchmark ontology has to be similar to the estimation ontology (similarity$_1$), and the simulation ontology must be similar to those of the benchmark (similarity$_2$). Therefore, the reliability of the simulation is measured as below.

$$reliability = similarity_1(O_{est}, R_{eb}, O_{ben}) * similarity_2(O_{ben}, R_{bs}, O_{sim}) \quad (1)$$

O_{est}, O_{ben}, and O_{sim} are the ontologies of estimation, benchmark, and simulation respectively. R_{eb} is a mapping rule from estimation ontology to benchmark ontology. R_{bs} is a mapping rule from benchmark ontology to simulation ontology. When given a rule for mapping R from the source ontology O_s to the target ontology O_t, the similarity between the two ontologies is measured as below.

$$similarity_i(O_s, R, O_t) = \sum_{i=1}^{n} \left(\frac{w_{R_i}}{W_R} * \frac{w_{O_{si}}}{\sum_{j=1}^{m}(w_{O_{sj}}) * r_{O_{si}}}\right) \quad (2)$$

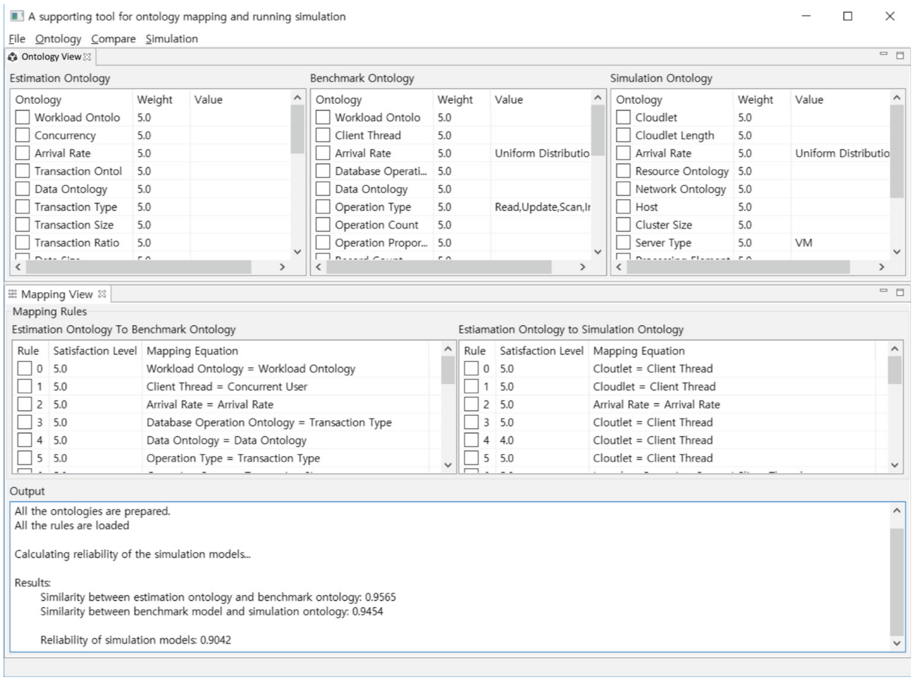
The variable n is the number of rules in the mapping rule R, W_R is the maximum weight value of the rule R, and w_{R_i} is the weight of each rule. The variable m is the total number of ontologies in O_s, $w_{O_{si}}$ is the weight of an ontology of O_s in a rule R_i, and $r_{O_{si}}$ is the number of mapping rules with which the ontology of O_s in the rule R_i is associated.

For example, let us assume that there are only three mapping rules $smap(YO_2) = CO_2$, $smap(YO_2) = CO_3$, and $smap(YO_3) = CO_3$. The maximum weight of the rule is assumed five and each rule has a weight of four. YO_2 and YO_3 have a weight of three. In this case, W_R is five, $w_{R_{1-3}}$ is four, $w_{O_{s1-3}}$ is three, and the summation of $w_{O_{s1-3}}$ is nine. Since YO_2 is associated with two rules, $r_{O_{s1-2}}$ is two. and $r_{O_{s3}}$ is one.

3.5 Step 5: Deriving Cloud System Architecture and Experimenting on Cloud

After building simulations, various workload and resource configurations can be simulated to derive architectural alternatives, such as the different number of concurrent users, servers, and various server types.

Let us assume a scenario that a CIO wants good performance (e.g., the throughput of over 10,000 operations per second), stable performance (e.g., stable throughput between 100 to 400 concurrent users), and inexpensive monthly cost (e.g., monthly fee less than 400 USD). Then the simulation shows the results that when the CIO uses three nodes of i2.xlarge instances, the good performance goal is satisfied because the throughput is from 11,000 to 15,000 (Ops/Sec). However, it costs 1,842.48 USD per month as the hour cost of the instance is 0.213 USD. On the other hands, when the CIO select three nodes of m4.large instances,


A supporting tool for ontology mapping and running simulation
File Ontology Compare Simulation
Ontology View
Mapping View

Estimation Ontology

Ontology	Weight	Value
Workload Ontolo	5.0	
Concurrency	5.0	
Arrival Rate	5.0	
Transaction Ontol	5.0	
Data Ontology	5.0	
Transaction Type	5.0	
Transaction Size	5.0	
Transaction Ratio	5.0	

Benchmark Ontology

Ontology	Weight	Value
Workload Ontolo	5.0	
Client Thread	5.0	
Arrival Rate	5.0	Uniform Distributio
Database Operati...	5.0	
Data Ontology	5.0	
Operation Type	5.0	Read,Update,Scan,Ir
Operation Count	5.0	
Operation Propor...	5.0	

Simulation Ontology

Ontology	Weight	Value
Cloudlet	5.0	
Cloudlet Length	5.0	
Arrival Rate	5.0	Uniform Distributio
Resource Ontology	5.0	
Network Ontology	5.0	
Host	5.0	
Cluster Size	5.0	
Server Type	5.0	VM

Mapping Rules

Estimation Ontology To Benchmark Ontology

Rule	Satisfaction Level	Mapping Equation
0	5.0	Workload Ontology = Workload Ontology
1	5.0	Client Thread = Concurrent User
2	5.0	Arrival Rate = Arrival Rate
3	5.0	Database Operation Ontology = Transaction Type
4	5.0	Data Ontology = Data Ontology
5	5.0	Operation Type = Transaction Type

Estiamation Ontology to Simulation Ontology

Rule	Satisfaction Level	Mapping Equation
0	5.0	Cloutlet = Client Thread
1	5.0	Cloudlet = Client Thread
2	5.0	Arrival Rate = Arrival Rate
3	5.0	Cloutlet = Client Thread
4	4.0	Cloutlet = Client Thread
5	5.0	Cloutlet = Client Thread

Output

```
All the ontologies are prepared.
All the rules are loaded

Calculating reliability of the simulation models...

Results:
    Similarity between estimation ontology and benchmark ontology: 0.9565
    Similarity between benchmark model and simulation ontology: 0.9454

    Reliability of simulation models: 0.9042
```

Fig. 4. A Prototype Tool

the monthly fee would be 259.2 USD as the hour cost is 0.12 USD; therefore, the goal of the inexpensive monthly price is achieved. However, the simulated throughputs are from 5,600 to 8,000 (Ops/Sec). By using the simulation, the CIO can derive Cloud design alternatives with trade-off analysis.

Since the requirements in the early design stage are ambiguous and changing, it is essential for CIOs to come up with Cloud design alternatives quickly, inexpensively, and also reliably in the initial design phases, which helps quickly make decisions and reduce trial and error in the later stages.

3.6 A Prototype Tool for Capturing Ontologies, Defining Mapping Rules, and Running Simulations

Figure 4 shows a tool helping the five-step process. In the Ontology View tap, users can define new ontologies or load pre-defined ontologies to refine the ontologies and weights. Likewise, pre-defined mapping rules can be loaded and refined in the Mapping View tap. After ontologies and mapping rules are configured, the tool automatically calculates the similarities among the ontologies, constructs simulation models, runs simulations, and measures the simulations reliability. The console tap displays the results.

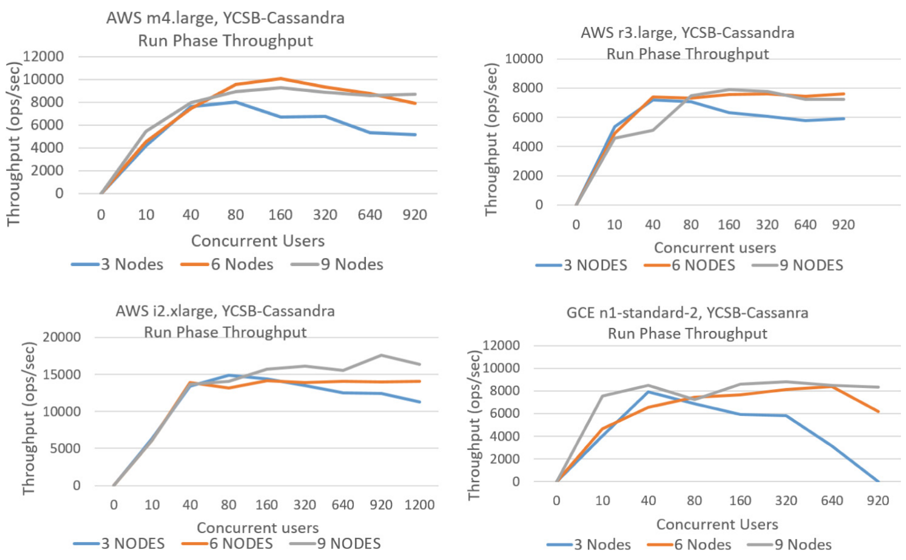


Fig. 5. Results of Cloud Benchmarking

4 Experiment

The objective of our experiments is to show proof of concepts in our approach, constructing reliable simulation models complemented by benchmarks. Since few papers describe how to build reliable Cloud simulations, our experimental results are compared with Cloud benchmark data.

4.1 Benchmark Data

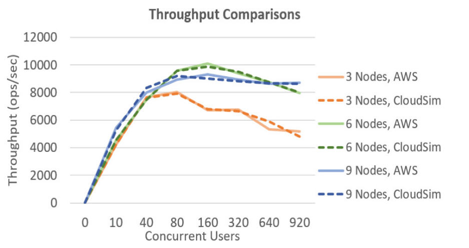
Figure 5 illustrates the Cloud benchmark data, collected from Amazon Web Service (AWS) and Google Compute Engine (GCE) by using the Yahoo! Cloud Serving Benchmark (YCSB). The benchmark data is utilized to construct the simulation models and used as a baseline to evaluate our simulation results.

The YCSB benchmark used the Cassandra NoSQL database and ran on m4.large, c4.large, r3.large, and i2.large instances of AWS and n1-standard-2 instances of GCE. The operation proportions for reading, updating, and inserting were respectively 63.25%, 21.64%, and 15.09%. The operation count and record count increased as the number of concurrent users rose. The operation count rose from 100,000 to 9,200,000. The record count increased from 255,550 to 8,922,472, and the number of concurrent users was 10, 40, 80, 160, 320, 640, and 920.

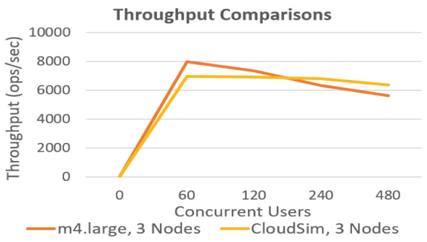
4.2 Simulation Experiments

In the simulation experiments using CloudSim, m4.large instances of AWS were selected. In every iteration of the experiments, the number of instances increased from 3 to 9, and the number of concurrent users increased from 10 to 960. Figure 6a shows a comparison between the simulation results and the original benchmark data. The dotted lines show the simulation throughput results and the straight lines depict the benchmark throughput results with the same workload and infrastructure settings as the simulation.

We cross-validated the simulation models by running Cloud benchmarks with workload configurations that have not been collected, such as 60, 120, 240, and 480 concurrent users and then compared the simulation models with the new benchmark results. Figure 6b shows throughput comparisons between the benchmarks and simulations. The experiments for cross-validation show 9.79(%) of Mean Absolute Percentage Error (MAPE) and 665.73 ops/sec of Mean Absolute Error (MAE). In other words, the simulation models show 90.21(%) accuracy.



(a) Validations of simulation models



(b) Cross-validations of simulation models

Fig. 6. Cloud Simulation Results

The validation results are compared to the reliability and similarity measurements described in the previous section. After applying the mapping rules, the reliability equation yields 0.9042, where similarity$_1$ is 0.9565 and similarity$_2$ is 0.9454. In other words, the measured reliability of the simulation models is 90.42%, which is similar to the accuracy evaluation result.

5 Observations and Discussion

We have focused on explaining the five steps for constructing simulation models complemented by benchmarks. Our approach is semi-automated helped by a prototype tool; the tool's users can either define new ontology, mapping rules, and reliability quantification scheme or refine the ones as references for future applications. The tool then automates constructing simulation models, running simulations, and measuring the models reliability. The ontologies, mapping rules, and reliability quantification scheme that are described in this paper seem to yield reliable simulation models. Moreover, by mapping the benchmark ontologies to those of a simulation, the semantics of ontologies in a simulation become understandable and traceable.

One threat to the validity of our approach, specifically regarding the ontologies, mapping rules, and reliability quantification, is its limitation to be generalized because the range of experiments and benchmark data were limited. Instead of fully generalizing our approach to fit all domain, we intend to offer as many references - i.e., reference ontologies, reference mapping rules, and reference reliability quantification defined by users in the different domain - as possible, then let users select and refine one, like a catalog. However, constructing and maintaining ontologies are non-trivial tasks in practice. Since we discussed throughput only in our evaluation, our approach must cover a broader range of QoS parameters. To approach those problems, more extensive case study is required. Last, the automation for constructing simulation models currently only partially works for specific mapping rules such as those described in this paper.

6 Conclusion

We presented a complementary approach, using both benchmarking and simulations together, as among the first of its kind, to estimating the performance of Cloud-based systems, in a fast, inexpensive and also reliable manner. In this approach, mapping benchmark ontologies to those of simulations enhance the reliability of simulation-based estimations, and in a more understandable manner. More specifically, our work proposes a five-step process for capturing the ontologies of estimation, benchmark, and simulation, and mapping the benchmark ontologies to simulation ontologies, while considering the similarities, and mismatches, between the two, with the incorporation of a quantification scheme for the simulation reliability and an algorithm for helping automate the construction of simulation models. A prototype tool has also been presented for supporting our five-step semi-automatic process. The results from our complementary approach-based simulation show that they are indeed reliable to the extent that they are similar to the YCSB benchmark results, when using Cassandra on AWS and GCE.

As future work, we plan to apply our approach to a wide variety of different types of domains and investigate more ontologies pertaining to such domains, while also considering mapping rules for them. Investigations of more systematic approaches for ontology mapping schemes and reliability quantification schemes, instead of just a few heuristic ones, lie ahead as well. Work is also underway towards fuller implementation of our prototype tool, e.g., for managing graphically-oriented goal-oriented models and mapping between them.

Acknowledgments. We would like to thank Amazon, Inc., for the AWS in Education Grant Award, and also Google, Inc., NTT Data, and NSF for their respective grants. We also thank Chengzu Yu for your helpful reviews.

References

1. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publisher, New York (2000)
2. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. In: 3rd IEEE International Symposium on Requirements Engineering, pp. 226–235 (1997)
3. Rolland, C., Souveyet, C., Achour, C.B.: Guiding goal modeling using scenarios. IEEE Trans. Softw. Eng. **24**(12), 1055–1071 (1998)
4. Mistry, S., Bouguettaya, A., Dong, H.: Economic Models for Managing Cloud Services. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-73876-5
5. Andrikopoulos, V., Strauch, S., Leymann, F.: Decision support for application migration to the cloud. In: Proceedings of CLOSER, vol. 13, pp. 149–155 (2013)
6. Bermbach, D., Wittern, E., Tai, S.: Cloud Service Benchmarking. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-55483-9
7. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.P.: Cost-benefit analysis of cloud computing versus desktop grids. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 226–235 (1997)
8. Iosup, A., Prodan, R., Epema, D.: IaaS cloud benchmarking: approaches, challenges, and experience. In: Li, X., Qiu, J. (eds.) Cloud Computing for Data-Intensive Applications, pp. 83–104. Springer, New York (2014). https://doi.org/10.1007/978-1-4939-1905-5_4
9. Chung, L., Subramanian, N., Hill, T., Park, G.: Silverlining: a simulator to forecast. CrossTalk, p. 29 (2015)
10. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur. Gener. Comput. Syst. **28**(5), 755–768 (2012)
11. Castañé, G.G., Nuñez, A., Llopis, P., Carretero, J.: E-mc^2: a formal framework for energy modelling in cloud computing. Simul. Model. Pract. Theory **39**, 56–75 (2013)
12. Núñez, A., Vázquez-Poletti, J.L., Caminero, A.C., Castañé, G.G., Carretero, J., Llorente, I.M.: iCanCloud: a flexible and scalable cloud infrastructure simulator. J. Grid Comput. **10**(1), 185–209 (2012)
13. Hill, T., Supakkul, S., Chung, L.: Run-time monitoring of system performance: a goal-oriented and system architecture simulation approach. In: 1st International Workshop on Requirements@ Run. Time, pp. 31–40 (2010)
14. Chung, L., Hill, T., Legunsen, O., Sun, Z., Dsouza, A., Supakkul, S.: A goal-oriented simulation approach for obtaining good private cloud-based system architectures. J. Syst. Softw. **86**(9), 2242–2262 (2013)
15. Shapira, G., Chen, Y.: Common pitfalls of benchmarking big data systems. IEEE Trans. Serv. Comp. **9**(1), 152–160 (2016)
16. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. **41**(1), 23–50 (2011)
17. Lera, I., Juiz, C., Puigjaner, R.: Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems. Sci. Comput. Program. **61**(1), 27–37 (2006)
18. Park, G., Chung, L., Zhao, L., Supakkul, S.: A goal-oriented big data analytics framework for aligning with business. In: Proceedings of 3rd IEEE International Conference On Big Data Service And Applications, pp. 31–40 (2017)

19. Johng, H., Kim, D., Hill, T., Chung, L.: Using blockchain to enhance the trustworthiness of business processes: a goal-oriented approach. In: Proceedings of IEEE International Conference on Services Computing, pp. 249–252 (2018)
20. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of 1st ACM Symposium on Cloud Computing, pp. 143–154 (2010)
21. Sancho, P.P., Juiz, C., Puigjaner, R., Chung, L., Subramanian, N.: An approach to ontology-aided performance engineering through NFR framework. In: Proceedings of 6th ACM International Workshop on Software and Performance, pp. 125–128 (2007)

Two-Phase Web Service QoS Prediction with Restricted Boltzmann Machine

Lu Chen[1,2], Yuyu Yin[1,2], Yueshen Xu[3(✉)], Liang Chen[4], and Jian Wan[1,2,5]

[1] School of Computer, Hangzhou Dianzi University, 310018 Hangzhou, China
{chenlu, yinyuyu, wanjian}@hdu.edu.cn
[2] Key Laboratory of Complex Systems Modeling and Simulation of Ministry of Education, Hangzhou, China
[3] School of Computer Science and Technology, Xidian University, 710126 Xi'an, China
ysxu@xidian.edu.cn
[4] School of Data and Computer Science, Sun Yat-Sen University, 510006 Guangzhou, China
chenliang6@mail.sysu.edu.cn
[5] School of Information and Electronic Engineering, Zhejiang University of Science and Technology, 310023 Hangzhou, China

Abstract. Collaborative filtering (CF) has been widely used in quality of service (QoS) prediction. However, most of traditional CF-based methods always suffer from overestimation of similarity computation and invalid neighbors. To address these problems, we propose a two-phase QoS prediction approach based on restricted Boltzmann machine (RBM). In the first phase, we propose an RBM-based approach to predict missing QoS values for invalid neighbors, which can identify similar neighbors with high accuracy. In the second phase, we propose a user-based CF method to predict, which utilizes user similar neighbors. Experimental results conducted in a real-world dataset show that our approaches can produce superior prediction accuracy and are not sensitive to parameter settings.

Keywords: Web service · QoS prediction · Collaborative filtering Restricted Boltzmann machine

1 Introduction

With the number of Web services growing dramatically, different service providers offer many Web services with the same or similar functions. It has been an urgent task to recommend suitable services for users from a large number of service candidates [1]. There are two types of properties for a service, functional properties and non-functional properties. The non-functional properties are also known as quality-of-service (QoS). It

The two authors Yuyu Yin and Lu Chen contribute equally to this paper, so they are co-first authors.

is a spotlight of how to improve the prediction accuracy of QoS and how to provide service recommendation with high quality [2].

In recent years, many researchers have applied collaborative filtering (CF) to QoS prediction and obtained better prediction accuracy [2]. A crucial task of CF-based methods is to identify neighbors. The prediction accuracy largely relies on the quality of the identified neighbors. To improve the quality of identified neighbors, some existing works focus on the improvement of similarity computation. But in the case of high data sparsity, we notice that the number of neighbors of a user is quite limited. Also, there exist some invalid neighbors that need to be filtered, which did not invoke the target service or are not observed by the target user (we name these neighbors as **invalid neighbors**) and those neighbors will be not used to predict missing QoS values. Such filtering is likely to lead to low prediction accuracy because of lack of available neighbors.

To address these problems, we propose a two-phase approach for QoS prediction, which is based on restricted Boltzmann machine (RBM). The first prediction phase is to predict missing values for invalid neighbors, and further identify similar neighbors. The second prediction phase is to compute the final prediction results by user-based CF method. From the experimental results, it can be seen that our approach can make better use of identified neighbors and alleviate data sparsity problem.

In summary, the main contributions of this paper are as follows.

1. It proposes a novel neighbors selection method. It employs RBM model to predict missing values for invalid neighbors, and further filters the neighbors with network location and geographical information. The proposed method can complete the task of selecting high-quality neighbors.
2. It conducts sufficient experiments on a real-world dataset, and compares the proposed methods to many existing methods. The experimental results demonstrate the effectiveness and high robustness to parameter values of our method.

The rest of this paper is organized as follows. Section 2 discusses the related work. Sections 3 and 4 elaborate the proposed approach, and the experimental results are presented in Sect. 5. Section 6 concludes the whole paper.

2 Related Work

The traditional CF methods can be classified into two categories: neighbor-based methods and model-based methods [2]. Many neighbor-based CF methods have achieved success in recommendation systems [3]. Wu et al. [4] proposed a ratio-based method to compute similarity, which improved the prediction accuracy and could be computed faster than other compared methods.

One difficulty in neighbor-based CF methods is the data sparsity problem [5]. Owing to the high sparsity of data, the neighbor-based CF methods cannot accurately obtain similar neighbors. In recent years, some studies try to solve this problem. Wu et al. [5] proposed a time-aware neighbor-based CF approach with better accuracy at high sparsity. These earlier works put attention to similarity computation, but this paper also focuses the selection of neighbors.

Moreover, it is difficult for neighbor-based methods to handle large amount of data. Therefore, researchers turned to study model-based CF methods. There are several representative model-based approaches, including Matrix Factorization (MF) [6] and restricted Boltzmann machine (RBM) [7]. In recent years, some researchers exploited the potential of RBM in extracting features and solving data sparsity problem [8].

To fully take advantage of neighbor-based methods and model-based methods, some researchers tried to combine the two types of approaches. Inspired by such an idea, in this paper, we propose a novel model by both leveraging the RBM-based model and neighbor-based model to predict missing QoS values.

3 The Whole Framework

We present the proposed whole framework in Fig. 1, which consists of two phases.

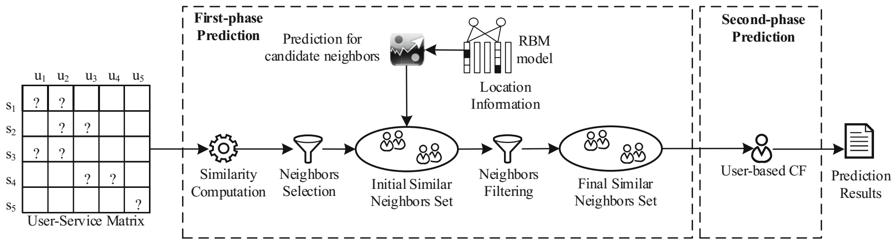


Fig. 1. The whole framework

In the first phase, we use the Euclidean distance to compute the similarity among users. The similarity computation result is used to build the initial similar neighbors set. Next, we employ RBM model to predict all missing QoS values for invalid neighbors. We further identify fine-grained neighbors from the initial neighbors set based on network location and geographical information. In the second phase, we propose the user-based CF model to predict final results.

4 The Proposed Prediction Approach

4.1 The First-Phase Prediction

Similarity Computation. In this section, we propose a Euclidean distance in similarity computation.

$$S_{u,v} = \frac{1}{1 + \sqrt{\frac{\sum_{i=0}^{M}\left(\left(q_{u,i}-\bar{q}_u\right)-\left(q_{v,i}-\bar{q}_v\right)\right)^2}{|M|}}} \tag{1}$$

where $S_{u,v}$ is the similarity of user u and user v. $M = M_u \cap M_v$ is the set of services that are invoked by both user u and user v, $q_{u,i}$ is the QoS value of target service i invoked by target user u, and $q_{v,i}$ is the QoS value of target service i invoked by user v. We add a one value in the denominator to prevent the denominator from being zero. $\bar{q}_u$ is the average QoS of user u and $\bar{q}_v$ is the average QoS of user v.

Neighbors Selection. After the similarity computation between different users and different services, we could have directly chosen the top K most similar neighbors. However, in the current neighbors set, some neighbors are not applicable, because such neighbors probably do not have invoked each target service, which lowers the number of available neighbors and further damages prediction accuracy. To fix this issue, we propose to use RBM model to predict missing QoS values for those neighbors which have not invoked a target service (in user side). Our aim is to ensure that all neighbors can be applicable for reliable prediction. Meanwhile, we further filter some other neighbors by utilizing the network location and geographical information. Such filtering can improve the quality of similar neighbors. Then, a similar neighbors' set $N(u)$ of user u is finally formed. $N(u)$ is composed of two subsets $N_1(u)$ and $N_2(u)$. $N_1(u)$ is the set of predicted neighbors and $N_2(u)$ is the set of valid neighbors.

Prediction Based on RBM Model. The next task is the missing QoS values prediction for the users in set $N_1(u)$. We propose an RBM model to finish this task. Suppose that we have M services, N users, rounded QoS values from 1 to K, and a user invoked m services. Each user is treated as a single training case of an RBM, and we still use an RBM to predict missing values. Each RBM shares the same number of hidden units H that represent features, but an RBM only has visible softmax units U for the services invoked by that user. An RBM can only have a few connections in high sparsity of the real case of services invocation. Let U be a $K \times m$ observed binary indicator matrix with $u_i^k = 1$ if the user u has invoked service i as the value of being k, and 0 otherwise. The energy function of user-oriented RBM is defined as

$$\begin{aligned} E(u,h;W,b) = &-\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=0}^{K} W_{i,j}^k h_j u_i^k + \sum_{i=1}^{M} \log Z_i \\ &-\sum_{i=1}^{M}\sum_{k=0}^{K} u_i^k b_i^k - \sum_{j=1}^{N} h_j b_j \end{aligned} \tag{2}$$

where $W_{i,j}^k$ is a symmetric interaction parameter between the QoS value k of the i-th service and the j-th feature. b_i^k and b_j are two biases to reflect the innovation preference of services i and j. $Z_i = \sum_{k=0}^{K} \exp(b_i^k + \sum_j h_j W_{i,j}^k)$ is the normalization term that ensures $\sum_{k=0}^{K} P(u_i^k = 1|h) = 1$. According to the conditional multinomial distribution and conditional Bernoulli distribution, the distributions of services and features are

$$P(u_i^k = 1|h) = \frac{\exp(b_i^k + \sum_{j=1}^{n} u_i^k W_{i,j}^k)}{\sum_{k=0}^{K} \exp(b_i^k + \sum_{j=1}^{n} h_j W_{i,j}^k)} \tag{3}$$

$$P(h_j = 1|u) = \sigma(b_j + \sum_{i=1}^{m} \sum_{k=0}^{K} u_i^k W_{i,j}^k) \tag{4}$$

where $\sigma(x) = 1/1 + \exp(-x)$ is the sigmoid activation function. With the conditional distributions in Eqs. (3) and (4), we can directly use the contrastive divergence algorithm for training [9], in which the updates for each parameter are as follows.

$$\begin{aligned} \frac{\partial \log P(u;\theta)}{\partial W_{i,j}} &= \varepsilon(\langle u_i h_j \rangle_{data} - \langle u_i h_j \rangle_{rec}), \quad u_i > 0 \\ \frac{\partial \log P(u;\theta)}{\partial a_i} &= \varepsilon(\langle u_i \rangle_{data} - \langle u_i \rangle_{rec}), \quad u_i > 0 \\ \frac{\partial \log P(u;\theta)}{\partial b_j} &= \varepsilon(\langle h_j \rangle_{data} - \langle h_j \rangle_{rec}) \end{aligned} \tag{5}$$

where $\langle \cdot \rangle_{data}$ represents the probability distribution of a hidden layer in the case of a visible unit. $\langle \cdot \rangle_{rec}$ represents the probability distribution of the model definition after the reconstruction using the contrastive divergence algorithm, and ε is the learning rate. Among these, $u_i > 0$ ensures that only the data that satisfy $u_i > 0$ are used in the model.

After the model is trained, the probability that a user v invokes a service i with a QoS value being k can be obtained directly based on the known QoS values set U. The RBM prediction of the missing QoS value $\hat{q}_{v,i}$ is:

$$\hat{q}_{v,i} = l \times \max(P(v_i^k = 1|V)_l) \tag{6}$$

where $\max(P(v_i^k = 1|V)_l)$ is the maximum value of the probability that the user v invoked the service i and the received QoS is k, and l is the score corresponding to the maximum value of the probability.

4.2 The Second-Phase Prediction

To improve the final prediction accuracy, we also propose a user-based CF method to predict in the second phase.

The neighbors set $N(u)$ for target user u is generated in the first-phase, which has two subsets, that is, the predicted neighbors set $N_1(u)$ and valid neighbors set $N_2(u)$. The final prediction result is computed as follows.

$$\hat{q}_{u,i} = \frac{\sum\limits_{v_1 \in N_1(u)} (\hat{q}_{v_1,i}) \times S_{u,v_1} + \sum\limits_{v_2 \in N_2(u)} (q_{v_2,i}) \times S_{u,v_2}}{\sum\limits_{v_1 \in N_1(u)} S_{u,v_1} + \sum\limits_{v_2 \in N_2(u)} S_{u,v_2}} \tag{7}$$

where $\hat{q}_{v_1,i}$ represents the first-phase prediction results of target service i after being invoked by user v_1. S_{u,v_1} is the similarity of target user u and user v_1. $q_{v_2,i}$ is the QoS value of target service i after being invoked by user v_2. S_{u,v_2} is the similarity of target user u and user v_2.

5 Experiment and Evaluation

We use the public dataset WSDream to conduct the experiments [10]. This dataset has been widely used by many researchers. WSDream dataset contains 5825 services and 339 users, including two QoS attributes: response time and throughput.

5.1 Evaluation Metric and Parameter Setting

We use the mean absolute error (*MAE*) and normalized mean absolute error (*NMAE*) metrics to evaluate the prediction accuracy. *MAE* and *NMAE* are computed as

$$MAE = \frac{1}{N}\sum_{u,i} |q_{u,i} - \hat{q}_{u,i}|, NMAE = \frac{MAE}{(\sum\limits_{(u,i)\in TestSet} q_{u,i})/N} \tag{8}$$

where $q_{u,i}$ represents the real QoS value, $\hat{q}_{u,i}$ represents the prediction result, and N is the number of values in test set.

5.2 Performance Comparison

In order to reflect the real case of service invocation, we randomly select a part of data from the original WSDream dataset as training set, and the rest data form the test set. In this study, we generate four training sets with different sparsities, where the sparsity degree d is 2.5%, 5%, 10%, 15%, and 20%, respectively.

To better evaluate the performance of the proposed method, we compare our methods with the following state-of-the-art competitive QoS prediction methods. The experimental results are present in Table 1. The parameters in the compared methods are set according to the default settings in their original papers.

1. RBM (restricted Boltzmann machine) [7]: This method uses the RBM-based CF algorithm to predict missing values.
2. WSRec [10]: A hybrid model composed of user-based CF and item-based CF.
3. LFM (latent factor model) [6]: LFM decomposes the user-service matrix by dimensionality reduction to learn implicit features and produce predictions.
4. CAP (credibility-aware prediction model) [11]: CAP is a novel credibility-aware QoS prediction method, which employs two-phase K-means clustering algorithm.

Table 1. Accuracy comparison (a smaller value means a higher accuracy)

Models	*d* = 2.5%		*d* = 5%		*d* = 10%		*d* = 15%		*d* = 20%	
	MAE	NMAE	MAE	NMAE	MAE	NMAE	MAE	NMAE	MAE	NMAE
RBM	0.9812	1.1986	0.8354	1.0291	0.7880	1.0082	0.7850	0.9868	0.7780	0.9830
WSRec	0.7040	0.8600	0.6794	0.8375	0.6211	0.7948	0.6037	0.7589	0.6020	0.7607
LFM	0.6983	0.8531	0.5783	0.7129	0.5644	0.7222	0.5438	0.6836	0.5350	0.6760
CAP	0.6503	0.7944	0.5452	0.6730	0.5040	0.6184	0.4865	0.5969	0.4635	0.5688
JLMF	0.6470	0.7904	0.5393	0.6612	0.5167	0.6335	0.5041	0.6180	0.4750	0.5824
LE-MF	0.6525	0.7971	0.5734	0.7066	0.5136	0.6571	0.4827	0.6068	0.4646	0.5870
U-RBM	**0.6375**	**0.7787**	**0.5333**	**0.6473**	**0.4982**	**0.6374**	**0.4747**	**0.5967**	**0.4621**	**0.5864**

5. JLMF [12]: JLMF is an MF model based on network location information and influence of neighbors.
6. LE-MF (location-enhanced matrix factorization) [13]: A matrix decomposition model that introduces location information and trust mechanism.
7. U-RBM (user-oriented RBM): Our proposed model with being named as U-RBM.

In Table 1, *MAE* is the mean absolute error, *NMAE* is the normalized mean absolute error, and *d* is the sparsity of training sets. We can have following observations.

1. The proposed prediction methods U-RBM is superior to the compared methods in on both *MAE* and *NMAE* measures.
2. In all cases of training set densities, our proposed models achieve consistently lower errors. Specially, in the case of high data sparsity (e.g., *d* = 2.5% and 5%), the prediction accuracy of U-RBM model is still the highest, which indicates that U-RBM can better deal with data sparsity problem.

5.3 Sensitivity Analysis of Parameters

Impact of *TopKUser*. The parameter *TopKUser* denotes the number of similar neighbors. A higher sparsity means less available training data. The evaluated value of *TopKUser* is from 4 to 20.

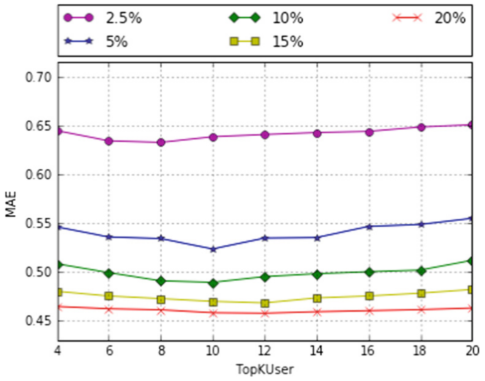


Fig. 2. Impact of *TopKUser*

As shown in Fig. 2, with the increase of *TopKUser*, *MAE* values decrease initially. This is because as the number of similar neighbors increases, the probability of selecting the real similar neighbors of a target user becomes larger. Also, the similar neighbor filtering adds those users that are highly similar to the target user. This is because there are no public invocation records, and those users are prevented from being similar neighbors on account of other users who do not have a strong similarity but are still selected. So the reliability of similar neighbors is enhanced.

6 Conclusions

In this paper, we propose an approach for QoS prediction based on RBM model. We proposed a novel similarity computation method. Then, the RBM was proposed to predict missing values for invalid neighbors. We also employed network location and geographical information to further improve the selection quality of neighbors. The extensive experiments conducted in a real-world dataset verified the effectiveness of our models.

Acknowledgments. This paper is supported by National Natural Science Foundation of China (No. 61702391), Natural Science Foundation of Zhejiang Province (No. LY12F02003) and Shaanxi Province (No. 2018JQ6050), National Key Technology Support Program (No. 2015 BAH17F02) and Fundamental Research Funds for Central Universities (JBX171007).

References

Wu, Y., Xie, F., Chen, L., Chen, C., Zheng, Z.: An embedding based factorization machine approach for web service QoS prediction. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 272–286. Springer, Cham (2017a). https://doi.org/10.1007/978-3-319-69035-3_19

Zheng, Z., Ma, H., Lyu, M.R.: QoS-aware web service recommendation by collaborative filtering. IEEE Trans. Serv. Comput. **1**(2), 140–152 (2010)

Yin, Y., Yu, F., Xu, Y., et al.: Network location-aware service recommendation with random walk in cyber-physical systems. Sensors **17**(9), 2059 (2017)

Wu, X., Cheng, B., Chen, J.L.: Collaborative filtering service recommendation based on a novel similarity computation method. IEEE Trans. Serv. Comput. **10**(3), 352–365 (2017b)

Wu, C., Qiu, W., Wang, X., et al.: Time-aware and sparsity-tolerant QoS prediction based on collaborative filtering. In: IEEE ICWS, pp. 637–640 (2016)

Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. ACM Comput. **42**(8), 30–37 (2009)

Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted Boltzmann machines for collaborative filtering. In: International Conference on Machine Learning, pp. 791–798 (2007)

Tramel, E.W., Manoel, A., Caltagirone, F., et al.: Inferring sparsity: compressed sensing using generalized restricted Boltzmann machines. In: Information Theory Workshop, pp. 265–269 (2016)

Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)

Zheng, Z., Ma, H., Lyu, M.R., et al.: WSRec: a collaborative filtering based web service recommender system. In: IEEE ICWS, pp. 437–444 (2009)
Wu, C., Qiu, W., Zheng, Z., et al.: QoS prediction of web services based on two-phase K-means clustering. In: IEEE ICWS, pp. 161–168 (2015)
Yin, Y., Song, A., Min, G., et al.: QoS prediction for web service recommendation with net-work location-aware neighbor selection. Int. J. Softw. Eng. Knowl. Eng. **26**(4), 611–632 (2016)
Xu, Y., Yin, J., Deng, S., et al.: Context-aware QoS prediction for web service recommendation and selection. Expert Syst. Appl. **53**(C), 75–86 (2016)

Service Engineering

Constructing and Evaluating an Evolving Web-API Network for Service Discovery

Olayinka Adeleye[1](✉), Jian Yu[1], Sira Yongchareon[1], and Yanbo Han[2]

[1] Department of Computer Science, Auckland University of Technology, Auckland 1010, New Zealand
{olayinka.adeleye,jian.yu,sira.yongchareon}@aut.ac.nz

[2] Beijing Key Laboratory for Large-scale Stream Data Processing, North China University of Technology, Beijing, China
yhan@ncut.edu.cn

Abstract. Web-APIs enable cross-organizational functionality integration over the Web and thus are the foundation of modern distributed service-based systems. However, despite the rapid increase in the number of Web-APIs available on the Internet, the discovery and uptake of appropriate Web-APIs by businesses on a Web scale is still a great challenge. One of the main reasons is that Web-APIs registered on directories such as ProgrammableWeb.com are in general *isolated*, as they are registered by diverse providers independently and progressively. In this paper, we present a method for analyzing the Web-API ecosystem and propose a complex-network-based approach for building an evolving social network for Web APIs. We conduct our analysis in two phases: First, from the complex network perspective, we investigate mashups and Web-APIs interactions and analyze the Web-API popularity distribution using the popular ProgrammbleWeb dataset. Second, we quantitatively measure the Preferential Attachment mechanism which is a key driver of an evolving network. Based on our analysis, we propose an approach to construct an evolving Web-API social network based on the theoretical procedure of the Barabási-Albert complex network model. Results presented in this work will not only provide insight into the topology of the Web-API ecosystems but also serve as a practical guide for designing an evolving-network-based solution for service discovery.

Keywords: Web APIs · Complex network analysis
Preferential attachment · Evolving networks · ProgrammableWeb

1 Introduction

The emergence of Web 2.0 coupled with the rapid development in Web service technology has led to a continual increase in the number of Web services and

This work was supported in part by the National Key Technology R&D Program of China (2017YFC0804406) and the National Natural Science Foundation of China (61672042).

their compositions. Nowadays many real-world applications such as online social media, online shopping, weather forecast, and disaster prevention [14,15] invoke web services via accessible endpoints to implement their functionalities. Modern web services with features such as *RESTful architecture*, *JSON data*, and/or *JavaScript interface* are usually called *Web-APIs*[1] in order to distinguish from the traditional SOAP-based web services; and multiple Web-APIs can be quickly composed into a webpage or application called *mashup*. This shortened software development life cycle leads to the formation of the so-called Web service ecosystem [5,16], where new services emerge, some old ones perish, and service vendors and developers collaborate to develop innovative software solutions. A typical representation of an evolving service ecosystem is ProgrammableWeb[2], which is currently the largest online Web-API directory, with over 19,000 Web-APIs belonging to more than 400 predefined categories, and over 6,000 mashups as at May 2018. It also provides information such as date of introduction, profile, and developers. The perishing of some existing Web-APIs and the emergence of new ones coupled with their dynamic collaborations drive the evolution of this service ecosystem over time [12].

For most service ecosystems, one of the main issues is the isolation of Web-APIs, which limits their discoverability. For instance, in ProgrammableWeb, Web-APIs have categories, and several Web-APIs can be involved in one mashup, but there is no *direct* connection or relationship between two Web-APIs. The reason behind this is that Web-APIs are usually registered by diverse service providers independently over time, and the connections or social relationships between Web-APIs are never directly created.

In this paper, we propose an evolving-complex-network-based approach to constructing a social network for Web-APIs based on their popularity in the service ecosystem. To achieve this, we first study the underlying topology of ProgrammableWeb and analyze the popularity distribution of Web-APIs in the ecosystem, using the dataset in a period of thirteen years (2005–2017). Then, we measure the Preferential Attachment (PA) of the ecosystem, which is the key mechanism that governs the evolution of many existing real world networks [11]. And finally, we incorporate our findings into the construction of an evolving social network of Web-APIs using the well-established Barabási-Albert model [3] in complex networks.

The main contribution of this paper includes:

1. We analyzed the popularity distribution of Web-APIs on ProgrammbleWeb based on mashup-API relationships and measured the PA mechanism which defines the topology of the ecosystem. To the best of our knowledge, this is the first time that PA is measured for a service ecosystem.
2. We designed and implemented an evolving social network model for ProgrammableWeb Web-APIs which facilitates service discovery and serves as a

[1] https://en.wikipedia.org/wiki/Web_API. Note that in this paper, we coin "Web" and "APIs" together as one term "Web-APIs" to emphasize the atomicity of this term.
[2] http://www.programmableweb.com.

stepping stone for developing advanced evolving network models for service ecosystems.

The rest of this paper is organized as follows. Section 2 is the background and related work; Sect. 3 presents the analysis of the ProgrammableWeb ecosystem and also the analysis results including popularity distribution and PA; In Sect. 4, we present an approach to the construction of an Web-API evolving network and discuss its application in service discovery; Finally, Sect. 5 is the conclusion and future work.

2 Background and Related Work

In this section, we discuss the background of this work and the related work in complex network analysis, evolving Web service ecosystem analysis, and existing service social network construction approaches.

Over the years, complex networks have been extensively studied and several significant discoveries have been made including the well-acclaimed small-world networks [24] and scale-free networks [4]. Various mechanisms that governs a network's topology and evolution have been investigated and found ubiquitous among many real world networks. In particular, *preferential attachment* and *growth* have garnered special attention in evolving complex networks research [3,21], not only because they are fundamental to explaining the topological features observed in many real world networks but also because they have been empirically validated to be the drivers of many evolving networks. For instance, the topology of the Internet, the World Wide Web and the citation network have been investigated using evolving network models and shown to be fundamentally governed by the PA and growth mechanisms [1–3]. In terms of evolving network models, the PA and growth driven Barabási-Albert (BA) model [3] is the foundation of other models such as the fitness-based Bianconi-Barabási model [6].

There have been a number of studies investigating the evolutionary properties of service ecosystems and the complementary features of services and their compositions particularly on ProgrammableWeb. Weiss et al. [25], examined the structure of the mashup ecosystem using the ProgrammableWeb dataset. The authors analyzed the relationships of mashups and Web-APIs using a bipartite graph, and found that while the growth rate of new Web-APIs and mashups is linear, the distribution of mashups over APIs follows a power-law. Huang et al. [12] used a network analysis approach to study both the usage patterns and the evolution traces of Web-APIs in the ProgrammableWeb. The authors conducted their analysis based on two derived networks: the Composition-Service network, which is the same bipartite graph of Mashup-APIs used in [25] and the Service-Service network, which is a network of services that are used together in the same mashups. The authors found that the service popularity distribution is highly concentrated, which is consistent with the findings in [25], and they also found that the reuse rate of services is low and the advanced use of many services together is still rare, which provides evidence to our motivation of building a social network for services/Web-APIs. To better present the ProgrammableWeb ecosystem, Lyu et al. [16] used a three-level hierarchical view to visualize it based on the Mashup-API graph, the tag graph, and domain graph. Wang et al. [23] also explored ProgrammableWeb data patterns from the user perspective with a User-API network.

As for the research on constructing service social networks, Fallatah et al. [9] proposed to add service-service, user-user, and user-service links to build a service social network. Based on the network, metrics such as user popularity, service market share, and user satisfaction can be measured. Simulation was done but how to build such network from real-world data was not discussed. Semantic information mined from service descriptions is a good reference for adding links among services. Wang et al. [22] used domain knowledge to calculate the degree of semantic match between any two services and then a threshold can be set to determine the number of links in the network. Similarly, Feng et al. [10] constructed three types of service networks based on the *subsume*, *sequential-total* (the output of service A covers the input of service B), and *sequential-part* (the output of service A partially covers the input of service B) semantic relations. Clearly such networks are static without considering any dynamical properties. From the evolving network perspective, Chen et al. [7] built a service social network partially based on the Bianconi-Barabási (BB) model. One limitation of their work is that the fitness parameter of an existing service node is calculated dynamically on the arrival of a new service, while the BB model requires a quenched/fixed fitness value for a node, which makes the closed-form solution of the BB model not applicable to this network.

3 Analysis and Results

In this section, we investigate the topology and dynamical mechanism of the ProgrammbleWeb registry. For the topology, we look at the popularity, or degree distribution of the Web-APIs based on the mashup-API bipartite graph. We first discuss data acquisition and processing, then we analyze the data in three steps: *visualization*, *model fitting*, and *comparison with existing classical network models* such the Poisson, exponential and log-normal distributions. For dynamical/evolving mechanisms, we investigate and measure *preferential attachment*.

3.1 Data Acquisition and Processing

We collected the time-stamped raw data, which contains information regarding Web-APIs and mashups from *June* 2005 to *November* 2017 in ProgrammableWeb. Since the $ProgrammableWeb$ backend database is not publicly accessible, only its web pages can be employed for collecting the data. We used data scraping to crawl data from ProgrammableWeb web pages. The web pages are separated into two categories: Web-APIs and mashups, where every Web-API has properties including *name*, *description*, *publication date*, and *category*; similarly, each mashup also contains the above metadata plus the list of Web-APIs

Table 1. Summarize features of the programmableweb dataset

Number of Web APIs acquired	16,138
Number of Mashups acquired	5,883
Average number of Web APIs invoked by Mashups	2.1
Number of Mashups with less than 2 services	241
Number of Web APIs invoked in at least one Mashup	1,525

Table 2. Top 5 most consumed Web-APIs

Web APIs	Number of links
GoogleMap	2,072
Twitter	663
Youtube	557
Flickr	484
Facebook	377

invoked within it. Table 1 gives an overview of the ProgrammableWeb dataset. After pre-processing and removing redundant mashup points, we have 16,138 Web APIs and 5,883 Mashups for our analysis.

3.2 Affiliation Network of Web-APIs and Mashups

To extract the popularity distribution of Web-APIs in ProgrammableWeb, we model the ecosystem in the form of an *affiliation network* that depicts the invocation relation between mashups and Web-APIs. As shown in Fig. 1, technically, the network is a bipartite graph, where the edges indicate which Web-APIs are invoked by which mashups: $G = (M \uplus A, E)$ where M is the set of Mashups and A is the set of Web-APIs, and for any edge $(m, a) \in E, m \in M$ and $a \in A$.

Although there are over 16,000 Web-APIs in ProgrammableWeb, only 1,525 of them appear in one or more mashups. We found that the Google Map Web-API takes a center stage in the affiliation network, attracting 2,072 edges/mashup-consumption, which account for about 35 percent of the total mashups in the ecosystem. As shown in Table 2, Popular social media Web-APIs such as Twitter, Youtube, Flickr, and Facebook also appear 663, 557, 484, and 377 times respectively in the network. We also found that less than 7% of the Web-APIs involved in the network are consumed more than 100 times, and over 47% of the Web-APIs are used less than 4 times.

The complete affiliation network is visualized using the Force-Atlas 2 layout in Gephi[3] as shown in Fig. 2. The hubs as listed in Table 2 are clearly visible in the figure as disks with Google-Maps API being the largest one sitting at the bottom.

[3] https://gephi.org/.

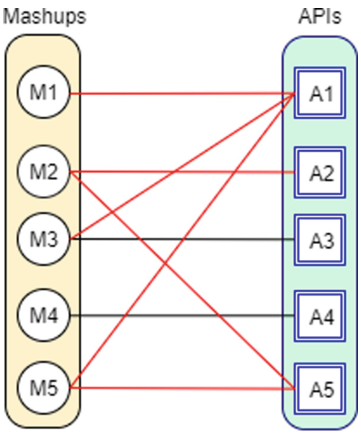


Fig. 1. Illustration of the Mashup-API bipartite graph

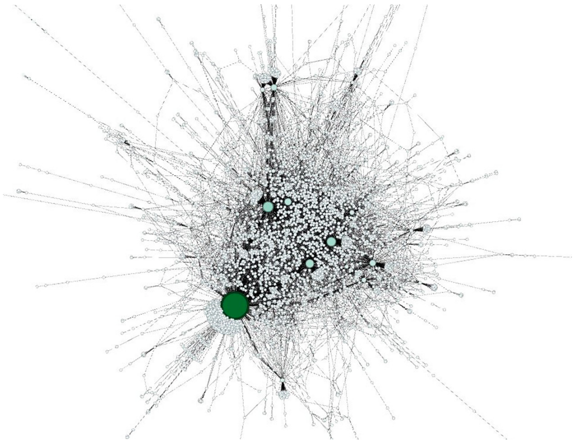


Fig. 2. Visualization of the Mashup-API affiliation network

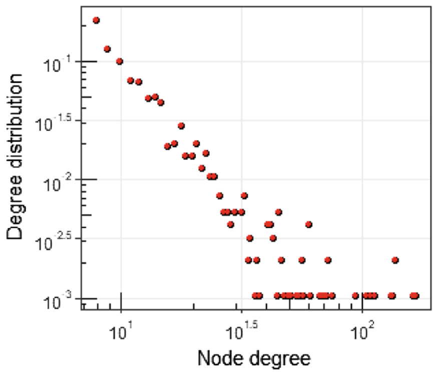
3.3 Web-API Nodes Degree Distribution

An integral part of analyzing the topology of a network is the plotting and fitting of its degree distribution $p(k)$. Networks with long-tailed degree distribution that follows a power-law are known to exhibit the *scale-free* topology. Most real networks such as the internet, WWW and the citation network are *scale-free* networks [3]. On the other hand, networks with exponentially-decaying-tail degree distribution are collectively referred to as *exponential Networks*.

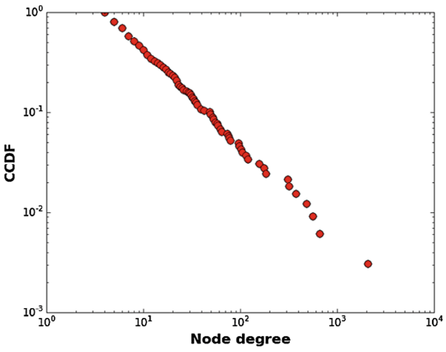
Plotting. To gain insight into the popularity of Web-APIs, we plot the degree distribution of the 1,525 Web-APIs based on their degrees in the affiliation network. As shown in Fig. 3, both the PDF (Probability Density Function) in log-log scale, linear binning, and the CCDF (Complementary Cumulative Distribution Function) in log-log scale are plotted.

In Fig. 3a, the small degree region demonstrates a log-linear relation between $p(k)$ and k ($\log p(k) \sim -\gamma \log k$, or $p(k) \sim k^{-\gamma}$), which is a typical feature of the scale-free network; while a plateau is formed at the large k region as typically we have only one copy of each large-degree node and this plateau affects our ability to estimate the degree exponent γ [3]. One way to extract information from the tail of the distribution is to use the CCDF (Fig. 3b), which enhances the statistical significance of the large-degree region, and if $p(k)$ follows the power-law, then the CCDF is also power-law: $P(k) \sim k^{-\gamma+1}$.

Fitting. In order to determine the best fit for the Web-API degree dataset, we first fit the data to four classical models including *Power-law*, *Exponential*, *Log-normal*, and *Poisson*. Figure 4 shows the result of the fitting when $k_{min} = 4$. We can see that both the power-law and the log-normal offer a good fit to the data, while the exponential and the Poisson fit poorly to the data.



(a)



(b)

Fig. 3. Degree distribution plot of the Web-APIs nodes in the affiliation network 3a shows the Log-log plot (linear-binning) of the Web-APIs degree distribution 3b shows the CCDF plot of the degree distribution in log-log scale.

Table 3. Plausibility of fitting Power-law, Log-normal, Exponential, and Poisson models to the Web-API degree data

Parameter	Power-Law	Exponential	Log-normal	Poisson
γ	2.200837	-	-	-
p-value	0.783	0.000	0.667	0.000

To quantitatively measure the plausibility of each distribution, next we conducted a *goodness-of-fit* test based on the Kolmogorov-Smirnov (KS) distance which measures the difference between the model and the empirical data, and a *p-value* $\in [0, 1]$ is calculated to measure the model plausibility. The closer p is to 1, the more likely that the difference between the model and the empirical data is attributed to statistical fluctuations alone. If p is very small, the model is not a good fit to the empirical data [3].

Table 3 shows the resultant *p-values* for each distribution. Clearly, power-law is the most plausible fit (*p-value* = 0.783) and next to it is log-normal (0.667); for both exponential and Poisson, the *p-value* is zero.

Exponent Estimating. In the above, we have justified that the power-law model provides the best fit to our data, next we use MLE (Maximum Likelihood Estimation) to estimate the scaling parameter/degree exponent γ [17]:

$$\hat{\gamma} = 1 + n\left[\sum_{i=1}^{n} \ln \frac{k_i}{k_{min}}\right]^{-1} \tag{1}$$

where k_i, $i = 1 \dots n$ are the observed values of k such that $k_i \geq k_{min}$, k_{min} represents the minimum degree of node in the network.

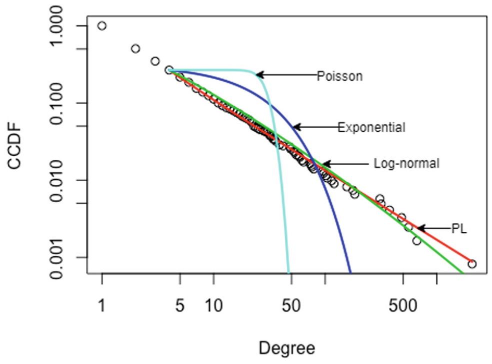


Fig. 4. Fitting Power-law (PL), Log-normal, Exponential, and Poisson models to the Web-API degree data

The assumption for estimating the parameter is that $\gamma > 1$, since the case of $\gamma \leq 1$ does not exist in real world [8].

When $k_{min} = 1$, the appropriate estimator for γ was given as:

$$\frac{\zeta'(\hat{\gamma})}{\zeta(\hat{\gamma})} = -\frac{1}{n}\sum_{i=1}^{n} \ln k_i \tag{2}$$

where $\zeta(\hat{\gamma})$ is the Riemann Zeta function.

Otherwise, when $k_{min} > 1$, the appropriate estimator for γ is:

$$\frac{\zeta'(\hat{\gamma}, k_{min})}{\zeta(\hat{\gamma}, k_{min})} = -\frac{1}{n}\sum_{i=1}^{n} \ln k_i \tag{3}$$

Using the method described in [8], which is also based on the KS distance, we can find the optimal k_{min} with respect to each data point and select the value that gives the minimal KS distance between the CCDF of our data and the fitted model. The resultant γ value for the CCDF is around 2.2 (or the γ value for the PDF is 3.2), which is close to that of the Internet ($\gamma = 3.42$) [3].

3.4 Measuring Preferential Attachment

Real-world networks reach their current size by adding new nodes to the network progressively, and a common phenomenon occurs, where new nodes tend to connect to existing nodes with high degree. This phenomenon is called *Preferential Attachment (PA)* [4]. If the probability that a newly arrive node connects to an existing node i is proportional to the degree of that node k_i, or

$$\Pi(k_i) = \frac{k_i}{\sum_j k_j} \tag{4}$$

Table 4. Preferential attachment measurement

Node span	α (Newman)	α (PAFit)
10	0.97 ± 0.05	1.09 ± 0.06
20	0.96 ± 0.05	1.08 ± 0.06
50	0.95 ± 0.07	1.06 ± 0.07
100	0.94 ± 0.06	1.05 ± 0.08
Monthly	0.96 ± 0.09	1.03 ± 0.09

then we call it *linear-PA*. The combination of growth and linear-PA play a critical role in shaping a network's topology and are responsible for the emergence of the scale-free property [3].

We can use the exponent α to classify different types of PA

$$\Pi(k) \sim k^{\alpha} \tag{5}$$

if α is 1, then PA is *linear*; if α is less than 1, then PA is *sub-linear*; otherwise PA is *super-linear* [3].

We aim to detect the presence of PA in the Web-API node set of the affiliation network of ProgammableWeb and also measure its α value. To do so, we can examine the degree increase of a node i between a fixed span Δt: $\Delta k_i = k_i(t + \Delta t) - k_i(t)$. For example, if $\Delta t = 5$, $k_i(t + \Delta t)$ is the degree of node i after five new nodes joined the affiliation network. The relative change $\Delta k_i / \Delta t$ should follow

$$\frac{\Delta k_i}{\Delta t} \sim \Pi(k_i) \tag{6}$$

Actually, to reduce the noise we can measure the cumulative preferential attachment:

$$\pi(k) = \sum_{k_i=0}^{k} \Pi(k_i) \tag{7}$$

We employ both the *PAFit* method [20] and Newmans's method [18] to estimate PA. As we can see in Table 4, Node Spans 10, 20, 50, 100, and monthly all output consistent results of $\alpha \approx 1$, which demonstrates the existence of linear-PA, or scale-free property, of Web-APIs in the ProgrammableWeb affiliation network.

4 Constructing an Evolving Web-API Network

In this section, we propose a complex-network-based approach to constructing an evolving social network for Web-APIs. We first discuss the limitation of the projection-based approach that projects the affiliation network to an one-mode API-API network; then, we present the evolving network model, and the strategy and procedure we use to construct the social network for ProgrammableWeb APIs; then we discuss the topological property of the constructed network; and finally we discuss an application of the constructed network in service discovery.

4.1 Limitation of the Projection-Based Approach

In order to build a social network for service ecosystems such as ProgrammableWeb, a simple approach is directly applying one-mode projection to mashup-API affiliation network to derive an API-API network. This approach has been used in [13,16]. Figure 5 illustrates how to project an affiliation network onto an one-mode API-API network.

The limitation of this approach is apparent: *Only Web-APIs used in mashups (suppose every mashup contains at least two Web-APIs) will appear on the projected network.* For example, the ProgrammableWeb affiliation network contains only 1,525 Web-APIs, which is less than 10% of the total 16,138 Web-APIs on the registry. Furthermore, the popularity/number-of-links of a Web-API node on the projected network is discounted as there are mashups that use only one Web-API, and such links are not counted in the projected network (for example the link between M_5 and A_6 in Fig. 5).

4.2 Network Model and Construction Strategy

As discussed in Sect. 3.4, the combination of *growth* and *PA* are the two generic mechanisms that drive many real-world networks, and we have validated the presence of both growth and PA in the ProgrammableWeb affiliation network. Based on that, we aim to build a growing/evolving network of Web-APIs that preserve both the topology properties of affiliation network and the popularity information of the Web-API nodes, while including all the Web-APIs in the ecosystems.

We base our model on the Barabási-Albert evolving network model [3]. For the *growth* aspect, it involves continuous addition of new nodes (Web-APIs) into the network, therefore increasing the number of nodes in the network throughout its life span. To do so, first, we initialize the network, starting with fully connected m_0 number of nodes. At every time step, we add a new node with m links.

To incorporate *PA*, we dynamically estimate the probability that a link of the new node connects to an existing node i depends on its degree k_i using the linear-PA equation (Eq. 4) described in Subsect. 3.4.

Figure 6 illustrate the network's growth and PA mechanism (assuming $m_0 = 4, m = 1$): at time 0, four fully connected nodes form the initial network; at time 1, *Node 5* joins the network; based on linear-PA, as *Node 1–4* each has the same degree, each of them has the same probability to attract *Node 5* to connect to it, which is 3/12, or 1/4; at time 2, based on linear-PA, the probabilities for *node 1–5* to attract the new node are $[3/K, 3/K, 4/K, 3/K, 1/K]$ where $K = 13$ is the total degree of the network and the numerator is the degree of each existing node.

Based on the growth and the linear-PA, older nodes always have better chance to attract links than newer nodes as they have better degree/popularity. Specific to ProgrammableWeb, to preserve the popularity information in the affiliation network, we need to define a strategy on *when* a Web-API joins the growing

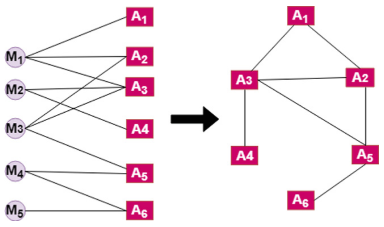


Fig. 5. Affiliation network projection

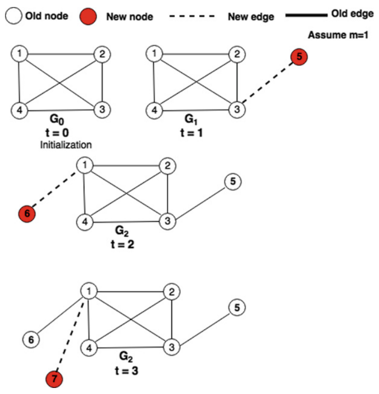


Fig. 6. Illustration of Web-API network growth procedure

network; i.e., we need to sort the full list of Web-APIs and put them onto the network one-by-one (or step-by-step) based on their position in the sorted list. A simple strategy is stated below:

– First, We sort the Web-API nodes based on their degree (or popularity) in the affiliation network in a descending order so that higher-degree nodes are in the front to produce L_1;
– Then, for Web-APIs that do not appear in the affiliation network we sort them based on their date of publication/birth in an ascending order so that older nodes are in the front to produce L_2;
(it is worth noting that *date-of-birth* is also used in [19] as a measure of popularity.)
– Finally, we just append L_2 in the end of L_1 and put the node into the network one-by-one based on their order in the list.

The complete network construction procedure is described below:

Procedure
Input Parameters (N: number of nodes, m_0: number of initial nodes, m: number of links added at each time step)

1. Creating node list: Sort Web-APIs nodes based on popularity and date-of-birth;
2. Initializing network: Start with a fully connected m_0 number of most popular nodes;
3. Growth: At each time step, a new node with m number of links is added and connected to m number of already existing nodes in the network, where $m \leq m_0$;

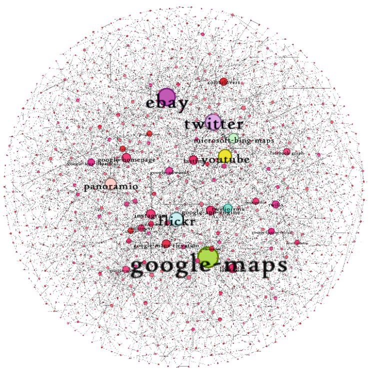


Fig. 7. Overview of the Web-API network

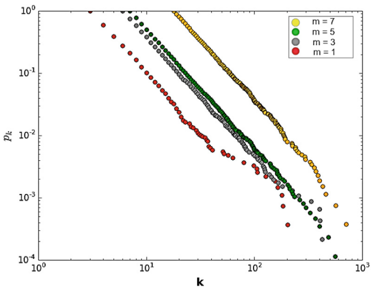


Fig. 8. Degree distribution of the Web-API Network; from bottom to top: m = 1 (red), m = 3 (grey), m = 5 (green) and m = 7 (yellow). (Color figure online)

4. Preferential attachment: With probability $\Pi(k_i)$, the new node connects to an already existing node i with degree (k_i). Probability $\Pi(k_i)$ is estimated dynamically based on Eq. (5).
5. After all N nodes join the network, we obtain the popularity-based, evolving Web-API network.

An overview of the constructed ProgrammableWeb API network containing all its Web-APIs is shown in Fig. 7, with popular nodes labelled. Next, we analyze the topological property of this network.

4.3 Topological Properties of the Web-API Network

In this subsection, we discuss three main topological property of the constructed Web-API network in terms of *degree distribution*, *network diameter*, and *clustering coefficient.*

Degree Distribution. Figure 8 shows the degree distribution (log-binning) with $N = 16138$, $m_0 = 4$, and $m = 1$(red, $\gamma = 2.782$), $m = 3$ (grey, $\gamma = 2.823$), $m = 5$ (green, $\gamma = 3.05$) and $m = 7$ (yellow, $\gamma = 3.002$).

Theoretically, we can use the continuum theory [3] to estimate the degree exponent of the degree distribution

$$p(k) \approx 2m^{1/\beta} k^{-\gamma} \tag{8}$$

where $\beta = 1/2$ is the dynamical exponent and $\gamma = 1/\beta + 1 = 3$.

So the constructed Web-API network is a scale-free network with degree exponent $\gamma \approx 3$.

Network Diameter. Network diameter d is the maximum of the shortest distances between any two nodes. If a network's diameter is proportional to $\ln N$, then it is a *small-world network* [24]. Theoretically, the expected value of the diameter of the Web-API network is [3]:

$$\langle d \rangle \sim \frac{\ln N}{\ln \ln N} \tag{9}$$

If we plug in $N = 16138$, we get $\langle d \rangle \approx 4.27$. For $m = 5$ and $m = 7$, the actual diameters are 6 and 5 respectively, which are consistent with the expected value. As d grows slower than $\ln N$, the Web-API network is *ultra-small.*

Clustering Coefficient. A clustering coefficient measures the density of links in a node's immediate neighborhood, and the average clustering coefficient of a network can be obtained by averaging over all its nodes. Theoretically, the average clustering coefficient of the Web-API network is [3]:

$$\langle C \rangle \sim \frac{(InN)^2}{N} \tag{10}$$

If we plug in $N = 16138$, we get $\langle C \rangle \approx 0.006$. For $m = 5$ and $m = 7$, the actual clustering coefficient are 0.005 and 0.007 respectively, which are consistent with the theoretical value.

4.4 Applying Web-API Network in Service Discovery

In this section, we discuss one application of the Web-API network in service discovery. Consider a new service consumer, who wants to leverage different Web-APIs from different domains (say Dictionary, Translation and Social) to create a mashup that allows users to find the meaning of a word in English, translate to French and post it on social media. In order to navigate through the Web-API network and discover require Web-APIs, a *link-as-you-go* approach [7] can be used. As illustrated in Fig. 9, the user can *zoom-in* to the network, starting from *Twitter API* and then navigate through by following social links to discover *Oxford dictionary (OX)* and then *Google Translate (GT)* API. Such user activity pattern is very similar to surfing the WWW; just in our case the user is surfing the service network.

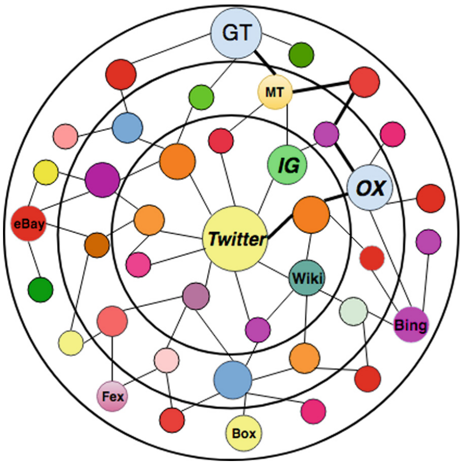


Fig. 9. Web APIs Discovery with *link-as-you-go* approach.

5 Conclusion and Future Work

In this paper, we propose an evolving-network-based approach for constructing a social network for Web-APIs based on their popularity in the service ecosystem. We achieve this by first studying the underlying topology of the ProgrammableWeb service ecosystem from the complex network perspective and analyze the popularity distribution and preferential attachment of Web-APIs in the ecosystem; then, we incorporate our findings into the construction of an evolving social network of Web-APIs using the well-established Barabási-Albert model. In the future we want to do further research in the following directions: (1) As the construction strategy of the network is purely based on node popularity, we plan to investigate other strategies that consider factors such as node fitness and similarity; (2) The clustering coefficient of the constructed network is low compared to real-world networks; we plan to investigate strategies that can build better clustered networks.

References

1. Albert, R., Jeong, H., Barabási, A.-L.: Internet: diameter of the world-wide web. Nature **401**(6749), 130 (1999)
2. Barabási, A.-L.: Network science: luck or reason. Nature **489**(7417), 507 (2012)
3. Barabási, A.-L.: Network Science. Cambridge University Press, Cambridge (2016)
4. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
5. Barros, A.P., Dumas, M.: The rise of web service ecosystems. IT Prof. **8**(5), 31–37 (2006)
6. Bianconi, G., Barabási, A.-L.: Competition and multiscaling in evolving networks. EPL (Europhys. Lett.) **54**(4), 436 (2001)

7. Chen, W., Paik, I., Hung, P.C.K.: Constructing a global social service network for better quality of web service discovery. IEEE Trans. Serv. Comput. **8**(2), 284–298 (2015)
8. Clauset, A., Shalizi, C.R., Newman, M.E.J.: Power-law distributions in empirical data. SIAM Rev. **51**(4), 661–703 (2009)
9. Fallatah, H., Bentahar, J., Asl, E.K.: Social network-based framework for web services discovery. In: 2014 International Conference on Future Internet of Things and Cloud (FiCloud), pp. 159–166. IEEE (2014)
10. Feng, Z., Lan, B., Zhang, Z., Chen, S.: A study of semantic web services network. Comput. J. **58**(6), 1293–1305 (2015)
11. Hébert-Dufresne, L., Allard, A., Marceau, V., Noël, P.-A., Dubé, L.J.: Structural preferential attachment: network organization beyond the link. Phys. Rev. Lett. **107**(15), 158702 (2011)
12. Huang, K., Fan, Y., Tan, W.: An empirical study of programmable web: a network analysis on a service-mashup system. In: 2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, 24–29 June 2012, pp. 552–559 (2012)
13. Huang, K., Fan, Y., Tan, W.: Recommendation in an evolving service ecosystem based on network prediction. IEEE Trans. Autom. Sci. Eng. **11**(3), 906–920 (2014)
14. Kavitha, R., Anuvelavan, S.: Weather master: mobile application of cyclone disaster refinement forecast system in location based on gis using geo-algorithm. Int. J. Sci. Eng. Res. **6**, 88–93 (2015)
15. Lee, J., Niko, D.L., Hwang, H., Park, M., Kim, C.: A GIS-based design for a smartphone disaster information service application. In: 2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering (CNSI), pp. 338–341. IEEE (2011)
16. Lyu, S., Liu, J., Tang, M., Kang, G., Cao, B., Duan, Y.: Three-level views of the web service network: an empirical study based on programmableweb. In: 2014 IEEE International Congress on Big Data (BigData Congress), pp. 374–381. IEEE (2014)
17. Muniruzzaman, A.N.M.: On measures of location and dispersion and tests of hypotheses in a pare to population. Calcutta Stat. Assoc. Bull. **7**(3), 115–123 (1957)
18. Newman, M.E.J.: Clustering and preferential attachment in growing networks. Phys. Rev. E **64**(2), 025102 (2001)
19. Papadopoulos, F., Kitsak, M., Serrano, M.Á., Boguná, M., Krioukov, D.: Popularity versus similarity in growing networks. Nature **489**(7417), 537 (2012)
20. Pham, T., Sheridan, P., Shimodaira, H.: PAFit: a statistical method for measuring preferential attachment in temporal complex networks. PloS one **10**(9), e0137796 (2015)
21. Pham, T., Sheridan, P., Shimodaira, H.: Joint estimation of preferential attachment and node fitness in growing complex networks. Sci. Rep. **6**, 32558 (2016)
22. Wang, H., Feng, Z., Chen, S., Xu, J., Sui, Y.: Constructing service network via classification and annotation. In: 2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE), pp. 69–73. IEEE (2010)
23. Wang, J., Chen, H., Zhang, Y.: Mining user behavior pattern in mashup community. In: IEEE International Conference on Information Reuse and Integration, IRI 2009, pp. 126–131. IEEE (2009)
24. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393**(6684), 440 (1998)
25. Weiss, M., Gangadharan, G.R.: Modeling the mashup ecosystem: structure and growth. R&D Manag. **40**(1), 40–49 (2010)

Stigmergic Service Composition and Adaptation in Mobile Environments

Andrei Palade(✉), Christian Cabrera, Gary White, and Siobhán Clarke

School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
{paladea,cabrerac,whiteg5,siobhan.clarke}@scss.tcd.ie

Abstract. Users within a limited geographic area can form service-sharing communities using the services deployed on their mobile devices. Creating Quality of Service (QoS) optimal service compositions in such decentralised and dynamic environments is challenging because of the service providers' mobility and the inherent dynamism in the available services. Existing proposals for mobile environments either use template-matching composition or require a-priori knowledge about the QoS objectives' weights, which limits the composition's flexibility in such environments. This paper presents a stigmergic-based approach to model the decentralised, flexible and dynamic service interactions of providers in a mobile environment. A nature-inspired optimisation mechanism is used to approximate the set of QoS optimal compositions that result from these interactions. To facilitate adaptation of the composite during execution, we introduce a procedure that encourages the exploration of service composition configurations that emerge as a result of providers' mobility. We evaluate the performance of the proposed approach with a no-adaptation variant, a Dijkstra-based, a Greedy and a Random approach. The results show that the proposed approach can obtain superior solutions compared with current optimisation methods for flexible service composition in mobile environments at the cost of increased overhead.

Keywords: Stigmergic · Flexible · QoS-aware service composition

1 Introduction

Recent developments in wireless communications technologies allow the mobile users within a limited geographical area to share services deployed on their mobile devices to form a service-sharing community, which can enable new service-based applications. Such applications are generally distributed among several devices since a single device is not capable of executing an entire application. For example, a smart route planner for users in a mall can be opportunistically provisioned by exploiting the ad-hoc interactions of a personal-shopper's phone, a nearby car's satellite navigator, and the mall's information kiosk to enable the requested functionality [8,17]. Apart from the functional requirements, this composition process should create solutions with the best possible

QoS. In the envisioned environment this is challenging because services may fail, their qualities may decline because of the mobility of devices, or functionally similar services with better QoS may appear during the execution of the composition.

Most service composition proposals use a template-matching approach to provision QoS optimal configurations. This approach relies on functionally-equivalent services, but with different QoS, to be optimally assigned to a predefined workflow of tasks. Such an exactly-defined request affects the composition' flexibility when the environment is dynamic [8]. The services can be combined in an input-output dependency graph, where each node corresponds to one service, and an edge is a matching between two connected services. Planning-based algorithms can be used to find paths in this graph using goal-driven approaches [8]. Without QoS consideration, the shortest path is normally identified. With QoS considerations, finding an optimal path is difficult. The shortest path might not be optimal, since a longer path may have better QoS. The providers' mobility and the dynamism in services deployed on their devices may affect the search [30].

The existing QoS optimisation mechanisms for planning-based service composition in mobile environments require a-priori knowledge about QoS objectives' weights, and use exact or heuristics methods to find one optimal solution. This requirement reduces the composition' flexibility, and does not allow for different quality trade-offs to be thoroughly investigated. Also, the exact or heuristics methods trade optimality for computational efficiency, or vice-versa [7]. Nature-inspired metaheuristics address these limitations by constructing one or more solutions and gradually improve these through an iterative process. However, the existing proposals are limited to template-matching composition [16] or require a centralised perspective [26,28]. Mobile environments need flexible interactions, which may benefit from decentralised processing.

Stigmergic coordination has been used successfully in finding QoS optimal routes in networks with frequent link disconnections, and rapid topology changes such as Mobile (Vehicular) Ad-Hoc Networks [25]. However, the existing stigmergy-based proposals are limited to template-matching composition [16]. An efficient and effective QoS optimisation method for planning-based service composition that can quickly find a set of QoS optimal compositions in a mobile environment is required to provide higher utility with acceptable overhead.

This paper presents a stigmergic-based approach to model the decentralised, flexible service interactions in a mobile environment. New compositions (with better QoS) may emerge as a result of providers' mobility. The main contributions of this paper are: (1) a nature-inspired optimisation method to approximate the set of QoS optimal compositions; (2) to facilitate composition adaptation during execution, a procedure that encourages the exploration of new service composition configurations that emerge as a result of providers' mobility. We evaluate the performance of the proposed approach and compare the results with a no-adaptation variant, a Greedy, a Dijkstra-based and a Random approach. The proposed approach can obtain superior solutions compared with existing optimisation methods for flexible service composition in mobile environments.

The paper defines the QoS optimisation problem in planning-based composition in Sect. 2, presents the stigmergic modelling of composition, the proposed QoS optimisation and adaptation procedure in Sect. 3, the implementation and evaluation of these methods in Sect. 5, and the evaluation results in Sect. 6. Section 7 presents the related work, and Sect. 8 concludes the paper.

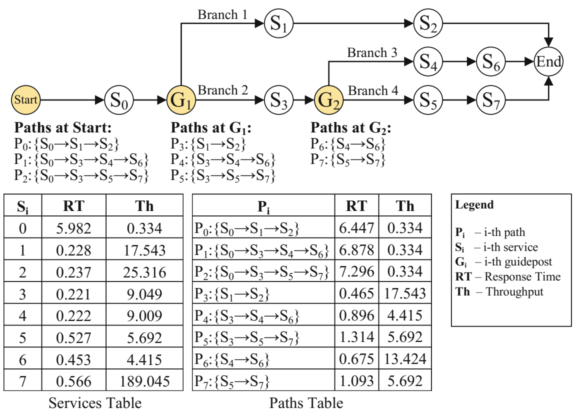
2 Problem Description

The services deployed on mobile devices in a geographic area such as a mall [8] or an university campus [9] can be used to create new service-based applications. To enable such applications, services can be clustered based on their syntactic or semantic similarity using a Service-Specific Overlay Network [1]. Goal-driven service composition using planning-based algorithms can automatically solve a user's request using the available services in these clusters [8].

Figure 1 shows the result to a user's request, which is a service dependency graph with three possible service composition solutions (paths): P_0, P_1 and P_2. In this example, the inputs of service S_2 can be satisfied by the outputs of service S_1. When two or more nodes (services) in the graph share the same input, an execution guidepost is created [8]. An execution guidepost $G_i = <R_{id}, D>$ is a split-choice control element in the composition process that maintains a set of execution directions (branches) D for a composition request R_{id}. Each element in the set D is defined as $d_j = <id, w>$, where j $\leq |D|$, w represents the set services in the branch and id represents the identifier of the branch. For example, at G_2 two paths can complete the execution: P_6: $\{S_4 \rightarrow S_6\}$ and $\{P_6 \rightarrow S_7\}$. In the envisioned environment, new service compositions can emerge as a result of providers' mobility. New nodes can be attached at runtime to the graph if they can provide the required inputs/outputs [8,9].

Apart from the functional requirements, a composite service needs to satisfy non-functional requirements. Given a set of QoS objectives, the challenge is how to efficiently construct a composite service that maximises the satisfaction of these QoS requirements. Such multi-objective optimisation problems are characterised by several objectives that have to be optimised simultaneously. When a user's preferences about each objective weight are known, the task is to find the service combination that can optimise a global utility function. However, when this information is not available, compromises among multiple objectives need to be made to identify the set of Pareto-optimal solutions. A Pareto-optimal solution is a solution that is not dominated by any other solution in the set. The population of such solutions form the Pareto-optimal set (Pareto-front) [14,23], and are generally identified using non-dominated sorting techniques [15].

While the QoS values of each service in the graph can generally be estimated using existing QoS prediction mechanisms [27], finding valid paths in this dynamic graph is challenging. The QoS values of services may be time-dependent, or services may become un-available because of the mobility of service providers. In contrast to optimisation towards a static optimum, the goal in a dynamic environment is to track as closely as possible the dynamically changing optima [4].



S_i	RT	Th
0	5.982	0.334
1	0.228	17.543
2	0.237	25.316
3	0.221	9.049
4	0.222	9.009
5	0.527	5.692
6	0.453	4.415
7	0.566	189.045

Services Table

P_i	RT	Th
P_0: {$S_0 \to S_1 \to S_2$}	6.447	0.334
P_1: {$S_0 \to S_3 \to S_4 \to S_6$}	6.878	0.334
P_2: {$S_0 \to S_3 \to S_5 \to S_7$}	7.296	0.334
P_3: {$S_1 \to S_2$}	0.465	17.543
P_4: {$S_3 \to S_4 \to S_6$}	0.896	4.415
P_5: {$S_3 \to S_5 \to S_7$}	1.314	5.692
P_6: {$S_4 \to S_6$}	0.675	13.424
P_7: {$S_5 \to S_7$}	1.093	5.692

Paths Table

Fig. 1. A service dependency graph with 3 available service compositions (paths): P_0, P_1 and P_2. Paths P_3, P_4, P_5, P_6, P_7 are alternative paths that can be selected at execution time. Services table shows the QoS for each service i in the graph, and path table shows the aggregated QoS for each path. These values are aggregated using the formulas in Table 1.

Prior optimisation methods for planning-based service composition in mobile environments used exact or heuristics algorithms to address this problem. Exact algorithms can find the optimal solution, but lack scalability because of their exponential complexity. Heuristics have polynomial complexity, but do not offer any worst-case guarantees on how close the QoS values of the returned solutions come to the QoS values of the Pareto-optimal ones [23]. Also, the existing proposals require a centralised perspective, which, in the envisioned environment, is a single point of failure or has the potential to cause processing and communication bottlenecks because of frequent state updates. Mobile environments need flexible interactions, which may benefit from decentralised processing.

Table 1. QoS aggregation formulas.

Parameter	Aggregation formula	Description
RT (Response time)	$\sum_{i=1}^{n} rt_i$	rt - response time of i-th service
Th (Throughput)	$min(th_i)$	th - throughput of i-th service

3 QoS Optimisation Mechanism

Stigmergic coordination, exhibited by the social insects to coordinate their activities, can be used to enable service-based applications in mobile environments. A set of mobile software agents interact with the environment by encoding application-specific information as pheromone to achieve certain tasks. These agents achieve collaboration and self-organisation by exchanging pheromone and performing several pheromone operations. A service is composed by a group of service agents that can form an organisation in order to collaborate. We use this abstraction to model the decentralised, flexible service interactions in a mobile environment. We are extending the notation used by Moustafa et al. [16] to account for the forward-backward motion of the mobile agents, which is used to reinforce the optimal paths [21].

Definition 1. ***Service Agent (sa)***. *A tuple sa = <id, F>, where id is the agent's identifier and F is the pheromone store used during composition.*

Definition 2. ***Pheromone Store (F)***. *A tuple F = <id, f>, where id is the identifier of the direction and f is the pheromone value associated with that direction.*

Definition 3. ***Memory Store (M)***. *A set* $M = \{sa_0, \ldots, sa_n\}$ *where each element in the set is a service agent.*

Definition 4. ***Mobile Agent (ma)***. *A tuple ma = <id, M>, where id is the identifier of each agent, and M is the memory store associated with that agent.*

The proposed QoS optimisation method is inspired by Ant Colony Optimisation [25]. The mechanism uses a decentralised architecture, which allows service composition configurations to adapt in mobile environments. A service composition request is satisfied by a requester service agent. Using a service discovery component, a user's request can be functionally modelled as a service dependency graph (Fig. 1). Each service in the service dependency graph is modelled using a service agent. A node in the graph is a service agent, and can function as a source, destination or intermediate node. The source and destination node can be the service requester agent. The intermediate nodes are the composition participants. Each intermediate node (service agent) stores a list of addresses of the service agents that can provide the next service in the graph. A pheromone level is associated with each service agent in the list. The quantity of pheromone is associated with the quality of the service provided by that service agent.

For each service composition optimisation task, a set of mobile agents is used. The mobile agents communicate through the pheromone store of each service agent. The mobile agents modify this value, to communicate the quality of solution to other mobile agents that visit the node. The number of mobile agents is task dependent. For simplicity, this number is set to the number of nodes in the service dependency graph. This number can affect the overhead, as well as the performance of the optimisation method. In this work we focus on the impact of the environment variables on the optimisation mechanism.

The mobile agents can iteratively traverse the graph in a forward-backward motion. The forward moving mobile agents use the intermediate nodes' probability routing tables to advance in the graph:

$$P_d = \frac{\tau_d^\alpha}{\sum_i^N \tau_i^\alpha} \tag{1}$$

where d is a branch in the set of branches D (as defined in Sect. 2), and τ_i is the pheromone level associated with the first node (service agent) from branch d. As they travel towards the destination node, the forward agents collect paths' QoS information. This information contains the QoS of each service component in the path and is added to the agent's memory store and aggregated. When all the forward agents reach the destination node, the identified paths are filtered using a non-dominated sorting technique [15] to identify the Pareto-optimal paths (optimal solutions). At the destination node, forward agents become backward agents only if their identified path is Pareto-optimal. As backward agents move in the reverse path, the intermediate nodes modify their pheromone store using: $\tau_i = \tau_i + \tau_i * (Q/l_i)$ (positive reinforcement) where τ_i is the pheromone level, Q is a constant and l is the length of the i-th path. To allow for new paths to emerge, an evaporation procedure $\tau_i = \tau_i * (1 - \rho)$ is used. Heuristic information is not used in this work to avoid expensive network state dissemination.

4 Adaptation Procedure

To facilitate composition adaptation during execution, a procedure that encourages the exploration of new service composition configurations that emerge as a result of providers' mobility is required. The stigmergic optimisation mechanism can converge to the optimal solution by selecting the path with the highest pheromone value. However, the envisioned environment introduces two additional challenges to finding the optimal configuration: (1) services in the service dependency graph may join or leave at any time because of the mobility of the providers; (2) the QoS of the services offered by such providers may vary in time. Therefore adaptation procedure must be promoted to effectively address these issues.

To facilitate composition adaptation during execution, and allow other parts of the solution space to be explored, we use a pheromone smoothing scheme. This scheme allows identified optimal paths to be reinforced with lesser pheromone [22]. This allows for other (some previously marked as non-optimal) paths to be explored. Each element in the pheromone store is updated as follows:

$$\tau_{ij'} = \tau_{ij} + \delta * (\tau_{max} - \tau_{ij}) \tag{2}$$

where $\tau_{ij'}$ is the new pheromone level, τ_{ij} is the previous pheromone level, δ is a smoothness coefficient and τ_{max} is the maximum pheromone level that can be associated with this path.

While evaporation adopts a uniform discount rate for every node, the pheromone smoothing technique places a greater reduction in the reinforcement of pheromone concentration on the optimal path(s), by reducing the rate of reinforcement of dominant paths [21]. Previous research showed that such technique can prevent the generation of dominant paths. Stuzle and Hoos [22] introduced a similar mechanism to solve the travelling salesman problem, but in a stationary environment. Also, this approach showed promising results in a dynamic, but stationary Wireless Sensor Networks environment [5].

In the envisioned environment, the topology of the service dependency graph can change continuously. The devices on which the services are deployed may move out of range, or may be power depleted. If these services are part of the optimal service composition paths, when the environment changes, other paths in the service dependency graph may become optimal during the execution of the composition. Also, new service composition may emerge because of the mobility of users, which may have more optimal QoS.

In this paper, we are proposing this approach as an adaptation handling procedure for planning-based service composition in mobile environments. Each service agent is allocated with two pheromone update rules, which will be executed periodically. These rules are the evaporation and pheromone smoothing of each element in the pheromone store.

5 Implementation and Evaluation

The proposed QoS optimisation method is implemented and evaluated using the Simonstrator platform [19]. The proposed QoS optimisation algorithm, both with and without the adaptation handling, produces a set of solutions for each traversal of the service dependency graph. First, we evaluate how well the proposed QoS optimisation mechanism approximates the set of Pareto-optimal solutions. We perform the evaluation with and without adaptation handling. Second, we measure utility of the produced solutions and the introduced overhead of the proposed QoS optimisation mechanism, with and without adaptation handling. We compare the results with a Dijkstra-based, a Greedy and a Random approach. The two objectives considered in this evaluation are the response time and the throughput, but multiple objectives can be considered such as battery lifetime of devices and service availability. We assume that each node in the service dependency graph stores the estimated QoS value.

5.1 Performance Metrics

PS (A) - Size of Dominated Space: This metric was introduced by Zitzler and Thiele [35] and indicates how well the Pareto-optimal set is approximated by the set of solutions A. Here, the set A contains only non-dominated solutions. The greater the size of the space dominated, the closer the solutions are to the Pareto-optimal set [26]. This metric is calculated as follows:

$$\begin{gathered} PS(A) = \frac{S(A)}{Max\ I * Max\ II}, \text{and} \\ S(A) = (Max\ I - x_1) * (Max\ II - y_1) + \sum_{i=2}^{n} (Max\ I - x_i) * (y_{i-1} - y_i) \end{gathered} \tag{3}$$

where $Max\,I$ and $Max\,II$ are the maximum values of Objective I and Objective II. These two values are used to create the Reference Point X_R. As an example, Fig. 2 shows the composition of set A, which is a Pareto front made of 5 points. The dominated space is given by the orange surface.

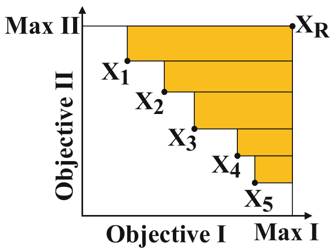


Fig. 2. PS (A) metric. Example of space dominated (in orange) by a given Pareto set (points X_1, X_2, X_3, X_4 and X_5) when two objectives are minimised. (Colore figure online)

Utility. This metric indicates the overall satisfaction of QoS requirements of a service composition configuration [2]. Certain QoS values such as response time are considered negative criteria and need to be minimised to increase user satisfaction, whereas others such as throughput are considered positive criteria and need to be maximised.

$$U_i min = \begin{cases} \frac{Q_i^{max} - Q_i}{Q_i^{max} - Q_i^{min}} \\ 1 \end{cases} \quad U_i max = \begin{cases} \frac{Q_i - Q_i^{min}}{Q_i^{max} - Q_i^{min}} & if Q_i^{max} - Q_i^{min} \neq 0 \\ 1 & if Q_i^{max} - Q_i^{min} = 0 \end{cases} \tag{4}$$

where Q_i is a QoS value, and Q_i^{min} and Q_i^{max} are the minimum/maximum QoS values available for Q_i. Equation 5 computes the utility of each service composition configuration as follows:

$$Utility_g = \sum_{1}^{n} U_i * W_i \tag{5}$$

where W_i represents the importance weights assigned to the i-th metric. Each weight is a number in the range [0, 1] and the sum of all weights is equal to 1. In this evaluation, all objectives carry an equal weight.

Overhead. The overhead measures the number of exchanged messages between the nodes used during the composition. This metric includes all the types of messages that are exchanged, including retransmission or probing messages.

5.2 Evaluated Algorithms

We chose several QoS optimisation mechanisms designed for planning-based service composition to compare their utility and overhead with our proposed model. Each algorithm was implemented in the simulator. Also, two versions of proposed QoS optimisation were formalised:

1. **ACO**: the implementation of the QoS optimisation mechanism described in Sect. 3, without the adaptation procedure.

2. **E-ACO**: an extension to **ACO**, which uses the proposed adaptation procedure introduced in Sect. 4.

An important limitation of the selected baseline algorithms is that they require user's QoS preferences and they produce a single solution, whereas the proposed QoS optimisation mechanism can does not have a requirement for user input (for QoS weights) and may produce a set of solutions. We use the utility metric to select a single solution from the set of Pareto-optimal solutions produced by ACO and E-ACO algorithms. We compare the utility values and overhead with the existing proposals for QoS optimal service composition in mobile environments:

1. **SimDijkstra**: An exact algorithm that uses a Dijkstra-based algorithm to find the shortest path in the service dependency graph [11]. The execution of the algorithm finishes when the end node is reached.
2. **GoCoMo**: A heuristics-based algorithm that uses a Greedy-based approach to select the path with the highest utility value to the user. It uses monitors to collect QoS of potential paths. Probes are sent regularly to update the QoS information from the leaf nodes to each guidepost in the service dependency graph [8].
3. **Random**: Randomly selects the next service in the service dependency graph. This algorithm shows the expected value of a random solution and provides a baseline that the specialised algorithms should easily outperform [13].

5.3 Test Case Generation

For the purpose of evaluation, we use a scenario based on an adaptive route planner application [8]. Such an application can be built using services provided by the available (mobile) devices in the environment. Many services can provide similar functionality and more than one service composition configuration may be available. The composition with the optimal QoS should be selected to maximise user satisfaction of non-functional requirements.

The response time and throughput of the final composite service were measured. These values are used to compute the utility of the final composition plan. For ACO and E-ACO, we also measure the size of the dominated space based on a reference point fixed using the best QoS values for response time and throughput. In this evaluation, the reference point was set to the highest throughput plus 5 units and the lowest response time minus 0.1 units. For each algorithm presented in Sect. 5.2 we also measured the ovearhed. To minimise the impact of external factors on our results, the presented algorithms were executed 100 times per problem instance to compute the averages.

5.4 Environment Setup

The mobile devices in the environment move at a speed of 7.5 to 13.5 m/s. The other environment's properties are: size - 1000*1000 m, communication range -

250 m, and movement model - Gauss Markov. Each device is set to offer one service. Each service has QoS values. We use a QoS dataset to initialise the QoS of each service participant, which consists of a matrix of response time and throughput values for 339 users by 5,825 services [34]. To address the dynamism in the services available in the environment, these QoS values are randomly changed after every service iteration by multiplying every QoS value with a random number in the interval [0.9, 1.1]. For simplicity, only QoS discrete numerical values are considered in this work.

In ACO and E-ACO a number of parameters need to be set before running the algorithm such as the number of mobile agents, the factor α, parameter Q, the pheromone coefficient ρ, and the initial pheromone level in each node. A larger number of mobile agents improves the convergence rate, however, overhead is introduced. We set the number of mobile agents to be equal to the number of nodes in the service dependency graph. The other parameters are initialised as follows: $\alpha = 0.9$, $\rho = 0.01$, $\tau_{initial} = 10.0$, and $Q = 10$. The adaptation procedure is initialised using $\tau_{max} = 20.0$ and $\delta = 0.5$. The smoothing rate is set to (evaporation rate) * 0.5.

6 Results

6.1 Size of Dominated Space

Figure 3 shows how the size of the dominated space achieved by ACO and E-ACO evolves as the number of iterations is increased. The size of the dominated space indicates how well the Pareto-optimal set is approximated by the identified solutions. The greater the size of the space dominated, the closer the solutions are to the Pareto-optimal set. This metric was defined in Sect. 5.1. We show the results of this evaluation for the cases when the number of available paths (size of the solution space) is 16, 32, 64, and 128. When 16 paths are available, the size of the dominated space for ACO is 63.32% after 10 iterations and decreases to 60.54% after 250 iterations, whereas for E-ACO maintains a level of 64%. When the number of paths increases to 32, ACO obtains 64.07% after 10 iterations and decreases to 60.56% after 250 iterations, whereas E-ACO slowly increases from 64.35% to 64.76%.

Both ACO and E-ACO increase the size of their dominated space as the number of available paths increases. For example, when 128 paths are available ACO achieves 67.59% and E-ACO achieves 66.96%. However, ACO maintains the same decreasing slope and achieves 64.51% after 250 iterations, whereas E-ACO achieves 68.45% after 250 iterations. We also report that when the size of the solution space is 64, ACO achieves 67.05% after 10 iterations and 62.90% after 250 iterations, whereas E-ACO achieves 66.96% after 10 iterations and 68.45% after 250 iterations.

6.2 Utility

Figure 4 shows the results of the utility evaluation as the number of available paths (service composition configurations) is increased. We show the results of

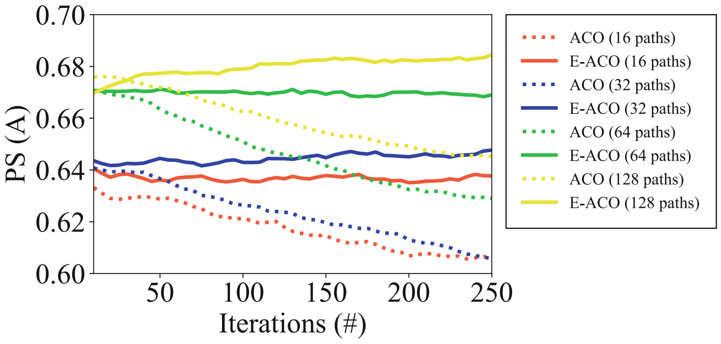


Fig. 3. Evolution of PS (A) metric (defined in Sect. 5.1) after 250 iterations for different number of paths (higher is better).

this evaluation for the cases when the number of available paths (size of the solution space) is 16 (Fig. 4a), 32 (Fig. 4b), 64 (Fig. 4c), and 128 (Fig. 4d). The utility is calculated using the configurations produced by ACO and E-ACO in the last 5 iterations, and the configurations produced by SimDijkstra, GoCoMo and Random. This metric was defined in Sect. 5.1. Both objectives have equal weights.

By increasing the number of iterations, two trends can be observed for ACO and E-ACO: (1) the utility of both increases as the number of paths increases, and (2) the utility of ACO is higher than the utility of E-ACO after 100 iterations. This difference decreases as the number of paths increases. When 16 paths are available, ACO achieves an utility of 86.58% and E-ACO achieves an utility of 80.38%. This decreases to 93.26% for ACO and 88.52% for E-ACO, when 128 paths are available. The SimDijkstra and Random algorithms have the lowest utility for almost all runs. GoCoMo performs comparably to ACO and E-ACO.

6.3 Overhead

Figure 5 shows the overhead (number of exchanged messages) after 250 iterations as the number of available paths in the service dependency graph varies. In GoCoMo, a global state mechanism periodically disseminates network state to all participating nodes, which explains the high overhead. This mechanism is not used in SimDijkstra and Random and the overhead introduced by these two algorithms is insignificant. However, the utility of solutions of these algorithms is considerably lower than the utility produced by ACO and E-ACO and GoCoMo. ACO and E-ACO introduce a higher overhead than the other approaches because of the large number of agent messages. This number is task dependent, and, for simplicity, it was set to the number of nodes in the service dependency graph. Tuning this parameter would likely reduce the overhead, though this needs to be verified.

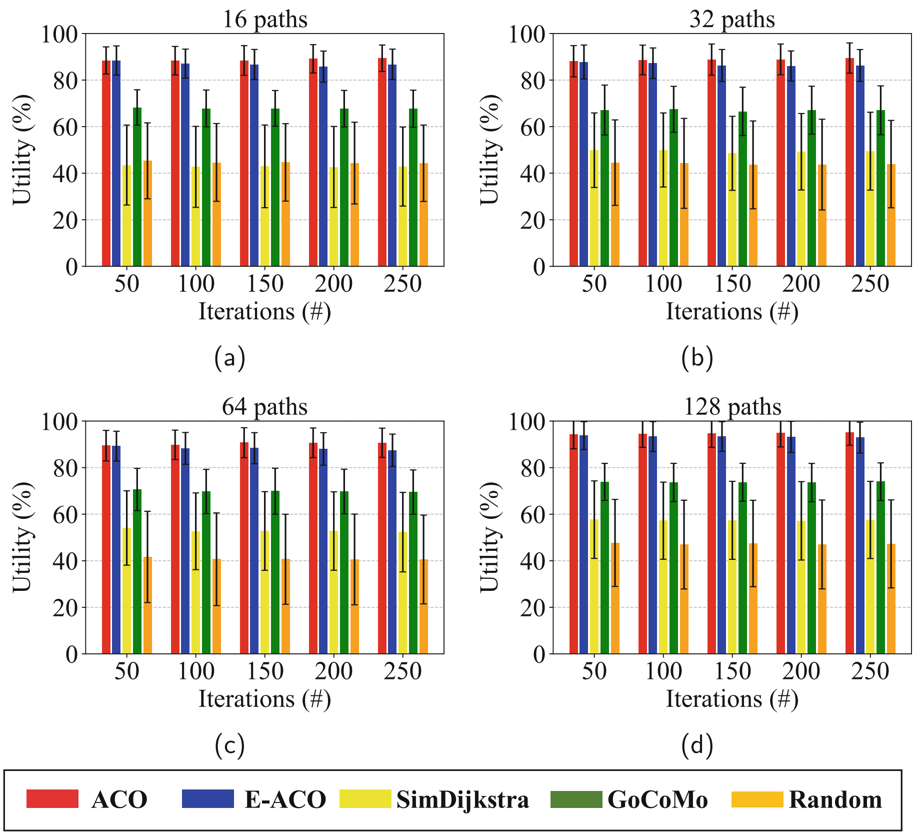


Fig. 4. The utility of solutions produced by the evaluated algorithms (higher is better) after 250 iterations for different number of paths. The results are averaged over 100 executions.

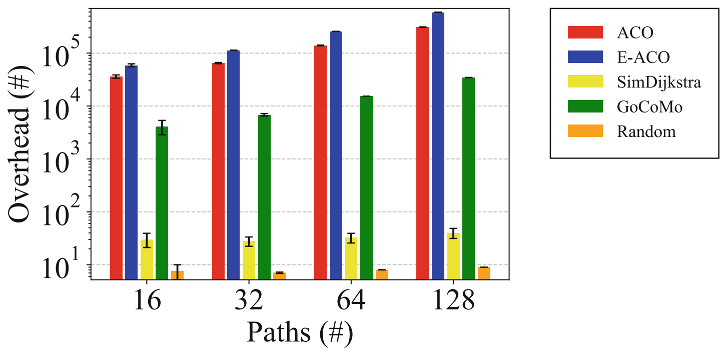


Fig. 5. The overhead when various number of paths are available (lower is better). The results are averaged over 100 executions.

7 Related Work

Most QoS-aware service composition mechanisms use a template-matching approach where the user's request is a predefined set of abstract tasks and the goal is to find an assignment of services to pre-defined abstract tasks to maximises a QoS-related user benefit. Such approaches used exhaustive search, genetic algorithms [6], linear and integer programming [2,3,32] to select, for each task, an optimal service candidate from a functionally-equivalent list of candidates.

Planning-based service composition was introduced to address the functional limitation of the template-matching model. However, the existing proposals have limited support for non-functional requirements in mobile environments. They use exact and heuristic approaches to find an optimal QoS composition plan. GraphPlan [30], SimDijkstra [11], and Rodriquez et al. [20] presented a Dijkstra-based algorithm to find compositions paths that satisfy users' QoS objectives subject to constraints. To improve resilience of the composition, Jiang et al. [11] return the top-k optimal paths. However, these works do not address the dynamic nature of a mobile environment. GoCoMo [8] and Kalasapur et al. [12] use a heuristics approach based on greedy selection to find the service composition configuration with the highest utility to the user. These planning-based service composition approaches require a-priori knowledge about the user's objectives, which reduces the flexibility of composition in a dynamic environment.

Nature-inspired metaheuristics have been designed to address challenges that are common in large-scale, dynamic and decentralised networks [10]. Such algorithms can support self-organisation, self-configuration, and collaboration in changing environmental conditions. They can dynamically adapt to ensure end-to-end communication between devices and provide efficient management of available resources. These algorithms can balance the trade-off between computation efficiency and optimality [21], and can obtain a set of solutions having two important properties: good convergence and diversity in solutions. The convergence represents the distance to the set of Pareto-optimal solutions. The diversity property refers to obtaining a uniform-spaced set of solutions to indicate a good exploration of the search space without losing any valuable information [14]. Such methods are approximate because they start from constructing one or more complete solutions and gradually improve these solutions during the iterative process. This makes it possible to track the quality of the solutions found during each iteration, and stop as soon as acceptable results are identified.

The existing stigmergic-based proposals for QoS-aware composition used ACO-based algorithms to identify the set of optimal solutions. MO_ACO [33] use an ideal vector with the best QoS value to guide the exploration of solution space and identify the path with the highest utility. Wu and Zhu [29] combine a transaction-aware service composition and a QoS-aware service composition model to find the optimal candidate for each abstract task. AACO [24] adjusts the pheromone evaporation factor at runtime based on the dynamics of the environment. ACO-WSC [31] present an ant-colony optimisation to minimise the length of the composition plans when composition participants are cloud services. However, these proposals do not maintain populations of non-dominated solutions. MOACO4WS [18] uses a reset strategy to address adaptation in dynamic environments. The existing proposals either are limited to template-matching composition [16,28] or require a centralised perspective [26].

8 Conclusion and Future Work

The main contributions of this work are a nature-inspired optimisation method to approximate the set of QoS optimal compositions when composing services deployed on mobile devices, and an adaptation procedure to facilitate the exploration of new service composition configurations (with potentially better QoS) that emerge as a result of providers' mobility. The existing proposals for service composition in such environments are limited to template-matching composition or require a-priori knowledge about the QoS objectives' weights. Previous work in service composition showed that template-matching composition can limit the flexibility of composition when the environment is dynamic. Also, new services may emerge as a results of mobility of providers, and the requirement of user input may affect the time of providing the user with new service composition configurations. The proposed mechanism is designed for planning-based service composition, and does not require user input. The mechanism, with and without using the adaptation procedure, is evaluated in a dynamic scenario where various number of compositions can be created. In our evaluation, we control this number to measure the performance of the proposed QoS optimisation mechanism, and compare the results with GoCoMo, SimDijkstra and a Random approach.

The results show that the proposed QoS optimisation mechanism can achieve a higher utility at the cost of increased overhead compared to the existing baseline proposals for planning-based service composition for mobile environments. Also, the results show that the proposed adaptation procedure improves the approximation of the set of QoS optimal service composition configurations that emerge as a results of providers mobility. The decrease in utility of solutions when the adaptation procedure is used can be motivated by the selection of weights that are used to evaluate this metric. However, the adaptation procedure can achieve a better approximation of the Pareto-optimal solutions than the no-adaptation variant. In our future work will explore the case when different utility weights are used. Also, in our future work we will investigate how the number of agents used in ACO and E-ACO affects the overhead.

Acknowledgment. This work was funded by Science Foundation Ireland (SFI) under grant 13/IA/1885.

References

1. Al-Oqily, I., Karmouch, A.: A decentralized self-organizing service composition for autonomic entities. ACM TAAS **6**(1), 7 (2011)
2. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient QoS-aware service composition. In: Proceedings of the 18th International Conference on World wide web, ACM (2009)

3. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Softw. Eng. **33**(6) (2007)
4. Blum, C., Li, X.: Swarm intelligence in optimization. In: Blum, C., Merkle, D. (eds.) Swarm Intelligence. Natural Computing Series. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-74089-6_2
5. Cai, W., Jin, X., Zhang, Y., Chen, K., Wang, R.: ACO based QoS routing algorithm for wireless sensor networks. In: Ma, J., Jin, H., Yang, L.T., Tsai, J.J.-P. (eds.) UIC 2006. LNCS, vol. 4159, pp. 419–428. Springer, Heidelberg (2006). https://doi.org/10.1007/11833529_43
6. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for qos-aware service composition based on genetic algorithms. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1069–1075. ACM (2005)
7. Chattopadhyay, S., Banerjee, A.: Qos constrained large scale web service composition using abstraction refinement. IEEE Trans. Serv. Comput. (2017)
8. Chen, N., Cardozo, N., Clarke, S.: Goal-driven service composition in mobile and pervasive computing. IEEE Trans. Serv. Comput. **11**(1), 49–62 (2018)
9. Deng, S., Huang, L., Taheri, J., Yin, J., Zhou, M., Zomaya, A.Y.: Mobility-aware service composition in mobile communities. IEEE Trans. Syst. Man Cybern.: Syst. **47**(3), 555–568 (2017)
10. Di Caro, G.A., Ducatelle, F., Gambardella, L.M.: Ant colony optimization for routing in mobile ad hoc networks in urban environments. IDSIA (2008)
11. Jiang, W., Hu, S., Liu, Z.: Top K query for QoS-aware automatic service composition. IEEE Trans. Servi. Comput. **7**(4), 681–695 (2014)
12. Kalasapur, S., Kumar, M., Shirazi, B.A.: Dynamic service composition in pervasive computing. IEEE Trans. Parallel Distrib. Syst. **18**(7), 907–918 (2007)
13. Klein, A., Ishikawa, F., Honiden, S.: SanGA: a self-adaptive network-aware approach to service composition. IEEE Trans. Serv. Comput. **7**(3), 452–464 (2014)
14. López-Ibánez, M., Stutzle, T.: The automatic design of multiobjective ant colony optimization algorithms. IEEE Trans. Evol. Comput. **16**(6), 861–875 (2012)
15. Mishra, K., Harit, S.: A fast algorithm for finding the non dominated set in multi objective optimization. Int. J. Comput. Appl. **1**(25), 35–39 (2010)
16. Moustafa, A., Zhang, M., Bai, Q.: Trustworthy stigmergic service composition and adaptation in decentralized environments. IEEE Trans. on Serv. Comp. **9**(2), 317–329 (2016)
17. Palade, A., Cabrera, C., White, G., Razzaque, M., Clarke, S.: Middleware for internet of things: a quantitative evaluation in small scale. In: 6th IEEE Workshop on the IoT: Smart Objects and Services, pp. 1–6. IEEE (2017)
18. Qiqing, F., Xiaoming, P., Qinghua, L., Yahui, H.: A global ToS optimizing web services selection algorithm based on MOACO for dynamic web service composition. In: IFITA, vol. 1. IEEE (2009)
19. Richerzhagen, B., Stingl, D., Rückert, J., Steinmetz, R.: Simonstrator: Simulation and prototyping platform for distributed mobile applications. In: Proceedings of the 8th International Conference on Simulation Tools and Techniques (2015)
20. Rodriguez-Mier, P., Mucientes, M., Lama, M.: Hybrid optimization algorithm for large-scale QoS-aware service composition. IEEE Trans. Serv. Comput. **10**(4), 547–559 (2015)
21. Sim, K.M., Sun, W.H.: Ant colony optimization for routing and load-balancing: survey and new directions. IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. 33(5) 560-572 (2003)

22. Stützle, T., Hoos, H.H.: Max-min ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
23. Trummer, I., Faltings, B., Binder, W.: Multi-objective quality-driven service selection-a fully polynomial time approximation scheme. IEEE Trans. Softw. Eng. 40(2) 167-191 (2014)
24. Wang, D., Huang, H., Xie, C.: A novel adaptive web service selection algorithm based on ant colony optimization for dynamic web service composition. In: Sun, X., Qu, W., Stojmenovic, I., Zhou, W., Li, Z., Guo, H., Min, G., Yang, T., Wu, Y., Liu, L. (eds.) ICA3PP 2014. LNCS, vol. 8630, pp. 391–399. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11197-1_30
25. Wang, L., Shen, J.: A systematic review of bio-inspired service concretization. IEEE Trans. Serv. Comput. **10**(4), 493–505 (2017)
26. Wang, L., Shen, J., Luo, J.: Facilitating an ant colony algorithm for multi-objective data-intensive service provision. J. Comput. Syst. Sci. **81**(4), 734–746 (2015)
27. White, G., Palade, A., Clarke, S.: QoS Prediction for Reliable Service Composition in IoT. In: Braubach, L., Murillo, J.M., Kaviani, N., Lama, M., Burgueño, L., Moha, N., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10797, pp. 149–160. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91764-1_12
28. Wu, Q., Ishikawa, F., Zhu, Q., Shin, D.H.: QoS-Aware multigranularity service composition: modeling and optimization. IEEE Trans. Syst. Man Cybern.: Syst. **46**(11), 1565–1577 (2016)
29. Wu, Q., Zhu, Q.: Transactional and QoS-aware dynamic service composition based on ant colony optimization. Future Gener. Comput. Syst. **29**(5), 1112–1119 (2013)
30. Yan, Y., Chen, M., Yang, Y.: Anytime QoS optimization over the PlanGraph for web service composition. In: Proceedings of the 27th Annual ACM SAC (2012)
31. Yu, Q., Chen, L., Li, B.: Ant colony optimization applied to web service compositions in cloud computing. Comput. Electr. Eng. **41**, 18–27 (2015)
32. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30(5)
33. Zhang, W., Chang, C.K., Feng, T., Jiang, H.y.: QoS-based dynamic web service composition with ant colony optimization. In: 2010 IEEE 34th Annual COMPSAC 2010
34. Zheng, Z., Zhang, Y., Lyu, M.R.: Investigating QoS of Real-World Web Services. IEEE Trans. Serv. Comput. 7(1) 32-39 (2014)
35. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)

State of the Practice in Service Identification for SOA Migration in Industry

Manel Abdellatif[1,2](✉), Geoffrey Hecht[1,4], Hafedh Mili[1], Ghizlane Elboussaidi[3], Naouel Moha[1], Anas Shatnawi[1], Jean Privat[1], and Yann-Gaël Guéhéneuc[2]

[1] Département d'informatique, Université du Québec à Montréal, Montréal, Canada
manel.abdellatif@polymtl.ca
[2] DGIGL, Polytechnique Montréal, Montréal, QC, Canada
[3] Ecole de Technologie Supérieure, Montréal, QC, Canada
[4] DCC, University of Chile, Santiago, Chile

Abstract. The migration of legacy software systems to Service Oriented Architectures (SOA) has become a mainstream trend for modernizing enterprise software systems. A key step in SOA migration is the identification of services in the target application, but it is a challenging one to the extent that the potential services (1) embody reusable functionalities, (2) can be developed in a cost-effective manner, and (3) should be easy to maintain. In this paper, we report on state of the practice of SOA migration in industry. We surveyed 45 practitioners of legacy-to-SOA migration to understand how migration, in general, and service identification (SI), in particular are done. Key findings include: (1) reducing maintenance costs is a key driver in SOA migration, (2) domain knowledge and source code of legacy applications are most often used respectively in a hybrid top-down and bottom-up approach for SI, (3) industrial SI methods focus on *domain* services–as opposed to *technical* services, (4) there is very little automation of SI in industry, and (5) RESTful services and microservices are the most frequent target architectures. We conclude with a set of recommendations and best practices.

1 Introduction

Software maintenance consumes the bulk of IT budgets, as legacy applications become harder to extend, fix, or even *sustain in operation*–with the obsolescence of the hardware and software on which they were built [14]. Rewriting from scratch is seldom an option, for two major reasons: (1) the substantial effort required to rewrite *(tens of) millions* of lines of code, and (2) the amount of valuable domain or procedural knowledge embodied, but otherwise undocumented, in such applications. Hence, application *modernization* remains essential to ease the maintenance of legacy systems and make them more flexible without losing their business values.

The migration of legacy systems to Service-Oriented Architecture (SOA) is one avenue for their modernization. SOA makes it possible to develop complex

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 634–650, 2018.
https://doi.org/10.1007/978-3-030-03596-9_46

and inter-organizational applications by integrating/orchestrating high-quality and reusable services. The migration of legacy systems to SOA requires identifying services, which is considered the most challenging task of the overall migration process [13]. Service Identification (SI) consists in identifying, in legacy systems or/and from the system domain decomposition, reusable services that may embed valuable business logics. The reusable services must meet a range of expectations concerning their capability, quality of service, and efficiency of use.

Several SI approaches have been proposed in the literature of academic research [4,10,16]. However, these approaches are based on few evidence and out of touch with industry practices due to the little knowledge about the state-of-the-practice of SI as part of *'real'* migration projects. Therefore, in this paper, we want to minimize the gap with industry by understanding industrial practices and identifying best practices of legacy applications migration to SOA in general and SI in particular. Thus, we wanted to answer the following research questions:

- **RQ1.** What kind of systems are being migrated to SOA?
- **RQ2.** Why are such systems being migrated?
- **RQ3.** What approaches are being used for application migration, in general, and SI in particular?

To this end, we surveyed 45 SOA migration practitioners using an online survey, and interviewed eight of them to answer these questions. We identify key findings including: (1) reducing maintenance costs is a key driver in SOA migration, (2) domain knowledge and source code of legacy applications are most often used respectively in a hybrid top-down and bottom-up approach for SI, (3) SI focuses on *domain* services, (4) there is little automation–the process of migration remains *essentially* manual, and (5) RESTful services and microservices are the most frequent target architectures.

This paper is structured as follows. Section 2 presents the related work. Section 3 describes the study design. The results of the online survey are presented in Sect. 4. Section 5 reports the results of the interview sessions, which are discussed in Sect. 6. We conclude in Sect. 7 with recommendations and best practices for SI.

2 Related Work

Seldom are research surveys related to the migration of legacy systems to SOA. Razavian and Lago [15] conducted an industrial survey about legacy-to-SOA migration approaches by targeting seven SOA solution providers. They argued that all industrial migration approaches share the same set of activities as they start by transforming the business models of legacy systems to continue with service design and implementation. Taibi et al. [19] also performed a survey on migration to microservices architectures, filled by 21 practitioners. The survey focus on the migration reasons and report that reducing maintenance cost is the main reasons of legacy-to-microsevices migration. Although these study approaches and results are similar to ours, their focus differ deeply as we cover more in details state of the practice on SI in terms of (1) the methods used, (2) the artifacts used by these methods, (3) the processes of these methods, and (4) the outputs of these processes. We also cover more participants and report best practices for SI.

More in general, a number of primary studies have been proposed in the literature about SI. Many of the proposed techniques rely on Business Process Models (BPMs), to identify services within the context of legacy migration [11,17,18]. These techniques decompose processes into tasks and then map these tasks to legacy source code elements to identify candidate services. Other SI techniques use heuristics based on the *technical properties* of services, as reflected in various metrics [1,10,12]. Such techniques often use these metrics to drive clustering and machine learning algorithms that identify software artefact clusters as candidate services. However, they do not always produce good candidate services. Other AI-based techniques use ontologies and Formal Concept Analysis to identify services in legacy systems [2,8,20]. They too, are complex and not ready for industrial applications. Other techniques put service interfaces around *existing* functional components and subsystems [4,6,16,17] but do not *infer* such clusters from finer-grained software artefacts. These so-called *wrapping-based techniques* are suitable for integration problems, but do not solve the maintenance issues.

3 Study Design

The survey presented in this paper was conducted between October 2017 and March 2018 and aimed to investigate the state of the practice in SOA migration, and, in particular, service identification. Our study consisted of four main phases:

A- Preparation of the Online Survey. We created a web-based survey (see https://goo.gl/forms/EE31KeA7R7pUeTYI2) using Google forms. The survey was prepared based on our literature survey of the state-of-the art methods for SI and informal discussions with some subject matter experts. This helped identify the dimensions/aspects of the questionnaire, the individual questions, and the possible answers for each question. Before publishing the survey, we performed a pilot with six potential subjects, three from academia and three from industry, to validate the relevance of the questions, their wording, the coverage of the answers, etc. The six 'testers' went through the questions and suggested minor changes. The final survey contained six sections: (1) participants' professional and demographic data, (2) type of migrated system, and reasons for the migration, (3) general information about SI methods (perception of importance, strategy, inputs, level of automation), (4) detailed technical information about SI (technique/algorithm used, quality metrics considered), (5) Information on the types of services sought and targeted technologies, and (6) Information about the tools used, and the suggested best practices.

B- Selection of Participants. We targeted developers with an industrial experience in SOA migration. Identifying and soliciting such developers was challenging. We relied on (1) information about companies that offer modernization services, (2) online presentations and webinars made by professionals that had the professional's contact information, and (3) search queries on LinkedIn profiles, such as "*legacy migration OR legacy modernization OR SOA architect OR SOA migration OR Cloud migration OR service migration OR service mining*". Once we identified potential participants, we sent them invitations via e-mail, LinkedIn, Facebook, and Twitter. We chose *not* to solicit more than three professionals from any given company to: (1) have an as wide representation as possible, and (2) to not overburden a single organization with our request.

C- Online Survey. We invited 289 professionals to participate, and kindly asked them to forward our invitations to other people in their network who have experience in SOA migration and SI. The survey was completed by 47 people, two of which did not participate in SOA migration projects and whose responses were discarded, leaving us with 45 complete responses.

D- Validation. We assessed the reliability of the answers in the online survey by looking for spurious/facetious answers, contradictions between answers, etc. To be able to validate improbable answers, one question of the survey asked participants if they agreed to be contacted for a follow-up 30-min interview, and 24 out of 45 agreed; however only eight could be interviewed in the end and the results of those interviews are shown in Sect. 5. A two-pass method [5] was used to analyze our transcripts of the individual interviews (see https://goo.gl/ZYv2Ut for sample transcripts). The first pass of the analysis consists of *thematic coding* to identify broad issues related to legacy-to-SOA migration in general and SI in particular. The second pass of analysis was performed using *axial coding* to identify relationships among the identified issues. Major factors were identified using *meta-codes*. The *meta-codes* were then used to identify similar patterns across the data from the multiple interviewees. Overall, the answers were plausible, and the eight detailed interviews confirmed the questionnaire answers, although provided us with more in-depth information.

4 Analysis of the Results of the Online Survey

In this section, we describe the results of our survey. We allowed multiple answers to most questions of the survey, therefore the sum of the computed percentage may exceed 100% in some cases. The given percentages are computed based on the total number of participants who answered a given question.

A- Participants. We reached a total of 45 participants who were involved in legacy-to-SOA migration projects in different capacities: 50% were software architects, 23.7% were directors of technology, and 21% were software engineers. The remaining 5.3% of participants mentioned other positions such as migration specialists, project managers and CEOs. They work in different industries: 64% were in technology and telecommunication, 20% from banking and insurance, 12.8% from health, and 3.2% from education. In terms of experience, 78% had more than 10 years of experience, and this was somewhat reflected in their age distribution: 23% were less than 35 years old, 39% were between 36 and 45, 20.5% were between 46 and 55, and 17.5% were over than 55 years old.

B- Types of Legacy Systems. The results show that the legacy systems included mainframe applications, transactional applications, ERP systems, monolithic client-server applications, software-analysis tools, and visualization tools; 13% of these were less than 5 year old, 18% were between 5–10 year old, and 69% were more than 10 years old. In terms of size, 62% of the systems were deemed large, 36% were medium size, and 2% were deemed small. Cobol (52.6%) and Java (57%) were the two most prominent languages for legacy systems. Figure 1 shows the many other languages used in the migrated applications.

Finding 1: *Practitioners migrate different types of old legacy systems implemented mainly in Cobol and Java.*

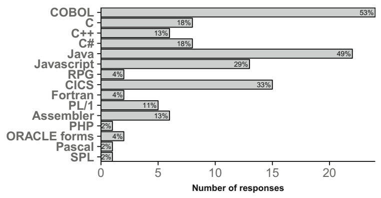


Fig. 1. Used languages in legacy systems

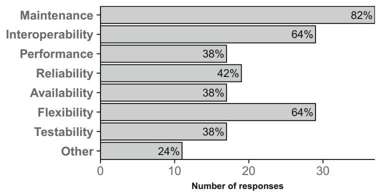


Fig. 2. Reasons for migration

C- Motivations for Legacy-to-SOA Migration. We asked about the motivations behind the migration of legacy systems to SOA. We provided a list of reasons for the migration as shown in Fig. 2. The most prevalent motivation was to reduce maintenance costs (82%). Practitioners reported during the interviews that the cost involved in maintaining legacy systems can be high due to (1) the poor/outdated documentation of these systems; (2) the obsolete/old programming languages used to implement these systems; (3) the decay and difficulty to understand the architectures, designs, and implementations of these legacy systems; and, (4) the lack of developers with the skills necessary to maintain these systems. The second most significant motivation to migrate legacy systems was to improve their flexibility (64%). We have been told during the interviews that practitioners have difficulties with legacy systems because they do not allow companies to have the flexibility required to carry out day-to-day tasks for evolving systems to meet new business requirements. The improvement of the interoperability of the legacy systems with the migration to SOA was the third most significant motivation (64%). During the interviews, practitioners told us that SOA eases the interoperability of heterogeneous systems by exploiting the pervasive infrastructure of the network. Thus, it offers the possibility to continue using and reusing the business capabilities provided by legacy systems in new, modern systems [9]. Improving system availability and testability as well as improving performance were other motivations of industrial legacy-to-SOA migration projects (38% each). Participants also mentioned other business and technical reasons for migrating legacy systems to SOA, such as improving business agility, having new user interfaces, and embracing new technologies.

Finding 2: *Reducing maintenance costs, improving the flexibility and interoperability of legacy systems are the main motivations to migrate legacy systems.*

D- Importance of Identifying Reusable Services from Legacy Systems. We asked about the importance of identifying *reusable* services in the source code of legacy systems during the migration process: 87% of the participants qualified it as important while only 13% thought that it is not. We explain this agreement by the benefits of software reuse, which (1) *increases software productivity* by shortening software-development time, (2) *reduces software development costs* by avoiding the reimplementation of existing services, (3) *reduces maintenance costs* because the reused services were functional and have been well-tested, and (4) reduces the risk of introducing new failures into the process of enhancing or creating new business services. We explain the 13% of disagreement as some participants undertook top-down migrations rather than bottom-up or mixed migrations and the former does not require identifying services in source code.

Finding 3: *Identifying services in legacy applications is an important step in legacy-to-SOA migration.*

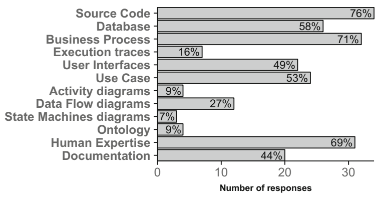


Fig. 3. Used inputs for SI in industry

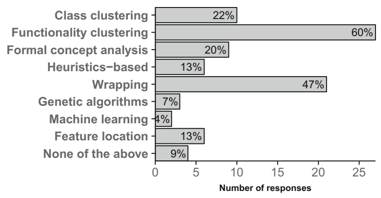


Fig. 4. Used techniques for SI in industry

E- Inputs of SI. Through a literature review, we identified several types of inputs used for SI. We listed these inputs in our survey and asked participants on which inputs they relied to identify services. Figure 3 shows that the most used inputs were source code, business process models, databases, and human knowledge. 76% of the participants relied on the recovery of the business logic of legacy systems through the analyses of the source code to identify services with high business value. 71% relied on the mapping of business processes with the legacy source code to extract reusable services through human expertise. These artifacts may help software engineers to have a better understanding of the legacy systems. Finally, participants rarely relied on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services. This observation may be due to their unavailability or complexity to establish especially since our practitioners deal with large systems.

Finding 4: *Many software artifacts can be used for SI. Practitioners mostly used source code, business process models, databases, and human expertise. There is a very low interest in relying on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services.*

F- Directions of SI. We asked participants about their choices of direction for identifying services. We proposed three directions: (1) Top-down: starting from domain-specific conceptual models, like business concepts and process models, to identify services, which are then specified and implemented through a forward engineering process; (2) Bottom-up: starting by analyzing the existing legacy system artifacts and identifying services from reusable legacy code; and (3) Mixed: starting both from domain-specific conceptual models and the analyses of the legacy system to identify services. We found that 53% of the participants use a mixed direction to identify services. Participants used almost equally top-down and bottom-up directions with 23% and 24% each. We explain these observations as follows: (1) practitioners relied on source code and business process models as reported in *Finding 4*, (2) practitioners also relied on extracting the business logic of legacy systems because documentation was not always available, (3) practitioners prioritized reuse and avoided development from scratch to reduce time and costs, and (4) practitioners faced limitations due to the lack of legacy experts/knowledge, unavailability of up-to-date documentation, program comprehension, and challenges of reverse-engineering legacy systems.

Finding 5: *Practitioners highly rely on a mixed direction to identify services during legacy-to-SOA migration process.*

G- Techniques for SI. We asked about the techniques that they used to identify services. As depicted in Fig. 4, we found that 60% of the participants relied on clustering functionalities of the legacy systems and exposing these clusters as services. 47% of them relied on some black-box techniques, like wrapping, because they either consider the migration as an integration problem or did not want to modify the core functionalities of the legacy systems because it provided useful services. We observed a low interest in using machine-learning techniques, formal-concept analysis, or meta-heuristic algorithms to identify reusable services. Using these techniques may be challenging for practitioners because they are dealing with large systems to migrate and so the knowledge required to establish these techniques could be time consuming and may not lead to optimal results. Also these techniques are researched by academics and not mainly by professionals (see Sect. 2). Finally, 9% of the participants mentioned that they did not use any techniques and performed SI manually.

Finding 6: *Functionality clustering and wrapping are the most used techniques of SI in industry.*

H- Analyses Types for SI. We asked about the types of analyses that they performed for SI (static, dynamic, textual, and–or historical analyses). We observed that 87% of the participants relied on static analyses of the source code of the legacy systems to identify services. 43% of them reported that they relied on run-time analyses. Participants also relied on textual analyses for the identification processes. Textual analyses include elements such as features identification techniques, natural language processing, legacy documentation analysis, etc. Only 18% of the participants reported that they relied on historical analyses (analyses of different versions of the legacy system) to extract candidate services, which may be due to (1) the unavailability of several versions of the legacy system and (2) the difficulty to study the evolution of a legacy system to gather valuable information to identify reusable services.

Finding 7: *Practitioners mostly relied on static analyses of the source code of their legacy systems for SI.*

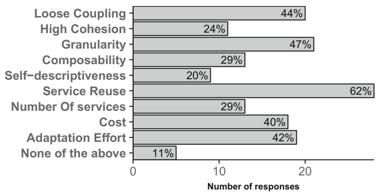


Fig. 5. Desired services quality criteria for SI in industry

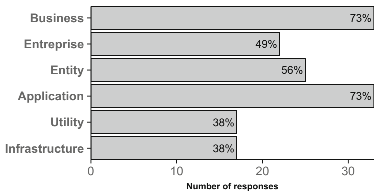


Fig. 6. Types of the migrated services

I- Services Quality Criteria. We asked the participants about the quality metrics/criteria that they sought during SI. We identified the quality criteria, listed in Fig. 5. Service reusability was the most sought quality criteria by the participants (62%), followed by service granularity (47%), and loose coupling (44%). Reusability was defined by participants as both a measure of the amount of source code reused in the services and the amount of services reused in the systems. Costs and the adaptation effort were also considered by the participants during the identification process (40% and 42% respectively). However, they did not consider self-descriptiveness, high cohesion, composability, and the total numbers of services when identifying services.

Finding 8: *Only few service quality criteria are desired by practitioners in the SI process: reusability, granularity, and loose coupling.*

J- Types of the Identified Services. We provided practitioners with a taxonomy classifying service types into domain-specific (business) services versus domain-neutral (technical) services. The provided domain-specific services are: (1) business services, enterprise services, application services and entity services. The technical services are utility services and infrastructure services. We report the results in Fig. 6. As domain-specific services represent the business core functionalities of SOA, they were the most targeted services (i.e., business and application services) during the SI processes compared to technical services. Utility and infrastructure services were the less targeted services because they are SOA-specific services and utility services are relatively easy to implement.

Finding 9: *SI is a business-driven process that prioritized the identification of domain-specific services rather than technical services.*

K- Service Technologies. We asked the participants about the services technologies that they targeted during migration. We found that 75% of them use REST services, 60% use SOAP and only 4.5% use Service Component Architecture (SCA). Surprisingly, half of the participants reported that they focused on identifying microservices in legacy systems. While there is no precise definition of this architectural style, microservices are gaining interest among organizations, especially with the growth of the Cloud and DevOps paradigms [7].

Finding 10: *Restful services are the most targeted service technology in legacy-to-SOA migration.*

L- Automation of SI. We asked the participants about the degree of automation of their SI techniques as well as the tools they used to this end (for the lack of space, we report the list of tools in https://goo.gl/ZYv2Ut). We found that many different tools are being used as well as manual analyses and in-house tools put together for the migration of particular legacy systems. However, not one set of tools supports adequately the migration of systems to SOA. We also found that the majority of the techniques used by the participants to identify services are either semi-automatic (51%) or manual (42.3%). Only three participants (6.7%) mentioned the use of tools to identify automatically reusable services. We highly believe based the reported tools that these fully-automatic approaches deal with re-engineering tasks as well as wrapping techniques that automatically expose legacy systems functionalities as services.

Finding 11: *There is a lack of automation of SI techniques in industry but input from human experts is essential to annotate/qualify intermediate or final results of SI.*

M- Threats to the Validity.

Construct validity threats refers to the extent to which operationalizations of a construct (in our case the survey and interview questions and terminology) do actually measure what the theory claims. To minimize this threat, we used both open and closed questions in the survey and tried to minimize the ambiguities through our pilot study as we mentioned in Sect. 3.

Internal validity acquiescence bias is a kind of response bias where respondents have a tendency to agree with all the questions in the survey or to indicate a positive connotation. It is sometimes referred to the tendency of a respondent to agree with a statement when in doubt. We mitigate this threat by doing interview sessions to validate the survey answers. We also eliminated responses where participants selected all the possible choices for all the questions. We also mitigate this threats by checking the responses to questions that are related to each others (e.g. the used input for SI and the identification direction, etc.). Finally, we decided not to have incentives for participating in our survey to minimize social desirability bias.

External validity the survey participants might not be representative of the general population of software developers migrating legacy systems to SOA. Thus, the generalizability of our survey might be limited. The mitigation of this threat to validity is very challenging because (1) we are targeting practitioners with very specific technical skills; and (2) professionals are in general not eager to communicate the details of their in-house tools and techniques of modernization approaches. To mitigate this risk, we advertised our survey through various channels (e.g., LinkedIn, Twitter, Facebook and email) and targeted professionals from different legacy modernization companies. Also, to the best of our knowledge, our sample size is one of the largest such size among many papers in empirical software engineering in general and modernization in particular (see Sect. 2). Participants could freely decide whether to participate in the study or not (self-selection). They were informed about the topic of the survey, the estimated time to complete the survey, the research purpose of the study and the guarantee of the anonymity of their identity and that of their answers.

5 Interview Sessions

Eight participants among the 45 agreed to carry phone interviews. Table 1 describes their profiles. The initial purpose of the interviews was to ask the participants to elaborate on some of their answers or resolve contradictions among their answers. However, the interviews often ended with discussions on issues not addressed in the survey. The interviews also allowed participants to rectify some of their answers and for our part to obtain presentations and white papers about their migrations. We now summarize salient facts gathered from the interviews in terms of the adopted migration strategies and directions of SI.

Table 1. Information about the participants in the interview sessions

Participant	Profession	Years of experience	Country
P1	Technical solution architect	25 years	Germany
P2	Legacy modernization and enterprise IT architect	18 years	India
P3	Mainframe modernization specialist	33 years	USA
P4	Legacy and data Center senior consultant	30 years	Italy
P5	Software modernization expert	15 years	Canada
P6	IT architect	20 years	Canada
P7	Director of technology	12 years	Canada
P8	Software engineer	7 years	France

5.1 Migration Strategies

We asked the interviewed participants about their adopted migration strategies to migrate legacy software systems to SOA. We identified three strategies: *rehosting*, *legacy system re-architecture*, and *rehosting followed by re-architecture.*

A- Rehosting (P1, P3, P6) consists of moving a legacy system with minimal changes from one platform, typically legacy mainframes, to more modern alternatives such as Linux, Unix, or Windows in two ways: (1) by running emulators or virtual machines of the source platform on the target platforms (e.g., a VMS or AS400 emulator/virtual machine on Linux) or (2) by rewriting the parts of the systems that interface with the target platforms. The business logic and data of the legacy systems remain unchanged on the new platform. Rehosting is done when the hardware or software platforms become too costly to support–or are no longer supported–by the manufacturer/vendor. The systems can be *wrapped* within services once they are integrated on the new platforms.

B- Legacy systems re-architecture (P1, P2, P5) is a migration strategy in three steps applied each on three different layers: the *application-code layer*, which contains the legacy code in Cobol, PL1, etc.; the *information layer*, which gathers data access through files, databases etc.; and, the *business-process layer* which describes the business logic of the system. The three migration steps are: (1) *Legacy system discovery and migration planning*, it focuses on cataloging and understanding all the assets in the legacy systems, *"we are importing the code in our toolset. We are looking for dependencies and capturing business processes. We just take a look if everything is complete"* said P1; (2) *Design*, this phase consists in designing the new system, a "*future case analysis repository*" that contains enhancements to the legacy business processes and all the modernized data and future SOA model are stored in the *information layer*; and (3) *Target system development and test*, this phase *"is a very classic software development phase just to develop and test the new SOA based system"* said P1.

C- Legacy systems re-hosting and re-architecture (P1, P4, P7, P8) aims to build new SOAs that yield the business values of the legacy systems while minimizing costs related to legacy hardware and ensuring a progressive and incremental replacement of the legacy code. This migration strategy is mainly used to "*minimize disruption while ensuring business continuity*" said P8. It avoids the "*big-bang*" migration strategy by (1) re-hosting the legacy systems to modern platforms to minimize hardware costs, (2) creating wrappers to hide the internal legacy functionalities, and (3) replacing progressively the legacy code.

5.2 Directions of SI

We detail in this section the adopted directions of SI by our interviewees.

A- Bottom-up strategies (P1, P3, P4, P5) consist of identifying artifacts of the legacy code that implement reusable business functions to be repackaged as services: *"Through the bottom-up SI strategy we want to reuse the existing legacy code certainly, but not the architecture. Most of legacy systems that we deal with have about 25 millions lines of code. If we want to write them again, it can take* *years"* said P5. The artifacts used by bottom-up approaches include the source code, data flow analyses, legacy system interfaces, databases, documentations, and human expertise. Reverse-engineering tools were used to understand the legacy systems and extract their business logic, especially when there is a lack of documentation and experts. Several interviewees reported using both in-house and open-source tools to reverse-engineer systems. For example P5 used an in-house tool based on the Knowledge Discovery Model (KDM) to obtain call and data-flow graphs of COBOL systems. P5 relies on functionality clustering and pattern matching to identify reusable services: *"We are searching for patterns and we are looking for business rules or business logic that match with these patterns and heuristics. We are doing data flow analysis with slicing to identify reusable business functions that can be grouped and deployed as services"*. Many interviewees (P1, P3, P4, and P5) also relied on techniques for detecting code clones to identify reusable services. *"What we also do in many cases is looking for duplicate code pattern because in many cases business rules are duplicated, you need to decide what to take out of this"* said P4.

B- Top-down strategies (P7) starts from the analysis of domain-specific conceptual models and requirements to specify the services of the targeted SOA. P7 recommended to use this strategy when (1) legacy source code is not available, (2) legacy source code is not reusable, (3) cost of re-engineering and integrating legacy systems is high, and (4) organizations are mature enough in terms of business processes. P7 reported that they adopted a semi-automatic top-down strategy for SI to migrate a legacy banking system to SOA. They used BPMN process models of the banking legacy system as input. They begun by identifying the entity-services and the application services. They then moved to higher-level services, such as task-centric services, and finally developed an orchestration layer that represented business services. This strategy is based on the analysis of "*information*" used in each activity of the business processes. P7 explained that *"information could be a document, reports, windows, screens, an entity etc. that is required in the execution of an activity"*. To identify entity services from business processes, *key information* manipulated in the business process models was identified. An *information* is considered as *key* by P7, if it meets at least one of the following conditions: (1) its number of occurrences exceeds a given threshold and (2) it is related to a highly solicited activities.

C- Mixed strategies (P1, P2, P3, P4, P5, P6, P8) rely on reverse-engineering techniques to document the legacy systems, extract the business logics, and identify reusable pieces of code that can be exposed as services. They also rely on forward-engineering techniques to define the business processes of the target SOAs and to design and implement the services. P2 said: *"Sometimes if the source code is available and documentation is not, we use some parser based tools to reverse-engineer these applications. These tools will create some documentation from the code and then that documentation is used to do the forward engineering and complete the targeted SOA road map"*. He also argued that *"through this documentation we create use cases for forward engineering to complete the identification, the design and the implementation of the services"*. P1 said *"we document everything in our system and then at the very end we identify the business rules mark them in the code and you can extract them afterwards [...] we have a list of business processes and core code description and we also document this, and based on this we are creating our service-oriented material".*

D- Final choice of the identified services is a manual process driven by subject-matter experts. P5 said: *"We make proposition about the services that we identify and ask the customer if it makes sense. Sometimes at technical level we have better knowledge than the customer but not from business process level".*

5.3 Threats to Validity

Internal Validity. Social desirability is a bias that leads any respondent to deny undesirable traits and report traits that are socially desirable. To minimize this threat, we did not put any incentives for the participants to participate in the interviews. We also guaranteed the interviewees their anonymity and emphasized that all the reported information will be only for research purposes.

External Validity. Information from our interviews is not generalizable as the number of the interviewees is a bit on the low end for software engineering studies. However, it is acceptable given that it is unquestionably difficult to find interviewees in legacy-to-SOA migration domain. We only sought to obtain a better understanding of the results of the online survey. Also, Table 1 shows that our interviewees are experts in legacy-to-SOA migration and, thus, that our sample is still reliable because we are dealing with subject-matter experts.

Conclusion Validity. The information from our interviews also show some threats to the validity of our conclusion because some interviewees contradict each other or, for one interviewee, change their answers to the survey. However, this threat is acceptable because we use these interviews with the purpose to mitigate and discuss the answers to the survey.

6 Discussions

After analysing the survey and interview data, we highlight the following facts.

Importance of Service Identification From Legacy Systems. We observed that SI is an important step in the overall legacy-to-SOA migration process for most practitioners, especially when it comes to the context of SI from legacy systems. As emphasized by P4 "It is important because we are able to identify the reusable of the code. SI is considered as the main helmet to measure the impact of the migration [...] you need to understand the migration cost which is in many cases too expensive, you have to cut the cost by identifying reusable pieces of the legacy code in a cost-effective way". Thus, the agreement about the importance of identifying reusable services in industry can be explained by the benefits of software reuse. However it should be noted that SI is not always fine-grained as mentioned by P6 "We basically wrap the legacy system and expose all its functionalities as services".

Business-Value Driven Service Identification. We notice that not all service quality criteria are equally targeted by practitioners. Unlike academia, efforts in industrial SI strategies are made to deal with business constraints such as the recovering of the business logic of legacy systems and extracting reusable functionalities with high business value. There are big investments by practitioners to preserve the business logic of legacy systems rather than to care about service quality constraints. As it is stated by P2, SI is mainly driven by the customers business needs: "Our customers do not really focus on these features. I am not saying that these quality criteria are not necessary but because of the business constraints, considering service quality metrics become a lower priority comparing to timing to finish the project and return in investment issues". Also, technical constraints may hinder the consideration of quality metrics: targeting quality metrics while identifying reusable pieces of code that can be exposed as services may not be suitable for all legacy technologies like mainframe legacy systems for example: "For banking mainframes systems it is not easy to use that kind of approach since we are dealing with routines" stated P4.

Automation and Experts Feedbacks. The full automation of SI process is not the primary focus of practitioners. It is even the case of big modernization companies as it is stated by P1 "In our SI methodology we are not doing everything automatic, automation is about 70% of all the migration project". However, there is automation in wrapping and reverse engineering techniques to document and extract the business logic of legacy systems when the documentation is absent. Feedback loop with business analysts and customers is considered essential by practitioners to decide about the pertinence of a candidate identified service. Practitioners also do not take the risk to try to fully automate the SI process as it is a challenging problem with unpredictable results, time consuming and needs a lot of research investments.

Gap Between Academia and Industry. None of the interviewed practitioners mentioned the use of research papers or academic resources for their migration projects. From the point of view of practitioners, "academics do not see the larger picture of the real industrial problems and challenges they are facing" as stated by P2. The lack of cost-effective academic SI technique and the lack of validation on real enterprise-scale systems is a problem that hinders knowledge transfer between academia and industry.

7 Conclusion and Recommendations

We presented a state of the practice of SI in industry to support the migration of legacy software systems to SOA. We surveyed 45 industrial practitioners and interviewed eight of them to collect, analyze, and report their experiences with the migration of legacy systems. Our results showed that reducing maintenance costs and improving the flexibility and interoperability of legacy systems are the main motivations to migrate these systems to SOA. They also showed that SI is perceived by practitioners as an important step for the migration, in particular to identify reusable code in the legacy systems. In addition, they showed that SI is a process driven by business value rather than quality criteria, even though some practitioners consider some quality criteria (mainly reusability, granularity, and loose coupling). Finally, our results showed that SI remains a manual process in which human experts' feedbacks is essential.

We drew several lessons from these results, which we summarize as follows.

Service Identification is a Business-Value Driven Process. When identifying services we must focus on the functional clusters that implement *useful* and *reusable* business functions. We must not focus on *technical/architectural* properties, as many academic techniques do (e.g., [1,2,11,12]). While the research literature identifies many service types, SI must first and foremost focus on identifying domain services, i.e., entity, business, and process services that have business values.

A Deep Understanding of the Domain and a Great Familiarity with the Legacy Systems are Necessary. Because SI is driven by business values, we must have a deep understanding of the *domain*, including its *main entities* and *processes*. We must also be familiar with the legacy systems in which to identify services. While the research literature assumes that the SI techniques are independent of the legacy experts, they should allow incorporating seamlessly knowledge from experts who are familiar with the systems.

The Input Must be Source Code and Production Data. According to the old military adage, if the terrain differs from the map, trust the terrain. With legacy systems, documentation (the map) may be absent or awfully out of date. The source code (the terrain) is the only reliable source of information about what the *current* system does. Production data, as stored for example in databases or JCL scripts, also contain valuable, up-to-date information about the systems. Therefore, while the research literature has been recently studying Q&A forums and other similar documentations, it should also strive to reconcile and improve the analyses of different up-to-date sources of information.

The Output Must be High-Value, Coarse-Grained Services. Regardless of the targeted SOA technology, be it SOAP, RESTful, or microservices, the output of any SI technique must be high-value, coarse grained *domain services*. While the research literature has been concerned by the implementation (and quality thereof) of services, it should seek to define, assess, and optimize the business values of the identified services.

The Process Must Follow a (Proven) Methodology. Migration projects are complex endeavors, regardless of the source and target technologies. There is value in adopting or adapting an existing *SOA migration methodology* because such methodologies prescribe *processes*, *deliverables*, and *quality metrics* to guide the migration. While the research literature proposes techniques, the participants recommended using existing methodologies including Oracle's *OUM Methodology*, IBM's *Service-Oriented Modelling and Architecture* (SOMA) methodology [3], and devising SI techniques as *parts* of these methodologies.

In future work, we plan to build an exhaustive catalogue of best practices for SI. We also want to identify issues that the research community can address to facilitate knowledge transfer between academia and industry in the context of legacy-to-SOA migration.

Acknowledgements. The authors would like to thank all the practitioners who replied to us. This work was supported by the Fonds Québécois de Recherches - Nature et Technologies. The second author is also supported by the Proyecto FONDECYT de Postdoctorado N3180561.

References

1. Adjoyan, S., Seriai, A., Shatnawi, A.: Service identification based on quality metrics object-oriented legacy system migration towards SOA. In: SEKE, pp. 1–6 (2014)
2. Amiri, M.J., Parsa, S., Lajevardi, A.M.: Multifaceted service identification: process, requirement and data. ComSIS **13**, 335–358 (2016)
3. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. IBM Syst. J. **47**(3), 377–396 (2008)
4. Canfora, G., Fasolino, A.R., Frattolillo, G., Tramontana, P.: Migrating interactive legacy systems to web services. In: CSMR, p. 10 (2006)
5. Charmaz, K., Belgrave, L.: Qualitative interviewing and grounded theory analysis. In: The SAGE Handbook of Interview Research, pp. 347–365 (2012)
6. Chenghao, G., Min, W., Xiaoming, Z.: A wrapping approach and tool for migrating legacy components to web services. In: ICNDC, pp. 94–98 (2010)
7. Di Francesco, P., Malavolta, I., Lago, P.: Research on architecting microservices: trends, focus, and potential for industrial adoption. In: ICSA, pp. 21–30 (2017)
8. Djeloul, M.: Locating services in legacy software: information retrieval techniques, ontology and FCA based approach. WSEAS Trans. Comput. (Greece) (2012)
9. Erl, T.: SOA Principles of Service Design. Prentice Hall PTR, Upper Saddle River (2007)
10. Gysel, M., Kölbener, L., Giersche, W., Zimmermann, O.: Service cutter: a systematic approach to service decomposition. In: ESOCC, pp. 185–200 (2016)
11. Huergo, R.S., Pires, P.F., Delicato, F.C.: A method to identify services using master data and artifact-centric modeling approach. In: ACM SAC, pp. 1225–1230 (2014)
12. Jain, H., Zhao, H., Chinta, N.R.: A spanning tree based approach to identifying web services. Int. J. Web Serv. Res. **1**(1), 1 (2004)
13. Khadka, R., Saeidi, A., Jansen, S., Hage, J.: A structured legacy to SOA migration process and its evaluation in practice. In: MESOCA, pp. 2–11 (2013)
14. Lewis, G., Morris, E., O'Brien, L., Smith, D., Wrage, L.: Smart: the service-oriented migration and reuse technique. Technical report, DTIC Document (2005)
15. Razavian, M., Lago, P.: A survey of SOA migration in industry. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 618–626. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25535-9_48
16. Rodríguez-Echeverría, R., Maclas, F., Pavón, V.M., Conejero, J.M., Sánchez-Figueroa, F.: Generating a REST service layer from a legacy system. In: Escalona, M.J., Aragón, G., Linger, H., Lang, M., Barry, C., Schneider, C. (eds.) Information System Development, pp. 433–444. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07215-9_35

17. Sneed, H.M., Verhoef, C., Sneed, S.H.: Reusing existing object-oriented code as web services in a SOA. In: MESOCA, pp. 31–39. IEEE (2013)
18. Souza, E., Moreira, A., De Faveri, C.: An approach to align business and it perspectives during the SOA services identification. In: ICCSA, pp. 1–7 (2017)
19. Taibi, D., Lenarduzzi, V., Pahl, C.: Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. IEEE Cloud Comput. **4**(5), 22–32 (2017)
20. Zhang, Z., Yang, H., Chu, W.C.: Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration. In: QRS, pp. 385–392 (2006)

A Truthful Mechanism for Optimally Purchasing IaaS Instances and Scheduling Parallel Jobs in Service Clouds

Bingbing Zheng[1], Li Pan[1(✉)], Dong Yuan[2], Shijun Liu[1(✉)], Yuliang Shi[1,3(✉)], and Lu Wang[1]

[1] School of Software, Shandong University, Jinan, China
bbingzheng@163.com, {panli,lsj,shiyuliang,luwang_hcivr}@sdu.edu.cn
[2] School of Electrical and Information Engineering, The University of Sydney, Sydney, Australia
dong.yuan@sydney.edu.au
[3] Dareway Software Co., Ltd., Jinan 250101, China

Abstract. Recently, more and more users outsource their job executions to service clouds. To reduce the costs and risks, many service providers purchase on-demand instances from IaaS clouds to provide services elastically. For maximizing social welfare, service providers need effective approaches to optimally purchase IaaS instances and schedule parallel jobs which have soft deadline, according to the valuations reported by users. In order to address the challenges such as NP-hardness and possible misreports of users, we design an auction-style randomized mechanism for the instance purchasing as well as job scheduling and pricing problem in service clouds. This mechanism can achieve an approximately optimal social welfare while scheduling jobs in a way without preemption. Many critical properties can be guaranteed simultaneously by our mechanism, including truthfulness in expectation, computational efficiency and individual rationality. Both the theoretical analysis and the extensive simulations based on synthetic data and real-world job traces validate the effectiveness of our mechanism on social welfare maximization.

Keywords: Service cloud · Auction mechanism · Scheduling · Pricing

1 Introduction

Considering cost-saving, expertise and management, more and more users begin to outsource their job executions to professional service providers, which promotes a new type of cloud offerings–service clouds. A service cloud operates a set of resources to provide professional services to users. Taking a rendering service cloud [2] as an example, it supplies rendering job execution services which accept rendering jobs from users and execute computation to produce pictures or videos as service results. For service providers, the cost of setting up, maintaining and operating data centers is generally high. Besides, the number of user arrivals and their demands for services are uncertain. In this situation, service providers can reduce their costs and risk by purchasing on-demand instances (pay per unit instance per unit time) from IaaS clouds such as Amazon [1].

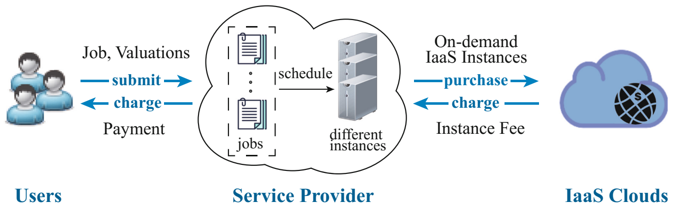


Fig. 1. A working scenario of service clouds.

For service clouds, this paper targets the goal of maximizing the social welfare (i.e., the total utility of all individuals in the system, including the service provider and all users). Figure 1 shows the working scenario of a service cloud. Users are generally job-oriented with soft deadline constraints, which means they only care about the completion time of their jobs and have different valuations for different completion time. To achieve the optimal social welfare, the service provider purchases instances and schedules jobs according to the valuations of users. Thus users are required to report the valuations they are willing to pay to the service provider when submitting jobs. The service provider then purchases on-demand instances from IaaS clouds, schedules users' jobs on these instances to produce service results and charges users corresponding payments. During this process, service providers should make decisions on IaaS instance purchasing as well as scheduling and pricing problems simultaneously. This is because the instance purchasing and scheduling schemes affect each other and ultimately influence the social welfare together.

The above decision-making process for arbitrary types of jobs in a service cloud is often a multifaceted puzzle, and thus is sometimes intractable. In this work we focus on pleasingly parallel jobs. They have flexible degree of parallelism and can be divided into an arbitrary but reasonable number of identical tasks to be executed in parallel on different instances, while no extra effort is needed. Thus different scheduling schemes of a job can result in different completion time. This type of job accounts for a large proportion in cloud market [7].

For optimally purchasing IaaS instances and scheduling parallel jobs with the goal of social welfare maximization, users' private valuations for their jobs are needed. Since users are rational and may increase their utilities by misreporting, well-designed auction mechanisms are needed to extract users' real valuations and make efficient decisions for service providers. However, there are many challenges when designing proper auction mechanisms. *First*, achieving optimal social welfare is NP-hard, even considering the parallel job scheduling problem only. *Second*, soft deadline constraints increase the difficulty of decision-making and truthfulness. *Finally*, the challenge further escalates when our mechanism makes decisions on instance purchasing and scheduling problems simultaneously.

In this paper, we propose a randomized auction mechanism for optimally purchasing IaaS instances and scheduling parallel jobs in service clouds. The main building blocks of this mechanism include an instance purchasing algorithm, a scheduling algorithm and a pricing algorithm. Our proposed randomized mechanism can achieve an approximately optimal social welfare and guarantee truthfulness in expectation, by which users will report their valuations truthfully. This mechanism is computationally efficient and individually rational, which means it has polynomial-time complexity and makes the utilities of all users non-negative. It can also schedule jobs while guaranteeing the non-preemption of tasks (i.e., an ongoing task cannot be interrupted before its completion). Through theoretical analysis and extensive simulations based on both synthetic and real data, we show that our mechanism can approximately maximize the social welfare.

The rest of the paper is organized as follows. Section 2 introduces the related works and Sect. 3 formulates the optimal instance purchasing and parallel job scheduling problem in service clouds as an integer programming. In Sect. 4, we propose a randomized mechanism. We then verify its performance through simulations in Sect. 5. Finally, we state concluding remarks in Sect. 6.

2 Related Work

Using mechanism design to schedule jobs has also been investigated in previous studies. Chen et al. [4] design a copula-based generic randomized truthful mechanism for scheduling on two unrelated machines. The goals of these works are all makespan minimization and neither of them considers the pricing problem of jobs. Varakantham et al. [8] consider the strategic variant of resource constrained project scheduling problems. They provide practical truthful mechanisms in which agents report their durations and costs of tasks as bids.

The mechanisms mentioned above assume that cloud providers or schedulers have their own resources, while in our problem service providers prefer to purchase resources from IaaS clouds. Because of the uncertainty of user demands, service providers should decide the instance purchasing scheme according to real-time situations to achieve a higher social welfare, rather than purchasing a fixed number of instances. Thus an instance purchasing algorithm is also needed.

Many works have been done for resource purchasing problem. In [11], Wang et al. propose two practical online algorithms that dynamically combine on-demand and reserved instances without any knowledge of the future. This work considers the resource purchasing problem from the perspective of end users, while in our problem the instance purchasers are also service providers who will provide services to end users. Zhao et al. [12] develop two resource purchasing models with the goal of minimizing purchase cost while meeting all the service demands. However, when providing services to users with the aim of social welfare maximization, service providers need to provide services to users selectively.

Based on these discussions, our main contribution lies in that we propose a randomized auction mechanism for optimally purchasing IaaS instances and scheduling parallel jobs in service clouds. In this mechanism, with the aim of social welfare maximization, instance purchasing as well as job scheduling and pricing schemes are determined simultaneously.

3 System Model

3.1 Fundamental Notations

Since the whole time axis is infinite and the users arrive constantly, we propose that the service provider makes decisions on instance purchasing, scheduling and pricing problems through round by round auctions with regular time intervals, which means the time intervals are of the same duration. At the start of each interval, the service provider purchases instances from IaaS clouds and processes all the job requests arrived so far. These jobs will either be completed in the next interval, or be rejected. The rejected jobs need to be re-submitted.

We assume that there are m types of on-demand instances which a service provider can purchase from an IaaS cloud platform, denoted by $\mathcal{M} = \{1, 2, \ldots, m\}$. The purchase price of instance j is $price_j$, which is fixed and set by IaaS providers. The number of units of instance j which are purchased by the service provider is r_j. In each interval, the time axis is divided into T discrete slots, denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$. There are total n users whose requests will be processed in an interval, denoted by $\mathcal{U} = \{1, 2, \ldots, n\}$. We assume that each user has a parallel job to execute and jobs are scheduled according to time slots. The runtime of a job on different instance types is varied, depending on the performance of the instances. We use $l_{i,j}$ to denote the estimated runtime of job i on instance j, i.e., the number of time slots needed to complete job i on one unit of instance j. This information can be obtained via historical data collected from previous execution records. The problem of estimating jobs' runtime is a complex issue that has been extensively researched ([9] et al.) and falls beyond the scope of this paper.

Considering the practical situations of job execution, we use threshold k_i to limit the degree of parallelism, i.e., the number of tasks that job i can be divided into. The valuations of user i are denoted by v_i. Specially v_i^e means user i is willing to pay v_i^e if its job is completed at time slot e. Let b_i and b_i^e represent corresponding bids reported by users, which may be different from v_i and v_i^e. Since jobs are scheduled according to time slots, each job has T possible completion time. Thus each user can submit at most T bids to represent discrete deadline options. Let p_i denote the payment charged to user i, which is calculated by the service provider. Let binary variable x_i^e indicate whether job i is fully completed at time slot e. Then the utility u_i of user i is:

$$u_i = \begin{cases} \sum_{e \in \mathcal{T}} v_i^e x_i^e - p_i & \text{if job } i \text{ is fully completed} \\ 0 & \text{otherwise} \end{cases}$$

3.2 Problem Formulation

We now formulate the optimal IaaS instance purchasing and parallel job scheduling problem in service clouds. Let $y_{i,j}^e(t)$ represent the number of units of instance j assigned to job i at time slot t, when job i will be completed at time slot e. Under the assumption of $b_i = v_i$, the problem can be formulated in more formal and mathematical terms:

$$\max \quad \sum_{i\in\mathcal{U}}\sum_{e\in\mathcal{T}} b_i^e x_i^e - \sum_{j\in\mathcal{M}} price_j \cdot r_j \tag{IP}$$

$$\text{s.t.} \quad \sum_{t\leq e}\sum_{j\in\mathcal{M}} y_{i,j}^e(t)\cdot\frac{1}{l_{i,j}} = x_i^e \qquad \forall\, i\in\mathcal{U}, e\in\mathcal{T} \tag{1}$$

$$\sum_{e\in\mathcal{T}} x_i^e \leq 1 \qquad \forall\, i\in\mathcal{U} \tag{2}$$

$$\sum_{i\in\mathcal{U}}\sum_{e\in\mathcal{T}} y_{i,j}^e(t) \leq r_j \qquad \forall\, t\in\mathcal{T}, j\in\mathcal{M} \tag{3}$$

$$\sum_{t\leq e}\sum_{j\in\mathcal{M}} y_{i,j}^e(t) \leq k_i \qquad \forall\, i\in\mathcal{U}, e\in\mathcal{T} \tag{4}$$

$$y_{i,j}^e(t)\in\mathbb{Z} \qquad \forall\, i\in\mathcal{U}, e\in\mathcal{T}, j\in\mathcal{M}, t\leq e \tag{5}$$

$$x_i^e\in\{0,1\} \qquad \forall\, i\in\mathcal{U}, e\in\mathcal{T} \tag{6}$$

$$r_j\in\mathbb{Z} \qquad \forall\, j\in\mathcal{M} \tag{7}$$

Constraint (5) means a unit of instance can only run one job per time slot for guaranteeing the non-preemption of tasks. Constraint (6) means partial completion of a job is not allowed. Besides, since in practice IaaS clouds only sell integer units of instances to users, Constraint (7) is needed.

By solving (IP) we can get the optimal instance purchasing and job scheduling schemes. However, finding the exact solution of (IP) is NP-hard. Besides, to satisfy the assumption of $b_i = v_i$, the truthful reports of users should be guaranteed. Considering these challenges, we design a randomized auction mechanism.

4 A Randomized Auction Mechanism

The main process of the mechanism is as follows.

- **Formulate** the optimal IaaS instance purchasing and parallel job scheduling problem in service clouds as an integer programming (IP), and relax its integer constraints to get corresponding linear programming (LP).
- **Calculate** the optimal fractional solution r^* and x^* (as well as corresponding y) by solving the linear programming (LP). This fractional solution is actually an infeasible scheme.
- **Transform** the optimal fractional r^* by rounding it up to the nearest integer, and purchase instances from IaaS clouds according to it.

– **Decompose** x^* into a series of feasible integer scheduling schemes, through a coloring decomposition algorithm. **Select** one feasible integer solution randomly and schedule jobs on the purchased instances according to it.
– If user i's job can be completed, **calculate** the payment $p_i = V_{\mathcal{U}\setminus i} - (V^*{}_{\mathcal{U}\setminus i})$.

Here p_i is the marginal harm caused by the participation of i to other users. $V_{\mathcal{U}\setminus i}$ is the optimal social welfare without i's participation. $V^*_{\mathcal{U}\setminus i}$ is the social welfare caused by x^* minus the corresponding valuation of i.

The coloring decomposition algorithm consists of two main steps. The first step is constructing a color set $\mathcal{I}$. Let a_i^e denote a *full allocation*, meaning if we schedule job i according to a_i^e, job i can be fully completed at time slot e. Let $a_{i,j}^e(t) = y_{i,j}^e(t)/x_i^e$. Then we round fractional $a_{i,j}^e(t)$ up to the nearest integer. Assume that $x_i^e = \frac{q}{N} + z$, where $q, N \in \mathbb{N}$ and $0 \leq z \leq \frac{1}{N}$. N is a parameter which can be set by the service provider. Let:

$$\overline{x}_i^e = \begin{cases} \frac{q+1}{N} & \text{with probability}\, N \cdot z \\ \frac{q}{N} & \text{otherwise} \end{cases}$$

For each user i, we add $N \cdot \overline{x}_i^e$ copies of a_i^e into $\mathcal{I}$. The second step is dividing the allocations in $\mathcal{I}$ into many *independent* groups, with the following rules: (i) No two allocations in the same group belong to the same job. (ii) No two allocations in the same group conflict with time and the capacity constraint.

Theorem 1. *The proposed randomized mechanism can achieve an approximately optimal expected social welfare in polynomial-time.*

The theoretical lower bound of the expected social welfare is $\frac{OPT^*}{\alpha} - \sum_j price_j(\frac{1}{\alpha} + \frac{\alpha-1}{\alpha}C_{max})$, where $\alpha = (1+\frac{C_{max}}{C_{min}-k})(1+\frac{nT}{N})$. Here C_{max}, C_{min} and k represent the maximum, minimum units of instances and the maximum threshold of jobs respectively, i.e., $C_{max} = \max_j m_j$, $C_{min} = \min_j m_j$, and $k = \max_i k_i$.

Theorem 2. *The proposed randomized mechanism is truthful in expectation.*

For each user, reporting its true valuations always maximizes its expected utility, regardless of the bids reported by other users.

Theorem 3. *The proposed randomized mechanism is individually rational.*

The utility of each user is non-negative, which means the payment a user should pay will not exceed its valuation. This property guarantees the voluntary participation of users.

5 Performance Evaluation

5.1 Simulation Setup

We evaluate the performance of our randomized mechanism through both synthetic and real data, by comparing with the optimal fractional result calculated by the fractional VCG mechanism [5,6,10]. The optimal fractional result is infeasible in practice and thus it only works as a benchmark for comparison.

We assume that there are $m = 5$ types of instances purchased by the service provider from IaaS clouds. We assume that the time interval of auctions is set to 1 h and the time axis of a round auction is divided into $T = 6$ time slots with 10 min per slot. The *price* is set according to Amazon EC2 on-demand instances. The bids of users equal jobs' runtime timing corresponding instance price, and then timing a value randomly picked in the range [0.8, 1.2] to reflect the preferences of users. The bids of a user are monotonically non-decreasing with completion time e. The parameter N which is used in the decomposition algorithm is set to 5000. Besides, considering the randomized nature, for each experiment we randomly select 100 integer scheduling schemes rather than only one to compute the average result.

5.2 EXP1: Influence of Different Runtime Distributions

We investigate the generated social welfare with different runtime distributions. Normal (20, 5) means the runtime is drawn from a normal distribution with mean 20 and standard deviation 5. Uniform (15, 25) means the runtime is drawn from a uniform distribution bounded by 15 and 25. Constant 20 means the runtime is always 20. The number of users n varies from 60 to 150, adding 10 to each experiment. The threshold k_i is drawn from [5, 30].

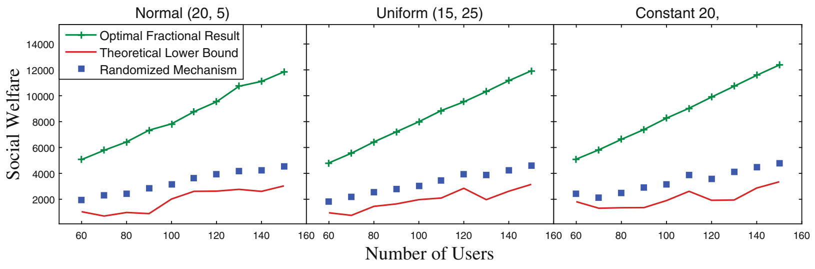
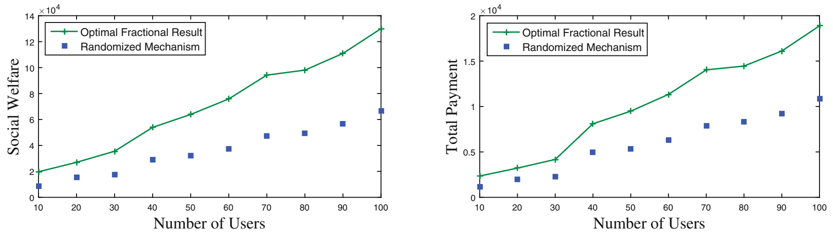


Fig. 2. Social welfare of our randomized mechanism compared with the optimal fractional result and theoretical lower bound, with three runtime distributions. (Color figure online)

In Fig. 2, the optimal fractional result (green curve) is the largest. The actual social welfare (blue point) has randomness and is always better than the theoretical lower bound (red curve) calculated by Theorem 1. We can see that the three figures corresponding to three different runtime distributions are much similar, which means the performance of our mechanism does not change significantly under different runtime distributions. This result can enable our mechanism to be more widely used.

5.3 EXP2: Performance of Varying the Number of Users

In this simulation, we study the performance of our mechanism when the number of users is varying, using Google cluster-usage data [3]. We vary the number of users n from 10 to 100, adding 10 to each experiment.



(a) Social Welfare (b) Total Payment

Fig. 3. Social welfare and total payment of our randomized mechanism compared with the optimal fractional result, when the number of users is varying.

From Fig. 3 we can see that as the number of users increases, the optimal fractional and actual social welfare as well as the total payment increase almost linearly. This is because when simultaneously considering the instance purchasing and job scheduling problems, our mechanism will purchase more instances if satisfying more users' demands increases the social welfare. This is different from the situation where service providers operate data centers themselves or purchase fixed resources. This simulation shows that under real-world data our randomized mechanism can approximately maximize the social welfare.

6 Conclusions

In this paper, we propose a randomized auction mechanism which approximately maximizes the social welfare for optimally purchasing IaaS instances and scheduling parallel jobs in service clouds. This mechanism is truthful in expectation, computationally efficient, individually rational and can guarantee the non-preemption of jobs. The theoretical analysis and extensive simulations validate the efficiency of our mechanism.

Acknowledgements. The authors would like to acknowledge the support provided by the National Key Research and Development Program of China (2017YFA0700601, 2018YFB1003800), the Key Research and Development Program of Shandong Province (2017CXGC0605, 2017CXGC0604, 2018GGX101019, 2016GGX106001, 2016GGX101008, 2016ZDJS01A09), the Natural Science Foundation of Shandong Province for Major Basic Research Projects (No. ZR2017ZB0419), the Young Scholars Program of Shandong University, and the TaiShan Industrial Experts Program of Shandong Province (tscy20150305).

References

1. Amazon EC2. https://aws.amazon.com/ec2/. Accessed 10 June 2018
2. Fox Renderfarm. http://www.foxrenderfarm.com/. Accessed 10 June 2018
3. Google cluster data. https://github.com/google/cluster-data/. Accessed 10 June 2018
4. Chen, X., et al.: Copula-based randomized mechanisms for truthful scheduling on two unrelated machines. Theory Comput. Syst. **57**(3), 753–781 (2015)
5. Clarke, E.H.: Multipart pricing of public goods. Public Choice **11**(1), 17–33 (1971)
6. Groves, T.: Incentives in teams. Econometrica **41**(4), 617–631 (1973)
7. Gunarathne, T., et al.: Cloud computing paradigms for pleasingly parallel biomedical applications. In: Proceedings of 19th HPDC, pp. 460–469 (2010)
8. Varakantham, P., et al.: Mechanism design for strategic project scheduling. In: Proceedings of 26th IJCAI, pp. 4433–4439 (2017)
9. Verboven, S., et al.: Runtime prediction based grid scheduling of parameter sweep jobs. J. Internet Technol. **11**(1), 47–54 (2010)
10. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. J. Finance **16**(1), 8–37 (1961)
11. Wang, W., et al.: Optimal online multi-instance acquisition in iaas clouds. IEEE Trans. Parallel Distrib. Syst. **26**(12), 3407–3419 (2015)
12. Zhao, H., et al.: Optimal resource rental planning for elastic applications in cloud market. In: Proceedings of 26th IPDPS, pp. 808–819 (2012)

Convenience-Based Periodic Composition of IoT Services

Bing Huang(✉), Athman Bouguettaya, and Azadeh Ghari Neiat

School of Information Technologies, The University of Sydney, Sydney, Australia
{bing.huang,athman.bouguettaya,azadeh.gharineiat}@sydney.edu.au

Abstract. We propose a novel service mining framework to personalize services in an IoT based smart home. We describe a new technique based on the concept of *convenience* to discover *periodic* composite IoT services to suit the smart home occupant's convenience needs. The key features of convenience is the ability to model the *spatio-temporal* aspects as occupants move in time and space within the smart home. We propose a novel framework for the *transient* composition of *spatio-temporal* IoT service. We design two strategies to prune non-promising compositions. Specifically, a *significance* model is proposed to prune insignificant composite IoT services. We describe a spatio-temporal *proximity* technique to prune loosely correlated composite IoT services. A periodic composite IoT service model is proposed to model the *regularity* of composite IoT services occurring at a certain location in a given time interval. The experimental results on real datasets show the efficiency and effectiveness of our proposed approach.

Keywords: IoT service · Periodic composite IoT services
Convenience

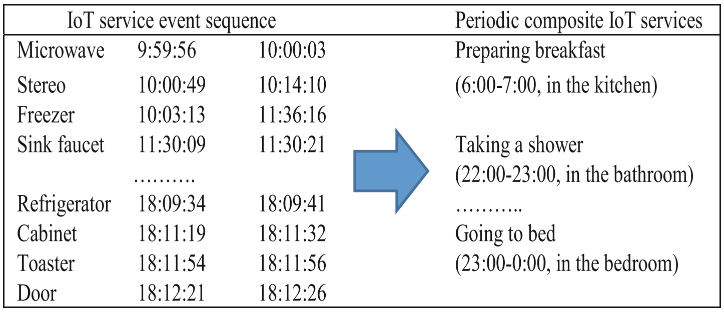
1 Introduction

The Internet is evolving from interconnecting computers to interconnecting things [1]. The Internet of Things (IoT) paradigm enables physical devices to connect and exchange information. IoT devices allow objects to be sensed or controlled remotely through the Internet [1]. The key challenge is that IoT devices are highly heterogeneous in terms of supporting infrastructure ranging from networking to programming abstraction [7]. Service-oriented Computing (SOC) is a promising solution for abstracting things on the Internet as services by hiding the complex and diverse supporting infrastructure [8]. This abstraction can shift the focus from dealing with technical details to how services are to be used [6]. We refer services for Internet of Things as IoT services. Daily life things such as a light is connected to the Internet and is represented as a light service.

An application domain for IoT is the smart home. A smart home can be considered as any regular home which has been augmented with various types of IoT services [15]. The purpose of a smart home is to make residents' life more

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 660–678, 2018.
https://doi.org/10.1007/978-3-030-03596-9_48

convenient and efficient [17]. Current research mainly focus on basic capabilities such as communication, computing, sensing and so on, which are indeed fundamental research topics [3]. However, these basic capabilities are not enough for IoT services. They should have more advanced intelligence i.e., the capability of understanding the physical world. To empower such high-level intelligence in smart homes, a key task is to discover *periodic composition of IoT services* which can represent periodic human activities. Periodic composite IoT services can be loosely defined as the composite IoT services' repeating occurrence at certain locations with regular time intervals. For example, a resident may have the habit of taking shower around 10 pm. It is of paramount importance to discover periodic composite IoT services. Periodic composition of IoT services can provide an insightful and concise explanation of IoT service usage patterns. These patterns can be used to design intelligent control of IoT services in smart homes to reduce residents' interactions with IoT services. Reducing those interactions provides more *convenience* for residents. Such periodic composite IoT services are also useful for human activity prediction. If an IoT service usage fails to follow its regular periodic composition, it could be a signal of abnormality.



IoT service event sequence		
Microwave	9:59:56	10:00:03
Stereo	10:00:49	10:14:10
Freezer	10:03:13	11:36:16
Sink faucet	11:30:09	11:30:21
	……….	
Refrigerator	18:09:34	18:09:41
Cabinet	18:11:19	18:11:32
Toaster	18:11:54	18:11:56
Door	18:12:21	18:12:26

Periodic composite IoT services
Preparing breakfast (6:00-7:00, in the kitchen)
Taking a shower (22:00-23:00, in the bathroom)
………..
Going to bed (23:00-0:00, in the bedroom)

Fig. 1. Examples of periodic composite IoT services

It is challenging to provide convenience by discovering periodic composite IoT services from IoT service usage history (i.e., IoT service event sequence). For example, Fig. 1 shows the IoT service usage history (on the left). We can see that it is difficult to extract periodic composite IoT services (on the right). We identify three key challenges.

- The set of IoT services are not known, which may be used collectively to fulfill a daily task. These IoT services are spatio-temporally correlated. We refer such set of IoT services as *composite IoT services.*
- There are many opportunities of establishing spatio-temporal relationships among IoT services, leading to an explosive number of possible composite IoT services. Many of these composite IoT services may be insignificant and loosely correlated. As a result, there is a need to prune insignificant and loosely correlated composite IoT services.

- The associated time interval and location for the periodic composite IoT services are not known. The composite IoT service may not occur exactly at the same time in a particular location. Therefore, there is a need to estimate the associated time interval and location.

In this paper, we focus on providing convenience by discovering periodic composite IoT services. At the first stage, we focus on discovering composite IoT services. Then, we employ *significance* and *proximity* strategies to prune insignificant and loosely correlated composite IoT services. At the third stage, we estimate associated time interval and location for the candidates generated in the second stage. Lastly, we measure how much convenience can be obtained by applying discovered periodic composite IoT services. The key contributions are as follows: (1) A new *IoT service model* and a *composite IoT service* model are proposed based on spatio-temporal features. (2) A *significance* model is proposed to prune insignificant composite IoT services. We also propose a *proximity* model in terms of *spatial-proximity* and *temporal-proximity* to filter out loosely correlated composite IoT services, (3) A *periodic composite IoT service* model is proposed to represent the regularity of composite IoT services occurring at a certain location in a time interval, (4) A *convenience* model is proposed to measure the benefits of applying periodic composite IoT services, and (5) A novel algorithm PCMiner (i.e., **P**eriodic **C**omposite IoT service **Miner**) is designed to discover periodic composite IoT services from event sequences.

The rest of the paper is organized as follows. Section 2 formally defines key concepts. Section 3 details the proposed algorithm PCMiner. Section 4 shows the experimental results. Section 5 surveys the related work. Section 6 conclude the paper and highlights some future work.

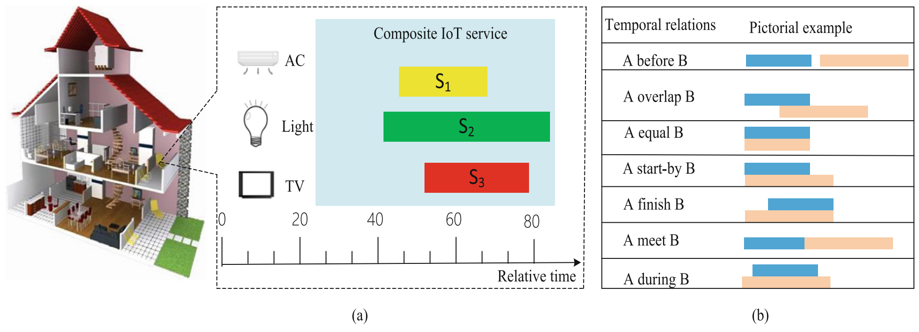


Fig. 2. (a) An example of a composite IoT service; (b) Time interval relations

Motivating Scenario

We use the smart home as our motivating scenario. Sarah lives alone in a smart home. We assume everything such as lights, TV, oven, window, and floors are connected to the Internet and represented as IoT services. This smart home aims to improve Sarah's life convenience. Intuitively, *convenience* can be interpreted as a smart home system which is aware of a resident's potential needs by understanding physical environment or situations and respond proactively at the right time and in the right place. This reduces a resident's interactions with IoT services. Let us imagine an interesting convenient life scenario in Sarah's home. In a weekday morning, the clock wakes Sarah up at 8 am. Then the lamp is turned on automatically. Sarah gets up and prepares stuff for taking a shower. Meanwhile, the heater in the bathroom starts to heat. When Sarah steps into the bathroom, it is already warm. While Sarah is taking shower, the music player is playing her favorite music. At the same time, the kettle in the kitchen starts to work and the coffee maker starts to make a cup of Mocha coffee. After Sarah finishes showering and grooming, she goes to the kitchen to prepare for breakfast. When she is enjoying the breakfast and coffee, the TV is turned on automatically and displays her favorite sport news. After finishing the breakfast, Sarah goes to work. The TV and all the lights are turned off automatically.

A fundamental task to provide convenience is to augment IoT services with capabilities of understanding the periodic usage of composite IoT services. Let us consider Sarah performs daily activities by interacting with IoT services. These interactions are recorded as IoT service event sequences shown in Fig. 3. For example, $<E^+E^-, (60, 75)>$ denotes the music player is playing music from time 60 to time 75 and E^+ (resp.E^-) denotes a turn on (resp. turn off) the music player event. There exist spatio-temporal relationships among IoT services. For example, in Day 1, the relationships between the music player service (i.e., E) and the shower service (i.e., F) shows that Sarah listens to music while taking a shower. In this regard, a collection of spatio-temporally correlated IoT services may represent an activity. We refer to such set of IoT services as *composite IoT services.* An example of a composite IoT service is shown in Fig. 2(a).

There are many ways of establishing spatio-temporal relationships among IoT services. According to Allen's temporal logic in Fig. 2(b), there are 57 ways of generating composite IoT services in Day 1 by a brute force approach (i.e., $C_6^2 + C_6^3 + C_6^4 + C_6^5 + C_6^6 = 57$). Many of them may be insignificant and loosely correlated. Therefore, we design the *significance* and *proximity* strategies to filter out these insignificant and loosely correlated composite IoT services.

The proximate and significant composite IoT services can represent the resident's daily activities. The residents usually performs his/her daily activities periodically in terms of time and location. For example, the resident usually goes to bed during 11pm to 12pm and wakes up during 8am to 9am. We refer such repeating composite IoT services at certain location with regular time intervals as *periodic composite IoT services.* The periodic composite IoT services can be serve as a knowledge basis for providing convenience. A *convenience* model is introduced to quantify how much benefits of applying periodic composite IoT services. In this paper, we focus on providing convenience by discovering periodic composite IoT services.

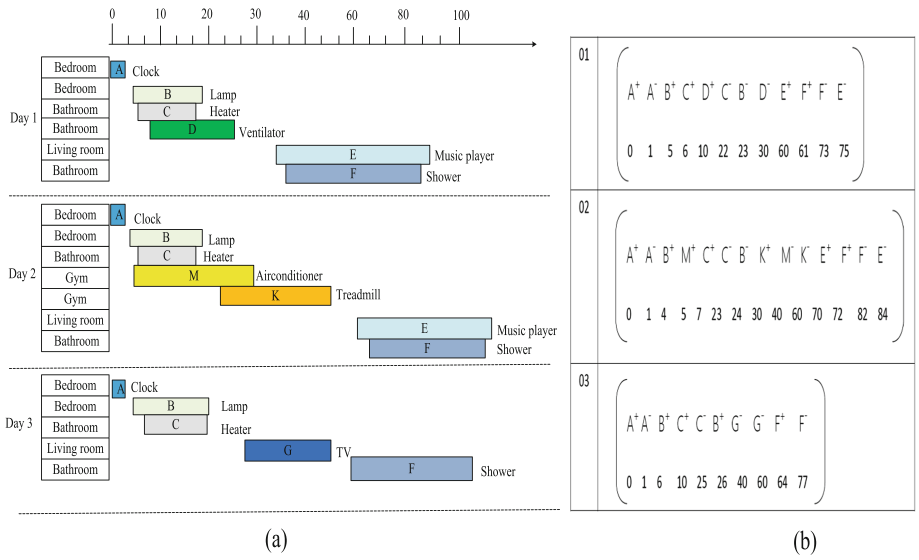


Fig. 3. An example of service event sequences

2 System Model

We first introduce the notion of IoT services and composite IoT services based on spatio-temporal features [25]. Then, a significance model and a proximity model are proposed to prune non-promising composite IoT services. We introduce the notion of periodic composite IoT service to model the occurrence regularity of composite IoT services. Lastly, we provide a convenience model to quantify how much benefits of applying periodic composite IoT services.

2.1 IoT Service Model

Definition 1: IoT Service. An IoT service S_i is a tuple $S_i = <s_i, F_i, IS, FS>$, where:

- s_i is a unique service identifier.
- F_i is a set of functions that are offered by S_i.
- IS (Initial State) is a tuple $<s_i^+, st_i, sl_i>$, where
  - s_i^+ is a symbol of IS.
  - st_i is a start-time of S_i.
  - $sl_i = <x_s, y_s>$ is a start location of S_i, where $<x_s, y_s>$ is a GPS point.
- FS (Final State) is a tuple $<s_i^-, et_i, el_i>$, where
  - s_i^- is a symbol of FS.
  - et_i is an end-time of S_i.
  - $el_i = <x_e, y_e>$ is an end location of S_i, where $<x_e, y_e>$ is a GPS point.

We focus on the spatio-temporal features in the remainder of this paper. Thus, the representation of an IoT service S_i is simplified as $<(s_i^+, st_i, sl_i), (s_i^-, et_i, el_i)>$. For example, a light service is represented as $<(light^+, 7pm, (1,2)), (light^-, 9pm, (1,2))>$ which is described as lighting from 7pm to 9pm in the bedroom where $(1, 2)$ is the GPS point in the bedroom and 7pm (resp. 9pm) is the start time (resp. end time).

2.2 Composite IoT Service Model

One IoT service may not accomplish a daily activity. Multiple IoT services may be composed to fulfill an activity [13]. These IoT services may be used collectively based on time and location correlations to accomplish a certain daily activity. We refer such spatio-temporally correlated IoT services as composite IoT services.

Definition 2: Composite IoT Service. A composite IoT service CS is a collection of IoT services that occur frequently in a particular spatio-temporal relationships. A composite IoT service is denoted by a tuple $CS = <S, sup(S)>$ where

- $S = \{<(s_1^+, st_1, sl_1), (s_1^-, et_1, el_1)>, ..., <(s_n^+, st_n, sl_n), (s_n^-, et_n, el_n)>\}$ represents n component IoT services where $<(s_i^+, st_i, sl_i), (s_i^-, et_i, el_i)>$ is a component IoT service S_i as defined in Definition 1 and $st_i \leq st_{i+1}$ and $st_i \leq et_i$. An example of a composite IoT service is shown in Fig. 2(a). By ordering all elements s_i^* ($*$ can be $+$ or $-$) in S in a non-decreasing order based on its associated time information st_i(or et_i), we can transform S into the following representation $S = <Seq, T, L> = \left\{ \begin{matrix} \alpha_1 & ... & \alpha_i & ... & \alpha_{2n} \\ t_1 & ... & t_i & ... & t_{2n} \\ l_1 & ... & l_i & ... & l_{2n} \end{matrix} \right\}$, where $Seq = \{\alpha_1 ... \alpha_i ... \alpha_{2n}\}$ is a symbol sequence and $\alpha_i = s_j^*$ ($*$ can be $+$ or $-$), $T = \{t_1 ... t_i ... t_{2n}\}$ is the time information and $t_i \leq t_{i+1}$, and $L = \{l_1 ... l_i ... l_{2n}\}$ is the location information. For example, the composite IoT service in Fig. 2(a) can be represented as $\left\{ \begin{matrix} s_2^+ & s_1^+ & s_3^+ & s_1^- & s_3^- & s_2^- \\ 48 & 50 & 58 & 65 & 70 & 75 \\ l_2 & l_1 & l_3 & l_1 & l_3 & l_2 \end{matrix} \right\}$ (i.e., $l_1 = (1, 2)$, $l_2 = (2, 4)$, $l_3 = (3, 5)$)
- $sup(S)$ is the support for S. The support sup is the total number of occurrence in a database $S = <Seq, T, L>$ and $S' = <Seq', T', L'>$ be two composite IoT services. S is referred to as a sub-composite IoT service of S' , denoted as $S \sqsubseteq S'$, if $Seq = \{\alpha_1 ... \alpha_i ... \alpha_{2n}\}$ is a subsequence of $Seq' = \{\alpha'_1 ... \alpha'_i ... \alpha'_{2m}\}$, denoted as $Seq \sqsubseteq Seq'$ with $n \leq m$. $Seq \sqsubseteq Seq'$ is satisfied if there exist integers $1 \leq k_1 \leq k_2 ... k_n \leq k_{2m}$ such that $\alpha_1 \subseteq \alpha'_{k_1}, \alpha_2 \subseteq \alpha'_{k_2}, \ldots, \alpha_n \subseteq \alpha'_{k_n}$. Given an IoT service event sequence DB, the tuple (sid, S') (i.e., sid is a sequence ID and S' is the composite IoT service) is said to contain a sub-composite IoT service S if $S \sqsubseteq S'$. The support sup of S in DB, denoted

as $sup(S)$ is the number of tuples containing S. $sup(S)$ can be formalized as follows.

$$sup(S) = |\{(sid, S) \in DB | S \sqsubseteq S'\}| \tag{1}$$

2.3 Significance and Proximity Model for Composite IoT Services

There are many possibilities of establishing spatio-temporal relationships among IoT services, leading to an explosive generation of composite IoT services. Many of the composite IoT services are insignificant and loosely correlated. Thus, there is a need to filter out these non-promising composite IoT services. We explore this problem from two aspects. On the one hand, those composite IoT services that occur frequently are more likely to be significant that those occur less frequently. Thus, we propose a $significance$ model to quantify how much significance these composite IoT services are from the statistic aspect. On the other hand, the IoT services that occur proximately in terms of time and location are more likely to be correlated. For example, from the spatial perspective, the relationship between the TV and the light in the same dining room may reveal a high correlation between these two IoT services. However, the co-occurrence of using TV in the dining room and using the light in the bedroom may merely be a coincidence. From the temporal perspective, using the TV and the light in the evening reveal that there may exist a high correlation between the two IoT services during that time. However, using the TV in the evening and using the light in the morning may not have any correlation. In this regard, we use *proximity* to characterizes correlation strength among component IoT services in terms of *spatial-proximity* and *temporal-proximity*. By *spatial-proximity*, it characterizes the location correlation strength among component IoT services. By *temporal-proximity*, it characterizes the temporal correlation strength among component IoT services. The *proximity* model is adapted from the approach for measuring spatio-temporal interval data distance [11].

Definition 3: Significance. *Significance* is used for evaluating statistic importance of CS. Given a composite IoT service $CS = <S, sup>$, its *significance* is formalized as:

$$significance(S) = \frac{\sqrt{expect(S)}}{sup(S) - expect(S)} \tag{2}$$

where $expect(S)$ is the expected number of occurrence in a DB. To estimate $expect(S)$, we adapt the statics model proposed in [14] by considering IoT services' various usage frequency across different regions in smart homes [16]. In practice, IoT services' usage frequency may vary across different regions in smart homes. For example, if a resident spends most of his/her time in their living room during the day and only goes to their bedroom for sleeping, then IoT services in the living room will be used more frequently than those in the bedroom. Thus, composite IoT services for sleeping may be ignored when searching for frequent composite IoT services.

Given a composite IoT service CS whose symbol sequence is $Seq = \{s_1^*...s_i^*...s_m^*\}$, suppose that Seq is a possible outcome drawn from the symbol set $A = \{a_1...a_i...a_n\}$ with $P(a_i)$ following Bernoulli distribution such that $\sum_{i=1}^{n} P(a_i) = 1$. Given a DB and a region set $R = \{r_1, ..., r_k\}$, DB_{r_i} records IoT service event sequences occurring at region r_i. $Num(a_i)_{DB_{r_i}}$ is the number of the event a_i occurrence in database DB_{r_i}. $P(a_i)$ can be estimated by $\frac{Num(a_i)_{DB_{r_i}}}{\sum_{a_j \in A} Num(a_j)_{DB_{r_i}}}$. Therefore, the occurrence probability $P(S)$ and $expect(S)$ can be formalized as follows.

$$P(S) = \prod_{\forall s_i^+ \in Seq} P(s_i^*) \tag{3}$$

$$expect(S) = P(S) \cdot |DB_{r_i}| \tag{4}$$

where $|DB_{r_i}|$ is number of IoT service events in region r_i. Note that we only count $P(s_i^*)$ once because each s_i^* has two points (i.e., s_i^+ and s_i^-).

Definition 4: Proximity. Given a composite IoT service $CS = <S, sup>$ where S is in the form $S = <Seq, T, L> = \left\{ \begin{array}{ccccc} \alpha_1 & ... & \alpha_i & ... & \alpha_{2n} \\ t_1 & ... & t_i & ... & t_{2n} \\ l_1 & ... & l_i & ... & l_{2n} \end{array} \right\}$, its proximity function is defined as follows.

$$U = w_1 \cdot spatial\_proximity + w_2 \cdot temporal\_proximity \tag{5}$$

where $w_i (i = 1, 2)$ is a weight such that $w_i \in [0, 1]$ and $w_1 + w_2 = 1$. The *spatial_proximity* and *temporal_proximity* are formalized as follows.

- *Spatial_proximity*: The *spatial_proximity* measures the average location proximity of all composite IoT service instances. The *spatial_proximity* for a composite IoT service instance is first formalized in Eq. (6). Then the average *spatial_proximity* for the composite IoT service is formalized in Eq. (7).

$$Spa = \sum_{i=1}^{n} \frac{1}{|x_i - x_{i+1}| + |y_i - y_{i+1}|} \tag{6}$$

$$spatial\_proximity = \frac{\sum_{j=1}^{sup} Spa_j}{sup} \tag{7}$$

where n is the total number of component IoT services, $l_i = <x_i, y_i>$ and $l_{i+1} = <x_{i+1}, y_{i+1}>$ are two locations for two consecutive component IoT services, and sup is the support for the composite IoT service. For example, for the composite IoT service in Fig. 2(a), its *spatial_proximity* score is $Spa = \frac{1}{|l_2 - l_1|} + \frac{1}{|l_1 - l_3|} = \frac{1}{|2-1|+|4-2|} + \frac{1}{|1-3|+|2-5|} = 0.53$. In this paper, we use Manhattan distance proposed in [12] to measure the proximity because it is computing efficient.

- *Temporal_proximity*: The *temporal_proximity* measures the average temporal proximity of all composite IoT service instances. We adapt the technique of evaluating the distance between time-interval based data in [11]. For each component IoT service instance $S_i = \{(s_i^+, st_i, sl_i), (s_i^-, et_i, el_i)\}$, we utilize a function f_i with respect to t to map the temporal aspect of S_i. f_i is formalized as follows.
$$f_i(t) = \begin{cases} 1, & t \in [st_i, et_i] \\ 0, & otherwise \end{cases} \tag{8}$$
Then we have a set of functions $\{f_1, f_2, ...f_n\}$ corresponding to the composite IoT service instance. The *temporal_proximity* for the composite IoT service instance is calculated by Eq. (9). The average *temporal_proximity* is calculated by Eq. (10).
$$Temp = \frac{\int_{t_1}^{t_{2n}} \sum_{i=1}^{n} f_i(t)\, dt}{(t_{2n} - t_1) \cdot n} \tag{9}$$
$$temporal\_proximity = \frac{\sum_{j=1}^{sup} Temp_j}{sup} \tag{10}$$
where t_1 and t_{2n} are the first and the last time information of CS, respectively, and n is the number of component IoT services. For example, the *temporal_proximity* score of the composite IoT service {<stove, [18:00, 19:00]>, <washing machine, [18:40, 19:20]>} can be calculated as $\frac{(18{:}40-18{:}00)+(19{:}00-18{:}40)\cdot 2+(19{:}20-19{:}00)}{(19{:}20-18{:}00)\cdot 2} = 0.625$. This composite IoT service can be interpreted as *when the resident is cooking, he/she is also doing laundry*. Another composite IoT service is {<stove, [18:00, 19:00]>, <fan, [18:00, 19:00]>} and its *temporal_proximity* score is 1. Thus the latter composite IoT service is considered to be more temporally proximate than the former.

2.4 Periodic Composite IoT Service Model

In this section, we introduce the novel notion of periodic composite IoT service to model the regularity of repeating composite IoT services.

Definition 5: Periodic composite IoT service. A periodic composite IoT service PC is defined as the repeating composite IoT services at certain locations with regular time intervals. It is denoted by a tuple $PC = <CS, T, L, P>$ where

- CS is a composite IoT service.
- $T = <T_s, T_e>$ is a representative time interval associated with CS, where T_s and T_e are the start time and end time of CS, respectively. Suppose all start time and end time of CS in DB constitutes the set $\tau = \{<st_1, et_1>, <st_2, et_2>...<st_m, et_m>\}$. We need to find the representative time interval $<T_s, T_e>$ which minimizes the dissimilarity between the instance $<st_i, et_i>$. We define the dissimilarity dis between two time intervals.
$$dis = |T_s - st_i| + |T_e - et_i| \tag{11}$$

Thus, the total dissimilarity between $<T_s, T_e>$ and τ can be defined by Eq. (12).

$$Dis(T,\tau) = \sum_{i=1}^{m} |T_s - st_i| + |T_e - et_i| \tag{12}$$

To minimize $Dis(T,\tau)$, Eq. (12) can be transformed into two known minimization problems, that is, find T_s and T_e to minimize $\sum_{i=1}^{m} |T_s - st_i|$ and $\sum_{i=1}^{m} |T_e - et_i|$, respectively. T_s is the median of the start time set $\{st_1, st_2...st_m\}$ and T_e is the median of the end time set $\{et_1, et_2...et_m\}$. The proof can be found in [19].

- L is the region location of CS such as the bedroom and the bathroom.
- P is the probability of CS occurring around time interval T at location L. Suppose the time information of a CS instance is $<st_j, et_j>$. The CS is said to occur around time interval T if their dissimilarity dis is no more than a tolerance threshold ζ, that is, $|T_s - st_j| + |T_e - et_j| \leq \zeta$. P can be formalized as follows.

$$P = \frac{Num}{TNum} \tag{13}$$

where Num is the number of CS occurrence around time interval T at location L. $TNum$ is the total number of CS occurrence in the database.

2.5 Convenience Model

The discovered periodic composite IoT services can be served as knowledge basis for building an intelligent system to provide *convenience* for the residents. By convenience, it is interpreted as the benefits of applying periodic composite IoT services via reducing residents' interactions with IoT services. The convenience can be quantified as follows.

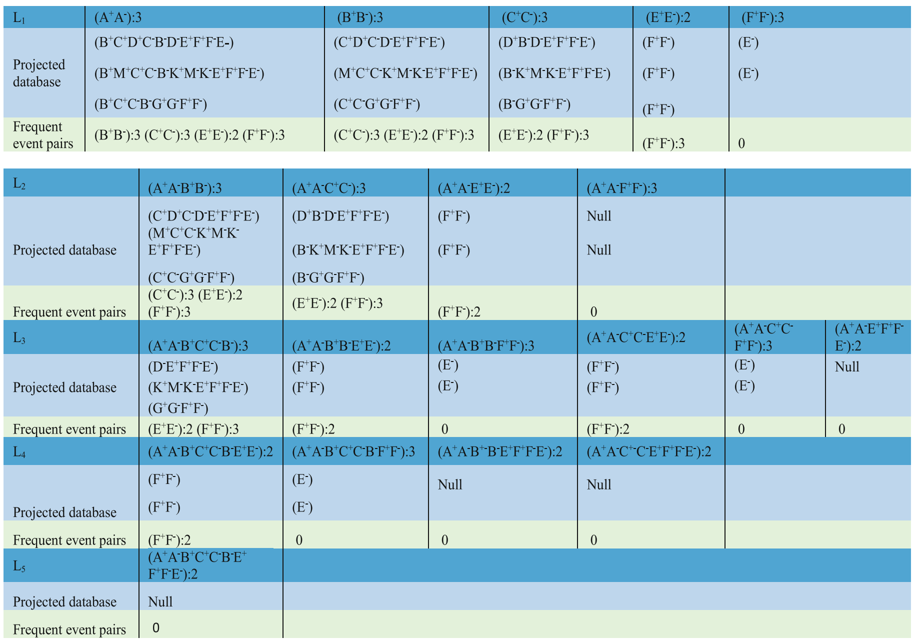
Definition 6: Convenience. Given an IoT service event sequence $\{<(a_1^+, st_1, sl_1), (a_1^-, et_1, el_1)> \ldots <(a_n^+, st_n, sl_n), (a_n^-, st_n, sl_n)>\}$ during the time period $[st_1, et_n]$, suppose this sequence is initialized from a set of periodic composite services $\{ PC_1, PC_2 \ldots PC_m\}$. According to the representative time information of the PC_m, we can roughly estimate the next PC_{m+1} occurrence. The IoT service events involved in PC_{m+1} is $\{ b_1, b_2...b_m \}$. Suppose the actual event set occurs next is $\{ c_1, c_2 \ldots c_k \}$. Therefore, the amount of convenience can be quantified by Eq. (14).

$$convenience = \frac{|\{b_1, b_2...b_m\} \cap \{c_1, c_2...c_k\}|}{|\{c_1, c_2...c_k\}|} \tag{14}$$

where $|\{c_1, c_2...c_k\}|$ is the number of events and $|\{b_1, b_2...b_m\} \cap \{c_1, c_2...c_k\}|$ is the number of correctly estimated events.

3 Discovering Periodic Composite IoT Service Approach

We develop the algorithm PCMiner to efficiently discover periodic composite IoT services from IoT service event sequences. Algorithm 1 shows the details of PCMiner. The algorithm consists of four phases. The mining process starts with *dividing the search space*. Then PCMiner *searches all composite IoT services* in a determined space. Third, PCMiner applies the *significance* and *proximity* strategies to remove non-promising composite IoT services. Finally, PCMiner collects time information and location information for candidates generated in the third phase. Based on these information, time period and location corresponding to the candidates are estimated which leads to generating a set of *periodic composite IoT services*. For the sake of consistency with the terms from data analysis techniques, we use *event patterns* to refer to composite IoT services occurring in service event sequences. We use the running example in Fig. 3 to illustrate the process of PCMiner shown in Fig. 4.



L_1	(A^+A^-):3	(B^+B^-):3	(C^+C^-):3	(E^+E^-):2	(F^+F^-):3
Projected database	$(B^+C^+D^+C^-B^-D^-E^+F^+F^-E$-) $(B^+M^+C^+C^-B^-K^+M^-K^-E^+F^+F^-E^-)$ $(B^+C^+C^-B^-G^+G^-F^+F^-)$	$(C^+D^+C^-D^-E^+F^+F^-E^-)$ $(M^+C^+C^-K^+M^-K^-E^+F^+F^-E^-)$ $(C^+C^-G^+G^-F^+F^-)$	$(D^+B^-D^-E^+F^+F^-E^-)$ $(B^-K^+M^-K^-E^+F^+F^-E^-)$ $(B^-G^+G^-F^+F^-)$	(F^+F^-) (F^+F^-) (F^+F^-)	(E^-) (E^-)
Frequent event pairs	(B^+B^-):3 (C^+C^-):3 (E^+E^-):2 (F^+F^-):3	(C^+C^-):3 (E^+E^-):2 (F^+F^-):3	(E^+E^-):2 (F^+F^-):3	(F^+F^-):3	0

L_2	$(A^+A^-B^+B^-)$:3	$(A^+A^-C^+C^-)$:3	$(A^+A^-E^+E^-)$:2	$(A^+A^-F^+F^-)$:3		
Projected database	$(C^+D^+C^-D^-E^+F^+F^-E^-)$ $(M^+C^+C^-K^+M^-K^-E^+F^+F^-E^-)$ $(C^+C^-G^+G^-F^+F^-)$	$(D^+B^-D^-E^+F^+F^-E^-)$ $(B^-K^+M^-K^-E^+F^+F^-E^-)$ $(B^-G^+G^-F^+F^-)$	(F^+F^-) (F^+F^-)	Null Null		
Frequent event pairs	(C^+C^-):3 (E^+E^-):2 (F^+F^-):3	(E^+E^-):2 (F^+F^-):3	(F^+F^-):2	0		
L_3	$(A^+A^-B^+C^+C^-B^-)$:3	$(A^+A^-B^+B^-E^+E^-)$:2	$(A^+A^-B^+B^-F^+F^-)$:3	$(A^+A^-C^+C^-E^+E^-)$:2	$(A^+A^-C^+C^-F^+F^-)$:3	$(A^+A^-E^+F^+F^-E^-)$:2
Projected database	$(D^-E^+F^+F^-E^-)$ $(K^+M^-K^-E^+F^+F^-E^-)$ $(G^+G^-F^+F^-)$	(F^+F^-) (F^+F^-)	(E^-) (E^-)	(F^+F^-) (F^+F^-)	(E^-) (E^-)	Null
Frequent event pairs	(E^+E^-):2 (F^+F^-):3	(F^+F^-):2	0	(F^+F^-):2	0	0
L_4	$(A^+A^-B^+C^+C^-B^-E^+E^-)$:2	$(A^+A^-B^+C^+C^-B^-F^+F^-)$:3	$(A^+A^-B^{+-}B^-E^+F^+F^-E^-)$:2	$(A^+A^-C^{+-}C^-E^+F^+F^-E^-)$:2		
Projected database	(F^+F^-) (F^+F^-)	(E^-) (E^-)	Null	Null		
Frequent event pairs	(F^+F^-):2	0	0	0		
L_5	$(A^+A^-B^+C^+C^-B^-E^+F^+F^-E^-)$:2					
Projected database	Null					
Frequent event pairs	0					

Fig. 4. The Process of PCMiner

Phase I: Dividing Search Space. The layout of a smart home consists of multiple regions such as a bedroom and a kitchen. Each IoT service event is associated with a region. For example, turn on the lamp event occurs in the bedroom. Given a region set $r = \{r_1, r_2...r_n\}$, we divide the database DB into multiple smaller databases DB_{r_i}. Each DB_{r_i} records IoT service event sequences occurring in the region r_i. In later phases, the discovering process performs on each sub-databases. For the purpose of illustrating PCMiner, we assume all IoT service event sequences in the running example shown in Fig. 3 are from the same region and constitute a sub-database.

Phase II: Searching Event Patterns. PCMiner employs a divide-and-conquer, pattern-growth principle from Prefixspan [23] as follows: event sequence databases are recursively projected into a set of smaller *projected databases* based on the current event patterns. Event patterns are then grown by searching the smaller projected databases.

Definition 7: Projected database. Let p be an event pattern in a database DB. The p-projected database, denoted as $DB|_p$, is the collection of suffixes of event sequences in DB with regard to the prefix p.

The searching process consists of three sub-phases. PCMiner first finds the set of 1-length event patterns. Then, PCMiner constructs projected databases for each 1-length event pattern generated in the phase one. Third, the event patterns are grown by searching their corresponding projected databases. Each of these sub-phases is detailed as follows.

1. Find the set of 1-length event patterns L_1. Given a database as shown in Fig. 3, PCMiner first scans the database to count the number of each event pairs and discards those events whose support is less than the minimum support threshold. If the *minsup* threshold is 2, all discovered 1-length event patterns whose support is not less than 2 constitute the 1-length event pattern set L_1. For example, in Fig. 4, (A^+A^-): 3 denotes the event pattern and its associated support count.

2. Construct projected databases for each 1-length event pattern. Let $L_1 = \{\alpha_1^1, \alpha_2^1 ... \alpha_n^1\}$ be the set of 1-length event patterns. For each α_i^1, a corresponding projected database $DB|_{\alpha_i^1}$ is created. $DB|_{\alpha_i^1}$ is a collection of suffix event sequences with regard to the prefix α_i^1.

3. k-length event pattern α is grown to the $(k+1)$-length event pattern α' through searching the projected database $DB|_\alpha$ corresponding to α $(k \geq 1)$. For a prefix α, PCMiner scans its projected database $DB|_\alpha$ once to find the set of local frequent event pairs $\{e_1, e_2 \ldots e_n\}$ and discards infrequent ones. Note that since event pairs are counted, these single events in the projected database will not be counted again. Frequent event pairs e_i are appended to the prefix α, generating the new frequent event pattern α' with the length increased by 1. Therefore, the set of $(\mathrm{k}+1)$-length event patterns prefixed with α are generated.

We illustrate the process of finding event patterns prefixed with (A^+A^-). By scanning the (A^+A^-) projected database $DB|_{(A^+A^-)}$, its local frequent event pairs are $(B^+B^-$: 3), $(C^+C^-$: 3), $(E^+E^-$: 2), and $(F^+F^-$: 3). Thus, the set of all 2-length event patterns L_2 prefixed with (A^+A^-) are found, and they are: $(A^+A^-B^+B^-$: 3), $(A^+A^-C^+C^-$: 3), $(A^+A^-E^+E^-$: 2), and $(A^+A^-F^+F^-$: 3). Recursively, all 2-length event patterns are used to find 3-length event patterns by constructing and searching their projected databases. By projecting $(A^+A^-B^+B^-)$, we find frequent event pairs from its projected database which are (C^+C^-: 3), (E^+E^-: 2), and (F^+F^-: 3). By appending these frequent event pairs to the prefix ($A^+A^-B^+B^-$), we have 3-length event patterns ($A^+A^-B^+C^+C^-B^-$: 3), ($A^+A^-B^+B^-E^+E^-$: 2), and ($A^+A^-B^+B^-F^+F^-$: 3). Similarly, we find ($A^+A^-C^+C^-E^+E^-$: 2) and ($A^+A^-C^+C^-F^+F^-$: 3) prefixed with ($A^+A^-C^+C^-$), and ($A^+A^-E^+F^+F^-E^-$: 2) prefixed with ($A^+A^-E^+E^-$). We find L_4 and L_5 in the same approach.

Phase III: Calculate Significance and Proximity for Event Patterns. For each event pattern generated in phase II, we collect the time information and location information from the event sequences. Based on these information, we calculate the statistic significance for each event pattern by Definition 3. We also discard insignificant ones if its significance is less than the significance threshold *minsig*. Given proximity threshold *minpro*, we calculate average proximity for each event pattern by Definition 4 and filter out those patterns whose proximity are less than *minpro*. For the running example, if the weight for *spatial_proximity* and *temporal_proximity* is set to be 0 and 1, respectively, the proximity of all 2-length event patterns are $Prox(A^+A^-B^+B^-) = 0.418$, $Prox(A^+A^-C^+C^-) = 0.358$, $Prox(A^+A^-E^+E^-) = 0.098$, $Prox(A^+A^-F^+F^-) = 0.082$, $Prox(B^+C^+C^-B^-) = 0.906$, $Prox(B^+B^-E^+E^-) = 0.224$, $Prox(B^+B^-F^+F^-) = 0.215$, $Prox(C^+C^-E^+E^-) = 0.209$, $Prox(C^+C^-F^+F^-) = 0.199$, $Prox(E^+F^+F^-E^-) = 0.879$. If the *minpro* is set to be 0.3, event patterns whose proximity is less than 0.3 are filtered out. The ultimate outcomes of our example are $<E^+F^+F^-E^->$ and $<A^+A^-B^+C^+C^-B^->$ and their respective proximity are 0.879 and 0.494.

Phase IV: Generating Periodic Event Patterns. After performing phase III, we obtain significant and proximate event patterns. Based on the time and location information collected in phase III, the algorithm estimates the time period, location, and probability for each event pattern by Definition 5. The outcomes of this phase is a set of event patterns associated with time intervals, location, and a probability (i.e., periodic composite IoT services).

4 Experimental Results

We systematically evaluate the approach proposed in this paper. The language used is Java and the experiments are performed on a 1.6 GHz AMD processor and 2 GB RAM under Windows 7. We evaluate the proposed approach using three real datasets, namely, Data1, Data2, Data3. Specifically, Data1 and Data2 are from CASAS datasets, which are collected in smart home environment [18]. For location information, we refer to the layout of sensors attached on objects for grouping objects into corresponding locations. Data1 and Data2 are in the format of <date, time stamp, sensor ID, on/off> (e.g., <2008-02-27, 12:46:37, M13, OFF >). Data3 is collected from a single apartment for two weeks [10]. The Data3 are in the format of <id, start time, end time, location> (e.g., <light,

Algorithm 1. PCMiner algorithm

```
Input: A spatio-temporal database ($DB$), region set $r = \{r_1, r_2...r_n\}$, support threshold ($minsup$),
    significance threshold ($minsig$), proximity threshold ($minpro$), weight of the $spatial\_proximity$
    ($w_1$), and weight of the $temporal\_proximity$ ($w_2$).
Output: A set of event patterns, a set of periodic event patterns.

    Procedure Divide_Search_Space ($DB$, $r$)
 1: transform $DB$ into $|r|$ numbers of sub-databases based on $r$;
 2: return $DB_{r_i}$;

    Procedure Search_Event_Pattern ($DB_{r_i}$, $minsup$)
 3: $L_1$ = find_1-length_pattern($DB_{r_i}$, $minsup$ );
 4: Prefixspan($\alpha$, $l$, $DB_{r_i}|_\alpha$)//$\alpha$ is an event pattern, $l$ is the length of $\alpha$ ($l \geq 1$), $DB_{r_i}|_\alpha$ is the
    $\alpha$-projected database.
 5: Scan $DB_{r_i}|_\alpha$ once, find the set of local frequent events $\{e_1,e_2...e_n\}$;
 6: for each frequent event $e_i$ do
 7: $\alpha' = \alpha + e_i$; //append $e_i$ to $\alpha$ to form the new event pattern $\alpha'$ .
 8: for each $\alpha'$ do
 9: add $\alpha'$ to the event pattern set $patternset$;
10: construct $\alpha'$-projected database $DB_{r_i}|_{\alpha'}$, and call Prefixspan ($\alpha'$, $l+1$, $DB_{r_i}|_{\alpha'}$);
11: return $patternset$;

    Procedure Compute_Significance_Proximity ($patternset$, $minsig$, $minpro$, $w_1$, $w_2$);
12: for each pattern $\alpha$ in $patternset$ do
13: computeSignificance($\alpha$, $minsig$); // Calculate significance for each pattern by Definition 3 and
    discard those patterns whose significance are less than $minsig$.
14: computeProximity ( $\alpha$, $w_1$, $w_2$, $minpro$);// Calculate proximity for each pattern by Definition
    4 and discard those patterns whose proximity are less than $minpro$.
15: return $patternset'$;

    Procedure Generate_Periodic_Event_Patterns ($patternset'$, $DB_{r_i}$);
16: for each pattern $\alpha$ in $patternset'$ do
17: find associated time and location information from $DB_{r_i}$;
18: estimate the time interval $T$, location $L$, and probability $P$ for $\alpha$ by Definition 5;
19: return $<\alpha, T, L, P>$;
```

7:00, 8:00, bedroom>). In addition, all datasets are annotated with corresponding daily activities. There are 5 and 8 activities in Data1 and Data2, respectively. For Data3, 23 activities are recorded and annotated. We conduct four sets of experiments. The first set is to evaluate the performance and scalability of PCMiner. The second set is to evaluate the effectiveness of the pruning strategies (i.e., significance and proximity). The third set is to evaluate the applicability of the proposed approach by showing the discovered periodic composite IoT services from real datasets. The fourth set is to measure the convenience by applying the discovered periodic composite IoT services.

The first set of experiments is conducted on a dataset which is a combination of three datasets i.e., Data1, Data2, and Data3. We vary the support threshold *sup* from 4% to 10%. Figure 5(a) shows the execution time of PCMiner decreases by increasing the support threshold. Figure 5(b) illustrates that the number of discovered event patterns decreases by increasing the support.

In the second set of experiments, we assess the effectiveness of $significance$ and *proximity* in pruning non-promising event patterns. Similar to our previous experiment, we use the combined dataset. We set the $significance$ to be 0.01. We test the effectiveness of $significance$ in reducing insignificant IoT service event patterns while varying different support threshold. Figure 6(a) depicts

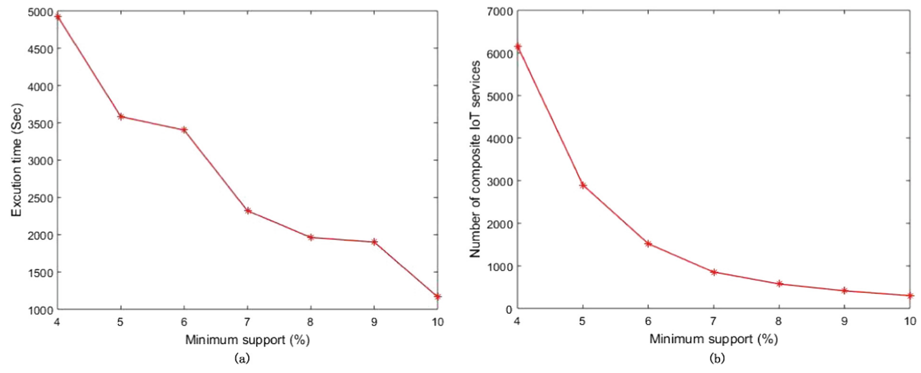


Fig. 5. Performance and scalability of PCMiner

the number of discovered patterns and significant patterns at different support threshold. The results show that the *significance* strategy performs effectively in pruning insignificant event patterns, which is an expected results. For example, the *significance* strategy can prune event patterns from 2954 to 1108 at the 5% support threshold. In addition, we test the effectiveness of *proximity* strategy in filtering out loosely correlated event patterns. Since the GPS point of each service in not available in the datasets, we set the weight of *spatial_proximity* to be 0 and the weight of *temporal_proximity* to be 1. We set the *proximity* to be 0.39. Figure 6(b) illustrates the number of discovered event patterns and proximate event patterns at different support threshold. The results show that the *proximity* strategy is effective. For example, the *proximity* strategy can prune loosely correlated event patterns from 2954 to 1705 at the 5% support threshold. These are expected results because the *significance* and *proximity* strategies enable PCMiner to filter out non-promising event patterns in each iteration and the search scope is shrunk for the next iteration.

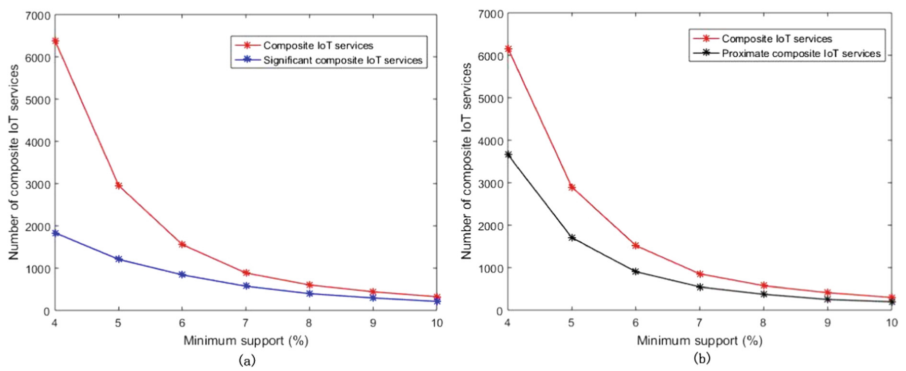


Fig. 6. Effectiveness of significance and proximity strategies

We perform the third set of experiments on Data3 to evaluate the applicability of our proposed approach. Table 1 shows the primary discovered composite IoT services. Some of the composite IoT services are indeed difficult to be discovered because they are less frequent. For example, the "lawn work" and "going out for entertainment" compositions occur only once during two weeks. Next, we check the discovered periodic compositions. Ideally, we want to associate one time interval with a composition. However, we discover that some IoT service compositions may associate multiple time intervals. For example, the "taking medication" service composition occurs in the morning and in the evening. This is a very practical issue. In this experiment, we group the discovered composition instances using a preliminary technique, that is, two time intervals are grouped together if they overlap. The tolerance threshold ζ is set to be 2 h. We can see from Table 1 that the resident performs some activities regularly. For example, one of the striking periodic activities is the "preparing breakfast". There is 75% chance that he/she will "prepare breakfast" during 5:16 and 6:51.

We conduct the fourth set of experiments on Data3 to measure how much convenience can be obtained by applying the discovered results in Table 1. We showcase some preliminary results. For example, based on the representative time, the "preparing breakfast" is likely to be followed by "watching TV" and "watching TV" is likely to be followed by "going out for shopping". Given the "preparing breakfast" activity on 4/22/2003, we can obtain 50% convenience.

Table 1. Primary discovered periodic composite IoT services from Data3

Periodic composite IoT services	Representative time intervals, location, probability
Taking medication	(1:54–2:00, in the kitchen, 0.5), (19:23–19:38, in the kitchen, 0.29)
Preparing breakfast	(5:45–6:20, in the kitchen, 0.75)
Preparing lunch	(11:15–12:05, in the kitchen, 0.64)
Preparing dinner	(18:15–18:48, in the kitchen, 0.5)
Going out for shopping	(7:52–8:05, in the hallway, 0.67)
Watching TV	(7:00–7:23, in the living room, 0.27), (14:27–15:22, in the living room, 0.33)

5 Related Work

A Web service mining framework is proposed to discover interesting composite services from available services [4]. This framework models Web services using ontologies and an efficient algorithm is proposed to discover composite services. In [24], a graphic model is proposed to represent the dependency among services. A general service mining framework is proposed based on an ontology service model [5]. Service relationships are established via ontology attributes. A Correlation Degree method is presented to evaluate the correlation strength among services. Most of existing work consider service relationships based on their input/output correlations, pre/post condition correlations etc. However, in the context of IoT, spatio-temporal relationships among IoT services are implicit and subtle. In [12], an efficient algorithm *CoPMiner* is developed to mine the temporal relationships among appliances in the smart home environment. The key idea of *CoPMiner* is to transform interval-based event sequences into endpoint based sequences. It also reformulates the problem of discovering temporal patterns among appliances as discovering frequent patterns from endpoint sequences. Location information regarding appliances is utilized to filter out insignificant temporal patterns. However, temporal distance is not considered in [12], which may result in undesirable frequent temporal patterns. In [9], an efficient algorithm IEMiner is proposed to discover temporal patterns for classification. In [2], a novel graph-based approach is proposed to capture the subtle relationships among things based on things' usage time, location, and users' social network information. The Random Walk with Restart method is applied to discover things relationships.

There are many research on human activity discovery. In [16], an efficient algorithm COM is proposed to discover human activity pattens from sensor event data. These patterns are used to build a HMM model for recognizing human activities. In [20], a probabilistic and Markov chain approach is proposed to discover complex human activity patterns. These patterns associated with context information are used to recognize activities. A general framework is proposed to address the problem of complex activity prediction by mining temporal sequence patterns from video [21]. A probabilistic suffix tree model is introduced to model activities. There have been little research into the human activity recognition which considers the periodic feature. For example, [22] discovers periodic activities from trajectory data such as staying in the office during daytime and staying at home in the evening.

6 Conclusion and Future Work

We addressed the problem of discovering periodic composite IoT services to provide personalized convenience to residents. An IoT service model and a composite IoT service model are proposed in terms of spatio-temporal aspects. The experimental results show our proposed *significance* and *proximity* strategies are effective in pruning non-promising composite IoT services. The periodic composite IoT service model is introduced and is applied to provide convenience. We introduce a new algorithm PCMiner to discover periodic composite IoT services. Future work includes improving the performance of PCMiner. Furthermore, we will apply the discovered periodic composite IoT services to build an intelligent system for providing convenience.

Acknowledgment. This research was partly made possible by NPRP 9-224-1-049 grant from the Qatar National Research Fund (a member of The Qatar Foundation) and DP160100149 and LE180100158 grants from Australian Research Council. The statements made herein are solely the responsibility of the authors.

References

1. Atzori, L., Iera, A.: The Internet of Things: a survey. J. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Yao, L.: Unveiling correlations via mining human-thing interactions in the web of things. ACM Trans. Intell. Syst. Technol. **8**(5), 62 (2017)
3. Miorandi, D.: Internet of Things: vision, applications and research challenges. Ad Hoc Netw. **10**(7), 1497–1516 (2012)
4. Zheng, G., Bouguettaya, A.: Service mining on the web. IEEE Trans. Serv. Comput. **2**(1), 65–78 (2009)
5. Huang, B., Bouguettaya, A., Dong, H., Chen, L.: Service mining for Internet of Things. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 566–574. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_36
6. Ghari Neiat, A., Bouguettaya, A.: Crowdsourcing of Sensor Cloud Services, pp. 1–116. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91536-4
7. Issarny, V., Bouloukakis, G., Georgantas, N., Billet, B.: Revisiting service-oriented architecture for the IoT: a middleware perspective. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 3–17. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_1
8. Bouguettaya, A., et al.: A service computing manifesto: the next 10 years. Commun. ACM **60**, 64–72 (2017)
9. Patel, D., Hsu, W.: Mining relationships among interval-based events for classification. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 393–404. ACM (2008)
10. Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home using simple and ubiquitous sensors. In: Ferscha, A., Mattern, F. (eds.) Pervasive 2004. LNCS, vol. 3001, pp. 158–175. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24646-6_10
11. Shao, W., Bouguettaya, A.: Clustering big spatiotemporal-interval data. IEEE Trans. Big Data **2**, 190–203 (2016)
12. Chen, Y.-C., Chen, C.-C., Peng, W.-C., Lee, W.-C.: Mining correlation patterns among appliances in smart home environment. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014. LNCS (LNAI), vol. 8444, pp. 222–233. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06605-9_19
13. Bouguettaya, A., et al.: End-to-end service support for mashups. IEEE Trans. Serv. Comput. **3**, 250–263 (2010)
14. Chen, Y.-C.: Significant correlation pattern mining in smart homes. ACM Trans. Intell. Syst. Technol. (TIST) **6**(3), 35 (2015)
15. Wu, Q.: Cognitive internet of things: a new paradigm beyond connection. IEEE Internet of Things J. **1**, 129–143 (2014)
16. Rashidi, P., Cook, D.J.: COM: a method for mining and monitoring human activity patterns in home-based health monitoring systems. ACM Trans. Intell. Syst. Technol. (TIST) **4**(4), 64 (2013)
17. Meyer, S.: A survey of research on context-aware homes. In: Proceedings of the ACSW, pp. 159–168 (2003)
18. Singla, G., Cook, D.J.: Tracking activities in complex settings using smart environment technologies. Int. J. Biosci. Psychiatry Technol. (IJBSPT) **1**(1), 25 (2009)
19. De Souza, R.M.C.R., De Carvalho, F.A.T.: Clustering of interval data based on cityblock distances. Pattern Recognit. Lett. **25**(3), 353–365 (2004)

20. Saguna, S., Zaslavsky, A.: Complex activity recognition using context-driven activity theory and activity signatures. ACM Trans. Comput.-Hum. Interact. (TOCHI) **20**(6), 32 (2013)
21. Li, K., Fu, Y.: Prediction of human activity by discovering temporal sequence patterns. IEEE Trans. Pattern Anal. Mach. Intell. **36**, 1644–1657 (2014)
22. Li, Z.: Mining periodic behaviors for moving objects. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1099–1108(2010)
23. Pei, J.: Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Trans. Knowl. Data Eng. **16**(11), 1424–1440 (2004)
24. Huang, F., Smidts, C.: Causal mechanism graph – a new notation for capturing cause-effect knowledge in software dependability. Reliab. Eng. Syst. Saf. **158**, 196–212 (2017)
25. Neiat, A.G.: Crowdsourced coverage as a service: two-level composition of sensor cloud services. IEEE Trans. Knowl. Data Eng. **29**, 1384–1397 (2017)

CrowdMashup: Recommending Crowdsourcing Teams for Mashup Development

Faisal Binzagr and Brahim Medjahed(✉)

Department of Computer and Information Science,
University of Michigan - Dearborn, Dearborn, USA
{faisalb,brahim}@umich.edu

Abstract. Mashups involve the collaboration of multiple developers to build Web applications out of pre-existing APIs. A large body of research focused on recommending APIs for mashups. However, very few contributions looked at recommending developers. In this paper, we propose *CrowdMashup*, a crowdsourcing approach for mashup teams recommendation. We analyze online developer communities and API directories to infer developers' interests in APIs through natural language processing. We predict missing interest values using the alternating least square method for collaborative filtering. We also model interactions (comments and replies) among developers as a weighted undirected graph and introduce a sociometric to identify socially related developers. We propose an algorithm, based on the concept of cliques in graph theory, that combines developers' skills and sociometric to recommend efficient and balanced teams. We describe a prototype implementation and conduct extensive experiments on real-world data and APIs to evaluate our approach.

Keywords: Mashup · Crowdsourcing · Team recommendation
Sociometric · Skills

1 Introduction

The past decade has witnessed an increasing interest in *mashup* development [4]. For instance, the popular `programmableWeb`[1] API directory currently includes about 8,000 mashups. *Mashups* are Web applications that aggregate pre-existing APIs (or services) to create valuable services with added functionality [27]. Mashup development generally involves several APIs requiring a variety of technological skills such as REST, SOAP, JSON, XML, and security. This often calls for the collaboration of multiple developers to reduce the overall mashup cost (e.g., development time). A large body of research focused on recommending APIs for mashups [13]. However, very few contributions looked at recommending

[1] https://programmableweb.com.

developers to be part of mashup development teams. With the substantial number of available APIs and programmers, finding skilled mashup developers is not straightforward. For instance, `programmableWeb` lists more than 19,000 APIs. The `StackOverflow`[2] and `GitHub`[3] developer community platforms report estimated 9 and 27 millions subscribers, respectively. Besides, the software industry has recently seen a new trend where *crowdsourcing* companies (e.g., `Topcoder`[4]) sell services to corporate, mid-size, and small-business clients, and pay community members (i.e., developers) for their work. These companies also organize open tournaments and programming challenges in which programmers are organized in teams to compete against each other. Therefore, it is important to form balanced teams with skilled developers.

Crowdsourcing is a powerful sourcing model to perform a broad range of hard tasks by splitting the work between workers [22]. It has been used in software development to perform vital activities such as implementation, design, coding, or testing [24]. Selecting appropriate developers should be performed carefully to improve productivity [20]. In the context of mashups, two factors contribute to successful developer recommendation. First, mashups involve various APIs that require a large array of *skills*. A recent study shows that the interest of project members toward specific tasks leads to better outcomes [15]. Hence, it is vital to pick developers that possess the right skills, demonstrate significant interest in the mashup, and have a good reputation among their peers. Second, it is necessary to form teams with members that can get along with each other. Studies have confirmed that strong social relationships among members increase team performance [10]. Most interactions among mashup and API developers take place via online communities such as `StackOverflow` and `GitHub`. Positive discussions between developers, through questions and answers, tend to increase their social ability and productivity.

In this paper, we propose *CrowdMashup*, a crowdsourcing-based approach for recommending teams of developers for mashups. We analyze `StackOverflow` and `programmableWeb` to generate teams that best statisfy mashup requirements. To the best of our knowledge, this is the first work to address recommendation in mashups from developer's perspective. The main contributions of the paper are summarized below:

– We use natural language processing [17] to assign *interest scores* to developers in using APIs. As developers may omit to comment on certain APIs, we predict *missing scores* using the alternating least square method for collaborative filtering [21]. We combine the computed interest scores and reputation values of developers in the community to quantify their *skills*.
– We define a *sociometric* to assess social relationships among developers in the community. *Sociometry* is a quantitative method in psychology for measuring social relationships [26]. We model interactions (comments and replies) among

[2] https://stackoverflow.com.
[3] https://github.com.
[4] https://topcoder.com.

developers as a weighted undirected graph. The weight of each edge represents the number of interactions between developers modeled as nodes.
- We propose an algorithm to generate teams from *mashup queries*. The query is a specification of the mashup requirements. We adopt the concept of *cliques* from graph theory to identify strongly related developers [5]. A *clique* is a subset of vertices from the sociometric graph where every two distinct vertices are adjacent. We compare the skills of the developers in the clique along with their sociometric scores to recommend *top-t teams*. We also describe a prototype implementation and conduct experiments on real-world data and APIs to evaluate our algorithm.

The rest of this paper is organized as follows. We propose the *CrowdMashup* approach in Sect. 2. We describe the implementation and performance study in Sect. 3. In Sect. 4, we overview related work. We conclude in Sect. 5.

2 The CrowdMashup Approach

The *CrowdMashup* architecture (Fig. 1) is composed of two major components: *Analysis of the Developer Community* (ADC) and *Crowdsourcing Team Generation* (CTG).

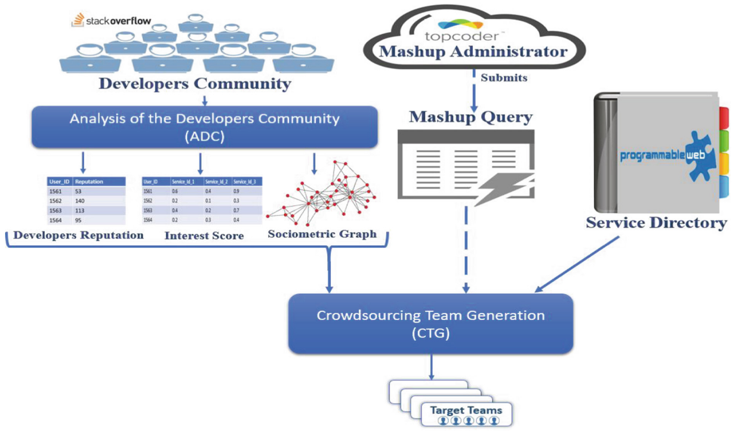


Fig. 1. CrowdMashup architecture

ADC runs *offline*, i.e., independently of any request to create mashup development teams. It analyzes the `StackOverflow` community to calculate and predict the interest of developers in adopting and using APIs. Nowadays developer communities become a troubleshooting manual, where many developers share experiences, issues, and solutions [18]. For instance, `StackOverflow` has more than 16 million questions and 24 million answers in 2018. The `LinkedIn` API uses

StackOverflow as a reference, at their official page, to support programmers in technical issues. Developer communities also showcase the level of affinity among developers. Many programmers may end up collaborating in projects as a result of their interactions in online communities [10].

CTG runs *online* at the reception of a *mashup query* from the *mashup administrator*. It returns efficient teams that best satisfy the mashup query requirements. The *mashup administrator* is a user or entity looking for teams of developers to collaborate on a mashup. Topcoder is an example of potential mashup administrator. It offers software development services to third party clients, contracting individual community programmers to work on specific tasks. It also holds design competition, thus offering design services to clients.

2.1 Analysis of the Developer Community (ADC)

ADC analyzes StackOverflow to generate three data structures (Fig. 1): *interests table* (U_I), *reputation table* ($\hat{U_R}$) and *sociometric graph* (SG).

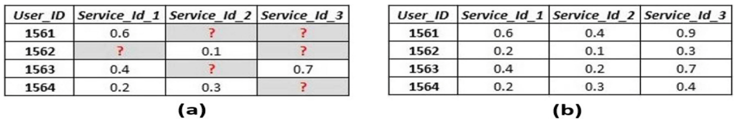
User Interests Table (U_I) - The initial step before analyzing the developer community is to prepare the list of APIs used in that community. To that end, we crawled all APIs from programmableWeb and extracted the *name* and *primary category* of each service using the Scrapy framework[5]. Since StackOverflow has about 66 millions comments (questions and answers), we focused on the ones that are related to APIs. We filtered StackOverflow comments using the API names retrieved from programmableWeb.

The next step is to analyze developers' comments and assign scores of interest in using APIs. For that purpose, we applied sentiment analysis to get the interest score $U_I(u_i)$ for each user u_i. We parsed comments using Stanford NLP[6] parser, which utilizes *recursive neural networks* (tree-structured models) for sentiment analysis [17]. For example, the comment "... Google Visualization API has several ways to do each task so it's important to know what you have already done and we could start there..." returns a positive interest value about Google Visualization API. An example of negative interest about Google Maps API is: "I simply have no experience with the Google Maps API ...".

Since certain APIs are not discussed by some developers, we ended-up with missing interest scores (Fig. 2). To solve this problem, we utilized the *Alternating Least Squares* (ALS) collaborative filtering technique [21]. In ALS, developers and their scores are described by a small set of latent factors used to predict the missing interest scores for all developers. Accordingly, we completed interest scores for all developers and APIs as shown in Fig. 2. If an API is listed on programmableWeb but unknown (i.e., not discussed) on StackOverflow, then ALS cannot complete the missing interest scores for this API. To deal with this issue, we average the interest scores of u_i for all APIs on StackOverflow that have the same category as the unknown API. Then, we assign the average score

[5] https://scrapy.org/.
[6] https://nlp.stanford.edu/.

as u_i's interest score for this API. If no API with similar category is commented by u_i, we average u_i's interest scores for all APIs discussed by u_i (Fig. 2).



User_ID	Service_Id_1	Service_Id_2	Service_Id_3
1561	0.6	?	?
1562	?	0.1	?
1563	0.4	?	0.7
1564	0.2	0.3	?

(a)

User_ID	Service_Id_1	Service_Id_2	Service_Id_3
1561	0.6	0.4	0.9
1562	0.2	0.1	0.3
1563	0.4	0.2	0.7
1564	0.2	0.3	0.4

(b)

Fig. 2. Interests table

User Reputation Table ($\hat{U_R}$) - `StackOverflow` has a reputation system which provides the level of expertise $U_R(u_i)$ for each user u_i. Since the extracted reputation has highly distributed values, we applied the *z-score* normalization to write reputation values into a standardized structure. The following formula shows the final reputation $\hat{U_R}$ for u_i, where μ and σ represent the mean and standard deviation of all reputation values, respectively:

$$\hat{U_R}(u_i) = \frac{U_R(u_i) - \mu}{\sigma}$$

Sociometric Graph (SG) - Another major aspect in teams formation is the social ability, or *sociometry*, among developers [26]. The idea is to make sure that members of the same team can actually work together. Studies showed that social relationships among members of the same team have a positive impact on improving the team productivity [3]. In our approach, we use interactions among developers via questions and replies in `StackOverflow` as a mean to estimate their social relationships. Developers that engage in more conversations with each other in online communities have more chances to successfully collaborate.

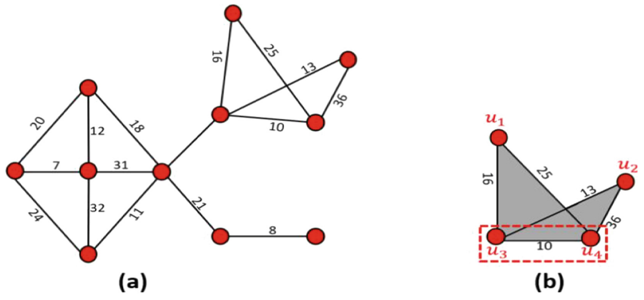


Fig. 3. Sociometric graph

We scanned the history of interactions among developers in `StackOverflow` regardless if the questions/replies are related to APIs or not. Then, we modeled those interactions as an undirected weighted graph, called *sociometric graph*

(SG). Each node in the graph represents a user (Fig. 3a). An edge (u_i, u_j) states an existing interaction (question or reply) between users u_i and u_j. Developers may interact at various levels, from few questions/replies to thousands. To capture this aspect, we label each edge (u_i, u_j) with a weight $W_e(u_i, u_j)$ that gives the number of interactions between users u_i and u_j:

$$W_e(u_i, u_j) = \#\ interactions\ between\ (u_i, u_j)$$

2.2 Mashup Query Specification

Mashup administrators interact with *CrowdMashup* through *mashup queries.* A *mashup query* Q defines the mashup requirements through a tuple $Q = (t, m, A)$ where:

- t: is the number of required teams.
- m: is the number of members within each team.
- A: is a list of APIs that compose the mashup.

Each element in the list A is defined as $<API_{ID}, API_w>$. API_{ID} is an ID that uniquely identifies the API. API_w is the weight (in the range 0 to 1) of the API. It represents the level of importance of the corresponding API in the mashup. For instance, a location-based mashup (e.g., transportation) may rely on a mapping API; the mapping API should be given a significant weight value to make sure the most skilled developers are recommended for this API. A small API_w implies that the API may not be mastered by all teams members; a big API_w indicates that the API should be mastered by most teams members.

Example 1. Assume we want to build 5 teams of 3 developers for a mashup that composes `GoogleMaps` (with ID 1 and weight 0.6), `Foursquare` (with ID 4 and weight 0.4) and `Last.fm` (with ID 5 and weight 0.1). The mashup query is specified by $t = 5$, $m = 3$, and $A = [<1, 0.6>, <3, 0.4>, <5, 0.1>]$.

Mashup administrators may not want to limit mashups to specific APIs by providing the list of API categories instead of APIs. For instance, they may refer to "Social" as a required category instead of `Facebook` or `Twitter`. In this case, we automatically fetch all APIs that belong to the categories listed by the administrator from `programmableWeb` and replace each category by the matching APIs.

Example 2. Assume we want to build 5 teams of 3 developers for a mashup that composes APIs from the *Mapping* and *Social* categories with 0.6 and 0.4 weights, respectively. Assume that APIs with IDs 1, 30, and 47 belong to *Mapping* and APIs with IDs 3, 17, and 22 relate to *Social.* The query is specified by $t = 5$, $m = 3$, and $A = [<1, 0.6>, <30, 0.6>, <47, 0.6>, <3, 0.4>, <17, 0.4>, <22, 0.4>]$.

2.3 Crowdsourcing Team Generation (CTG)

CTG generates teams that best satisfy the mashup query requirements. It uses as input the sociometric graph SG as well as interests and reputation tables, U_I and $\hat{U_R}$. Before describing the CTG algorithm, we introduce the metrics to calculate the performance of a team based on SG, U_I, and $\hat{U_R}$.

We evaluate the *skills* of each user (i.e., developer) u_i in the community based on u_i's reputation and interest in each $API^j \in A$ specified in the mashup. The user's interest in API^j is multiplied by API^j_w to take into account the weight (i.e., importance) assigned by the administrator to each API:

$$User_{skills}(u_i) = \hat{U_R}(u_i) * \sum_{API^j \in A} U_I(u_i, API^j) * API^j_w \tag{1}$$

Based on the skills of each user u_i given in formula (1), we define the skills of a team T composed of m members as the sum of the skills of all members:

$$Team_{skills}(T) = \sum_{u_i \in T} User_{skills}(u_i) \tag{2}$$

Using the sociometric graph SG, we also introduce the *sociometric score* of T to quantify the level of collaboration between members. The sociometric score $Team_{sociometric}(T)$ of T accumulates the weights of all edges that connect members in T and divide it by the number m of team members:

$$Team_{sociometric}(T) = \frac{\sum_{ui \in T, uj \in T, (ui,uj) \in SG} W_e(u_i, u_j)}{m} \tag{3}$$

From formulas (1) and (2), we define the overall performance of T by summing the skills and sociometric of the team:

$$Team_{Performance}(T) = Team_{skills}(T) + Team_{sociometric}(T) \tag{4}$$

The CTG algorithm (Algorithm 1) identifies strongly connected members in the sociometric graph SG using the concept of *cliques* in graph theory. A *clique* C is a subset of vertices of an undirected graph such that every two distinct vertices in C are adjacent [5]. We use the *Bron Kerbosch* algorithm [5] to return cliques in the *AllCliques* list (line 3). Another important data structure is *SharedCliques* (lines 1 and 16). Each element SC in this list contains *common vertices* between cliques as well as the remaining vertices (called *potential vertices*) in the cliques. For example, Fig. 3b depicts two adjacent cliques C_1 =$\{u_1, u_3, u_4\}$ and C_2 =$\{u_2, u_3, u_4\}$. The common and potential vertices are defined by *SC.common*=$\{u_3, u_4\}$ and *SC.potential*=$\{u_1, u_2\}$, respectively. Due to space limitation, we omit the algorithm for the *GetSharedCliques()* function.

CTG uses *AllCliques* and *SharedCliques* to recommend the *top-t* (t is the number of required teams). Each element in the returned *TeamsList* is composed of the team's members and performance of the team as defined in formula (4). The algorithm first looks for cliques of size m (i.e., cliques with required number of members). If more teams still need to be generated ($TeamsList.size()<t$), then CTG explores the shared cliques.

Algorithm 1. Crowdsourcing Team Generation (CTG)

```
Input  : $\hat{U_R}$ $Table, U_I$ $Table$, Sociometric Graph $SG$, Mashup Query $Q$
Output: $TeamsList$ (recommended teams)
1  $SharedCliques \leftarrow null$;
2  $TeamsList \leftarrow null$;
3  $AllCliques \leftarrow BronKerboschAlgorithm(SG)$;
4  foreach $C \in AllCliques$ do
5      if $(C.size() == m)$ then
6          T = All users from C;
7          Calculate Performance$_{Team}$(T) of team T;
8          $TeamsList$.add($T, Performance(T)$);
9          $AllCliques = AllCliques - C$;
10     end
11 end
12 if $( TeamsList.size() >= t)$ then
13     $TeamsList \leftarrow sort()$;      //By team performance
14     Return Top-t teams from $TeamsList$;
15 end
16 $SharedCliques = GetSharedCliques(AllCliques)$;
17 foreach $SC \in SharedCliques$ do
18     if $(SC.common.size() >= m)$ then
19         T = Top users from SC.common; // By skills or sociometric
20         Calculate Performance$_{Team}$(T) of team T;
21         $TeamsList$.add($T, Performance(T)$);
22         $SharedCliques = SharedCliques - SC$;
23     end
24 end
25 if $( TeamsList.size() >= t)$ then
26     $TeamsList \leftarrow sort()$;      //By team performance
27     Return Top-t teams from $TeamsList$;
28 end
29 foreach $SC \in SharedCliques$ do
30     if $(SC.common.size() < m)$ then
31         T = Users from SC.common + remaining top from SC.potential;
           // By skills or sociometric
32         Calculate $Performance_{Team}$(T) of team T;
33         $TeamsList$.add($T, Performance(T)$);
34         $SharedCliques = SharedCliques - SC$;
35     end
36 end
37 $TeamsList \leftarrow$ sort();      //By team performance
38 Return Top-t teams from $TeamsList$;
```

We identify the following three cases during team recommendation:

Case 1: Cliques have m members (lines 4–15) - CTG first parses cliques with the exact number of members. If the size of a clique C is m, then all members of C are used to form a team. We calculate the performance of T, insert T and its performance to $TeamsList$, and remove C from *AllCliques*. If $TeamsList$ reaches the desired number t of teams (lines 12–15), $TeamList$ is sorted based on performance and the top-t teams are returned, hence ending the algorithm. Otherwise, we process shared cliques (Case 2).

Case 2: Shared cliques have al least m members (lines 16–28) - CTG processes shared cliques that have enough members in their common vertices. It picks the top-m members from common vertices using one of two selection options (line 19). (i) *CTG by Skills*: m members with the highest skills are selected; and (ii) *CTG by sociometric*: m members with the highest sociometric scores are selected. The corresponding teams are inserted into $TeamsList$ as described in Case 1; the shared cliques used to build the teams are removed from $SharedCliques$. If $TeamsList$ reaches the desired number t of teams (lines 25–28), $TeamList$ is sorted based on performance and the top-t teams are returned, hence ending the algorithm. Otherwise, we proceed to Case 3.

Case 3: Shared cliques have less than m members (lines 29–38) - CTG handles the shared cliques that do not have enough members in their common vertices. It picks the remaining members from the potential vertices in the shared cliques. The remaining members are selected using *CTG by Skills* or *CTG by Sociometric* as described in Case 2 (line 31). Teams along with their calculated performance are added to $TeamsList$ and the top-t teams are returned.

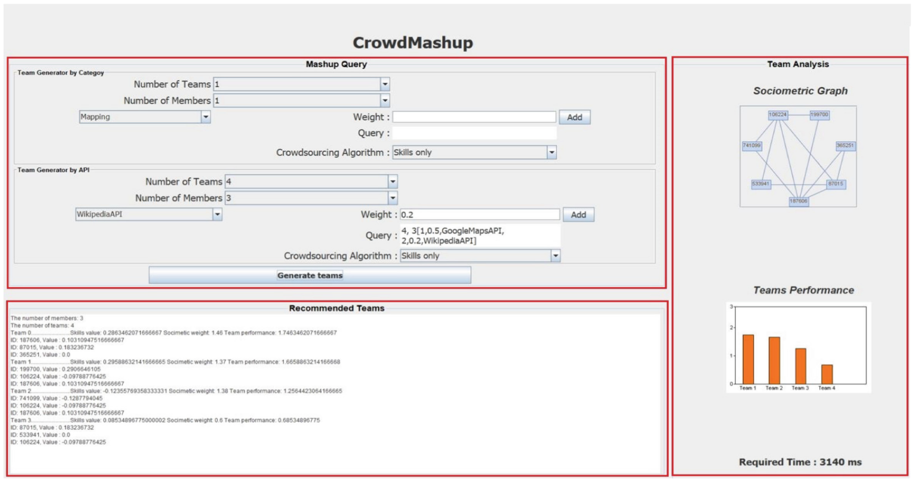
3 Implementation and Performance

In this section, we describe the *CrowdMashup* prototype implementation. Then, we evaluate the performance of our approach using real-world data and APIs.

3.1 CrowdMashup Prototype

We implemented a *CrowdMashup* prototype in Java. We used `Google BigQuery`[7] to retrieve comments about APIs from `StackOverflow`. We collected 8,617 comments related to 583 APIs. We used the `Jgrapgt` library[8] to handle graphs and identify cliques. We utilized `Stanford Natural Language Processing` library to calculate developers' attitude (interest) toward APIs. We used `Apache Spark's scalable machine learning` (MLlib) library[9] to deal with missing developers' interest values.

7 https://cloud.google.com/bigquery/public-data/stackoverflow.
8 http://jgrapht.org/.
9 https://spark.apache.org/.



Team Generator by Categoy

Number of Teams: 1
Number of Members: 1
Mapping
Weight :
Query :
Crowdsourcing Algorithm : Skills only

Team Generator by API

Number of Teams: 4
Number of Members: 3
WikipediaAPI
Weight : 0.2
Query : 4, 3[1,0.5,GoogleMapsAPI, 2,0.2,WikipediaAPI]
Crowdsourcing Algorithm : Skills only

```
The number of members: 3
The number of teams: 4
Team 0.......................Skills value: 0.2863462071666667 Socimetic weight: 1.46 Team performance: 1.7463462071666667
ID: 187606, Value : 0.10310947516666667
ID: 87015, Value : 0.183236732
ID: 365251, Value : 0.0
Team 1.......................Skills value: 0.29588632141666665 Socimetic weight: 1.37 Team performance: 1.6658863214166668
ID: 199700, Value : 0.2906646105
ID: 106224, Value : -0.09788776425
ID: 187606, Value : 0.10310947516666667
Team 2.......................Skills value: -0.12355769358333331 Socimetic weight: 1.38 Team performance: 1.2564423064166665
ID: 741099, Value : -0.1287794045
ID: 106224, Value : -0.09788776425
ID: 187606, Value : 0.10310947516666667
Team 3.......................Skills value: 0.08534896775000002 Socimetic weight: 0.6 Team performance: 0.68534896775
ID: 87015, Value : 0.183236732
ID: 533941, Value : 0.0
ID: 106224, Value : -0.09788776425
```

Required Time : 3140 ms

Fig. 4. The *CrowdMashup* user interface

Figure 4 shows *CrowdMashup*'s graphical interface. Mashup administrators specify their queries through the *Mashup Query* pane (top left). They assign the number of required teams and members in each team. Administrators enter either a list of specific APIs or generic API categories along with their weights. They also pick the algorithm to be used to generate teams: (1) *Skills Only*: members are selected based on skills only. (2) *Sociometric Only*: members are selected based on sociometric only. (3) *CTG-Skills*: uses both skills and sociometric but gives priority to skills in dealing with shared cliques (lines 19 and 31 in Algorithm 1). (4) *CTG-Sociometric*: uses both skills and sociometric but gives priority to sociometric in dealing with shared cliques (lines 19 and 31 in Algorithm 1). The generated teams are shown in the *Recommended Teams* pane (bottom left). The pane shows each recommended team as a list of developer IDs. It also displays the calculated performance of each team and orders the generated teams based on their performance. The *Team Analysis* pane (right) displays the two metrics for team recommendation: sociometric sub-graph and team performances illustrated in a bar graph to visualize the performance of different teams. The time to generate teams is also shown in this pane.

3.2 Experiments

The aim of the experiments is to assess the ability of CTG to select teams with the best performance. We ran our experiments on a 64-bit Windows 10 environment, in a machine equipped with an Intel i7-7700HQ and 16 GB RAM. We measured the performance of the generated teams using three non-CTG algorithms: Random (members are randomly selected), Skills Only, Sociometric Only; and two CTG algorithms: CTG-Skills and CTG-Sociometric. We ran all experiments on real-world data and APIs from `StackOverflow` and `programmableWeb`.

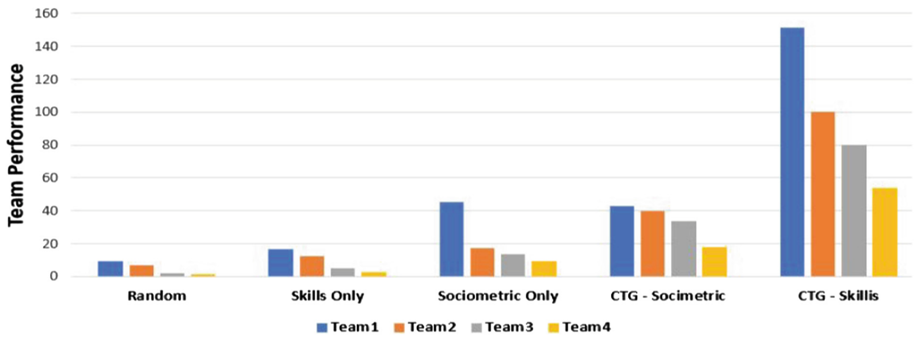


Fig. 5. Single query team performance for non-CTG (random, skills only, sociometric only) and CTG (CTG-skills, CTG-sociometric) Algorithms

Figure 5 compares the five algorithms using the same mashup query to generate four teams with seven members per team. First, we compare CTG vs. non-CTG algorithm in terms of team performance. CTG algorithms perform better than non-CTG algorithms due to combining sociometric and skills. Besides, CTG-Skills generates better teams than CTG-Sociometric. This is because vertices that are outside cliques are unlikely to return high sociometric values. Then, we compare the distribution of the performance of the four teams recommended by each algorithm. Figure 5 shows that team performance decreases steadily from the first to the last team in both CTG algorithms. Hence, CTG shows more balanced teams than non-CTG algorithms. For instance, there is significant difference (more than double) between the performance of the first and second teams in the Sociometric-Only algorithm.

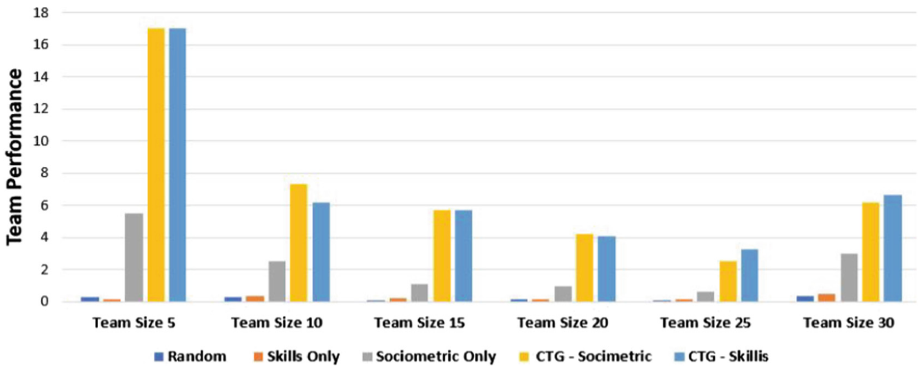


Fig. 6. Multiple queries team performance for different team sizes

We also conduct experiments to explore how forming teams with various sizes is handled by CTG. We randomly generated queries with sizes 5, 10, 15, 20, 25, and 30. We had 5 queries for each time size (for total of 30). As shown in Fig. 6, CTG algorithms always show better team performances than the non-CTG algorithms regardless of the team size. This is because non-CTG algorithms ignore sociometric, skills, or both (in the case of random). Overall, generating teams with bigger sizes (more than 10 members) leads to lower performance, as it is harder to find a large number of developers with the right skills and social relationships. Studies have shown that 3–7 developer teams are key to successful software projects (3–5 person teams would be the best)[10]. Hence, this makes CTG a suitable technique for team recommendation. For large team sizes (e.g., 25), CTG-Skills shows better team performance than CTG-Sociometric as finding cliques or shared cliques with larger sizes becomes challenging. For teams of size 2–5, CTG-Skills and CTG-Sociometric are comparable, and they largely outperform the three other algorithms: Random, Skills Only, and Sociometric Only. In teams of size 6–10, CTG-Sociometric shows better team than CTG-Skills as finding cliques or shared cliques with size 10 is still possible and improves the overall team performance.

4 Related Work

The growth and popularity of crowdsourcing has led to significant research on forming teams to facilitate collaborative software development [7]. Part of this research has focused on team structure, while other contributions focused on the complexity of the algorithm and economic factors for team building. [9] shows that network structure between members has a vital effect on team formation. It uses four different network structures to model team formation and compares the performance of each structure. [6] takes advantage of social network information and uses hierarchical structures (e.g., using "report to") between team members. [16] defines a self-organized team formation technique by allowing members to rate each other and use other information such as demographics (e.g., age, gender). [8] proposes a framework that recommends teams based on the skills and connection among members. It uses co-authorship in DBLP and clustering algorithms to find expert teams (sub-graph). [22] employs a dynamic programming technique in crowdsourcing based on the prior familiarity of members to generate target teams. It considers the availability (response time) of the members to find most familiar alternative members. [26] defines heuristic algorithms based on notions such as weak and strong ties in social networks. It utilizes two metrics to find social connection from an undirected weighted graph.

Several techniques dealt with the issue of improving the efficiency of the team formation process. [28] proposes a genetic algorithm with the goal of finding the best groups that can meet the defined tasks based on members availability, skills, and price. [11] introduces an approach for forming teams with specific skills from a vast professional community using network communication costs to optimize team formation. It calculates communication costs by using minimum spanning tree and the largest shortest path from the graph. [2] describes a greedy approach

[10] http://www.qsm.com/process_improvement_01.html.

for better performance considering team size and workload such as the number of tasks allocated to each member.

[20] and [14] propose a team formation technique based on pricing to find cost effective teams. [12] studies task coordination cost in crowdsourcing teams. It aims to facilitate self-coordination and communication among teams by distributing and synchronizing the project tasks. [10] introduces a technique for forming multiple teams to maximize the global efficiency of the teams considering skills, availability, sociometric (relationship), and allowed time (part-full time) members. [25] proposes a negotiation-based team formation technique where the deal to join the team is used as a formation factor. [15] investigates how personality affects team performance by applying the DISC (dominance, inducement, submission, compliance) personality test. [23] discusses team elasticity in software development such as the skills, experiences, response time and reliability of the workers. [1] proposes a data leak-aware system in crowdsourcing team by applying clustering algorithms that detect social interactions between members to avoid data leakage. [19] conducts a statistical analysis to investigate how to extract influence factors from successful teams.

CrowdMashup differs from existing approaches in multiples ways. First, to the best of our knowledge, this paper is the first to look at team recommendation for mashups. Second, we define a two-level approach to analyze developer communities. At the individual developer's level, we infer developer's interests in APIs through natural language processing and collaborative filtering. At the community level, we consider social relationships among developers as an important factor to recommend team members. We model interactions among developers as a weighted undirected graph and find cliques to identify strongly related developers. Note that our approach is different from the one introduced in [26] where members of the same team are selected from different cliques to ensure the impartiality of the execution result of a task. We use cliques to recommend teams composed of (socially) strongly connected members to improve productivity.

5 Conclusion

We propose the *CrowdMashup* approach to recommend teams for mashup development. The first *CrowdMashup* phase analyzes the `StackOverflow` developer community to infer developers' skills in using APIs. It also models the ability of developers to collaborate with each other via a sociometric graph. The second phase recommends crowdsourcing teams that best satisfy the requirements of a mashup query. We introduce a team recommendation algorithm that combines developers' skills and sociometric. We provide a prototype implementation and conduct experiments on real-world data and APIs from `StackOverflow` and `programmableWeb` to evaluate our approach. Experiments show promising results in generating efficient and balanced teams for mashup development.

References

1. Amor, I.B., Benbernou, S., Ouziri, M., Malik, Z., Medjahed, B.: Discovering best teams for data leak-aware crowdsourcing in social networks. ACM Trans. Web (TWEB) **10**(1), 2 (2016)
2. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: forming teams in large-scale community systems. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, pp. 599–608 (2010)
3. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: Proceedings of the 21st International Conference on World Wide Web, pp. 839–848 (2012)
4. Bouguettaya, A., et al.: A service computing manifesto: the next 10 years. Commun. ACM **60**(4), 64–72 (2017)
5. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973)
6. Ding, C., Xia, F., Gopalakrishnan, G., Qian, W., Zhou, A.: Teamgen: an interactive team formation system based on professional social network. In: Proceedings of the 26th International Conference on World Wide Web Companion, pp. 195–199 (2017)
7. Doan, A.H., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM **54**(4), 86–96 (2011)
8. Farhadi, F., Hoseini, E., Hashemi, S., Hamzeh, A.: Teamfinder: a co-clustering based framework for finding an effective team of experts in social networks. In: 12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, 10 December, pp. 107–114 (2012)
9. Gaston, M., Simmons, J., DesJardins, M.: Adapting network structure for efficient team formation. In: Proceedings of the AAAI 2004 Fall Symposium On Artificial Multi-agent Learning (2004)
10. Gutiérrez, J.H., Astudillo, C.A., Ballesteros-Pérez, P., Mora-Melià, D., Candia-Véjar, A.: The multiple team formation problem using sociometry. Comput. Oper. Res. **75**, 150–162 (2016)
11. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June – 1 July, pp. 467–476 (2009)
12. Lee, S.W., Chen, Y., Klugman, N., Gouravajhala, S.R., Chen, A., Lasecki, W.S.: Exploring coordination models for ad hoc programming teams. In: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp. 2738–2745 (2017)
13. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Comput. Surv. **48**(3), 1–41 (2016)
14. Liu, Q., Luo, T., Tang, R., Bressan, S.: An efficient and truthful pricing mechanism for team formation in crowdsourcing markets. In: 2015 IEEE International Conference on Communications (ICC), pp. 567–572 (2015)
15. Lykourentzou, I., Antoniou, A., Naudet, Y., Dow, S.P.: Personality matters: balancing for personality types leads to better outcomes for crowd teams. In: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work and Social Computing, pp. 260–273 (2016)
16. Lykourentzou, I., Wang, S., Kraut, R.E., Dow, S.P.: Team dating: a self-organized team formation strategy for collaborative crowdsourcing. In: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp. 1243–1249 (2016)

17. Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., McClosky, D.: The stanford coreNLP natural language processing toolkit. In: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55–60 (2014)
18. Nasehi, S.M., Sillito, J., Maurer, F., Burns, C.: What makes a good code example? A study of programming q&a in stackoverflow. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 25–34 (2012)
19. Pobiedina, N., Neidhardt, J., Calatrava Moreno, M.D.C., Werthner, H.: Ranking factors of team success. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 1185–1194 (2013)
20. Rokicki, M., Zerr, S., Siersdorfer, S.: Groupsourcing: team competition designs for crowdsourcing. In: Proceedings of the 24th International Conference on World Wide Web, pp. 906–915 (2015)
21. Ryza, S., Laserson, U., Owen, S., Wills, J.: Advanced Analytics with Spark: Patterns for Learning from Data at Scale (2017)
22. Salehi, N., McCabe, A., Valentine, M., Bernstein, M.: Huddler: convening stable and familiar crowd teams despite unpredictable availability. arXiv preprint arXiv:1610.08216 (2016)
23. Saremi, R.L., Yang, Y., Ruhe, G., Messinger, D.: Leveraging crowdsourcing for team elasticity: an empirical evaluation at topcoder. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pp. 103–112 (2017)
24. Stol, K.-J., Fitzgerald, B.: Researching crowdsourcing software development: perspectives and concerns. In: Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering, pp. 7–10 (2014)
25. Wang, W., Jiang, J., An, B., Jiang, Y., Chen, B.: Toward efficient team formation for crowdsourcing in noncooperative social networks. IEEE Trans. Cybern. **47**(12), 4208–4222 (2017)
26. Yin, X., et al.: Social connection aware team formation for participatory tasks. IEEE Access **6**, 20309–20319 (2018)
27. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. IEEE Internet Comput. **12**(5), 44–52 (2008)
28. Yue, T., Ali, S., Wang, S.: An evolutionary and automated virtual team making approach for crowdsourcing platforms. In: Li, W., Huhns, M.N., Tsai, W.-T., Wu, W. (eds.) Crowdsourcing. PI, pp. 113–130. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47011-4_7

A Variation Aware Composition Model for Dynamic Web Service Environments

Soumi Chattopadhyay[1] and Ansuman Banerjee[2(✉)]

[1] Indian Institute of Information Technology, Guwahati, Guwahati, India
[2] Indian Statistical Institute, Kolkata, Kolkata, India
ansuman@isical.ac.in

Abstract. Contemporary approaches for automated web service composition mostly deal with static web services. The underlying assumption here is that the web services participating in resolving a query are static and thereby, their functional and non-functional parameters change very infrequently or do not change at all. However, in reality, this assumption does not hold. New services are added to the repository, existing unpopular services are removed from the repository, service interfaces change due to changes in the specification. Classical service composition approaches therefore fall short to handle the dynamic behavior of a web service during composition. In this paper, we present a stochastic model of the web service composition problem to capture the dynamic behavior of web services from the functional perspective. We present experimental results on the ICEBE-2005 benchmarks to show the effectiveness of our proposed methods.

1 Introduction

The quest for efficient methods for automated web service composition [15,21,22,27] that can meet business needs and deliver guaranteed performance has drawn a significant research attention since long. A significant body of early work in this direction considered composition of static service repositories and proposed a number of composition solutions [14,16,25]. The underlying theory assumes that the functional parameters of a web service remain unchanged [9,19] or change rarely. However, in reality, this is often not a valid assumption. Service repositories have grown to exhibit significant dynamic characteristics today. A large number of new services are added to the service repository, while unpopular services are removed from the service repository. Due to continuous change in user requirements, the interface of a web service is often modified and thereby, the functional and non-functional parameters of a service may change. The dynamic characteristics over non-functional parameters have been studied in [3,23,26]. The authors in [12] conducted an 11 week-long survey of Web services on the Internet, collected from seekda.com, webservicelist.com, and xmethods.net to show the dynamic nature of the web services. It is evident from their experiments that the number of web services and their inputs, outputs fluctuate considerably.

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 694–713, 2018.
https://doi.org/10.1007/978-3-030-03596-9_50

Classical web service composition approaches that deal with static web services, may not fit well in this scenario and in most of the cases, these methods may end up producing a solution that may not be available at the time of execution. This is the main motivation behind our work.

In [10,13], the authors proposed dynamic service binding during composition. Uncertainty events in service composition are dealt with in [2,18]. In [12,17], the authors considered dynamic service composition and proposed an algorithm based on a variation of the Dijkstra's shortest path [7] algorithm. The authors in [26] identified a set of backup services, and using these backup services, the authors tried to improve the reliability of the composition solution. In this work, however, authors did not consider the dynamism of the functional parameters of a service (except, one characteristics, i.e., a service may be unavailable). This is the main focus in our paper. In [11], an adaptation approach for web service composition has been proposed. This paper measured the changed information values, which is potentially introduced when a service is updated in a business process and demonstrated how this adaptation takes place for different workflow patterns. However, most of the above proposals are able to handle the situation when the service workflow (a specific order of execution) is known. In reality, some times a query is specified in terms of input and output parameters. In such cases, the workflow is not known beforehand and therefore, these methods fail to handle it.

In this work, we propose automated web service composition approaches that can capture the dynamic behavior of a service from the functional perspective in the situation when the workflow is unknown. To make composition more realistic in dynamic environments, we assign a probability value with each service and its functional parameters and demonstrate how composition can be done in this situation. The inherent scalability limitation of optimization formulations renders them infeasible for large problem dimensions in real time. Therefore, in our final proposal, we propose a heuristic method that can generate a solution in a reasonable time limit. In summary, this paper has the following contributions:

- We model the dynamic behavior of a service.
- We propose an optimal and a heuristic approaches to handle this modeling.
- We perform an extensive experiment of our proposed methods on synthetically generated datasets and benchmark datasets to show the effectiveness of our proposal.

2 Background and Problem Formulation

A web service is a software component that takes a set of inputs, performs a specific task and produces a set of outputs. In classical web service composition, we are given:

- A set of web services $S = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$.
- For each $\mathcal{S}_i \in S$, a set of inputs $\mathcal{S}_i^{(ip)}$ and outputs $\mathcal{S}_i^{(op)}$.

- A query $\mathcal{Q}$, expressed in terms of a set of provided inputs $\mathcal{Q}^{(ip)}$ and a set of desired outputs $\mathcal{Q}^{(op)}$.

The objective of the classical web service composition problem is to serve a query by providing a solution, in terms of a set of services with a specific execution order, so that the functional dependencies [4] are preserved. However, the dynamic characteristics of the services, as discussed in Sect. 1, are mostly missing in the classical setting. In this paper, we augment the above composition problem with the following features to incorporate a dynamic setting.

- **Feature-1:** *A service in the service repository may or may not be available for execution, when the composition is done at query time.*
- **Feature-2:** *The interface of a service may change, and thereby, the functional parameters of a service (i.e., input-output parameters) may change at query time.*

The above features capture the dynamic aspects in functional behavior of a service. In other words, these allow us to capture the possible differences in functional parameters that may occur at query time, with respect to the original service description provided in the repository. We begin by describing the details of the model for dynamics.

3 Modeling Architecture

We now present the modeling of dynamic behavior of functional parameters of a service.

3.1 Modeling of Functional Characteristics

In order to model the dynamic characteristics of the functional behavior of a service, we assign a probability value with each service and its functional parameters, as shown below.

- Each service $\mathcal{S}_i \in S$ is available in the repository at query time with probability p_i, i.e.,
$$P(\mathcal{S}_i \text{ is available}) = p_i \tag{1}$$
- An output $io_j \in \mathcal{S}_i^{(op)}$ is produced by $\mathcal{S}_i$ at query time with probability $\beta_{i,j}$, i.e.,
$$P(io_j \text{ is produced by } \mathcal{S}_i \mid \mathcal{S}_i \text{ is executed}) = \beta_{i,j} \tag{2}$$

It may be noted that Eq. 1 models Event-1, i.e., the probability of a service being available in the repository. Equation 2 models Event-2, which is a conditional probability that captures the probability of producing an output io_j by $\mathcal{S}_i$, given that $\mathcal{S}_i$ is executed (denoted by the classical|operator) [20].

As already discussed earlier, once a query comes into a system, a dependency graph [4] or planning graph [5] is constructed based on the input-output dependency relationship. However, the conventional dependency graph or planning graph does not capture the dynamic characteristics of the services. Therefore, instead of constructing the dependency graph or planning graph, we propose to build a dependency network. The dependency network, which is also constructed based on service input-output dependency, is a variant of the classical dependency graph and also a variant of the planning graph. In the next subsection, we demonstrate the modeling of the dynamic behavior through our proposed dependency network.

3.2 The Dependency Network for Service Composition

Our proposed dependency network $\mathcal{G} = (V_S \cup V_{io}, E)$ is a variant of the classical AND-OR graph [8] consisting of two types of nodes: AND nodes (V_S) corresponding to the services and OR nodes (V_{io}) corresponding to the inputs/outputs of services. Since a node is associated with either a service or an input/output, the node is required to be *activated* in the network. Initially, the nodes corresponding to the query inputs are activated in the dependency network. Once a node is activated, its corresponding output links are available in the network. As the network is traversed, more nodes are activated depending on the availability of the links. Depending on the activation criteria, we now formally define AND and OR nodes.

Definition 1 [OR Node]: *A node $u_i \in V_{io}$ is called an OR node, if it is activated on availability of one of its input links.* ■

Definition 2 [AND Node]: *A node $v_i \in V_S$ is called an AND node, if the node is activated on availability of all its input links.* ■

Throughout this paper, we represent an AND node using v_i and OR node using u_i. We now discuss the properties of $\mathcal{G}$.

- $\mathcal{G}$ is a layered network.
- Each layer of $\mathcal{G}$ consists of either a set of OR nodes or a set of AND nodes.
- The OR layer (i.e., the layer consisting of only OR nodes) and the AND layer (i.e., the layer consisting of only AND nodes) alternate in $\mathcal{G}$.
- No links exist between nodes in the same layer. Links are available only between AND layer to OR layer/OR layer to AND layer.

To capture the dynamic characteristics of services, the dependency network has some additional features, as below.

- A probability value is assigned to each node and link of the dependency network.
- Each AND node is activated with probability p_i on availability of all its input links.
- Once an AND node v_i is activated, an output link (v_i, u_j) is available with probability $\beta_{i,j}$, where u_j is an OR node.

Example 1. Consider the service repository shown in Table 1. The first column of Table 1 shows the service name. Columns 2 and 3 represent the set of inputs and the outputs of the service. Column 4 represents the probability of generating an output io_j by a service $\mathcal{S}_i$ given that $\mathcal{S}_i$ is executed (as shown in Eq. 2). Finally, Column 5 represents the probability of availability of the service in the repository (as shown in Eq. 1).

Table 1. Definition of services

Service	Inputs	Outputs	Probability of output production	Available Probability
$\mathcal{S}_1$	io_1	io_4	$\beta_{1,4}$	p_1
		io_5	$\beta_{1,5}$	
$\mathcal{S}_2$	io_2	io_6	$\beta_{2,6}$	p_2
$\mathcal{S}_3$	io_3	io_7	$\beta_{3,7}$	p_3
$\mathcal{S}_4$	io_4	io_8	$\beta_{4,8}$	p_4
		io_9	$\beta_{4,9}$	
$\mathcal{S}_5$	io_5, io_6, io_7	io_9	$\beta_{5,9}$	p_5
		io_{10}	$\beta_{5,10}$	
		io_{11}	$\beta_{5,11}$	
$\mathcal{S}_6$	io_8	io_{12}	$\beta_{6,12}$	p_6
$\mathcal{S}_7$	io_9	io_{13}	$\beta_{7,13}$	p_7
$\mathcal{S}_8$	io_{10}	io_{12}	$\beta_{8,12}$	p_8
$\mathcal{S}_9$	io_{11}	io_{13}	$\beta_{9,13}$	p_9

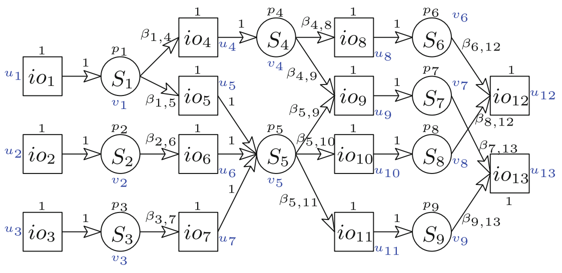


Fig. 1. Dependency Network corresponding to $\mathcal{Q}$

Consider a query $\mathcal{Q}$ with inputs $\{io_1, io_2, io_3\}$ and outputs $\{io_{12}, io_{13}\}$. The dependency network $\mathcal{G}$ constructed based on the query is shown in Fig. 1. Each AND node (as represented by a circle in the figure) from $V_S = \{v_1, v_2, \ldots, v_9\}$ represents a service, whereas, each OR node (as represented by a square box in the figure) from $V_{io} = \{u_1, u_2, \ldots, u_{13}\}$ represents an input/output. Nodes in the first OR layer do not have any input link and thereby, these nodes are activated by default. It may be noted that each AND node is annotated with a probability value, representing the probability of the service corresponding to the AND node being available in the service repository. Further, each link from an AND node v_i to an OR node u_j is also annotated with a probability value $\beta_{i,j}$, that represents the probability that the service $\mathcal{S}_i$ corresponding to v_i generates the output io_j corresponding to u_j with probability $\beta_{i,j}$, given that $\mathcal{S}_i$ is activated or executed. In order to make the dependency network uniform, we annotate each OR node and each link from an OR node to an AND node with probability value 1. It may be noted, unlike a service, an input/output is not controlled externally by the service provider. Therefore, the availability of an input/output is a certain event, though its activation depends on the query inputs or a service that produces it as its output. Hence, assigning the probability value 1 to each OR node and each link from an OR node to an AND node do not have any impact on constructing the solution. ■

We now discuss some properties of the dependency network.

- Each AND node v_i corresponds to a service $\mathcal{S}_i$. A probability p_i (i.e., $P(\mathcal{S}_i$ is available)) is assigned to v_i, denoting v_i is available with probability p_i. $Av(v_i)$ denotes that the service corresponding to v_i *is available.*

$$P(Av(v_i)) = p_i \tag{3}$$

- Each OR node u_i corresponds to an input / output. A probability 1 is assigned to u_i. Since a probability value is assigned to each AND node, to maintain uniformity, we assign a probability 1 to each OR node.

$$\forall u_i \in V_{io}, P(Av(u_i)) = 1 \tag{4}$$

- If an AND node v_i is activated, this implies the corresponding service $\mathcal{S}_i$ is also activated. $\mathcal{S}_i$ produces an output io_j with probability $\beta_{i,j}$. Consider u_j be the corresponding OR node of io_j. Hence, $\beta_{i,j}$ is assigned as the probability of having a link (v_i, u_j) available in $\mathcal{G}$, given that v_i is activated.

$$P(Av(v_i, u_j) \mid Ac(v_i)) = \beta_{i,j} \tag{5}$$

  $Ac(v_i)$ denotes v_i *is activated.*
- To maintain uniformity, we assign a probability 1 to each link from an OR node to an AND node in $\mathcal{G}$.

$$\forall (u_i, v_j) \in E, P(Av(u_i, v_j) \mid Ac(u_i)) = 1 \tag{6}$$

- A link in $\mathcal{G}$ is activated, if the source node of the link is activated and the source node generates the link.

$$\begin{aligned} P(Ac(u_i, v_j)) &= P(Ac(u_i) \cap Av(u_i, v_j)) \\ &= P(Ac(u_i)).P(Av(u_i, v_j)|Ac(u_i)) \end{aligned} \tag{7}$$

$$\begin{aligned} P(Ac(v_i, u_j)) &= P(Ac(v_i) \cap Av(v_i, u_j)) \\ &= P(Ac(v_i)).P(Av(v_i, u_j)|Ac(v_i)) \end{aligned} \tag{8}$$

This follows from the definition of conditional probability $P(A|B) = \frac{P(A \cap B)}{P(B)}$, where $\cap$ denotes the intersection operator.

– An AND node v_i is activated, if all its input links are activated and v_i is available.

$$P(Ac(v_i)) = P(\bigcap_{\substack{u_j \in V_{io} \\ \&(u_j, v_i) \in E}} Ac(u_j, v_i)) \ . \ P(Av(v_i)) \tag{9}$$

– An OR node u_i is activated, if any of its input links is activated and u_i is available, represented as a union over the corresponding inputs.

$$P(Ac(u_i)) = P(\bigcup_{\substack{v_j \in V_S \\ \&(v_j, u_i) \in E}} Ac(v_j, u_i)) \ . \ P(Av(u_i)) \tag{10}$$

where $\bigcup$ denotes the union operator.

3.3 Dependency Network Construction

We now present the details of the procedure of dependency network construction. We start with the query inputs and identify the set of services that can be directly or eventually (i.e., with the outputs of the services that are directly or eventually activated by the query inputs) activated by the query inputs. We construct an AND node corresponding to each service and an OR node corresponding to each input/output. The network links are constructed depending on the set of inputs and outputs of a service.

Once the network is constructed, we add two dummy nodes v_s and v_e to the network to represent the start and the end nodes of the network. v_s does not have any input link, however, a set of links is created from v_s to the nodes corresponding to the query inputs. Similarly, v_e does not have any output link, hence a set of links is created from the nodes corresponding to the query outputs to v_e. While constructing the dependency network, we start with the query inputs. Therefore, each node is connected to v_s through some path. However, it is not necessary that each path of the network starting from v_s ends at v_e. Therefore, we traverse the dependency network backward and identify the set of nodes belonging to the paths from v_e to v_s and finally, we remove the remaining set of nodes, which do not belong to any path from v_s to v_e, since these nodes do not take any part in resolving the query. It may be noted that v_s in the network is available and activated with probability 1, whereas, v_e has to be available with probability 1 for the query to be fulfilled.

$$P(Av(v_s)) = P(Ac(v_s)) = 1; P(Av(v_e)) = 1 \tag{11}$$

It may also be noted further that if v_e is activated with probability p_e, this implies we obtain the query outputs with probability p_e.

Layering the Dependency Network. Once the dependency network is constructed, the network is divided into multiple layers. This step is required for constructing a solution to a query. A node $v_i \in V_S$ in $\mathcal{G}$ belongs to a layer L, if the following condition is satisfied: for all $u_j \in V_{io}$, such that (u_j, v_i) is a link in $\mathcal{G}$, u_j belongs to a layer L', where $L' < L$. Similarly, a node $u_i \in V_{io}$ in $\mathcal{G}$ belongs to a layer L, if for all $v_j \in V_S$, such that (v_j, u_i) is a link in $\mathcal{G}$, v_j belongs to a layer L', where $L' < L$. The first layer of the network contains only v_s. A layer of the network is mathematically defined as:

$$V_L = \begin{cases} \{v_s\}, & L = 0 \\ \{u_i \in V_{io} | \forall v_j \in V_S, (v_j, u_i) \in E, v_j \in V_{L'}, L' < L\}, & L \text{ is odd} \\ \{v_i \in V_S | \forall u_j \in V_{io}, (u_j, v_i) \in E, u_j \in V_{L'}, L' < L\}, & L \text{ is even} \end{cases} \quad (12)$$

In order to ensure that the input-output edge relationship does not span across multiple layers, we add some dummy nodes in the network. In other words, if a link does not connect two nodes between two consecutive layers, we add a set of dummy nodes in the network. Consider a link $(u_i, v_j) \in E$, such that u_i belongs to a layer L and v_j belongs to a layer L' and $(L' - L) > 1$. In this case, we do the following:

- $\forall L_k$, such that $L < L_k < L'$, we add a dummy node w_{L_k} in layer L_k. If L_k is even, w_{L_k} is an OR node. Otherwise, w_{L_k} is an AND node.
- We add the following set of links in the network: $\{(u_i, w_{(L+1)}), (w_{(L+1)}, w_{(L+2)}), \ldots, (w_{(L'-1)}, v_j)\}$.
- $\forall w_{L_k} \in \{w_{(L+1)}, w_{(L+2)}, \ldots, w_{(L'-1)}\}$, we assign $P(Av(w_{L_k})) = 1$. This essentially means, the dummy nodes are always available with probability 1. This is required to ensure that the probability of generating a solution from the dependency network is not affected due to the insertion of the dummy nodes.
- $\forall (x, y) \in \{(u_i, w_{(L+1)}), (w_{(L+1)}, w_{(L+2)}), \ldots, (w_{(L'-1)}, v_j)\}$, we assign $P(Av((x, y))|Ac(x)) = 1$. This condition implies, once a dummy node is activated, it generates all its outputs with probability 1.

4 Dynamic Service Composition

It may be noted that the dependency network of a query contains all possible solutions to the query. The probability of generating a solution to a query from the dependency network is maximized, if the entire dependency network is returned as a solution to the query. However, in this case, the solution becomes inefficient in terms of its cost and quality, since the solution contains a set of redundant services (i.e., services without which the solution construction is possible). Therefore, here our objective is to *minimize the number of services*. In addition to the classical web service composition problem, we have the following constraint: the desired solution has to be obtained with a probability greater than or equal to $\alpha_{solution}$, $(0 \leq \alpha_{solution} \leq 1)$, where $\alpha_{solution}$ is provided by the user.

5 Solution Generation

We now discuss the procedure for generating a solution to a query from the dependency network. It may be noted that each solution to a query is a subnetwork of the dependency network. However, the converse is not true, i.e., each subnetwork of the dependency network is not a solution to the query. This is mainly because of the following reason. Each AND node of the dependency network is associated with a service and a service is activated only when all its inputs are available. Therefore, an AND node requires all its input links to be available. However, each OR node is associated with an input/output, which is available if a service produces it or it is a part of the query inputs. Therefore, only one link is sufficient to activate an OR node. Hence, a solution subnetwork requires (a) each AND node, belonging to it, to have the same number of incoming links as in the dependency network and (b) each OR node, belonging to it, to have at least one incoming link. Therefore, for ease of analysis, to find the optimal solution satisfying all the constraints, we transform the dependency network to a hyper dependency network, where each path of the hyper dependency network provides a solution to the query. We now define a few terms related to the hyper dependency network.

Definition 3 [Hyper Node]: *A node containing multiple homogeneous nodes (i.e., either AND nodes or OR nodes) of a dependency network is called a hyper node.* ■

A hyper node consisting of multiple OR nodes of a dependency network is an OR hyper node. Similarly, a hyper node consisting of multiple AND nodes of a dependency network is an AND hyper node.

Definition 4 [Hyper Edge]: *An edge containing multiple edges of a dependency network is called a hyper edge.* ■

Definition 5 [Hyper Dependency Network $\mathcal{HG} = (HV_S \cup HV_{io}, HE)$]: *A dependency network consisting of hyper nodes and hyper edges is called a hyper dependency network.* ■

We now present the construction of the hyper dependency network $\mathcal{HG} = (HV_S \cup HV_{io}, HE)$ from a dependency network $\mathcal{G}$. We start with v_e of $\mathcal{G}$. We then identify the set of nodes that are responsible for activating v_e and construct a hyper node consisting of this set. Once we have an OR hyper node $hu \in HV_{io}$, we construct all possible combinations of AND nodes that are responsible for activating each OR node corresponding to hu and for each combination, we construct an AND hyper node. In the process of constructing an AND hyper node hv, we keep track of all the edges that have been considered during this construction and create a hyper edge (hv, hu) consisting of the set of edges.

Example 2. Consider the dependency network $\mathcal{G}$ as shown in Fig. 2(a). Figure 2(b) shows the hyper dependency network $\mathcal{HG}$ corresponding to $\mathcal{G}$. Consider an OR hyper node hu_9 consisting of u_{12} and u_{13}. The set of edges that can activate u_{12} is $\{(v_6, u_{12}), (v_8, u_{12})\}$, while the set of edges that can activate u_{13} is $\{(v_7, u_{13}), (v_9, u_{13})\}$. Therefore, four combinations are possible to activate hu_9, i.e., $\{(v_6, u_{12}), (v_7, u_{13})\}$, $\{(v_6, u_{12}), (v_9, u_{13})\}$, $\{(v_8, u_{12}), (v_7, u_{13})\}$ and $\{(v_8, u_{12})\ , (v_9, u_{13})\}$. Considering these combinations, we have four AND hyper nodes $hv_6 : \{v_6, v_7\}$, $hv_7 : \{v_6, v_9\}$, $hv_8 : \{v_8, v_7\}$ and $hv_9 : \{v_8, v_9\}$ and four hyper edges $he_{21} : \{(v_6, u_{12}), (v_7, u_{13})\}$, $he_{22} : \{(v_6, u_{12}), (v_9, u_{13})\}$, $he_{23} : \{(v_8, u_{12}), (v_7, u_{13})\}$ and $he_{24} : \{(v_8, u_{12}), (v_9, u_{13})\}$. ■

Once we have an AND hyper node $hv \in HV_S$, we construct the set of OR nodes that can activate each AND node corresponding to hv and construct an OR hyper node hu consisting of this set. We then construct an hyper edge (hu, hv) in a similar manner as discussed above.

Example 3. Consider an AND hyper node hv_6 consisting of v_6 and v_7. The set of edges that can activate v_6 is $\{(u_8, v_6)\}$, while the set of edges that can activate v_7 is $\{(u_9, v_7)\}$. Therefore, the hyper dependency network contains an OR hyper node $hu_5 : \{u_8, u_9\}$ and a hyper edge $he_{17} : \{(u_8, v_6), (u_9, v_7)\}$. ■

Note that an AND hyper node has always inDegree (i.e., the number of incoming links) 1, an OR hyper node may have inDegree more than 1. We now define two more terms.

Definition 6 [Solution Dependency Network (SDN)]: *A subnetwork SDN $\mathcal{G}' = (V'_S \cup V'_{io}, E')$ of the dependency network $\mathcal{G} = (V_S \cup V_{io}, E)$ is a connected network, such that the following conditions hold:*

- $V'_S \subseteq V_S$, $V'_{io} \subseteq V_{io}$, $E' \subseteq E$ and $v_s, v_e \in V'_S$.
- *Each node and link in $\mathcal{G}'$ belongs to at least one path from v_s to v_e in $\mathcal{G}'$.*
- $\forall u_i \in V'_{io}$, $1 \leq inDegree(\mathcal{G}', u_i) \leq inDegree(\mathcal{G}, u_i)$.
- $\forall v_i \in V'_S$, $inDegree(\mathcal{G}', v_i) = inDegree(\mathcal{G}, v_i)$.
- *The probability assignments for each node and link in $\mathcal{G}'$ are same as in $\mathcal{G}$.*

where $inDegree(\mathcal{G}, v_i)$ denotes the inDegree [7] of v_i in $\mathcal{G}$. ■

It may be noted that the SDN is a subnetwork of the dependency network. Therefore, while the dependency network constructed based on a query consists of all solutions to the query, the SDN of the dependency network consists of only a subset of solutions. We now define the notion of a unique solution dependency network representing a unique solution to a query, where a unique solution refers to a solution, in which each service is dependent only on one service for a specific input.

Definition 7 [Unique Solution Dependency Network (USDN)]: *An USDN $\mathcal{G}' = (V'_S \cup V'_{io}, E')$ of a dependency network $\mathcal{G} = (V_S \cup V_{io}, E)$ is a SDN of $\mathcal{G}$, such that, $\forall u_i \in V'_{io}$, $inDegree(\mathcal{G}', u_i) = 1$.* ■

It may be noted that each path of the hyper dependency network is an USDN.

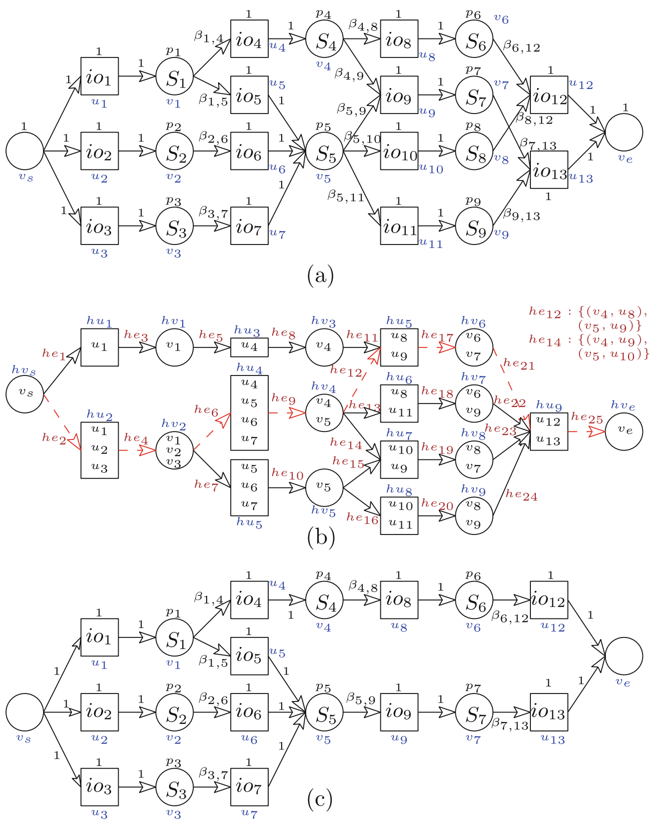

(a)
(b)
(c)
v_s
v_e
$he_{12} : \{(v_4, u_8), (v_5, u_9)\}$
$he_{14} : \{(v_4, u_9), (v_5, u_{10})\}$

Fig. 2. (a) Dependency network (b) Hyper dependency network (c) An USDN (Color figure online)

Example 4. Figure 2(c) shows an USDN corresponding to the dependency network shown in Fig. 2(a), which is generated from a path of the hyper dependency network as shown in Fig. 2(b). In the figure, the path is shown by a red dashed line. ■

It may be noted that being a SDN, an USDN constitutes a solution to a query. We now prove the following lemma.

Lemma 1. *An USDN contains a single solution to a query.* ■

Proof. In order to prove the above lemma, it is sufficient to prove that if a node is removed from an USDN, no solution is produced from the USDN. We prove this by contradiction. We first assume that *a solution can be produced by removing a node from the USDN.* We first consider the case where a solution can be produced by removing an OR node $u_i \in V'_{io}$ from the USDN. Since u_i belongs to at least one path from v_s to v_e, thereby, after removal of u_i, the path from u_i to v_e becomes invalid. We now consider the OR node $u_j \in V'_{io}$ belonging to the path from u_i to v_e, such that $(u_j, v_e) \in E'$. Being part of the USDN, u_j has a single input and thereby, u_j becomes inactivated, as the entire path from u_i to v_e becomes invalid. Therefore, the link (u_j, v_e) also becomes inactivated. Hence, v_e becomes inactivated. As a result, no solution to the query is produced, which contradicts our assumption. Using a similar argument, it can be proved that no solution can be produced by removing an AND node from the USDN.

It is evident from the above lemma that a solution to a query consists of at least one USDN. Though a single USDN is enough to generate a solution with a certain probability, however, multiple USDNs in a solution to a query are required just to increase the probability of generating a solution. The following lemma helps to compute the probability of the solution characterized by a USDN.

Lemma 2. *In an USDN $\mathcal{G}'$ of a dependency network $\mathcal{G}$,*

$$P(Ac(v_e)) = \prod_{v_i \in V'_S} P(Av(v_i)). \prod_{(v_i,u_j) \in E'} P(Av(v_i,u_j)|Ac(v_i)).$$

i.e., $P(Ac(v_e))$ is equal to the product of the probabilities of all nodes and links belonging to any path from v_s to v_e. ■

The proof of this lemma is omitted from this text due to space limitation. If a SDN $\mathcal{G}'$ of a dependency network $\mathcal{G}$ consists of m USDNs $\mathcal{G}_1, \mathcal{G}_2, \ldots . \mathcal{G}_m$, the activation probability of v_e of the SDN can be expressed as:

$$P(Ac^{\mathcal{G}'}(v_e)) = P(\bigcup_{i=1}^{m} Ac^{\mathcal{G}_i}(v_e)) \tag{13}$$

It may be noted that each $\mathcal{G}_i$ is an USDN of $\mathcal{G}'$, for $i = 1, 2, \ldots, m$ and $Ac^{\mathcal{G}_i}(v_e)$ denotes the activation event of v_e of $\mathcal{G}_i$. $P(\bigcup_{i=1}^{m} Ac^{\mathcal{G}_i}(v_e))$ can be computed using principle of inclusion and exclusion [6]. The end node v_e of a SDN $\mathcal{G}'$ can be activated through any of the USDNs belonging to the SDN. Therefore, the activation event of v_e in a SDN can be expressed as the union of the activation events of v_e for each USDN corresponding to $\mathcal{G}'$.

The Principle of inclusion and exclusion is stated as below.

$P(A_1 \cup A_2 \cup \ldots \cup A_n) = \sum_i P(A_i) - \sum_{i,j} P(A_i \cap A_j) + \sum_{i,j,k} P(A_i \cap A_j \cap A_k) - \ldots$ where each A_i, for $i = 1, 2, \ldots, n$, represents an event of a random experiment. In case of two USDN networks, the above expression can be applied as:

$$P(Ac^{\mathcal{G}_1}(v_e) \cup Ac^{\mathcal{G}_2}(v_e)) = P(Ac^{\mathcal{G}_1}(v_e)) + P(Ac^{\mathcal{G}_1}(v_e)) - P(Ac^{\mathcal{G}_1}(v_e) \cap Ac^{\mathcal{G}_1}(v_e))$$

$$P(Ac^{\mathcal{G}_1}(v_e) \cap Ac^{\mathcal{G}_1}(v_e)) = P(Ac^{\mathcal{G}_1}(v_e)).\ P(Ac^{\mathcal{G}_2}(v_e)|Ac^{\mathcal{G}_1}(v_e))$$

$P(Ac^{\mathcal{G}_2}(v_e)|Ac^{\mathcal{G}_1}(v_e))$ is computed by multiplying all the link and node probabilities of the SDN containing $\mathcal{G}_1$ and $\mathcal{G}_2$.

We now demonstrate the algorithm for generating a solution to a query from the dependency network. The first step of our algorithm is to generate the hyper dependency network $\mathcal{HG}$ from a given dependency network $\mathcal{G}$. Once we have a hyper dependency network, we construct an USDN from each path of the hyper dependency network. To construct an USDN from a path of the hyper dependency network, we split each hyper node and hyper edge into the set of nodes and edges of the dependency network. Consider G^* be the set of all possible USDNs constructed from $\mathcal{G}$. We compute the activation probability of v_e for each USDN as stated in Lemma 2. We then construct the power set $\wp^{G^*}$ of G^*. It may be noted, each set belonging to $\wp^{G^*}$, which is a subset of G^*, forms a SDN of $\mathcal{G}$. Since our objective is to minimize the number of services in a solution, we next construct a list of SDNs sorted in ascending order based on the number of AND nodes without considering the dummy nodes in that network. Finally, we consider the SDNs one by one from the sorted list and compute the activation probability of v_e of each SDN as stated in Eq. 13. For any SDN, if the activation probability of v_e is greater than or equal to $\alpha_{solution}$, we return the SDN. If no such SDN $\mathcal{G}'$ of $\mathcal{G}$ exists for which $P(Ac^{\mathcal{G}'}(v_e)) \geq \alpha_{solution}$, this implies the query cannot be answered with probability $\alpha_{solution}$.

Here, we have demonstrated an optimal algorithm for service composition for a dynamic environment. Though the optimal algorithm is able to generate the optimal solution satisfying all the constraints, however, it suffers from scalability issues for large problem dimensions in real time. This is mainly because of the following two expensive operations.

1. *Generation of the hyper dependency network from a dependency network*, while constructing the AND hyper layer, we construct all possible AND hyper nodes for each OR hyper node. This step combinatorially explodes.
2. *Computation of the power set of the set of USDNs.*

In the next section, we, therefore, propose a suboptimal algorithm that can produce a solution faster than the optimal one. However, it compromises on solution quality.

6 A Heuristic Algorithm

In this section, we propose a suboptimal solution using a variant of the classical memory bounded A* algorithm [24]. Here, our objective is to find a SDN of a dependency network constructed in response to a query without constructing the entire hyper dependency network. Given the dependency network $\mathcal{G}$ constructed in response to a query, our objective is to generate a solution path of the hyper dependency network $\mathcal{HG}$ corresponding to $\mathcal{G}$, such that, all constraints are satisfied *as much as possible*. To do so, we first associate a level with the nodes of the dependency network as follows:

$$\begin{aligned} Level(v_s) &= 0; \\ Level(v_i) &= Level \text{ (predecessor of } v_i \text{)} + 1; \end{aligned} \tag{14}$$

We now present the detailed algorithm. The state space of the A* algorithm is $2^{|V_S|}$, where V_S is the set of AND nodes in the dependency network. A state is a collection of one or multiple nodes. The initial state of our algorithm is the state consisting of only v_e and the goal state is the state consisting of v_s. Depending on the node type, a state is classified into two categories: AND state and OR state. The state consisting of a set of AND nodes is called an AND state and the state consisting of a set of OR nodes is called an OR state.

We now discuss the operator of our algorithm. To be precise, the operator of this algorithm indicates how we generate one state from another state. Once we encounter a state, we construct its neighboring state set. Here, we construct maximum n number of neighbor states, where n is a given parameter. The parameter n is used to bound the number of nodes we explore in this algorithm. If we encounter an AND state, only one neighboring OR state is possible. However, if we encounter an OR state, the number of possible neighbor states is equal to $\prod_{i=1}^{k}(2^{n_i}-1)$, where we assume that an OR state consists of k number of OR nodes and each OR node has n_i number of input links, for $i = 1, 2, \ldots, k$. Although, one incoming link is sufficient to activate an OR node belonging to an OR state, we may need multiple incoming links for an OR node to increase its activation probability. Therefore, the number of ways we can choose more than one incoming link from the set of incoming links of an OR node is $2^{n_i}-1$, where n_i is the number of incoming links of the OR node. The expression refers to the cardinality of the power set of the set of predecessor AND nodes associated with an OR node excluding the empty set. Therefore, the number of ways we can choose more than one link for each OR node is equal to $\prod_{i=1}^{k}(2^{n_i}-1)$, which refers to the cardinality of the Cartesian product of the power set of the set of AND nodes associated with each OR node excluding the empty set.

Example 5. We now illustrate the neighboring state construction methodology on the dependency network shown in Fig. 2(a). The initial state s_1 is an AND state consisting of v_e. The only possible neighbor of s_1 is an OR state s_2 consisting of $\{u_{12}, u_{13}\}$. The incoming links for u_{12} are $(v_6, u_{12}), (v_8, u_{12})$ and the incoming links for u_{13} are $(v_7, u_{13}), (v_9, u_{13})$. The number of possible AND states is therefore, $(2^2 - 1) * (2^2 - 1) = 9$. The set of neighboring AND states is computed as follows. The power set of $\{v_6, v_8\}$ excluding the empty set is: $\wp_1 = \{\{v_6\}, \{v_8\}, \{v_6, v_8\}\}$. The power set of $\{v_7, v_9\}$ excluding the empty set is: $\wp_2 = \{\{v_7\}, \{v_9\}, \{v_7, v_9\}\}$. Cartesian Product of $\wp_1 \times \wp_2 = \{\{v_6, v_7\}, \{v_7, v_8\}, \{v_6, v_7, v_8\}, \{v_6, v_9\}, \{v_8, v_9\}, \{v_6, v_8, v_9\}, \{v_6, v_7, v_9\}, \{v_7, v_8, v_9\}, \{v_6, v_7, v_8, v_9\}\}$. ■

We now discuss the cost function for each state. The cost function $f(s)$ in each state s is calculated as $f(s) = g(s) + h(s)$, where, $g(s)$ is the number of AND nodes without considering the dummy nodes and $h(s)$ is the heuristic function, which denotes how far the current AND state is from the goal state and defined as:

$$\begin{aligned} h(s) = &\ \ Level(v_i); && v_i \in s \text{ and } s \text{ is an AND state} \\ = &\ \ 0; && s \text{ is an OR state} \end{aligned} \tag{15}$$

Finally, we discuss about constraint validation. We validate the constraint on the cumulative set. The cumulative set consists of the set of nodes from the initial state to the current state. If multiple incoming links of an OR node are considered in a solution, we choose the maximum probability value among all the assigned probability values corresponding to all incoming links of the OR node that has been considered, in order to compute the activation probability of the OR node. However, the actual activation probability of the OR node is more than the computed value. This happens because of the following reason. We first consider an OR node u_i belonging to the cumulative set having two incoming links e_1 and e_2. We further consider two events: (i) u_i is activated through e_1 and (ii) u_i is activated through e_2. We need to compute the probability of the following event : u_i is activated through either e_1 or e_2. Since these two events are not independent, we cannot simply multiply the probability value of the individual events. On the other side, we cannot compute the actual probability value, since in this approach, we traverse the dependency network in the backward direction. Therefore, at the moment of computation, we cannot compute the probability of the availability of e_1 and e_2. Hence, the actual probability of generating a solution is more than the probability value computed by the heuristic algorithm. We use the following equation to represent the constraint.

$$\prod_{\substack{v_i \in \\ \text{cumulative set}}} P(Av(v_i)) \times \prod_{\substack{(v_i, u_j) \in \\ \text{cumulative set}}} Max_{v_i} P(Av(v_i, u_j) \mid Ac(v_i)) \geq \alpha_{(solution)}; \tag{16}$$

If any state violates any of the above constraints, we ignore the path through which the node is reached and in that case, we do not update the cost value of that node. If no state is found due to constraint violation, we choose a state that has the minimal violation. The final solution is obtained from the set of states from the initial state to the goal state having minimal cost. Since less number of hyper nodes are explored in this method, this method is expected to be faster than the optimal algorithm, however, it compromises on solution quality, since the entire solution space is not explored due to the memory bound parameter.

7 Experimental Results

We implemented our proposed algorithms in Java (version 1.7.0_95, 32 bit). All experiments were performed on an Ubuntu (version 14.04 LTS, 32 bit, kernel 3.13.0-77-generic) Linux system on a 2.53 GHz machine with 4 GB DDR3 RAM. The algorithms were evaluated against a synthetically generated dataset and the eight public repositories of the 2005 ICEBE Web Service Challenge (WSC) [1]. To the best of our knowledge, this work is the first of its kind. Hence, we provide comparative experimental results between the two approaches proposed in this paper, the optimal one and the heuristic.

Services		Inputs	Outputs	# Services
FS	(1)	FromAirport, ToAirport, DepartureDate, ReturnDate No.ofPersons	FlightDetails	5
	(2)	FromAirport, ToAirport, DepartureDate, ReturnDate No.ofPersons, Class	FlightDetails	5
FF		FlightDetails, FlightPreferenceCriteria	FlightToBook, DepartureTime, FlightArrivalTime, ReturnDepartureTime, ReturnArrivalTime	2
FB		FlightToBook, Credential	FlightTicket	8
TS	(1)	FromStation, ToStation, DepartureDate, ReturnDate No.ofPersons	TrainDetails	5
	(2)	FromStation, ToStation, DepartureDate, ReturnDate No.ofPersons, Class	TrainDetails	5
TF		TrainDetails, TrainPreferenceCriteria	TrainToBook, DepartureTime, TrainArrivalTime, ReturnDepartureTime, ReturnArrivalTime	2
TB		TrainToBook, Credential	TrainTicket	5
AB		FlightArrivalTime, HotelAddress	AirportCabBookingConfirmation	5
HS	(1)	ArrivalDate, CheckOutDate No.ofRooms, City	HotelDetails, HotelRatings	2
	(2)	ArrivalDate, CheckOutDate No.ofRooms, City, Budget	HotelDetails	2
LP		HotelAddress, VisitingPlaces, VisitingPreferenceCriteria, DetailsOfVisitingPlaces	Tour Plan: {(VisitingPlace, Date, Time), ...}	2
RS	(1)	VisitingPlace	RestaurantName, Cuisine PhoneNumber, Direction, Rating	2
	(2)	VisitingPlace, Cuisine	RestaurantName, PhoneNumber, Direction	4
	(3)	VisitingPlace, Cuisine	RestaurantName, PhoneNumber	2
SS		VisitingPlace	ShoppingMallAddress, Direction	5
WF		VisitingPlace	WeatherForecastReport	2
VF		City	VisitingPlaces,DetailsOfVisitingPlaces	5
HF		HotelDetails, HotelPreferenceCriteria	HotelName, HotelAddress, City, HotelToBook, Facilities	2
HB		HotelToBook, Credential	HotelBookingConfirmation	8
SB	(1)	TrainArrivalTime, HotelAddress, No.OfPersons	StationCabBookingConfirmation	5
	(2)	TrainArrivalTime, No.OfPersons HotelAddress, CarTypeToHire	StationCabBookingConfirmation	2
LB		HotelAddress, TourPlan CarTypeToHire	CityCabBookingConfirmations	8
PS		HotelAddress, No.OfPersons TourPlan	RouteDetails	2
EM	(1)	VisitingPlace	EmergencyService, ContactDetails	2
	(2)	VisitingPlace, EmergencyService	ContactDetails	2

Fig. 3. Service description

7.1 Dataset Description

We now present a brief description of our in-house dataset as shown in Fig. 3. We considered 19 different service categories. Each service category performs a specific operation/task. Under each category, there are 2 or 3 different sub categories. Each sub category is selected based on input-output parameters. The services under a specific sub category have identical set of inputs and outputs. We used an in-house web crawler and the open travel alliance[1] dataset to get the number of services for some service categories (e.g., searchFlight, bookFlight, searchHotel, bookHotel, forecastWeather, bookAirportTransport, bookLocalTransport, searchRestaurant etc.). Consider a query $\mathcal{Q}$ with inputs {*FromAirport, ToAirport, DepartureDate, ReturnDate, No.ofPersons, Class, FlightPreferenceCriteria, Credential, ArrivalDate, CheckOutDate, No. ofRooms, City, Budget, HotelPreferenceCriteria, VisitingPreferenceCriteria, Cuisine*} and desired output {*FlightTicket, HotelBookingConfirmation, AirportCabBookingConfirmation, CityCabBookingConfirmations, WeatherForecastReport, RestaurantName, PhoneNumber*}. The total number of services involved in resolving

[1] http://www.opentravel.org/.

the query was 73. Later, we randomly generated the number of services corresponding to each service category to analyze the performance of our algorithms. We used normal distribution with mean 0 and standard deviation 1 to generate the probability of the services being available in the repository and to generate the probability of producing each output by each service.

Table 2. Comparison of composition time for Case 1

n	# Services	Optimal(ms)	Heuristic(ms)	Constraint violation
3	73	19478	186	-
3	78	24378	9756	-
3	85	112789	201	8%
3	113	158793	339674	11%
3	220	time out	18576	17%
3	280	time out	32856	12%
3	370	time out	7745879	31%
2	280	time out	71	18%
4	280	time out	87342	11%
8	280	time out	197568	8%
12	280	time out	33658734	7%
15	280	time out	time out	-

We now compare the performance of the optimal algorithm with respect to the heuristic algorithm. We set $\alpha_{solution} = 0.7$. We experimented with a few different values of n for the heuristic method. We divided our experiment into two categories.

Case 1: Comparison on Our Dataset: We compare the optimal algorithm with the heuristic algorithm on our synthetically generated dataset. Table 2 shows a comparison between the performances of the optimal and the heuristic algorithms. It is evident from the table, the heuristic algorithm is significantly faster than the optimal algorithm. Columns 3 and 4 of Table 2 show the average composition time for both the optimal and the heuristic algorithms. Column 5 of Table 2 shows the performance degradation (which is measured as the percentage of constraint violation) of the heuristic algorithm. The first 7 rows of Table 2 above the horizontal line, show the comparison between the computation time and the constraint violation with increase in the number of services. The remaining rows (marked in a different shade) in Table 2 present the comparison between the computation time and the constraint violation with increase in the value of n. In 9 cases, the heuristic algorithm violates the constraint as evident from the table. However, in 7 cases, the heuristic algorithm is able to produce a solution, but the optimal algorithm is unable to produce any result. Furthermore, it is also evident from the table that as the value of n increases, the constraint degradation decreases and computation time increases.

Case 2: Comparison on the ICEBE-2005 Dataset: We compare the optimal algorithm with the heuristic algorithm on the ICEBE-2005 dataset. The

Table 3. Comparison of composition time for Case 2

Data set	# Services	Optimal time (ms)	Heuristic time (ms)	Constraint violation
Out composition	143	198765	7151	5%
Composition 1-20-4	2156	Time out	13869	8%
Composition 1-20-16	2156	Time out	34865	3%
Composition 1-20-32	2156	Time out	89625	2%
Composition 1-50-4	2656	Time out	1368	11%
Composition 1-50-16	2656	Time out	24596	17%
Composition 1-50-32	2656	Time out	14982	2%
Composition 1-100-4	4156	Time out	6619	7%
Composition 1-100-16	4156	Time out	348169	6%
Composition 1-100-32	4156	Time out	187964	18%
Composition 2-20-4	3356	Time out	2497645	21%
Composition 2-20-16	6712	Time out	32876	24%
Composition 2-20-32	3356	Time out	99164925	36%
Composition 2-50-4	5356	Time out	14785	11%
Composition 2-50-16	5356	Time out	5986732	9%
Composition 2-50-32	5356	Time out	6785297	23%
Composition 2-100-4	8356	Time out	7413921	8%
Composition 2-100-16	8356	Time out	3257691	53%
Composition 2-100-32	8356	Time out	913645889	31%

dataset contains 19 service repositories. Corresponding to the first service repository (Out Composition), there are 4 queries. For the remaining service repositories, there are 11 queries. Table 3 shows a comparison between the performances of the optimal and the heuristic algorithms when executed with n as 3. Column 2 of Table 3 shows the number of services in the ICEBE-2005 dataset. Columns 3 and 4 of Table 3 show the average composition time for both the optimal and the heuristic algorithms. It is evident from the table, the heuristic algorithm is significantly faster than the optimal algorithm. Column 5 of Table 3 shows the performance degradation of the heuristic algorithm. Though the heuristic algorithm violates the constraints, however, in most of the cases the optimal algorithm fails to generate a solution, whereas the heuristic algorithm is able to produce a solution. It is evident from our experiments that the heuristic algorithm is more efficient in terms of computation time.

8 Conclusion and Future Directions

This paper presents a dynamic variation aware service composition algorithm from the functional perspectives. As future work, we are currently working on extending our proposal to develop more sophisticated techniques for complex dependency service networks. Also we are looking at the scenario when new services are added in the repository or new outputs are available in the system. We believe that our work will open up a lot of new research directions in the general paradigm of composition for dynamic environments.

References

1. Home page of the web services challenge at the IEEE conference on e-business engineering ICEBE (2005). http://www.comp.hkbu.edu.hk/simctr/wschallenge/
2. Alfrez, G.H., Pelechano, V.: Facing uncertainty in web service compositions. In: ICWS, pp. 219–226 (2013)
3. Chattopadhyay, S., Banerjee, A.: QSCAS: QoS aware web service composition algorithms with stochastic parameters. In: ICWS, pp. 388–395 (2016)
4. Chattopadhyay, S., et al.: A scalable and approximate mechanism for web service composition. In: ICWS, pp. 9–16 (2015)
5. Chen, M., Yan, Y.: QoS-aware service composition over graphplan through graph reachability. In: SCC, pp. 544–551. IEEE (2014)
6. Cohen, D.I.: Basic techniques of combinatorial theory. Technical report (1978)
7. Cormen, T.H.: Introduction to Algorithms. MIT Press, Cambridge (2009)
8. Dechter, R., Mateescu, R.: Mixtures of deterministic-probabilistic networks and their and/or search space. In: UAI, pp. 120–129. AUAI Press (2004)
9. El Hadad, J., et al.: Tqos: transactional and qos-aware selection algorithm for automatic web service composition. IEEE TSC **3**(1), 73–85 (2010)
10. Feng, Y., et al.: Dynamic service composition with service-dependent QoS attributes. In: ICWS, pp. 10–17 (2013)
11. He, Q., Yan, J., Jin, H., Yang, Y.: Adaptation of web service composition based on workflow patterns. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 22–37. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89652-4_6
12. Jiang, W., et al.: Continuous query for QoS-aware automatic service composition. In: ICWS, pp. 50–57 (2012)
13. Kazhamiakin, R., et al.: Data-flow requirements for dynamic service composition. In: ICWS, pp. 243–250 (2013)
14. Klein, A., Ishikawa, F., Honiden, S.: Efficient heuristic approach with improved time complexity for QoS-aware service composition. In: ICWS, pp. 436–443 (2011). https://doi.org/10.1109/ICWS.2011.60
15. Kona, S., Bansal, A., Gupta, G.: Automatic composition of semanticweb services. In: ICWS, pp. 150–158 (2007). https://doi.org/10.1109/ICWS.2007.52
16. Li, Y., Lin, C.: QoS-aware service composition for workflow-based data-intensive applications. In: ICWS, pp. 452–459 (2011). https://doi.org/10.1109/ICWS.2011.18
17. Lv, C., Jiang, W., Hu, S., Wang, J., Lu, G., Liu, Z.: Efficient dynamic evolution of service composition. In: TSC (2017). https://doi.org/10.1109/TSC.2015.2466544
18. Mostafa, A., Zhang, M.: Multi-objective service composition in uncertain environments. In: TSC (2015). https://doi.org/10.1109/TSC.2015.2443785
19. Oh, S.C., Lee, D., Kumara, S.R.: Effective web service composition in diverse and large-scale service networks. IEEE TSC **1**(1), 15–32 (2008)
20. Papoulis, A.: Probability, Random Variables, and Stochastic Processes. McGraw Hill, New York (1984)
21. Rao, J., Su, X.: A survey of automated web service composition methods. In: Cardoso, J., Sheth, A. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30581-1_5
22. Rodriguez-Mier, P., et al.: Automatic web service composition with a heuristic-based search algorithm. In: ICWS, pp. 81–88 (2011). https://doi.org/10.1109/ICWS.2011.89

23. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic QoS and soft contracts for transaction-based web services orchestrations. TSC **1**(4), 187–200 (2008)
24. Russell, S.J.: Efficient memory-bounded search methods. In: ECAI, vol. 92, pp. 1–5 (1992)
25. Sepulveda, C., et al.: QoS aware descriptions for restful service composition: security domain. WWW **18**(4), 767–794 (2015). https://doi.org/10.1007/s11280-014-0278-0
26. Wagner, F., Klöpper, B., Ishikawa, F., Honiden, S.: Towards robust service compositions in the context of functionally diverse services. In: Proceedings of the 21st international conference on World Wide Web, pp. 969–978. ACM (2012)
27. Zeng, L., et al.: Quality driven web services composition. In: WWW, pp. 411–421 (2003)

A Model-Driven Framework for Automated Generation and Verification of Cloud Solutions from Requirements

Hamid R. Motahari Nezhad[1], Taiga Nakamura[1], Adi Sosnovich[2], Peifeng Yin[1], and Karen Yorav[2](✉)

[1] IBM Almaden Research Center, San Jose, CA, USA
motahari.hamid@gmail.com, {taiga,peifengy}@us.ibm.com
[2] 2IBM Haifa Research Lab, Haifa, Israel
{adisos,yorav}@il.ibm.com

Abstract. Cloud computing projects require the design of a so-called Cloud Solution, which is an architectural blueprint for a particular cloud environment. A cloud solution defines the hosting infrastructure (servers, VMs, etc.), software stack, and services such as network, backup, disaster recovery, management, etc. The design of a cloud solution needs to consider existing client environments and future environment's requirements, and at the same time comply with the cloud provider's portfolio and limitations. As such, the design of enterprise cloud solutions is a very complex and challenging problem. In this paper, we present a novel framework for provider-side cloud solution design based on model-driven and formal methods that facilitates the job of automated solution generation, starting from client requirements and resulting in a complete and correct cloud solution. We present a set of novel methods and a tool, called COOL, which implements the method and is used in production in a large Cloud service provider.

Keywords: Model-driven engineering · Enterprise cloud · Cloud applications engineering

1 Introduction

Over the last decade, the rate of adoption of cloud computing in the enterprise has accelerated significantly. Commonly, enterprises buy infrastructure and managed services from a cloud provider. These services may include migrating all existing applications from the enterprise's own data centers to cloud and managing the infrastructure and applications for them [1]. The first step in an enterprise cloud migration project is understanding and specifying the client requirements for compute and infrastructure, network, storage, middleware, databases, software, and application stack. There could

H. R. Motahari Nezhad—The work was done while Hamid was with the IBM Almaden Research Center.

also be non-functional requirements such as service level, disaster recovery, resiliency, backup, installation, monitoring, and management of the entire stack. The next step is considering different cloud offerings with various capabilities and price points to come up with the design of a so-called cloud solution. A cloud solution consists of the architecture of the proposed solution, and a bill of material of all elements and services. Cloud solution design is a very challenging problem due to the complexity and inter-dependency of the requirements, and the sheer number and complexity of options.

In this paper, we investigate the problem of automatic cloud solution design for enterprise from the point of view of a cloud service provider. The goal is to assist cloud sales architects in producing technical solution proposals that fit the client's requirements, and at the same time comply with the design specification supported by the service provider's offerings. We make the following novel contributions:

- We present an end-to-end model-driven cloud solution design framework, from client requirements to deployable cloud solutions.
- The model-driven method takes advantage of a formal modeling backend and a constraint satisfaction engine, which drive various functionalities.
- We have implemented the proposed methods in a tool called ClOud sOlution design tooL (a.k.a COOL), which showcases the viability of the methods, and it is in use by hundreds of technical cloud solution designers in a large cloud service provider.

An early version of COOL was demonstrated in [2]. Since then we have extended the framework with new methods for model-driven requirements management, solution checking, automated UI generation, and guided solutioning. This paper is the first that describes the techniques behind the demo presented in [2].

The paper is structured as follows: Sect. 2 reviews related work, Sect. 3 gives an overview of the model-driven framework, Sect. 4 describes models and model management, Sect. 5 describes the framework in detail, and Sect. 6 concludes.

2 Related Work

We review related work in two related categories: cloud solution modeling, and software engineering methods in support of cloud applications.

There exist various languages and format for specifying the architecture of cloud applications and their deployment: TOSCA [3], CloudML [4], OCCI [5], MAP [6]. We, on the other hand, address the complementary open space of cloud solution design.

Software engineering methods for cloud application development, management and deployment have been studied [7]. Also, practitioners have investigated patterns and conceptual models for provisioning and scaling of cloud applications [8]. There has also been research discussing strategies for migrating legacy applications to the cloud [9, 10]. Most of these methods either offer abstractions for an architect to manually create the migration plan, or offer methods for migrating existing applications to cloud environments. COOL offers a comprehensive and automated model-driven method for transformation and verification checking of AS-IS and To-BE cloud applications.

2.1 Model-Driven Design Tooling

COOL's process is driven by a set of models for cloud offerings. The model is consulted to determine the correctness of a solution relative to the offering constraints. In [11], a CSP (constraint satisfaction) model is used for systems engineering design, and a CSP solver [12] to verify correctness. COOL's model-driven approach applies the same methods to a different domain. COOL also enhances the model technology with the ability to analyze conflicts, functions for locating offending values in the solution, and the ability to use the model for non-verification tasks (e.g. solution generation).

3 COOL Solution Design Framework: An Overview

COOL's solution design framework is created for cloud solution architects designing business-level solutions from resources of cloud service providers. What makes COOL unique is that it is a model-driven tool, i.e. it relies on models to drive the solutioning process. COOL uses two types of models. The *offering model* is a static model that defines what components are available and what is a legal configuration. A *Configuration* (or solution) is a dynamic model of a specific cloud design. Thus, the offering model defines the space of legal configurations (which are instances of the model).

Figure 1 shows COOL's work flow. We briefly describe the flow here (Sect. 5 elaborates on the major steps).

First, *Requirements* are gathered through the business requirements form and IT requirements spreadsheet, which is a structured spreadsheet designed for that offering. This collects existing infrastructure and workloads to be migrated to the cloud, or requirements for a new cloud project. Alternatively, the requirements may be captured directly in the tool's UI. The *Requirement Analysis* function identifies requests that cannot be met by the cloud provider and makes suggestions for alternatives. Next, *Solution Generation* automatically creates a draft solution. It creates a solution object from the requirements object using the offering model for correctness, and incorporating domain knowledge regarding best-practices and performance considerations. Next, the user moves to the solution view, where the configuration may be fine-tuned. The solution object is continuously *validated* against the entire set of offering constraints, after every change performed by the user. Any violation is flagged to the user in two ways: a descriptive message is displayed, and the solution properties responsible for the violation are flagged in red. In addition, the tool implements what we call *Guided Solutioning*. For each unassigned field the UI offers a drop-down box of values to choose from, containing only values that are consistent with the choices made so far. For example, if the user chooses a value in the 'OS' field then the 'Storage Type' is instantly updated with only those storage types that are supported with the chosen OS. The offering model includes both error and warning constraints, where a violation of an error constraint renders the solution invalid, whereas a violation of a warning constraint is displayed but does not prohibit conclusion of the solutioning process. *Output Generation* is enabled only when all error constraints are satisfied. COOL produces outputs for both pricing and deployment, in formats consumable by down-stream tools.

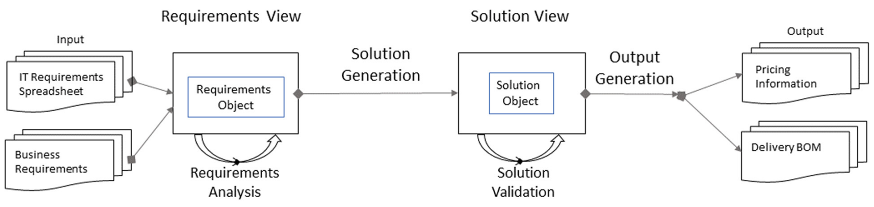


Fig. 1. Cloud solution design flow

4 Cloud Solutions: A Model-Centric Design

In this section, we give a detailed explanation of how a cloud offering is modeled, and how offering models are managed in a model-driven design tool.

4.1 Cloud Offering Models

COOL keeps a model for each offering, capturing all aspects of a cloud configuration using a data model and a set of constraints. The *Data Model* captures the cloud components that are being offered and the information to be gathered for each component. This includes all aspects of the offering, from infrastructure, platform and software components, to services and more. Each component, is modeled as a type (or class) that contains properties. These properties can be of various atomic types, customized enumerated types, or of a user-defined type resulting in a hierarchical design. For example, a VM type may contain a CPU property of type CPU. Properties are annotated with information controlling how they display in the UI (e.g. whether it can be edited).

The *Constraints* capture relationships between data items, limiting values to the sub-space supported by the cloud vendor. For example, a constraint may describe what level of availability can be guaranteed on each supported OS. There are two types of constraints: expressions and tables. Expressions constrain data model properties using a simple language for Boolean Predicates. Table constraints describe Boolean predicates over properties, with the columns corresponding to predicate parameters and the rows corresponding to legal combinations of values, using sets and wild cards for generalization. A legal configuration must match exactly one row in the table.

An offering model contains two aspects: Requirements and Solution, each described by a data model and a set of constraints. The Requirements Model is designed to capture what clients may ask for and is therefore a rather loose model, allowing anything that can occur in the real world (whether or not supported by the offering). The Solution Model represents a cloud configuration that is to be solutioned, and therefore is much more detailed and restricts the configuration to what is supported by the offering.

4.2 Model Management

Figure 2 depicts COOL's framework for defining and managing the (static) cloud offering model. The offering model is kept in a human-readable form and is created and maintained by expert cloud offering staff. The model automation component creates the internal model structures that are used by COOL at runtime.

The machine-readable offering model includes two semantically equivalent representations. The back-end and UI use the JSON format, whereas the CSP solver [12] uses a CSP format to do validation. The model automation ensures that all aspects of the offering model are defined in one place, eliminating the need to maintain multiple copies, which is both labor intensive as well as error-prone. Also, translating from a single source into multiple formats guarantees that all parts of the tool are using semantically equivalent models. Finally, it allows us to manage the model in a human-friendly format while enabling the code to use an efficient internal representation.

5 Model-Driven Framework: From Requirements to Solution

5.1 Casting Solution Design as a Constraint Satisfaction Problem

COOL casts the problem of verifying a cloud solution as a CSP formula, and utilizes a specialized CSP solver [13]. We briefly explain how a CSP solver works and how a cloud offering model is formalized as a CSP formula. We also describe how CSP is exploited by COOL for validation and auto-population of cloud solutions.

A CSP solver receives a formula over a set of variables and determines whether there exists a satisfying assignment for this formula. The solver performs *Arc Consistency* [12], which is an algorithm that prunes variable domains by propagating them through constraints, until it has a minimal consistent set. That is, the reduced domain of each variable is such that if we assign any value from that domain there exists an extension of that partial assignment to a satisfying assignment. This reduction algorithm is employed by the solver before starting the search for a satisfying assignment. It does not produce a satisfying assignment, but it can identify when the formula is unsatisfiable[1]. In this case, we can ask the solver to produce an unsat-core, which is a subset of constraints that are still unsatisfiable, while removing any constraint will result in a satisfiable instance. CSP solvers accept formulas in a very low-level language. We use an IBM tool called PRB [13], which wraps the underlying solver with a rich language for CSP. A PRB model consists of hierarchical type definitions, variable definitions, and constraints. Constraints can be either hard or soft. A hard constraint must be satisfied for the problem to be satisfiable. A soft constraint will be used to influence variable values as long as it is satisfied and will be ignored if it contradicts with hard constraints. COOL invokes PRB in arc-consistency-only mode, so that given a model instance, and if it is satisfiable, we get back a reduction of the variable domains that satisfies all hard constraints and as many soft constraints as possible. If the model

[1] In theory, it is possible for arc-consistency not to identify unsatisfiability [12]. However, this happens in a very particular situation, which cannot occur in COOL's models.

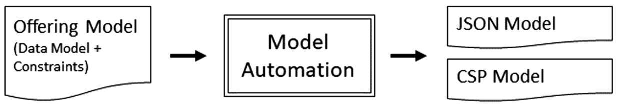


Fig. 2. COOL's framework for model automation

instance is unsatisfiable PRB will generate the unsat core. Hard constraints are used to model "error constraints", and soft constraints are used to model "warning constraints" as well as to assign default values. The offering model is static, as is the CSP version of it. At runtime, COOL uses PRB in order to verify that the configuration that the user created is legal. Upon each invocation of validation, a PRB file describing the particular configuration is created and concatenate to the static offering model, and PRB is invoked.

5.2 Model-Driven UI

COOL is implemented as a web application. The UI renders the JSON object of a requirement or solution based on the data model. The solution editor recognizes the basic property types and enforces what values can be entered. Furthermore, as properties are populated, the validator determines fewer valid choices and the UI updates accordingly. All UI behaviors are completely driven by the model, so that when the model definition is updated, the UI automatically reflects the changes with no code changes required.

5.3 Requirements Analysis

Sometimes, there is a gap between user requirements and the offering. For example, the user may request an OS that the offering does not support, or the user may ask for an invalid configuration of a VM. The requirements model does not include constraints on what the offering supports, because its purpose is to document what the client asked for. When Requirement Analysis is invoked it highlights requests that cannot be met and provides suitable replacements if necessary.

Requirements analysis is composed of a collection of scripts, each of which implements a particular analysis and defines how the check should be conducted as well as the actions that should be taken when a violation is found. The analysis scripts consult with the offering model to discern valid configurations supported by the offering, which means that when the offering changes the analysis scripts are automatically updated and require no modification.

Each analysis script consists of a series of "rules". Rules are independent of each other and are responsible for a subset of the analysis. Each rule defines an analysis location and an analysis action. The analysis first navigates to the specified location in the requirements object where the rule should be applied and performs a check based on the offering model constraints. If the check fails, the analysis applies the specified actions. For instance, one rule is to check the combination of OS and DB types. This analysis will navigate to each required VM and find the installed DB servers, to obtain

OS and DB type values. The rule verifies these against the supported combinations of OS and DB, as specified in the offering model. If an invalid combination is found, the rule's actions are invoked – in this case to highlight the violation and provide a list of legal OS/DB pairs for users to migrate to.

5.4 Solution Generation Through Model Transformation

Given a valid requirements object, Solution Generation generates a draft of the solution. This functionality driven by instructions for best practices, as defined by offering experts. Offering model constraints are consulted so that the generated solution is a valid and efficient configuration. The process involves scanning the JSON object of the requirements and applying transformation rules to each element, which in turn generate a JSON object for the corresponding solution object.

5.5 Solution Analysis and Validation

The validator runs PRB on the combined offering and configuration models and asks it to perform arc consistency. If the instance is satisfiable then the result specifies a legal sub-domain for each variable. These domains are written back and rendered in the UI. If the domain of a particular variable contains multiple legal values then these values will populate the drop-down menu in the UI for that property. If there is only a single legal value the tool will set that value as a system-defined value, and display it.

If the instance is found to be unsatisfiable the solver will produce a set of constraints that are violated. The offering model associates an explanation string with each constraint, so that when a constraint is violated this explanation is displayed for the user in the UI. In addition, the solver identifies the set of variables involved in the conflict, and these are marked in the solution view.

5.6 Post Processing and Output Generation

There are several operations that are invoked when the solution is complete, and the user wishes to generate the full pricing document and the deployment document. The most interesting example is the adaptation of the solution for High Availability (HA). The manner in which HA is achieved is defined by the offering team and depends on specific implementation details of the cloud stack. To avoid users having to understand the intricacies of building HA solutions, COOL asks the users to specify the level of availability, and the tool adds the necessary infrastructure and services automatically.

COOL supports outputs for pricing and deployment. The pricing output lists all elements of the solution that incur a fee, in a format compatible with a downstream pricing tool. The deployment output is a BOM, detailing everything that is to be deployed.

6 Conclusion and Future Work

We have introduced an end-to-end model-driven framework for design of verifiable cloud, implemented in the COOL tool. COOL has been successfully deployed in a large cloud service provider for designing large cloud solutions for enterprise clients. It has shown to significantly reduce the time to produce the solution, enable standardization of the solutioning practice by different cloud architects, and reduce errors and omissions in the cloud solutions designed. As for future work, we plan to construct an eco-system around the tool via publishing APIs for both upstream and downstream applications.

Acknowledgement. We would like to acknowledge and thank all colleagues that have been part of the project at various times, including Scott Trent, Takayuki Kushida, and Uma Subramanian.

References

1. Linthicum, D.: The case for managed service providers in your cloud strategy. http://www.infoworld.com/article/2923441/cloud-computing/the-case-for-managed-service-providers-in-your-cloud-strategy.html. Accessed January 2018
2. Motahari Nezhad, H.R., et al.: COOL: a model-driven and automated system for guided and verifiable cloud solution design. In: Drira, K., et al. (eds.) ICSOC 2016. LNCS, vol. 10380, pp. 194–198. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68136-8_23
3. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) Advanced Web Services. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7535-4_22
4. Ferry, N., Rossini, A., Chauvel, F., Morin, B., Solberg, A.: Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In: IEEE CLOUD 2013, pp. 887–894. IEEE Press (2013)
5. Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., Tata, S.: A precise metamodel for open cloud computing interface. In: IEEE CLOUD 2015, pp. 852–859. IEEE Press (2015)
6. Microsoft Assessment and Planning (MAP) Toolkit for Windows Azure Platform. https://technet.microsoft.com/en-us/solutionaccelerators/gg581074.aspx
7. Grundy, J., Kaefer, G., Keong, J., Liu, A.: Software engineering for the cloud. IEEE Softw. **29**(2), 26–29 (2012)
8. Duan, Q.: Cloud service performance evaluation: status, challenges, and opportunities – a survey from the system modeling perspective. Digit. Commun. Netw. **3**(2), 101–111 (2017)
9. Zhao, J.-F., Zhou, J.-T.: Strategies and methods for cloud migration. Int. J. Autom. Comput. **11**(2), 143–152 (2014)
10. Jamshidi, P., Ahmad, A., Pahl, C.: Cloud migration research: a systematic review. IEEE Trans. Cloud Comput. **1**(2), 1 (2013)
11. Boni, O., et al.: Applying constraint programming to incorporate engineering methodologies into the design process of complex systems. In: 24th Conference on Innovative Applications of Artificial Intelligence, Toronto, Canada (2014)
12. Dechter, R.: Constraint Processing, Morgan Kaufmann Series in Artificial Intelligence. Elsevier, Amsterdam (2003)
13. Bilgory, E., Bin, E., Ziv, A.: Solving constraint satisfaction problems containing vectors of unknown size. In: Beck, J.C. (ed.) CP 2017. LNCS, vol. 10416, pp. 55–70. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66158-2_4

Service Applications

Healthcare Application Migration in Compliant Hybrid Clouds

Anca Sailer[1], Bo Yang[2](✉), Siddharth Jain[1], Angel E. Tomala-Reyes[1], Manu Singh[3], and Anirudh Ramnath[4]

[1] IBM Watson Health, IBM, New York, USA
{ancas,aetomala}@us.ibm.com, Sid.Jain@ibm.com
[2] IBM Research – China, IBM, Beijing, China
yangbbo@cn.ibm.com
[3] Amazon Video, Amazon, Seattle, USA
mmnsin@amazon.com
[4] Uber Freight, Uber Technologies, San Francisco, USA
ani@uber.com

Abstract. Key challenges in managing healthcare applications lie in the area of compliance of the deployment environments and the usage of hybrid clouds. Our approach, as reported in this paper, utilizes two innovative concepts: compliance conformance validation and environment reconstruction supported by a Platform as a Service (PaaS) environment performing healthcare application automated migrations in hybrid clouds. We show how the migration process is conducted with dynamic reconstruction of the application dependencies on the PaaS services. For system administrators, this approach can lead to significant time savings on migrations to compliant environments. Implementation details and experimental results are presented to validate our methodology.

Keywords: Healthcare · Platform as a Service · Migration · Hybrid cloud

1 Introduction

Modern healthcare application developers make effective use of Platform as a Service (PaaS) development techniques which provide services that speed-up the development, building and deployment of online applications. However, native PaaS environments may be insufficient for applications with special requirements for deployment on a third party environment. For example, native PaaS environments may not support compliant handling of end-user sensitive data which requires special security policies as prescribed by the Health Insurance Portability and Accountability Act (HIPAA) [1]. Another example is the case of applications that bind sensitive data to local (or in country) data support. As a further example, the native environment of PaaS may not support the approval processes associated with healthcare deployment environments upgrades.

To address this issue and to allow developers to continue taking advantage of the PaaS development convenience while running applications in a compliant environment, we have developed a migration technology which includes a compliance conformance validator and a compliance control point. The compliance conformance validator analyzes an application's profile in the source environment while the compliance control points analyze an application's profile in the target environment. The migration system also includes a reconstructor that recreates the application's dependencies in the target environment based on the analysis of the application's profile. We show how the migration process is conducted with dynamic reconstruction of the application dependencies on the PaaS services. For system administrators, this approach can lead to significant savings in time spent on migration to a compliant environment. Implementation details and experimental results are presented to validate our methodology.

The remainder of the paper is organized as follows. In Sect. 2 we provide a brief overview of the HIPAA compliance context and its Security Rule. Section 3 presents the existing work related to compliant clouds in the industry and various aspects of the migration technology. In Sect. 4 we introduce our migration solution architecture and in Sect. 5 we detail the implementation and experimental results. Section 6 presents the conclusions of our findings and future work.

2 Background on HIPAA and HL7

The Health Insurance Portability and Accountability Act (HIPAA) [1] has two major goals: (1) to enable the transfer and storage of electronic Protected Health Information (e-PHI, called here after simply PHI as we only refer to electronical PHI in this paper) and (2) to enable the insurance portability from one insurer to another. Thus, HIPAA fosters the replacement of medical paperwork with standardized electronic medical records (EMRs) enabling healthcare and insurance providers to reduce cost and to be more efficient.

2.1 Transfer and Storage of PHI

Storing and transferring PHI electronically raises the question of data loss and exposure due to theft or systems failure. To address these concerns, IBM Cloud for Health in particular, as recipients of PHI from our customers, comply with HIPAA by implementing specific controls as defined and published in the HIPAA Security Rule [1].

The HIPAA Security Rule specifies a series of administrative, technical, and physical security procedures to assure the confidentiality, integrity, and availability of PHI - healthcare providers use these procedures. In general, the administrative controls refer to policies and procedures for protection of PHI, the technical controls refer to technological tools that protect the integrity and availability of resources, while the physical controls refer to the protections of the environment where the computing devices are deployed. In this paper we address the technical controls only, as these controls raised specific challenges for our data services use cases and technology, while the administrative and physical controls have been inherited from the Health Cloud HIPAA readiness.

The following HIPAA technical controls are specified by HIPAA Security Rule:

1. Access Control to implement policies and procedures that allow only authorized persons to access PHI.
2. Person or entity authentication to implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed.
3. Audit Controls to implement hardware, software, and/or procedural mechanisms to record and examine access and other activity in information systems that contain or use PHI.
4. Integrity Controls to implement policies and procedures to protect PHI from improper alteration or destruction.
5. Transmission Control to implement technical security measures that guard against unauthorized access to PHI that is being transmitted over an electronic network.

The healthcare provider is responsible to adopt the Security Rule and maintain written records of the required activities for six years.

2.2 Standardization and Portability

Healthcare standards materialize by bills approved by Congress. HIPAA standards cover both the storage and transmission of PHI (e.g., electronical medical records) as well as the billing and administrative processes (e.g., claims). The HIPAA administrative processes, including Eligibility Inquiry and Response, Claims Status Inquiry and Response, Authorizations and Referrals, Claims, Remittance Advice follow the ASC X12 [2] as the primary standard for administrative transactions.

The standard for clinical data, including Discharge Summary, Imaging Report, Admission & Physical, Pathology Report, follows the Health Level 7's [3] HL7 Clinical Document Architecture (CDA) and Messaging, which together with the Continuity of Care Document (CCD) and SNOMED CT, LOINC, and RxNorm code sets are used to standardize vocabularies.

HL7 offers a framework and standards for the distribution and retrieval of EMRs. These standards describe how PHI is packaged and transferred to support the clinical practice and the management, delivery and assessment of health services. The latest HL7 data format standard is the Fast Healthcare Interoperability Resources (FHIR) [4]. FHIR builds on the best practices of HL7's v2.x, HL7 v3 and CDA standards while leveraging the latest technology such as RESTful protocol, HTML and Cascading Style Sheets for user interface integration, a choice of JSON or XML for data representation, OAuth for authorization and Atom for results. Today, 95% of US healthcare organizations use HL7 V2.x [5].

3 Related Work

Cloud providers, such as Amazon [6], Google [7] and Microsoft [8] have been upgrading their services to comply with health regulations. Similarly, to IBM, a customer organization must first sign a written agreement called the Business Associate Agreement (BAA) before using IBM's regulated environments and services. However, in the case of Amazon, Google and Microsoft, each customer is responsible for their own compliance mechanisms, policies, and procedures to follow the HIPAA technical controls while using their Clouds' HIPAA services. Thus, the customer is expected to encrypt or deidentify the data and separately store the identifying/encryption key, monitor all access to the databases to prevent unauthorized and detect security breaches, security incidents or disclosure of PHI, and ensure fault tolerance and backup of their data. On the other hand, our Health Compliant Cloud services are managed according to HIPAA and implement the HIPAA Security Rule control points mentioned above allowing the customer to focus on their health business. Section 4 will detail the design of our Health Complaint Cloud services to natively support the HIPAA regulations.

There is a rich body of literature that addresses the migration of legacy or enterprise applications to cloud [9–15] and its security and cloud hybrid aspects [12, 16–20]. Because migration to cloud is a major change for a service provider, carefully thought-out decisions factoring in technical, economical and compliance related aspects [16] need to be made and executed. Decision models may consider factors affecting migration in a holistic way and quantify them to provide a formalized decision support like the system called "InCLOUDer" [21]. In the case where the full migration of applications is not possible, due to compliance or complex interdependencies, the decision framework can facilitate only partial migration [16]. A survey on cloud migration decision making methodologies has been conducted in [13].

Migration to hybrid or federated clouds is another aspect addressed by current research. This type of migration requires partitioning of the application. The authors in [17] have proposed a static analysis based on automation guided by the developer's annotations in code. From the security and compliance perspective, ideally the application or system migrated to a target cloud environment is desired to be at least as secure and compliant as it was originally in the source environment. An analysis of cloud migration methodologies has been conducted [19], and from the security perspective the authors conclude that there is little research on the migration of the security and compliance aspects.

Moving to cloud remains a complex endeavor which requires planning and execution of multiple steps and various vendors such as AWS [22] and Cisco [18] have published guides for the service providers illustrating the required processes on their respective platforms.

Our intent in this paper is to contribute to the migration aspects in the context of HIPAA regulated hybrid cloud with a focus on providing or maintaining compliance. On an abstract level, some of the concepts used for this migration are similar to the existing migration technologies. However, our focus on compliance and on the migration implementation within this context differentiates our approach from the above solutions. For example, the extraction of the source architecture, generation and adaptation presented in the CloudMIG approach [12] are comparable to the steps in our implementation. However, the dependency analysis conducted in our methodology to recreate the target environment in a health compliant fashion is distinct.

4 System Architecture

4.1 Overall System Architecture

In this section, we introduce the overall architecture of our migration system. Figure 1 shows the constituent components: (i) conformance validator, (ii) imitator, (iii) deployer and (iv) reconstructor.

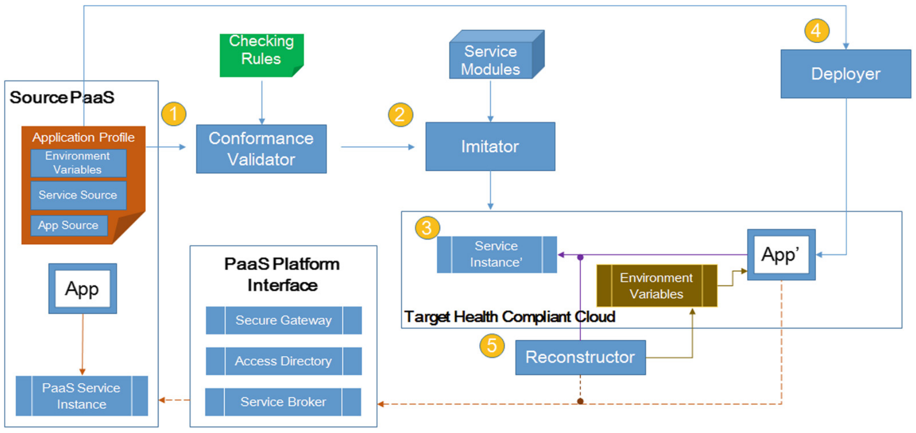


Fig. 1. Overall architecture of migration system with compliance validation

The **conformance validator** (see #1 in Fig. 1) analyzes an application profile, explained hereafter, in the source environment based on validation policies. Our migration system includes an **imitator** (see #2 in Fig. 1) to mirror the configuration in the target environment and to stand up compliant **PaaS service instances** (see #3 in Fig. 1) in the target environment when the dependent original PaaS service instance is non-compliant. This migration system also includes a **deployer** (see #4 in Fig. 1) configured to migrate the application with updated dependencies from the source environment to the target environment. After the target environment was recreated by the imitator and the application was migrated by the deployer, the **reconstructor** (see #5 in Fig. 1) configures the application dependencies in the target environment based on the above described analysis of the application's profile in the source environment, producing an updated environment with reconstructed dependencies.

We will describe now these artifacts in detail.

An application profile describes the application software configuration (e.g., the hosting PaaS profile) which is stored in the PaaS platform. This profile is utilized by our system to generate equivalent dependencies for the migrated application in the target environment. The application software configuration is defined by the environment variables, the service source, and the application's source. The environment variables include application runtime dependencies customized by the application developer. The service source includes application runtime dependencies such as the information of the bind to service instances to which the developer subscribed to, e.g. a database service, and used in the application's code. The application source comprises deployment dependencies, examples of which include git repository links, war/zip/tar file urls, etc.

The conformance validator automatically checks an application's source code scan or a dependency validation test against a specification. The conformance validator retrieves the application's profile from the PaaS platform. Then, it uses the rules stored in a rule engine to automatically check conformance requirements against the application's profile (e.g. analyze the PaaS environment source profile for a PaaS application). The conformance requirements of the target environment can be fine grained, like in precise rules for the PaaS development template and deployment guide. Note that rules checking can be included on the conformance validator or can be provided by a third party. Examples of rules checking include: the Heuristic 1, Heuristic 2, and Heuristic 3 provided below. For each item that is found non-compliant, the conformance validator can also provide fix suggestions as illustrated in the examples Heuristic 4 and Heuristic 5.

The conformance validator is again used after the deployment to check the compliance heuristics in the target environment, as illustrated in Heuristic 6, Heuristic 7, and Heuristic 8.

Heuristic 1:

If Application Source is
{app_type: NodeJS, source_type: github, source_url: "git://***"}

Then Check
{source code availability on $ source_url}
{source code structure conformance to NodeJS}
{Package.json file is complete}
{App.js file is complete} | {Server.js file is complete}

Heuristic 2:

If Platform Source is
{platform: Bluemix, Environment variables: Set}

Then Check
{source code is developed with Environment variables reference in VCAP_Services and USER_Defined}

Heuristic 3:

If Service Source is
{DB: cloudantNoSQLDB, Environment variables: Set}

Then Check
{credentials are set in Environment variables}
{access endpoint and credentials are accessible from target environment}
{API version is compliant (manual or programmatic input)}

Heuristic 4:

If Service Source has
{inaccessible access endpoint from target environment}

Then Action
{open change request for firewall/network device configuration}

Heuristic 5:

If Service Source has
{non compliant API version (detected via manual or programmatic input)}

Then Action
{Instantiate compliant offering for service instance}

Heuristic 6:

If Application Target is
{platform: target environment, app_type: NodeJS, source_type: github, source_url: "git://***", HIPAA compliant: True}

Then Check
{SSL/TLS is enabled, reachable via HTTPS ONLY and NOT reachable via HTTP}
{local files are encrypted}

Heuristic 7:

If Platform Target is
{platform: target environment, HIPAA compliant: True}

Then Check
{security agents are deployed and enabled}
{audit agents are deployed and enabled}
{backup agents are deployed and enabled}

Heuristic 8:

If Service Target is
{DB: cloudantNoSQLDB, HIPAA compliant: True}

Then Check
{encryption is enabled (manual or programmatic)}
{API version is compliant (manual or programmatic)}
{access endpoint and credentials are valid for target environment}

The HIPAA technical controls described in Sect. 2 have been mapped to technology requirements as follows:

Access Control and Person or entity authentication are implemented by authentication and authorization procedures. Technology like AppScan can provide insights on security compliance including access being guarded by authentication and authorization.

Audit Controls are implemented by log agents at hardware, software and application level (e.g., Heuristic 5).

Integrity and Transmission Controls are implemented by encryption at rest and in transit and by backup and recovery measures (e.g. Heuristic 4 for encryption in transit, Heuristic 5 for backup agent and Heuristic 6 for encryption at rest).

For the compliance control points (introduced in Sect. 2) that cannot be validated programmatically, a GUI is available for a developer to confirm the implementation of the compliance requirements for each control point.

The imitator's purpose is to recreate/imitate in the target environment the application's profile artifacts as defined by the source environment variables and dependent services. The imitator stands up the mapped runtime environment variables and services as service instances in the target environment based on the analysis of the application profile in the source PaaS environment. Note that the service instances in the target environment, while they are a copy of the service instance in the source environment, they can be different than the instances used by the source PaaS, e.g. in the case of non-complaint PaaS services. The imitator also stands up the middleware stack where the application will be migrated. The image used for the deployment stack is expected to support the compliance control points presented in Sect. 2. They will be checked by the compliance validator as exemplified above.

The deployer transfers the application packages to the target environment and executes its deployment to stand up a migrated application instance.

The reconstructor configures the dependencies of the environment variables and service instances bindings on the target environment based on the analysis of the application's profile in the source PaaS environment. The reconstructor can also be configured to update the environment's variables and the service instances' bindings with auto mapping based on predefined compliant service access endpoints in the target environment. For example, when the application is running in PaaS, it calls the environment variables and the service instances bindings of the PaaS. Once the migrated application is running in the target environment, the environment's variables and service instances' bindings in the target environment need to be updated so that the migrated application can make similar calls. Thus, the new service instances initialized by the imitator must have their configuration (e.g. conformance requirements) setup and updated in the environment variables. These updated environment variables are used by the reconstructor to recreate the dependencies for the migrated application in the target environment. The code of the migrated application does not need to be changed based on the operations of the reconstructor, we assume that the new dependent services provide the same APIs and drivers as in the source environment.

Figure 2 shows the process flow of our system migration from a hosting PaaS environment to a compliant target environment. The process begins with the conformance validator checking the application profile on the source PaaS environment. The validator automatically checks the application environment, its source code scan and available dependency validation tests against specification rules (such as source packages access, proper configuration formatting). The rules are composed of fine grained predicates suitable for automatic checking based on the specification of the PaaS development guidance and the application environment definition template, as illustrated in the heuristics above. For each item that is found in nonconformance, an automatic fix option is provided.

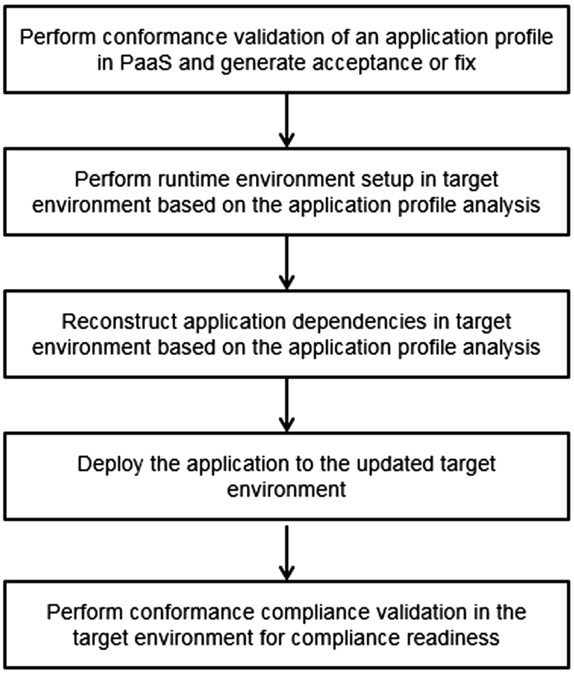

Perform conformance validation of an application profile in PaaS and generate acceptance or fix
Perform runtime environment setup in target environment based on the application profile analysis
Reconstruct application dependencies in target environment based on the application profile analysis
Deploy the application to the updated target environment
Perform conformance compliance validation in the target environment for compliance readiness

Fig. 2. Migration system process flow

Based on the profile analysis, the imitator performs a runtime environment setup to provide the dependent service instances and a stack where the application can be deployed. A reconstructor can now recreate the dependencies configuration in the target environment, including the variables and service instances' bindings. With the dependencies reconstructed, the environment is now updated and ready for the application migration. The deployer of the migration system migrates/deploys the application to the updated target environment.

Finally, the conformance compliance validator checks that all required compliance control points have the implementation tracked and the operational processes approved.

4.2 PaaS Platform Interface Components

A key component to enable the target environment reconstruction is the PaaS platform interface which is leveraged by the imitator and reconstructor to orchestrate the services instances creation, configuration and access binding to the application. The PaaS platform interface includes the components to onboard services in PaaS and make them available to the application developers. At a high level, there are three main workflows between the PaaS, service provider, and application consuming the services as follows.

Service Onboarding: This is the process to make services available in PaaS. This step essentially allows the services to be published in the PaaS catalog where they can be subscribed to. Service onboarding ensures that all metadata related to the service, e.g. price plans, service broker url (explained below), is available to the PaaS in the appropriate format. This step is executed prior to any application creation in PaaS in order to allow for the application development process find the desired services in the PaaS catalog and subscribe to them.

Service Provisioning and Binding: This process is responsible for the service instance creation upon subscription to a services in the PaaS catalog. This process is also responsible for creating API access credentials (e.g. token or password) to control the application access to its service instances at the runtime calls. The API access credentials, which bind an app to its services, are stored in the Access Directory. This step is executed via the services broker both for the application development and during the application migration to the target environment for creating the services instances desired by the original application and expected by the migrated application, respectively. The provisioning of the application dependent services is initiated (a) for the application in PaaS by the developer upon subscription to the service in the PaaS catalog, and (b) for the migrated application by the imitator as part of the migration automation after the application profile analysis determined the dependent services. Once the service instances are created and the application migrated, the reconstructor proceeds with the binding operation by calling the service broker API.

Application Runtime Calls to Services: Applications use their binding credentials every time they call their services instances. These runtime calls are made via a secure gateway which validates the credentials and then routes the call to the appropriate service instance. These calls expect the migration process was completed.

The above workflows come with their corresponding reverse processes workflows for service off-boarding, service instance deprovision and service instance unbinding which we will not detail here.

The PaaS components to support the above workflows are the services broker, the secure gateway and the access directory:

Service Broker: The Service Broker is the entity that manages provisioning and deprovisioning of service instance resources and establishes the binding relationships between services and applications making use of the services. The service broker is an interface to PaaS implemented and operated by the service providers with access to services metadata maintained is in synch with the services onboarded in the PaaS. The imitator and reconstructor call the service broker as part of the migration automation process illustrated in Fig. 2 to perform the runtime environment services instances provisioning and application dependencies bindings respectively.

Secure Gateway: The Secure Gateway is an interface that acts as an entry channel between the PaaS and the services running in the services providers cloud. During runtime, the secure gateway validates against the access directory the credentials presented by the API calls coming from the applications. In our deployment, we used IBM Security Gateway DataPower appliance to perform this functionality.

Access Directory: The Access Directory is responsible to store the API access credentials which are generated during the service instance binding call.

Our migration system has been deployed in our Health Compliant Cloud and in the following section we detail the options and findings of our experiments.

5 Experimental Results

We used IBM PaaS Cloud as our dev/test environment for an expedite application prototyping and verification, and our Health Compliant Cloud as the target environment to migrate our application to for being hosted in a HIPAA health compliant production environment. IBM Cloud provides PaaS DevOps services which speed-up the development, building, and deployment of online applications as well as the updating of their subsequent versions. For deploying hosted applications, we have two options: installation and migration.

The installation of a new application in the target environment encompasses primarily the following steps:

1. Build the installation package (based on a stable source code);
2. Download installation package to target environment;
3. Run Install/Setup program to install the application;
4. Configure the application according to the user's customization requirements.

This option to install an application from scratch presents however a few risks when compared to a migration process, as follows:

- The application installation behavior in the target environment could be different than in PaaS due to the configuration variations of the deployment in a regulated environment.
- The files to be installed are scanned following security hardening procedures, and hence the installation process could potentially introduce new files (e.g., 3rd party library, configurations, scripts). In the same time, certain programs cannot be scanned from the binary installation package.
- The network access to potential 3rd party dependent libraries could be blocked due to security and complaint concerns, resulting in the installation to fail.

In the case of migration, where all deployment artifacts come from a verified application instance in the Dev/Test environment, the files and programs deployed for the application are stable, with no suspicious files to introduce potential risk of exposure, while the configuration is tuned for optimal performance in this new environment.

Table 1 compares the installation vs. migration of five popular (see the popularity rank count) Github applications: SVGO [23], Hackathon-Starter [24], WordPress [25], Smart-crop [26], Hexo [27]. We identified that each installation changed the application file set after deployment, which means that the files introduced during installation might not undergo security scanning as required in a regulated environment.

If we set the firewall to block any unchecked files in the installation procedure, the installation will fail. To deploy a quality tested application in a regulated environment, we need to make sure the application can be deployed successfully and that all the files are compliant with the HIPAA and security requirements.

Based on this evaluation, we choose the migration as the solution to archive healthcare application deployment in regulated environment. This approach was farther beneficial towards meeting the mandatory compliance and security requirements effectively.

Table 1. Experimental results of installation vs. migration of Github applications

Metric	Application				
	SVGO	Hackathon-Starter	WordPress	Smartcrop	Hexo
Rank count (k)	8.7	22.1	9.5	10.1	19.8
Installation					
Files # (release package)	342	243	9,868	192	362
Files # (after install)	1,217	13,945	82,869	14,728	1,988
Files # (after config)	1,217	13,945	87,990	14,728	9,299
Timecost (sec)	4	55	386	34	9
Migration					
Files # (release package)	1,217	13,945	87,990	14,728	9,299
Files # (after migrate)	1,217	13,945	87,990	14,728	9,299
Timecost (sec)	3	10	59	11	5

Figure 3 illustrates the orchestration of our migration processes. We use Jenkins to coordinate the resources migration pipeline and UCD (IBM UrbanCode Deploy [15]) to automate the deployment across the two environments. Our implementation in UCD covers the automation of the main migration steps presented in Sect. 4, i.e., retrieval of the application profile and artifacts, validation of the conformance, setup of the environment, reconstruction of the dependencies, deployment of the application in the regulated environment and validation of the compliance control points.

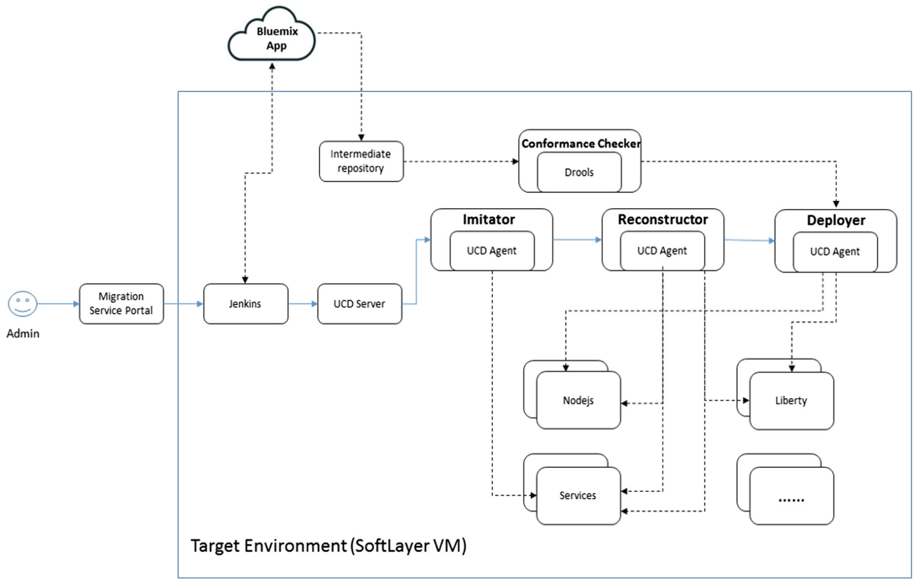


Fig. 3. Migration process implementation

Thus, we have developed Jenkins jobs to access the application profile and artifacts on the IBM Cloud, to perform conformance validation on the profile, and store the analysis result on an intermediate repository in the regulated environment for local access. The Jenkins job is configured to use a UCD component together with its UCD application, UCD application process and UCD environment, to deploy the application targeted for migration. After the profile analysis is uploaded, the Jenkins job will trigger UCD component to start setup the environment accordingly in view of moving the application artifacts to the target regulated environment, as illustrated in Fig. 4.

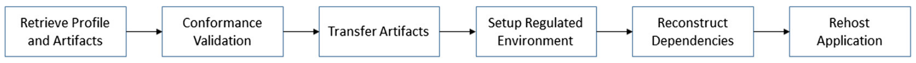


Fig. 4. Migration process build and deployment pipeline

UCD implements the end to end deployment automation. The UCD component process executes the following steps:

1. Initialization of the VMs. To boost the migration performance, the VMs can be selected from a preprovisioned pool.
2. Imitator service sets up the service instances the application depends on.
3. Downloading and unpackaging of the application artifacts on the VM.
4. Reconstruction of the relationships between the migrated application and imitated services instances (e.g., update of the API access endpoint and binding credential based on the profile, update of the environment variables based on target environment setting, update of the routing rules in regulated environment for application's access control.).
5. Validation of the HIPAA control points.

The migrated application with the dependencies configured and validated in the compliant target environment is ready for runtime operation. UCD can be used for deploying a test client in the target environment for executing prebuilt integration validation tests. After the tests pass, the test component is deprovisioned and the migrated application is ready for public access.

6 Conclusion

Modern healthcare applications present particular challenges in cloud and hybrid cloud environments. Keeping these applications up-to-date in live cloud environments can be costly and time consuming. HIPAA compliance, in particular, introduces technical and security challenges that are an overload to the developers and operators of cloud native solutions. Our experimentation has shown that through the use of open source technologies, best of breed automation tools and the PaaS Platform interface we are able to implement a DevOps methodology that addresses and meets the HIPAA requirements. Our migration methodology introduces simple yet powerful concepts aiming to efficiently migrate applications or solutions developed in PaaS to fully compliant environments. In the follow-up to this work, we will focus on the GxP compliance which brings an additional regulation overhead to the developers of clinical studies and medical devices applications.

References

1. The security rule: U.S. Department of Health and Human Services. http://www.hhs.gov/hipaa/for-professionals/security/index.html
2. X12: American National Standards Institute. www.x12.org
3. Health level seven international: Health Level Seven International. http://www.hl7.org/
4. Welcome to fhir: Health Level Seven International. http://www.hl7.org/fhir/
5. Hl7 version 2 product suite: Health Level Seven International. https://www.hl7.org/implement/standards/productbrief.cfm?productid=185
6. Hipaa in the cloud. https://aws.amazon.com/health/providers-and-insurers/hipaa/
7. Hipaa compliance and data protection with Google apps. https://static.googleusercontent.com/media/apps.google.com.br/pt/BR/files/hipaa-implementation-guide.pdf
8. Microsoft Azure trust center. https://azure.microsoft.com/en-us/support/trust-center
9. Wan, Z., Duan, L., Wang, P.: Cloud migration: layer partition and integration. In: 2017 IEEE International Conference on Edge Computing, EDGE, Honolulu, HI, pp. 150–157 (2017)
10. Kaviani, N., Wohlstadter, E., Lea, R.: J. Internet Serv. Appl. **5**, 14 (2014)
11. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving applications to the cloud: an approach based on application model enrichment. Int. J. Coop. Inf. Syst. **20**(03), 307–356 (2011)
12. Frey, S., Hasselbring, W.: Model-based migration of legacy software systems into the cloud: the CloudMIG approach. In: Proceedings of 12th Workshop on Software-Reengineering of the GI-SRE, Bad Honnef, Germany, 3–5 May 2010 (2010)
13. Wan, Z., Wang, P.: A survey and taxonomy of cloud migration. In: 2014 International Conference on Service Sciences, Wuxi, pp. 175–180 (2014)
14. VMware: Migration Strategies for Hybrid Cloud, Version 2.9, January 2018. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-migration-strategies-hybrid-cloud.pdf
15. IBM UrbanCode Deploy. https://www.ibm.com/us-en/marketplace/application-release-automation, https://developer.ibm.com/urbancode/products/urbancode-deploy/
16. Juan-Verdejo, A., Baars, H.: Decision support for partially moving applications to the cloud: the example of business intelligence. In: Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services, HotTopiCS 2013, pp. 35–42. ACM, New York (2013)
17. Smit, M., Shtern, M., Simmons, B., Litoiu, M.: Partitioning applications for hybrid and federated clouds. In: Jacobsen, H.-A., (Jenny) Zou, Y., Chen, J. (eds.) Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research, CASCON 2012, pp. 27–41.IBM Corp., Riverton (2012)
18. Cisco: Planning the migration of Enterprise Applications to the Cloud, Cisco White paper (2010)
19. Rosado, D.G., Gómez, R., Mellado, D., Fernandez-Medina, E.: Security analysis in the migration to cloud environments. Future Internet **4**(2), 469–487 (2012)
20. Jamshidi, P., Ahmad, A., Pahl, C.: Cloud migration research: a systematic review. IEEE Trans. Cloud Comput. **1**(2), 142–157 (2013)

21. Juan-Verdejo, A., Zschaler, S., Surajbali, B., Baars, H., Kemper, H.G.: InCLOUDer: a formalised decision support modelling approach to migrate applications to cloud environments. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, pp. 467–474 (2014)
22. Varia, J.: Migrating your Existing Applications to the AWS Cloud (2010). http://media.amazonwebservices.com/CloudMigration-main.pdf
23. Project SVGO. https://github.com/svg/svgo.git
24. Project Hackathon-Starter. https://github.com/sahat/hackathon-starter
25. Project WordPress. https://github.com/Automattic/wp-calypso.git
26. Project Smartcrop. https://github.com/jwagner/smartcrop.js.git
27. Project Hexo. https://github.com/hexojs/hexo.git

DAliM: Machine Learning Based Intelligent Lucky Money Determination for Large-Scale E-Commerce Businesses

Min Fu[1,2], Chi Man Wong[3], Hai Zhu[1], Yanjun Huang[1], Yuanping Li[1], Xi Zheng[2](✉), Jia Wu[2], Jian Yang[2], and Chi Man Vong[3]

[1] Alibaba Group, Hangzhou, China
{hanhao.fm, marvin.zh, jenny.hyj, yuanping.lyp}@alibaba-inc.com
[2] Department of Computing, Macquarie University, Sydney, Australia
{james.zheng, jia.wu, jian.yang}@mq.edu.au
[3] University of Macau, Macau, China
{mb55501, cmvong}@umac.mo

Abstract. E-commerce businesses compete in the market by conducting marketing strategies consisting of four aspects: customers, products, marketplaces and intermediaries. One of the widely-used marketing strategies, called Lucky Money, is capable of encouraging customers to buy products from marketplaces. However, the amount of luck money for each customer is usually randomly determined or even manually determined and cannot fully achieve the business objectives. This paper proposes a machine-learning based lucky money determination approach, called DAliM, for e-commerce businesses to achieve their desired goals. We implement DAliM for the "Double 11 Global Shopping Festival 2017" initiated by Alibaba Group and evaluate it using a few hundred million real customers from all over the world. The experimental results demonstrate that our method manages to decrease the lucky money spent by 41.71% and increase the final purchase rate by 24.94% compared to the state-of-the-art baseline.

Keywords: Machine learning · Lucky money · E-commerce · Data mining
Price prediction · Price optimization

1 Introduction

Large-scale worldwide e-commerce businesses, such as Amazon, e-Bay and Alibaba Group, are always competing for marketing shares, and the number of target customers attracted by the business to a large extent determines its profits [1, 2]. Hence, the e-commerce businesses and companies have launched several marketing strategies which are related to four key aspects: customers, products, marketplaces and intermediaries [1]. Throughout the four aspects, intermediaries are the most special one because they are able to build a link between the providers and the customers who buy products at certain marketplaces [1, 2]. As such, one of the widely-used intermediaries for

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 740–755, 2018.
https://doi.org/10.1007/978-3-030-03596-9_53

marketing plans is known as “Lucky Money”, which is a certain amount of e-cash bonus given to the buyers and can be used to offset the original price to be paid by a customer [3]. In other words, with luck money, customers can buy products at a lower price. Hence, luck money is becoming increasingly popular among e-commerce companies and even other companies [4, 5].

The ways of determining the amount of lucky money for each customer used by e-commerce businesses are usually by assigning a random amount of money to the customer or by issuing a manually-determined amount of money to the customer [6–9]. However, there are several drawbacks with these existing methods: (1) the random amount may not be satisfactory to the customer; (2) the issued random amount for each customer does not fully satisfy the business objectives (e.g. cost optimization and targeted purchase rate) set by stakeholders; (3) it provides little reason to explain why the customer is assigned with the money [10]. The use of these methods is because it is not straightforward to determine the accurate amount of lucky money for each customer, in terms of three challenges: (1) the money should be issued on an ongoing basis; (2) the characteristics of each customer are difficult to be obtained; (3) the budget for lucky money is quite limited.

In this paper, we propose a novel and intelligent approach, called DAliM, for better determining lucky money for customers. DAliM is based on historical data analysis and machine learning techniques. We first analyze hundreds of million real historical data related to customer, transaction and lucky money usage, and then we rely on machine learning techniques to train the historical data to obtain a purchase rate prediction model, which is used for calculating the predicted purchase rate of a particular customer with a certain amount of lucky money. Next, we apply a smoothing regression algorithm to figure out each target customer’s assigned lucky money that can drive the customer to buy products at the targeted purchase rate set by the stakeholders. According to our knowledge, it is one of the first kind to apply machine learning for lucky money determination.

We evaluate the feasibility and validity of DAliM in the “Double 11 Global Shopping Festival” initiated by Alibaba Group in 2017. In the experiment, we use an AB-Test strategy by simultaneously launching two sets of lucky money determination approaches: (1) Alibaba Group’s existing random lucky money based approach; (2) our newly proposed approach: DAliM. These two approaches are applied on a few hundred million real target customers of the “2017 Double 11 Global Shopping Festival”. We make a comparison between these two approaches, and the experimental results show that DAliM is able to decrease the lucky money spent by around 41% and increase the final purchase rate by 25% compared to the existing method, while satisfying the business objectives set by stakeholders.

The research contributions of this paper are: (1) we propose a novel and intelligent machine learning based approach for determining appropriate lucky money within e-commerce businesses; (2) we formulate the business objectives and requirements to be set by e-commerce business stakeholders; (3) we propose a methodology to evaluate the difference between two lucky money determination approaches, and demonstrate the applicability of the methodology.

The remainder of this paper is organized as follows: Sect. 2 introduces the background; Sect. 3 discusses the related work; Sect. 4 illustrates our proposed method of determining lucky money; Sect. 5 is the experimental evaluation; Sect. 6 provides the conclusions and our future work.

2 Background

We introduce the fundamentals of lucky money in e-commerce businesses, and discuss business requirements and objectives for lucky money.

2.1 Lucky Money in E-Commerce Businesses

E-commerce businesses, especially the large-scale ones such as Amazon and Alibaba Group, choose to use lucky money as one of their preferred marketing strategies, in order to compete for their marketing share and attract more customers [1, 2]. Lucky money is a type of e-cash bonus given to the customers by e-commerce corporations or product providers [3]. With the lucky money, customers can offset the original price they need to pay when they buy a particular product, which gives them a sense that they buy the products at a discounted price [3]. Normally, based on various business purposes, there are four types of lucky money: (1) visit-number oriented lucky money whose purpose is to increase the visiting number of users; (2) advertising oriented lucky money whose purpose is to publish and advertise a promotion activity; (3) purchase-rate oriented lucky money which aims at increasing the purchase rate of customers; (4) new-customer oriented lucky money which targets to attract more new customers who have never bought products from the e-commerce corporations. For instance, every year Alibaba Group launches a lucky money strategy called “Torch Red Packet” during big promotion activities, and launches the “Happy City Red Packet” lucky money strategy during the “Double 11 Global Shopping Festival” [3]. As for the “Double 11 Global Shopping Festival”, it achieved a turnover of 18 billion dollars during 11 November 2016, and 25.5 billion dollars during 11 November 2017, so a reasonable lucky money strategy plays and will continue to play a significant role in e-commerce businesses’ promotion activities like “Double 11 Global Shopping Festival”.

2.2 Business Requirements and Objectives for Lucky Money

The above-mentioned four types of lucky money reflect the business requirements and objectives which are to increase visits, advertise a promotion activity, increase purchase potential, and attract new customers [11–14]. All of the four objectives are highly related to increasing the overall profits of e-commerce businesses either by increasing the number of customers or by increasing the possibility for each customer to buy products. Moreover, these requirements also aim to give those e-commerce businesses a larger exposure to the market. In this paper, we are mainly focused on the objective of increasing the purchase rate.

3 Related Work

Machine learning techniques are widely employed to solve many problems in financial and economic areas, and we discuss a few of them in this section. The lucky money determination problem addressed in this paper highly resembles several economic decision problems such as price determination [15–17].

3.1 Share Market Price Prediction Using ANN

The researchers from SUST proposed a price prediction mechanism for share market based on Artificial Neural Networks (ANN) techniques in 2011 [15]. This proposed method addressed the challenge with the insignificant relationship between the variables of the share market chaos system and the share price [15]. The traditional multiple layer perceptron (MLP) model was applied to the neural network and the back propagation (BP) mechanism was used to calculate and update the weights of the intermediate inputs and outputs [15]. There are several drawbacks with this mechanism: first, only two hidden layers were constructed in the neural network, and there is no explanation for it; moreover, there is no comparison between the predicted effect of using two hidden layers and that of using more hidden layers; second, the variables of share market used in the neural network only contain the information about the company's financial situation and do not consider other useful features such as the company's customer size or launch time [15]. In addition, the proposed method was evaluated using only one company coupled with two sets of input data, which decreases the validity of the method [15].

3.2 Financial Time Series Forecasting

The researchers from Bond University, Australia found that Artificial Neural Networks (ANNs) were identified to be the dominant machine learning technique in the area of predicting financial time series in the stock market [16]. While the other methodology used for such a prediction was based on evolutionary and optimization based techniques, there is a clear trend to use and enhance established ANN models with new training algorithms or combine ANNs with emerging technologies into hybrid systems [16]. However, how the real-world constraints can impact the accuracy of financial time series forecasting and stock index prediction remains a question, and whether the investors' risk-return tradeoff can be improved or not should also be studied [16].

3.3 Crude Oil Price Prediction with ANN-Q

The researchers from the University of Manchester proposed an Artificial Neural Network-Quantitative (ANN-Q) model based approach for predicting the price of crude oil in 2010 [17]. This research addressed a big problem with crude oil price prediction: the price of crude oil is related to a complex set of factors and any change of the factors may have an exclusive impact on the price, because crude oil is one of the world' major commodities with high volatility level [17]. An ANN-Q model was developed based on a total of 22 key factors that have an impact on the price of crude oil. In order to develop a better model, the data related to these key factors were further divided into three categories: large impact class, medium impact class and small impact class [17]. BPNN was used for training the input variables [17]. The simulation results showed acceptable accuracy of the proposed method since it was still in progress, and the improved accuracy could be achieved by better tuning the parameters or applying other machine learning models.

4 Our Proposed Method

Large-scale e-commerce businesses like Alibaba Group have their own business goals and objectives to fulfil when it comes to determining the lucky money for their customers. For example, with a fixed amount of lucky money budget, the corporation may demand that the max number of customers should receive at least a minimum amount of lucky money and the final purchase rate of those customers should be optimized or meet the purchase rate goal set by the stakeholder. Our proposed method is designed to be able to satisfy these business objectives. However, there are some challenges: (1) whether a customer will buy products with the given lucky money or not depends on a complex set of factors, not only on the amount of lucky money; (2) the characteristics of every customer are quite different from each other, and it is difficult to use a universal way to model different customers; (3) some customers are not sensitive to lucky money at all, and they would not buy any products no matter how much lucky money is given to them. As such, we define three research questions: (1) apart from lucky money amount, what are the other factors and features that impact the overall purchase rate of customers? (2) how to model the set of factors and features to represent the data of every customer? (3) how to develop the model that is able to correctly filter the customers sensitive to lucky money and assign the appropriate amount of lucky money to these selected customers?

Our proposed method, DAliM, answers all of the three research questions defined. When applied within Alibaba Group, DAliM is able to fulfil all of the business objectives set by the stakeholders of Alibaba Group. As far as we know, it is the first time that such a method is ever proposed. The problem can be defined in the following way:

Suppose the lucky money budget is C, and the number of customers who receive lucky money is D, each customer's lucky money is Q_i ($1 \leq i \leq D$), and each customer's purchase rate is P_i, then we need to determine Q_i based on the following formula:

$$\boldsymbol{Q} = \underset{Q_i}{\operatorname{argmax}} \sum_{i \in D} P_i \tag{1}$$

Subject to:

$$\sum_{i \in D} Q_i P_i \leq C \tag{2}$$

Where:

$$P_i = g(\boldsymbol{X_i}), \mathrm{i} \in D \tag{3}$$

Where g(.) is the function that represents the machine learning model used for predicting the purchase rate and Xi denotes the set of customer features, including the feature of lucky money amount.

4.1 Overview of DAliM

The overview of DAliM is shown in Fig. 1.

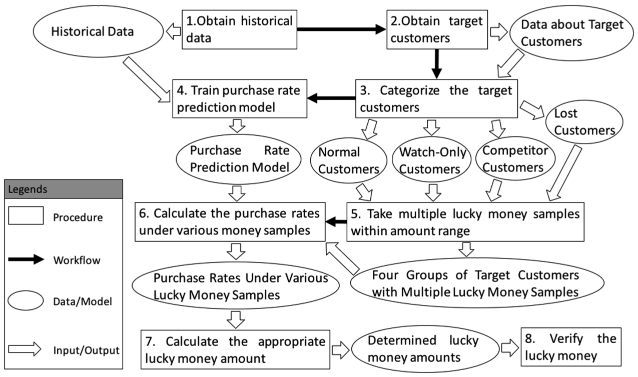

Historical Data
1.Obtain historical data
2.Obtain target customers
Data about Target Customers
4. Train purchase rate prediction model
3. Categorize the target customers
Lost Customers
Purchase Rate Prediction Model
Normal Customers
Watch-Only Customers
Competitor Customers
Legends
Procedure
Workflow
Data/Model
Input/Output
6. Calculate the purchase rates under various money samples
5. Take multiple lucky money samples within amount range
Purchase Rates Under Various Lucky Money Samples
Four Groups of Target Customers with Multiple Lucky Money Samples
7. Calculate the appropriate lucky money amount
Determined lucky money amounts
8. Verify the lucky money

Fig. 1. Overview of DAliM.

The method consists of eight procedures: (1) we obtain the historical data about the past customers and their usage of the lucky money given to them (i.e. whether they have used the lucky money to buy products or not); (2) we obtain the target customers who will attend the promotion activity (i.e. “Double 11 Global Shopping Festival 2017”); (3) we then make categorization for those target customers based on the extent to which they will participate in the promotion activity, and they are categorized into four groups: normal customers, watch-only customers, competitor customers and lost customers; (4) next we use the historical data about customers and lucky money usage to train the machine learning model that is used for predicting the purchase rate (probability) of a particular customer; (5) we take samples of the lucky money by selecting several amounts of money that are within the lucky money range specified by the stakeholder, and for each customer group we attach each selected sample amount with all the customers inside that group, to form up a series of customer sets with different lucky money samples, yet all of the sets share the same set of customers. In total, there are four customer groups and each group is attached with several lucky money samples, and the money sample works as one of the data features. (6) With the obtained purchase rate prediction model and a series of target customer sets with multiple lucky money samples, we calculate the purchase rate for each customer in each group with attached sampled lucky money; (7) then relying on the information about each customer's different purchase rates under different lucky money samples, we calculate how much lucky money should be given to each customer so that the customer will buy products at the stakeholder-specified purchase rate, and the calculated money is the result assigned to that customer. We do this for each of the four groups. (8) Finally, we verify the predicted lucky money for all the target customers.

4.2 Training Data and Prediction Data

The historical data about the characteristics, natures and lucky money usage of all the past customers of the promotion activities is used for training the machine learning model. The data about the characteristics and natures of all the target customers of the current promotion activity ("Double 11 Global Shopping Festival 2017") is used as prediction data.

We obtain the training data from the data generated in last year's promotion activity, which is "Double 11 Global Shopping Festival 2016". There were hundreds of million items of data generated in 2016's "Double 11 Global Shopping Festival", and each data item included 275 features and 1 label that specified whether the customer made any purchase or not. These 275 features are all related to the label or the purchase status. These features could be classified into the following 8 groups: (1) purchase information; (2) membership information; (3) personal information; (4) shopping cart actions (e.g., adding items to the cart); (5) products collection action e.g., adding products into the favorite store); (6) products visiting action; (7) customer coupon sensitivity; (8) customer purchase capacity. These 8 groups are independent of each other.

For the labelling of the historical training data, we use the label 0 to represent a customer made no purchase and we use label 1 to represent a customer made any purchase. We obtain the prediction data by analyzing who will be the target customers of this year's "Double 11 Global Shopping Festival". The target customers of the promotion activity are determined by summarizing those who have made at least one purchase action or visiting action in the past 3 months prior to this year's "Double 11 Global Shopping Festival", because the expert knowledge has indicated that the customers being proactive in the past 3 months prior to an upcoming promotion activity will also be proactive in the upcoming promotion activity, and those who have accessed the e-commerce platform will also be valuable for the upcoming promotion activity. The number of such customers is a few hundred million. The lucky money that should be given to a customer is highly affected by the customer's proactivity in the promotion activity [3]. As such, in order to provide a better lucky money determination mechanism, we differentiate these customers by their proactivity in the promotion activity and categorize them into the following 4 groups: (1) Normal Customers, (2) Watch-Only Customers, (3) Competitor Customers and (4) Lost Customers. The definitions of these 4 groups of prediction data and the percentages of customers in each group are illustrated in Table 1.

We divide the big prediction dataset into 4 prediction sub-datasets. All the prediction sub-datasets share the same features as the big training dataset, and the percentage data inside each prediction sub-dataset will be passed to the trained machine learning model as its inputs for predicting the purchase probability of each target customer. We have also checked for the noise inside the prediction dataset and found

Table 1. Four groups of prediction data.

Group	Definition	Percentage
Normal customers	The customers who have made at least 1 purchase during the past 3 months	90.5%
Watch-only customers	The customers who only accessed the e-commerce platform and merely browsed products on the platform without making any purchase during the past 3 months	4.5%
Competitor customers	The customers who have bought products only from the competitors of Alibaba Group during the past 3 months	3.8%
Lost customers	The customers who haven't made any purchase or products browsing during the past 3 months	1.2%
Overall customers:		100%

no noise in it, since the noise was filtered out by the big data storing and processing engine provided by Alibaba Group.

4.3 Feature Engineering

If we use all of the 275 features in the training dataset for training the machine learning model, it may take relatively much time. Although all the features are related to the labelling, not all of them are of the same importance and only a subset of them have heavier weights and contribute more to the labelling. The remaining features are with light weights and some can even be negligible. Hence, in order to reduce the model training time while maintaining the validity of the model, we need to filter the heavily-weighted features, and thus we apply feature engineering. While feature engineering also includes feature generation and feature transformation, we are only concerned with feature filtering, because we argue that the features in use are already complete enough.

Feature filtering can be performed in several ways, such as by expertise, random sampling or relying on external feature engineering tools, and we determine to perform the feature engineering by using a combination of three ways: (1) relying on the feature importance determination library provided by popular SDKs (e.g. Python machine learning libraries); (2) calculating and ordering the entropy values and Gini indexes of all the features; (3) relying on the gradient boosting decision tree machine learning library. A combination of these three steps is more suitable than any single method for determining feature importance and thus results in a better outcome. For a specific feature, suppose its importance determined by the first step is denoted as W_1, and its importance determined by the second step is denoted as W_2, and its importance determined by the third step is denoted as W_3; we assume these three importance values are of the same weight since there is no apparent clue to indicate which one is more accurate than another one; then the final importance of this feature is calculated as $(W_1 + W_2 + W_3)/3$. We calculate this importance value for each feature and extract the top 50 important features. This answers our research question 1. Among the top 50 important features, their weights range from 0.005 (occupation) to 0.35 (lucky money amount), and they come from all the eight feature groups: 28 of them belong to the group of purchase information related features, 7 of them belong to the group of membership information related features, 5 of them belong to the group of personal information related features, 2 of them belong to the group of shopping cart actions related features, 4 of them belong to the group of products collection actions related features, 2 of them belong to the group of products visiting actions related features, 1 of them belongs to the group of customer coupon sensitivity, and 1 of them belongs to the group of customer purchase capacity related features.

4.4 Training Model Selection and Parameter Tuning

We determine to investigate four training model candidates and choose the best one among them, and they are: Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF) and Gradient Boosting Decision Tree (GBDT). In our pioneer study, we noticed that GBDT can provide better accuracy, precision and recall than the other three, and hence we choose GBDT as the training model. In order to confirm the validity of the selected GBDT training model and its parameters, we further investigate the GBDT model by using the first half of the historical data as the training dataset and the second half of the historical data as the prediction dataset, and the results show that the accuracy is 83.6%, the precision is 0.82 and the recall is 0.77. Although the accuracy is not very high, considering the huge amount of base data and the limited lucky money budget, we assume that the data with correctly predicted results is large enough for assigning lucky money properly. The machine learning model, denoted as G(F), can be represented as below:

$$G(\boldsymbol{F}) = g(x_1, x_2, \ldots, x_n), \mathrm{n} = 50(\textit{features count}) \tag{4}$$

Where g(.) is the function that represents the GBDT model used for predicting the purchase status and x_i denotes each feature of any customer inside both the training dataset and the prediction dataset. This answers the research question 2.

4.5 Data Prediction

We use the result returned by the GBDT model to calculate the purchase rate of each customer in the prediction dataset. The features of each customer in the prediction dataset are denoted as $\mathbf{X}_i$, and the GBDT machine learning model can be denoted as g ($\mathbf{X}_i$), so the purchase rate of the customer, denoted as P_i, is calculated as:

$$P_i = g(\boldsymbol{X_i}), \mathrm{i} \in T \tag{5}$$

Where T is the count of the target customers inside the prediction dataset.

When we make the prediction, the feature of lucky money amount does not use a fixed value. Instead, we use a set of sample lucky money amounts for this feature. The minimum lucky money amount must be in accordance with the requirements of the stakeholders (e.g. 1 RMB); the maximum amount should also satisfy the business requirements (e.g. 25 RMB); the average lucky money amount can also be determined according to the overall lucky money budget and the number of lucky money target customers. Suppose the minimum lucky money amount is 1, the maximum lucky money amount is 25 and the average lucky money amount is 8, then the lucky money samples used in DAliM could be 1, 5, 8, 10, 15, 20, 25. The purchase rate values returned when using all of these lucky money samples can be denoted as P(1), P(5), P (8), P(10), P(15), P(20) and P(25), respectively.

4.6 Lucky Money Determination

After we obtain each customer's purchase rate for each sample lucky money amount, we can determine how much lucky money can hit the intended purchase rate specified by the stakeholders. We define the stakeholder-specified purchase rate as R and for each customer we need to find the target lucky money amount, denoted as M, to make sure R and M can satisfy the following formula:

$$R \leq g\left(X_i', M\right), \mathrm{i} \in T \tag{6}$$

Where X_i' denotes the customer's features excluding the feature of the lucky money amount and g(.) is the function that represents the GBDT model used for predicting the purchase rate. Considering the rule that larger amount of lucky money results in a higher purchase rate, we argue that the final lucky money can be determined by making and analyzing the regression over the sample lucky money amounts and the corresponding purchase rates. Suppose the lucky money samples are denoted as set **S**, and the predicted purchase rates for all the lucky money samples are denoted as set **P**, we apply a smoothing regression algorithm to determine each customer's target lucky money amount, as shown in the below algorithm. The inputs are the lucky money samples set **S**, the purchase rates set **P** in response to **S** and the stakeholder-specified target purchase rate R. This algorithm is able to assign the appropriate money to the customers sensitive to lucky money, which answers our research question 3.

Algorithm 1: Lucky Money Amount Determination Algorithm

Inputs: the lucky money samples set ***S***, the purchase rates set ***P***, the stakeholder-specified purchase rate *R*

Output: the final lucky money amount

```
function DetermineLuckyMoney ($S$, $P$, $R$) {
  for ($i \leftarrow 1$; $i < |S|$; $i$++) {//this is for smoothing the rates
    if ($P[i] < P[i-1]$) {
      $P[i] \leftarrow P[i-1]$;}}
  for ($j \leftarrow 0$; $j < |P|$; $j$++) {//this is for regression of the rates
    if ($P[j] >= R$) {
      if ($j == 0$) {return $S[j]$; }
      else if ($P[j] == R$) {return $S[j]$; }
      else if ($j > 0$) {
        $slope \leftarrow (P[j] - P[j-1]) / (S[j] - S[j-1])$;
        $amount\_to\_deduct \leftarrow (P[j] - R) / slope$;
        $lucky\_money \leftarrow S[j] - amount\_to\_deduct$;
        return $lucky\_money$; }}}
  return 0; //if all the rates are smaller than $R$, return 0
}
```

5 Experimental Evaluation

We have implemented DAliM for the promotion activity called "Double 11 Global Shopping Festival 2017" initiated by Alibaba Group, as such, our experimental evaluation is based on a real-world large scale scenario. In the following subsections, we describe the experimental environment, illustrate the experimental procedure, analyze the experimental results and provide some discussion for the experimental results.

5.1 Experimental Environment

The experimental environment is shown in Fig. 2. The historical training data are stored in Alibaba Group's big data processing platform (ODPS), which provides distributed data storage services and big data processing services. This is where we warehouse all the training data and prediction data. Then, the training data and the prediction data are passed to the distributed machine learning platform.

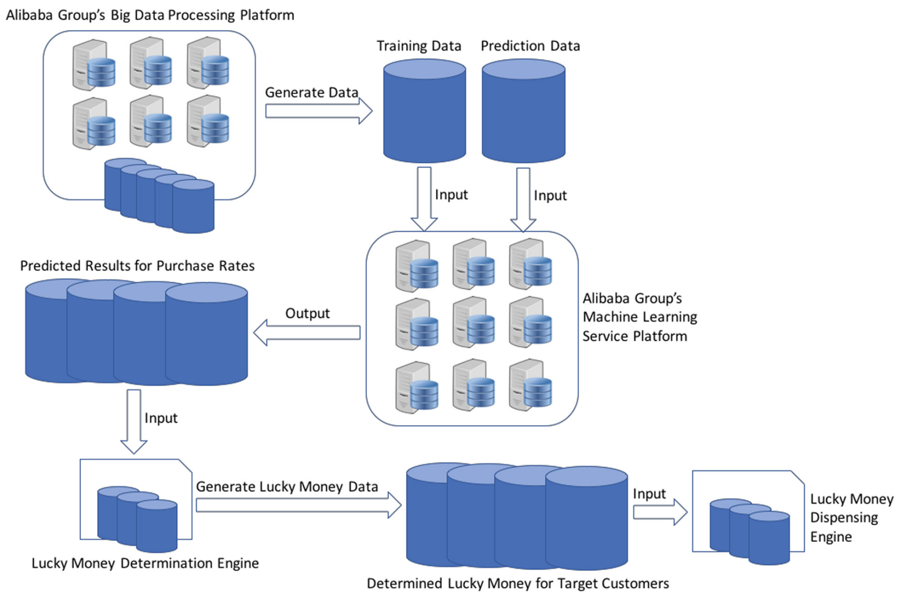

Alibaba Group's Big Data Processing Platform
Generate Data
Training Data
Prediction Data
Input
Input
Predicted Results for Purchase Rates
Output
Alibaba Group's Machine Learning Service Platform
Input
Generate Lucky Money Data
Input
Lucky Money Determination Engine
Determined Lucky Money for Target Customers
Lucky Money Dispensing Engine

Fig. 2. Experimental environment

The machine learning model is derived based on the training data, and 4 groups of prediction data will be inputs of the machine model for purchase rates prediction. Next, we employ our lucky money determination engine to figure out the appropriate lucky money that can satisfy the stakeholder-specified requirements for each target customer. Finally, the lucky money will be dispensed to these customers through our lucky money dispensing engine. Alibaba Group is using a web service-oriented architecture for all of the services involved in the experiment, and we make use of these web services via front-end browsers.

5.2 Experimental Procedure

We run two sets of lucky money dispensing mechanisms simultaneously: the first one uses our proposed method and the second one uses the random lucky money based method. By using the random lucky money base method, the customers who should receive lucky money are randomly selected and the lucky money amounts given to them are also randomly determined, and the random amounts are within the threshold set by the stakeholders. The second method meets Alibaba Group's business standards and requirements, and hence it is used for comparison with our proposed method. For each prediction sub-dataset, we divide it into two sets, where the first set uses the old method and the second set uses our method. As such, these two methods are applied on two sets of customers for each of the 4 prediction sub-datasets, and these two customer datasets have the same statistical distribution because we control both of their volumes to be statistically large enough, though they are different in size. In this way, essentially we apply the AB testing. The customers whose lucky money is determined by our method are called the B-group, and the customers whose lucky money is determined by the existing old method are called the A-group. The experimental procedure is strictly following the lucky money dispensing and spending workflow executed in the promotion activity of "Double 11 Global Shopping Festival 2017". Hence, the experimental procedure is as below:

Step 1: determination of the A-group and the B-group of target customers. For each of the 4 prediction data groups, we randomly select 2% of the customers in the group as the A-group and select the remaining customers in the group as the B-group. By doing so, we can make sure that the distribution of the A-group is the same as the distribution of the B-group, and we can also to a large extent avoid the negative effects that are brought by the existing old method.

Step 2: making all the target customers receive the pre-arranged lucky money. The promotion activity takes place on 11 Nov 2017; between 1 Nov 2017 and 7 Nov 2017, it is the time for dispensing lucky money to customers. During these 7 days, every customer will receive the lucky money after he/she logins into the e-commerce web page and clicks on the promotion activity button. The customers in the A-group will receive the money determined by the existing method while the customers in the B-group will receive the money determined by our proposed method. However, the received lucky money is not allowed to be used until 11 Nov 2017 (Double 11).

Step 3: customers purchasing products with the given lucky money. On the day of 11 Nov 2017, all the target customers who have received lucky money will make a purchase with the given lucky money, and we record the details of all of these transactions, including whether each customer has made any purchase.

Step 4: obtaining purchase status data of the A-group and the B-group and making a comparison between them. For each of the 4 groups of prediction data, we compare the purchase rate of its A-group and that of its B-group.

5.3 Experimental Results

We obtain the experimental results for each of the 4 groups of target customers: normal customers, watch-only customers, competitor customers and lost customers. Each group consists of the A-group and the B-group, and for both the A-group and the B-group, we calculate the actual purchase (denoted as Ra) based on the formula below:

$$R_a = \frac{N}{M} \times 100\% \tag{7}$$

Where N is the number of those customers who have made any purchase with the given lucky money, and M is the number of all the customers who have been given lucky money.

We first show the results for the overall target customers and walk through the experimental results for the 4 groups of target customers one by one. The experimental results for the overall customers are shown in Table 2.

Table 2. Experimental results for overall customers.

Group	AB-group	Lucky money methods	Decreased average amount (B over A)	Increased purchase rate (B over A)
Overall customers	A-group	Existing old method	41.71%	24.94%
	B-group	DAliM		

We can see that the average lucky money amount for the B-group which uses DAliM has decreased significantly by 41.71% compared to the average lucky money amount for the A-group which uses the existing old method, and the actual purchase rate when using DAliM has increased by 24.94% compared to the actual purchase when using the existing old method. Hence, DAliM improves on the existing old method. On the one hand, DAliM is able to save money; on the other hand, DAliM is able to increase the actual purchase rate.

For the experimental results for the normal customers, we notice that the average lucky money amount for the B-group which uses DAliM has decreased by 43.56% compared to the average lucky money amount for the A-group which uses the existing old method, and the actual purchase rate when using DAliM has increased by 26.4% compared to the actual purchase rate when using the existing old method. The results show that normal customers make frequent online purchases regularly, and there is no need to attract them by giving them a relatively large amount of lucky money.

For the experimental results for the watch-only customers, we notice that the average lucky money amount for the B-group which uses DAliM has increased by 100.73% compared to the average lucky money amount for the A-group which uses the existing old method, and the actual purchase rate when using DAliM has increased by 18.4% compared to the actual purchase rate when using the existing old method. The fact that the average lucky money amount when using DAliM is more than the average amount when using the old method is in accordance with the stakeholder's expectation, because a good possible way to drive customers who show interest in buying products but do not actually buy them to make the purchase is to provide them with more price discount. Our suggested approach is able to find out this rule and take advantage of it. Nevertheless, we notice that both methods do not result in a very high purchase rate, and this is because making watch-only customers buy products is intrinsically a challenge. Anyhow, DAliM improves on the existing old method.

For the experimental results for the competitor customers, we notice that the average lucky money amount for the B-group which uses DAliM has increased by 105.03% compared to the average lucky money amount for the A-group which uses the old method, and the actual purchase rate when using DAliM has increased by 16.0% compared to the actual purchase rate when using the existing old method. The relatively higher expenditure on lucky money is because a good possible way to drive customers who only buy products from an e-commerce corporation's competitor is to provide them with better price offers, and a relatively large amount of lucky money is equivalent to a better price offer. Both methods yield relatively low purchase rates because making the customers who have got accustomed to our e-commerce business's competitors switch the loyalty to our e-commerce business is not that straightforward and requires many other conditions.

For the experimental results for the lost customers, we notice that the average lucky money amount for the B-group which uses DAliM has increased by 103.91% compared to the average lucky money amount for the A-group which uses the old method, and the actual purchase rate when using DAliM has increased by 17.5% compared to the actual purchase rate when using the existing old method. The results show that a potential way to drive customers who no longer buy products from our e-commerce business is to provide them with better price offers and more discounts.

5.4 Discussion

The experimental results are in accordance with the business requirements defined by the business stakeholders. It is requested that the minimum number of customers receiving lucky money should be reasonable, and the expected number of customers who will buy products during the promotion activity should be a relatively large number. So, the overall target purchase rate can be calculated based on this information. With the lucky money budget set by the stakeholders, the average lucky money per customer can be calculated. For the sake of business confidentiality, we do not show these numbers in this paper. As such, we compute the actual average lucky money amount per customer resulting from our method and can find that it is within the stakeholder-defined average amount; the actual purchase rate when using our method is also larger than the stakeholder-specified target purchase rate. Moreover, with DAliM, 28% more customers have received the lucky money compared to last year's promotion activity, which well satisfies the stakeholder-specified requirement about the number of lucky money target customers.

6 Conclusions and Future Work

E-commerce businesses compete for the market by several ways, one of which is called the Lucky Money strategy. Lucky money is a type of intermediary that is capable of encouraging customers to buy products from marketplaces. However, the existing ways of lucky money determination usually cannot fully satisfy the business objectives and goals set by the stakeholders, especially for large-scale e-commerce businesses like Alibaba Group. As such, this paper proposes a lucky money intelligent determination approach, called DAliM, for e-commerce businesses to achieve their predefined business objectives such as purchase rates. We design and implement DAliM based on a set of machine learning based techniques. It is able to issue an appropriate amount of luck money to each customer to make sure the customers can buy products at the stakeholder-defined purchase rate. We evaluate DAliM in a real promotion activity named "Double 11 Global Shopping Festival 2017" initiated by Alibaba Group. A few hundred million real customers' data from all over the world have been used for the evaluation. The experimental results show that our method can decrease the lucky money spent by 41.71% and increase the final purchase rate by 24.94% on average compared to the existing old method.

Our future work includes: (1) try applying DAliM on other promotion activities and research on how to readjust it to cater for these activities; (2) obtain more historical training data, train a more accurate prediction model and propose a better way of selecting the target customers of the promotion activity.

Acknowledgement. This research is initiated by Alibaba Group and supported by Macquarie University and University of Macau. We express our special gratitude to all the organizations.

References

1. Sharma, S.: Internet marketing: the backbone of ecommerce. Int. J. Emerg. Res. Manag. Technol. **4**, 200–202 (2015)
2. Turban, E., King, D., Lee, J.K., Liang, T.P., Turban, D.C.: Electronic Commerce: A Managerial and Social Networks Perspective. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-58715-8
3. Alibaba Group's Lucky Money Packet. https://www.alibaba.com/showroom/lucky-money-packet.html. Accessed 28 Feb 2018
4. Tonnison, W., Tonnison, J.I.: Online E-commerce and networking system with user requested sponsor advertisements, patent, App/Pub Number: US20090292595A1, 26 November 2009
5. Clemons, E.K., et al.: Impacts of E-commerce and enhanced information endowments on financial services: a quantitative analysis of transparency, differential pricing, and disintermediation. J. Financ. Serv. Res. **22**(1–2), 73–90 (2002)
6. Yu, L., et al.: Method and system for communication in instant messaging application, patent, App/Pub Number: US20170178094A1, 22 June 2017
7. Liu, W., He, X., Zhang, P.: Application of red envelopes–new weapon of WeChat payment. In: 5th International Conference on Education, Management, Information and Medicine (EMIM 2015) (2015)

8. Yuan, Y., et al.: Online red packets: a large-scale empirical study of gift giving on WeChat. Cornell University library arXiv:1712.02926, 8 December 2017
9. Xie, C., Putrevu, J.S.H., Linder, C.: Family, friends, and cultural connectedness: a comparison between WeChat and Facebook user motivation, experience and NPS among Chinese people living overseas. In: International Conference on Cross-Cultural Design (CCD 2017), pp. 369–382, 14 May 2017
10. Alibaba Group's Buyer Official Forum. http://buyer.alibaba.com/forum. Accessed 28 Feb 2018
11. Chaffey, D.: E-Business & E-Commerce Management Strategy, Implementation and Practice, 5th edn. Pearson Education Limited, London (2011). ISBN 978-0-273-75201-1
12. Gordijn, J., Akkermans, J.M.: Value-based requirements engineering: exploring innovative e-commerce ideas. J. Requir. Eng. **8**(2), 114–134 (2003)
13. Tsalgatidou, A., Pitoura, E.: Business models and transactions in mobile electronic commerce: requirements and properties. J. Comput. Netw. **37**(2), 221–236 (2001)
14. Huang, Z., Benyoucef, M.: From E-commerce to social commerce: a close look at design features. J. Electron. Commer. Res. Appl. **12**(4), 246–259 (2013)
15. Khan, Z.H., Alin, T.S., Hussain, M.A.: Price prediction of share market using artificial neural network (ANN). Int. J. Comput. Appl. **22**(2), 42–47 (2011)
16. Krollner, B., Vanstone, B., Finnie, G.: Financial time series forecasting with machine learning techniques: a survey. In: 2010 European Symposium on Artificial Neural Networks: Computational and Machine Learning, April 2010
17. Abdullah, S.N., Zeng, X.: Machine learning approach for crude oil price prediction with artificial neural networks-quantitative (ANN-Q) model. In: 2010 International Joint Conference on Neural Networks (IJCNN 2010), October 2010

Service-Oriented Approach for Analytics in Industry 4.0

Philippe Lalanda[1(✉)] and Denis Morand[2(✉)]

[1] Grenoble University (UGA), 38058 Grenoble, France
philippe.lalanda@imag.fr
[2] Schneider Electric, 06510 Carros, France
denis.morand@imag.fr

Abstract. Pervasive computing promotes the integration of smart electronic devices in our living and working spaces in order to provide new, advanced services. Many technologies and architectural patterns are proposed to develop such services. The however number of real world applications is still limited. In this paper, we present a real world deployment related to Industry 4.0. This article details our technical choices and the benefits of the approach, mostly based on service-oriented technology. First and foremost, we also point out the specific requirements and constraints we had to face in this work.

Keywords: Pervasive computing · Service-oriented components · Industry 4.0

1 Introduction

Pervasive computing [1, 2] promotes the integration of smart electronic devices in our living and working spaces in order to provide new, advanced services. Such devices are mobile or static, can take multiple forms, and pick up a wide variety of signals in the environment. They collect contextual information, run local computation, and, in some cases, directly act upon the environment. This allows the implementation of simple, reactive services like opening an emergency door when a triggering condition is detected. Pervasive devices are also enhanced with networking capabilities so that they can communicate with each other or with more powerful computing elements, located in close proximity or in the cloud. This is of course necessary to implement complex services, based on various data sources and requiring significant computing and memory capacities.

The proliferation of such pervasive devices, coupled with the widespread availability of the Internet, makes pervasive computing more concrete every day. This new technology has profound effects on entire industries. This applies in particular to the manufacturing domain where the notion of Industry 4.0 is gaining increasing attention. The purpose of this initiative is to bring together new IT technologies and production processes to enable the emergence of smart, connected manufacturing. The term, coined in 2011 by a government-funded German project, refers to what could be the fourth industrial revolution. Industry 4.0 is based on the use of new production techniques, new materials, and the generalized adoption of digital technologies [3]. In this paper, we focus on the latter point, which constitutes a tremendous challenge.

Industry 4.0 thus promotes seamless integration between field devices controlling operations in the plant floor and supervision systems, usually located in IT facilities [4, 5]. Among many benefits, such integration should allow the systematic oversight and improvement of production activities and resource management. However, a large number of problems have yet to be fully addressed to realize that vision. In particular, new software architectures have to be designed and deployed in order to allow generalized connectivity. These architectures must integrate heterogeneous software components, including middleware, which are kept separate so far and usually built by different expert teams. This is a significant challenge for most industrial companies, which do not always have the right skills to do so.

The problem is made worse by the fact that architectural principles are missing today to successfully and repetitively drive Industry 4.0 developers. Potentially, new services can use a variety of computing resources distributed along a cloud-to-devices continuum [6, 7]. Getting the right architecture for a given service, that is making the right decisions about where to compute and store data is still an open issue. Today, this is essentially done on a case-by-case basis.

The purpose of this paper is to present an architecture and a service-oriented platform that have been designed by Schneider Electric to deploy new services supervising water management stations. Implementing this service required the design of a decentralized architecture where a number of analytics are run close to devices and longer-term analytics are run in a partner cloud, as it is usually the case today. Placing computing, control, and storage functions close to end user devices, that is in the so-called fog layer, turned out to be necessary in order to deal with stringent requirements related to performance, cost, security and agility. It led us, however, to build a fog infrastructure allowing the parallel and effective development of heterogeneous components by different teams.

The paper is structured as it follows. First, our driving service is presented. This use case has been entirely implemented and is now fully operational. In this section, we point out the specific requirements and constraints related to an effective industrial deployment. Section 3 presents and justifies the global architecture that has been developed. This architecture relies on the notion of smart gateway running in the fog layer. We show that the use of such gateway comes with some significant benefits but also brings important constraints. Section 4 focuses on the smart gateway and presents its architecture. Implementation is detailed in Sect. 5. Related work is presented in Sect. 6 and the paper is concluded by Sect. 7.

2 Running Example

This paper focuses on the supervision of Schneider Electric pumping stations, which are facilities for moving fluids from one place to another. Pumping stations are usually part of more global infrastructures managing water in large, populated areas. This management consists of all main activities inherent to water treatment including supply, transfer under gravity, drainage, sewage, etc. We focus in this paper on a network of stations located around Paris (see Fig. 1). The infrastructure includes around 10 pumping stations, many lifting stations and a major water treatment station.

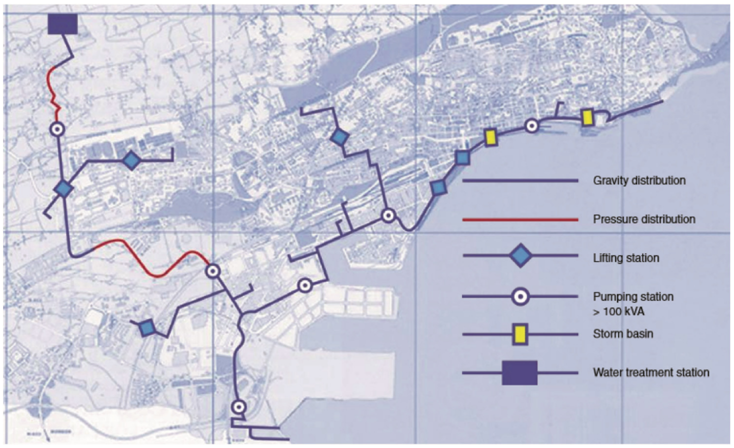


Fig. 1. Water management infrastructure near Paris.

Pumping stations have to maintain consistent water pressure and water flow in order to keep the global infrastructure in good working order. This is a complex process requiring close monitoring and continuous adjustment of a number of configuration parameters (for pumps, drives, PLCs, etc.). In particular, it is essential to avoid cavitation, which is the formation of vapor cavities in the water within the pump. Cavitation can be destructive, entailing significant costs and serious disruptions of operation.

In most installations, control and supervision are achieved by specific drives. Local technicians routinely visit installations to inspect and audit processes through dedicated interfaces. This approach suffers from many limitations. First, technicians are not always on site, which delays maintenance operations. Also, they are not always able to intervene for lack of knowledge about specific devices or lack of information and documentation about the overall installation. Optimization in that context is difficult, if not impossible, to achieve. Advanced analysis is often conducted offline, during post-incident reviews by experts. In our case, experts are located in Boston, USA.

As illustrated by Fig. 2, we connected field devices to the cloud in order to integrate local and remote supervision [5]. The goal was to bring operational data to supervising headquarters and, conversely, to push down information to local technicians in case of a problem. When abnormal situations are detected, alerts and related data are sent to experts on their smartphone. Experts then diagnose the technical problem and send instructions to technicians on site. Instructions come with all the useful information to fix a problem, including data, images or schemas.

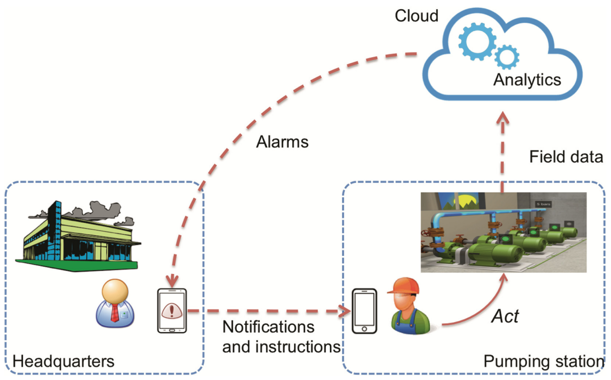


Fig. 2. Use case description.

In this solution, most software complexity is absorbed by the cloud infrastructure. Data collected in the plant floor is moved to the cloud and stored in centralized data centers. Powerful servers, deployed on demand, are then used to run analytics. Gateways in the pumping stations are very simple: their role is to collect data from field devices, possibly do some pruning to remove faulty data, and send up relevant data up to the cloud.

This architectural approach actually makes sense and has been already adopted by Industry 4.0. Analytics are based on large volumes of data that must be stored and accessed rapidly, and require significant computing power. Obviously, cloud infrastructures provide the necessary facilities to do so. They are known to offer great benefits in terms of computing power, elasticity, flexibility, pay-per-use facilities and security. Another reason relates to administration simplicity. Cloud providers are in charge of the management and control of the cloud infrastructure. Manufacturers and service providers can then focus on their own code, generally through virtualized gateways. Most of the time, this is easy and fast. In contrast, managing business code running on field gateways is more complicated and time demanding. A third major reason relates to evolution. It is clearly much easier to implement cloud-to-cloud integration in order to connect new devices instead of gateway-based integration. This is especially true when integrating devices using new field buses or lacking appropriate APIs.

It turned out however that this approach has some limitations [8, 9]. First, some services are time-critical and cannot be executed in the cloud due to unpredictable delays or insufficient bandwidth. Security is also seriously challenging current architectures. First, users are not comfortable with the idea of sensitive data being stored in clouds they don't trust. Also, the way cloud-based services are run raises some issues. Cloud solutions for security rely on perimeter-based protection. If the perimeter is endangered, the common countermeasure is to take the system offline. This causes service disruption in all the facilities managed by the corrupted cloud, which is not acceptable for clients.

Finally, in economical and ecological terms, it does not appear opportune to transport and store big amount of data in clouds without a well thought out plan. Storage and communication can be very expensive. Also, to take a broader vision, clouds are big energy consumers that should not be used to store irrelevant data or data that could be kept near devices.

3 Global Architecture and Requirements

We have designed an architecture where computing and storage functions are distributed more effectively between cloud and fog. Simply put, time-stamped data is collected, stored, and analyzed near the supervised processes, while synthetized, less critical data is sent to the cloud for more advanced and time-consuming analysis. The fog infrastructure is materialized by smart gateways integrated in the water management stations. Connection between cloud and gateways is based on secure MQTT.

This architecture (see Fig. 3) is based on the complementarity of fog and cloud solutions. Clearly, some analytics are more naturally executed in the fog while others in the cloud. Determining the right distribution is actually a key issue that we had to tackle in our project. If correctly designed, such architecture allows overcoming many limitations of the existing solutions relying on computing in the cloud and on end-user devices. Precisely, this architecture is:

- **More efficient.** The gateway proximity to physical processes enhances the supervision service performance. Time-sensitive problems are identified sooner. They are reported to experts and then to technicians with smaller latency. This reduces the risks of the system being taken offline.
- **Cheaper.** Costs are drastically reduced because a significant portion of data is now kept in the plant floor. Only relevant, synthetized data are transmitted to the cloud and billed by the telecom operators. It also reduces the need for Internet bandwidth. Finally, costs related to the use of commercial cloud services, like storage or processing, are lowered due to data reduction.
- **More agile.** Analytics run at the gateway level are fully administered and controlled by industrials (Schneider Electric here). They can be more flexible and responsive to client needs. They can update, remove or add services to meet new demands or changing running conditions. It is more complicated to do so in cloud facilities where vendors cannot easily provide custom-made functions.
- **More secure.** The gateway serves as the aggregation and control points for data. It can be decided to keep sensitive data in the gateway or to perform encryption if they have to be sent to a cloud.

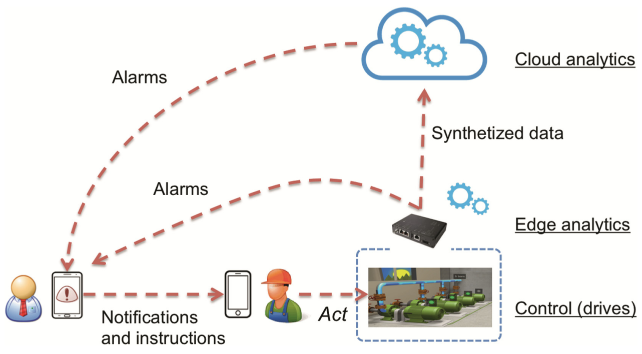


Fig. 3. Edge and cloud analytics.

Smart gateways constitute a critical component of the proposed architecture. However, their implementation turned out to be very challenging for industrials since they have to meet a number of stringent requirements. Let us review hereafter some important ones. First, gateways must provide multiple network connectivity. They must communicate with different devices using heterogeneous field buses, like KNX or Modbus for instance. Communication protocols also vary according to installations. In addition, gateways have to provide a sustainable solution to meet the challenges of future extensions. Control devices are likely to change overtime: pumping stations for instance are expected to run for several decades and are actually in constant evolution. Finally, gateways must integrate standard Internet protocols. This is needed to send data to the clouds over the Internet, but also to be accessible by various web browsers for supervision. The latter point becomes crucial since gateways now contain data and analytics that are not available elsewhere. It is necessary for technicians or experts to get synoptic views of the current situation, both of the supervised processes and the gateways status.

Second, gateways must exhibit a good level of performance. In the current installations, gateways typically have to collect, process and store hundreds items of data per second. Clearly, these figures can only increase given the industrials' growing appetite for added-value device-based services. In addition to that, our gateways must frequently run analytics implementing often-greedy algorithms. For instance, complex analytics may have to be triggered every minute. Finally, gateways must be available to provide real-time information about their status, collected data and analytics results. Implementing all these functions, while meeting the related requirements, is very challenging. For instance, process data cannot be missed because analytics are running. This requires high computation capacities and appropriate software design.

Third, gateways must provide facilities to store data locally (typically, thousands of items may have to be stored). Those facilities must provide fast access to time-sensitive data, preferably based on standard request languages. For instance, analytics require data belonging to a given temporal window or coming from a given set of devices.

Fourth, gateways must be highly available. In our case, pumping stations must be supervised 24 h a day, 7 days a week. Maintenance activities, including deployment, must be carefully planned in order to minimize service interruption. This has a great impact on the gateway internal structure, especially if autonomic solutions have to be used [10].

Fifth, gateways must support the dynamic integration of software components. Components deal with analytics, mediation operations (for alignment for instance), communication, etc. They are usually developed by different teams with different paces regarding development, deployment and maintenance.

Finally, installation cost is also a major requirement. Integrating smart gateways in existing stations should not require new skills. The approach of most industrialists is indeed to acquire and install gradually new devices in order to transform existing processes. Also, by no means, control functions should be impacted by the additional services.

4 Gateway

Platform

Executing services (applications) at the gateway level is complex. This is due to the dynamic, heterogeneous and stochastic nature of the pervasive environments and further complicated by the fact that resources in a gateway are limited and must be managed explicitly. We believe that streamlining the production of fog-level services requires developers and system administrators to be equipped with software engineering tools. A common approach is to introduce an execution platform, or middleware, providing a development model and a set of technical services, also called non-functional services. Making a distinction between the execution platform and the hosted services lowers complexity in terms of code, debug, configuration, administration operations, etc.

We have defined a pervasive platform based on the service paradigm and designed for the fog level [11]. It allows the production of context-aware applications that can be autonomically adapted at runtime. To do so, it provides a service-oriented development model and a set of facilities to communicate with heterogeneous devices, capture and store contextual information, and allow dynamic deployment of applications and components. More precisely, our platform is entirely built on top of the iPOJO service-oriented component model [12], which in its turn is based on the OSGi framework [13], as illustrated by Fig. 4. In contrast to most component models, iPOJO does not define explicit architectures. It specifies components capabilities and needs in terms of services, and leaves bindings resolutions to the execution framework. Bindings are very flexible and can be changed dynamically. Architectures are then built at runtime depending on the components available in a local registry and their properties. Components can be deployed anytime, using Java’s dynamic features (on demand class loading, multiple class loaders, typing verification before loading etc.). Dependencies are resolved by the execution platform when components are instantiated. Bindings are set and constantly re-evaluated by the execution platform. Precisely, every time a service used in a dependency specification changes, the dependency is re-computed.

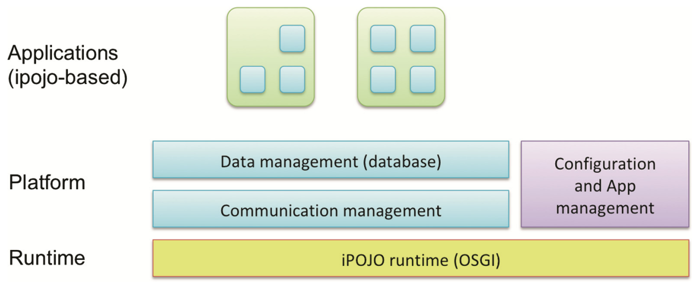

Applications (ipojo-based)
Platform
Runtime
Data management (database)
Communication management
Configuration and App management
iPOJO runtime (OSGI)

Fig. 4. Pervasive platform.

The data management module (Fig. 4) is about context. Contextual information covers many different aspects including physical quantities in the environment, measures about the supervised, available devices, and status of the supporting computing infrastructure itself. Collecting such information and keeping it up to date has proved to be particularly complex. By nature, context is heterogeneous, dynamic and stochastic. In most cases, it is necessary to integrate disparate devices, applications and network protocols in order to get the needed information. In the manufacturing industry, protocols are numerous and depend very much on automation solution providers. Our data management module is made of a dynamic set of iPOJO components and a database. It is dynamic in order to reflect the changing nature of the execution environment but also to deal with applications evolving needs. IPOJO components are used to collect information and to build higher-level information. Important information is kept in the database.

Finally, the communication module gives access to services and data provided by devices in the environment, and to the Cloud. This manager supports an open set of protocols, including Zwave, Zigbee, X10, UPnP, DPWS and Bluetooth, and generates proxy services [14]. Whenever a device disappears or a remote service becomes unreachable, the manager detects the disruption and removes the proxy. The set of managed protocols can be extended at runtime by adding a protocol manager, using generic deployment facilities. The device access manager is also in charge of monitoring concurrent invocations. Most of the protocols do not support concurrent accesses. To enforce integrity, read and write accesses are made sequential.

Application

On top of this platform, we have built our application to deal with the water management use case. The architecture of this application is partly presented in Fig. 5. This view highlights three major modules, each of them made of several iPOJO components. Modules are:

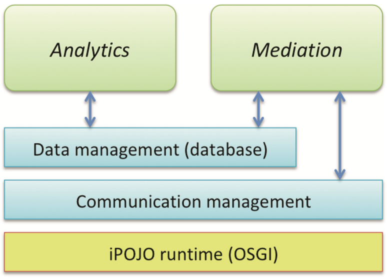


Fig. 5. Gateway software architecture.

- A mediation module [15, 16] that is responsible for collecting, filtering, aggregating, transforming, storing, selecting and sending data up to the cloud.
- A database that is optimized for handling time-sensitive data. It is used to store process data, device data and analytics data, including intermediary results. It is connected to a dashboard delivering synoptic views.
- An analytics module that contains an extensible set of analysis components, all connected to the database. The purpose of analytics is to closely follow the evolution of the supervised processes and related devices. In our case, analytics are used to detect cavitation, bad water flows, excessive pressures, etc.

The nominal behavior is rather straightforward. Field data are collected, processed and stored in the database. Different analytics are triggered on a regular basis to check the proper functioning of the supervised processes and related devices. To do so, analytics use data kept in the database through a variety of queries and store results, intermediary and final, in the database. If an issue is detected, a message is sent to the experts through Google Cloud Messaging. Finally, analytics results and some synthetized information are sent up to the cloud, after more mediation operations.

The mediation part is implemented as a chain of lightweight service-oriented components called mediators. Each mediator implements a simple mediation operation and is structured into three elements dealing with synchronization, processing and routing. Mediators are connected via bindings, describing a connection between an output port and an input port. Specifically, mediators are built in Java on top of Cilia [17], a framework entirely based on iPOJO. Cilia is an autonomic integration framework designed to meet the requirements of cyber physical systems. It offers a set of abstractions to support design, deployment and update of integration solutions. Specific mediators, called adapters, implement communication protocols and handle the dynamicity of resources. The modularity of the mediation solution brings flexibility. It is relatively straightforward to adapt the mediation chain, and the mediators, to specific field configurations. Specific mediators, called adapters, implement communication protocols and handle the dynamicity of resources.

The database is a central element of the gateway architecture. In addition to data collected on the plant floor, it stores all the data generated by the analytics, including intermediary results. In our architecture, the database can be seen as a shared blackboard. Indeed, it represents the only means of communication between the main components of the gateway, that is the analytics, the *in* and *out* mediation parts, and the dashboard. All these components only have indirect interactions through the database and do not know each other. This architectural pattern provides a number of benefits, including the following:

- It allows parallel developments of architecture components since coupling is well defined. This is very important in our projects, which are generally subject to time and budget constraints.
- It allows components to be updated at different paces because, once again, of the weak coupling between components and database.
- It is vary adapted to AI-based systems where control flow of analytics is hard to formalize and, in addition, may vary according to stations.
- It is also robust in the sense that faulty components do not block the gateway. The gateway can be rebooted in extreme cases while important items are saved in the database.

5 Implementation

Platform

The gateway takes the form of an industrial box that is easily pluggable in existing stations, in particular with Modbus. As said previously, safe, easy and cost-effective upgrades are especially important in the manufacturing industry that has to deal with a huge number of plants in operations. Specifically, the gateway is based on Schneider Electric Magelis G5U, a highly configurable and robust hardware using an Intel X86 processor (1.3 GHz) with 16 Gigabytes of data storage (CF Card) and a memory of 2 Gigabytes (RAM). It is equipped with several network interfaces, including Ethernet and the most common fieldbuses. It is named IIotBox (Table 1).

Table 1. Gateway footprint.

Modules	Footprint (kilobytes)
OSGi	483
OSGi bundles (JSON, WS server)	4449
iPOJO	399
Cilia core	165
Cilia runtime	459

The database implementation is based on Influx DB that ranked first in our lab experiments in terms of storage and retrieval times. This database is very efficient to store unstructured IoT-based data, indexed by time. It also provides very fast access through an SQL-like language with built-in time-centric functions for querying data structures composed of measurements, which is exactly what is needed by the analytics. Let us note that the database is also accessed by a dashboard component, which purpose is to provide immediate information to local or remote technicians. This dashboard makes use of web server integrated in the Magelis box.

Application
Figure 6 presents the detailed architecture implementing our use case. It is based on three analytics and a number of mediation components to align, store and transfer data. Precisely, we use OPC/UA and Modbus to collect data from the field. We also use MQTT, a lightweight messaging protocol optimized for IoT applications, for the cloud connection. Precisely, in the use case presented in this paper, we have implemented the following mediators:

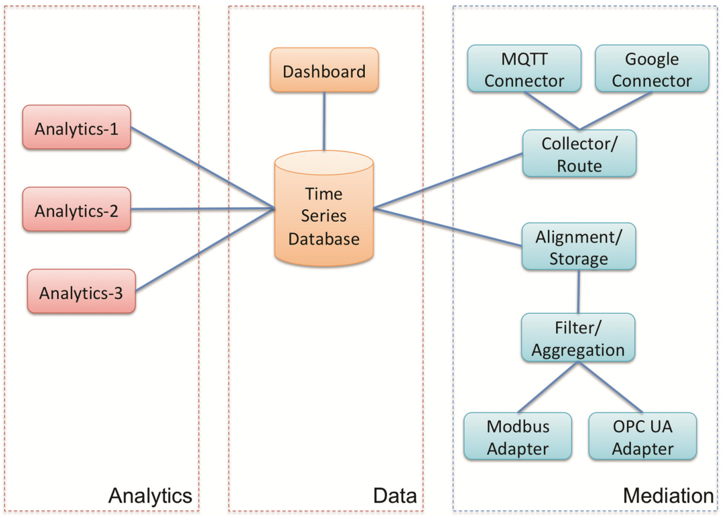


Fig. 6. Software architecture for our use case.

- OPC/UA adapter. This adapter collects data characterizing the supervised processes. Depending on the operational situation, different data sets can be collected, at various periodicities.
- Modbus adapter. This adapter connects to control devices and HMI and gets information about their running status.
- Filtering and aggregation. This mediator filters out incorrect data (out of pre-defined limits) and aggregate related items.
- Alignment and storage. This mediator transforms data into a common format used by the analytics. Then, it writes resulting items in the database. It is based on a node.js open source library.
- Collect and route. This mediator reads data in the time series database. Alarms are routed to the Google adapter while regular data are sent to the MQTT adapter. Frequencies are different.
- MQTT adapter. This adapter connects to a MQTT client in order to establish a secure and reliable connection to the cloud.

Google adapter. This adapter connects to Google Cloud Messaging. This mobile service allows developers to send notification data or information to Android-based smartphones.

Depending on the station characteristics, between 500 and 1000 measures are collected every second. A similar number of item is also written in the time series database every second. This range is conservative: we made sure that the architecture could support ten times more data. The amount of data sent to the cloud is way smaller. Around 50 synthetized data are sent every month to the cloud. This allowed us to meet the goal of significantly reducing communication costs.

In order to meet the performance requirements (see Sect. 2), we had to rewrite the analytics. Indeed, the algorithms initially used in the cloud were too time-consuming due to bad modularity. We had then to design small-grained algorithms in order to meet the time-related requirements of the gateway (that cannot be preempted too long to run "long" analytics). We ended up with three modular analytics characterized by different time frames and communicating through the database, like in pure blackboard-based problem solving. Each one has its own logic and can operate independently. Precisely:

- The first one is triggered every 10 s to check operational conditions and provide synthetized information about the process (stored in the database).
- The second one is run every day. As input, it takes results of the first analytics and also stores its results in the database.
- The third one is activated once a month and uses intermediately results provided by the previous calculations. Its outputs are also kept in the database for future use by the *out* mediation.

If an issue is detected, a message is sent to the experts through Google Cloud Messaging. Finally, analytics results and some synthetized information are sent up to the cloud, after more mediation operations.

6 Discussion and Related Work

As said, Industry 4.0 builds on generalized connectivity. However, its purpose is not to replace existing PLCs (Programmable Logic Controllers) with new Internet-based technology but rather to provide additional added-value services. In the industry, today, these services mainly rely on cloud solutions. Plant floor gateways can be seen as network bridges. Their main purpose is indeed to collect data, perform simple mediation operations and send the great majority of the gathered data to a cloud. In some cases, they run simple scenario to coordinate devices actions. They may also be used to implement actions on devices decided at the cloud level. This is more and more the case, for instance, in the domain of smart homes.

In order to be largely used to implement Industry 4.0 services, gateways need to get smarter. Several such platforms have thus been proposed in the pervasive community to ease application production. Early projects such as Gaia [19] or AURA [19] aim at providing an infrastructure for applications to integrate devices and modeling environmental and user context. Gaia provides a context model based on file-system like resources. However, it lacks tools for helping application development. AURA on the other hand provides an execution model based on user tasks. Application services are executed automatically depending on user's activity.

After this initial work, many platforms have been developed on top of the OSGi framework, which promotes the development of modular and dynamic applications. Indeed, Pervasive platforms are usually based on the principles of service-oriented computing where loosely coupled services enable the creation of flexible dynamic applications. An application is built from several services that can be distributed on different devices. Extensions were needed to simplify the OSGi development model and to deal with pervasive specific features like context modeling. For instance, GatorTech [20] extends the OSGi framework with context representation and heterogeneous device access services. It also provides touch points required to build autonomic behavior. SOCAM [21] is another platform based on OSGi. It uses ontologies to model the context and effectively simplifies the development of context-aware applications. Other platforms introduced autonomic features. For instance, AMUSE [22] and ACCORD [23] propose specific component models to build autonomic applications. MUSIC project [24] provides a service-oriented execution platform and a design studio for developing context-aware mobile applications. It contains a context model and repository that stores historical context data. This repository can be queried for deciding on application adaptations.

Similarly, PCOM proposes a component based on the BASE middleware [25]. BASE offers a service-oriented model that abstracts from concrete interoperability protocols and shifts the synchronization and interaction pattern into the middleware requiring only a one-way message service. BASE allows adapting to changes in connectivity by switching protocols on the fly. Adaptation of the application is only supported by signaling (un)availability of services. In order to support the programmer, the component system PCOM supports automatic adaptation of applications.

A different approach is employed by Cascadas [26] project, which aims at building pervasive systems by independent, self-organizing autonomic components. Each component has access to the knowledge repository that organizes and distributes context information. Context information about the pervasive environment is fed to the repository and passes through aggregators that can infer relationships and more knowledge. The project also includes runtime models for verifying execution of components. The resulting system in principle continuously re-organizes and optimizes itself. However, in such an approach, application development may suffer from the unpredictable nature of the system.

7 Conclusion

In this paper, we have presented a real world pervasive application, which purpose is to supervise water management stations and to trigger alarms when something goes wrong. We particularly focused on the architecture that effectively distributes computing and storage functions between cloud and fog. It includes a major component referred to as smart gateway. Its purpose is to collect, process, store data and to run modular analytics. The gateway also sends analytics results and synthetized data to the cloud through a secure connection and a reliable message-oriented middleware. The proposed solution meets a set of thorny requirements. Some of them are specific to Industry 4.0, like the need to limit communication costs in the absence of cheap Internet connection. Others, like those related to latency, security, performance, or availability, are shared by most pervasive applications.

The gateway development turned out to be quite challenging. It must host a variety of components that are developed by different teams with different programming. In order to smooth up this process, it was important to reduce coupling between components in order to allow parallel developments and independent deployments. Dynamism is brought by service-orientation that allows application development through late composition of independent software components.

Two directions are currently explored to improve the current solution. First, deployment has to be more automated [27]. Infrastructure as code is very promising, but must be extended with variability and constraint support. These additions would reflect environment disparities and dynamism. Also, we believe that components administration interfaces must be improved to better deal with starting or stopping phases.

References

1. Weiser, M.: The computer for the 21st century. In: Human-Computer Interaction, pp. 933–940. Morgan Kaufmann Publishers Inc. (1995). http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm
2. Satyanarayanan, M.: Fundamental challenges in mobile computing. In: Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 1–7. ACM, New York (1996)
3. Acatech (ed.): Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group (2013)
4. Lee, J., Bagheri, B., Kao, H.A.: A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. Manuf. Lett. **3**, 18–23 (2015)
5. Lalanda, P., Morand, D., Chollet, S.: Autonomic mediation middleware for smart manufacturing. IEEE Internet Comput. **21**, 32–39 (2017)
6. Chiang, M.: Fog and IoT: an overview of research opportunities. IEEE IoT J. **3**(6), 854–864 (2016)
7. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE IoT J. **3**(5), 637–646 (2016)
8. Lee, C.K.M., Zhang, S.Z.: Development of an industrial internet of things suite for smart factory towards re-industrialization in Hong Kong. In: Advanced Manufacturing and Automation (IWAMA) (2016)
9. Perera, C., Liu, C.H., Jayawardena, S., Chen, M.: A survey on internet of things from industrial market perspective. IEEE Access **2**, 1660–1679 (2014)
10. Lalanda, P., McCann, J.A., Diaconescu, A.: Autonomic Computing – Principles, Design and Implementation. Springer, Berlin (2013). https://doi.org/10.1007/978-1-4471-5007-7
11. Papazoglou, M.P.: Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering, Los Alamitos, CA, USA, pp. 3–12, December 2003
12. Escoffier, C., Hall, R.S., Lalanda, P.: iPOJO: an extensible service oriented component framework. In: IEEE International Conference on Services Computing, SCC 2007, pp. 474–481. IEEE (2007)
13. Pauls, K., McCulloch, S., Savage, D., Hall, R.S.: OSGi in action. Manning (2011)

14. Bardin, J., Lalanda, P., Escoffier, C.: Towards an automatic integration of heterogeneous services and devices. In: IEEE APSCC, pp. 171–178. IEEE Computer Society (2010)
15. Wiederhold, G.: Mediators in the architecture of future information systems. Computer **25**(3), 38–49 (1992)
16. Wiederhold, G., Genesereth, M.: The conceptual basis for mediation services. IEEE Expert **12**(5), 38–47 (1997)
17. Lalanda, P., Escoffier, C., Hamon, C.: Cilia: an autonomic service bus for pervasive environments. In: IEEE SCC, pp. 488–495 (2014)
18. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: Gaia: a middleware platform for active spaces. ACM SIGMOBILE Mob. Comput. Commun. Rev. **6**(4), 65–67 (2002)
19. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project aura: toward distraction-free pervasive computing. IEEE Pervasive Comput. **1**(2), 22–31 (2002)
20. Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., Jansen, E.: The Gator Tech Smart House: a programmable pervasive space. Computer **38**(3), 50–60 (2005)
21. Gu, T., Pung, H.K., Zhang, D.Q.: Toward an OSGi-based infrastructure for context-aware applications. IEEE Pervasive Comput. **3**(4), 66–74 (2004)
22. Lupu, E., et al.: AMUSE: autonomic management of ubiquitous e-health systems. Concurr. Comput.: Pract. Exp. **20**(3), 277–295 (2008)
23. Liu, H., Parashar, M., Hariri, S.: A component-based programming model for autonomic applications. In: Autonomic Computing (2004)
24. Rouvoy, R., et al.: MUSIC: middleware support for self-adaptation in ubiquitous and service-oriented environments. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 164–182. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_9
25. Becker, C., Handte, M., Schiele, G., Rothermel, K.: PCOM - a component system for pervasive computing. In: Proceedings of International Conference on Pervasive Computing and Communications, pp. 67–76. IEEE (2004)
26. Baresi, L., Fernando, A.D., Manzalini, A., Zambonelli, F.: The CASCADAS framework for autonomic communications. In: Vasilakos, A., Parashar, M., Karnouskos, S., Pedrycz, W. (eds.) Autonomic Communications, pp. 147–168. Springer, Berlin (2012). https://doi.org/10.1007/978-0-387-09753-4_6
27. Günalp, O., Escoffier, C., Lalanda, P.: Rondo: a tool suite for continuous deployment in dynamic environments. In: IEEE International Conference on Service-oriented computing (SCC), pp. 720–727 (2015)

eTOUR: A Two-Layer Framework for Tour Recommendation with Super-POIs

Chunwei Wang, Yuanning Gao, Xiaofeng Gao(✉), Bin Yao, and Guihai Chen

Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai 200240, China
{weiwei0224,gyuanning}@sjtu.edu.cn, {gao-xf,yaobin}@cs.sjtu.edu.cn,
gchen@nju.edu.cn

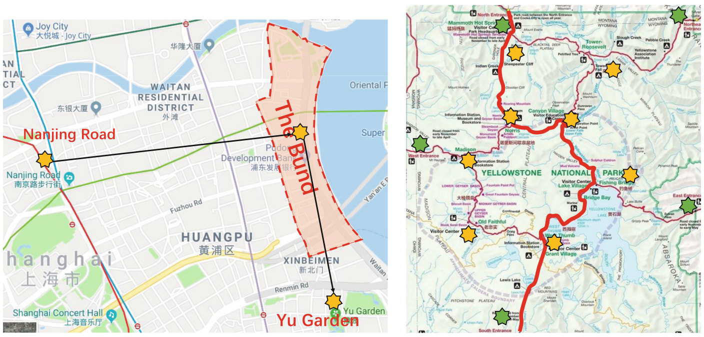
Abstract. Tour recommendation is popular nowadays for providing the best-fit route plans to tourists. Existing applications only focus on the sequences design of Points of Interest (POIs) while ignore the detailed information in large-scale POIs. To further satisfy tourists' demands, we propose *Embedded Tour* (eTOUR), a two-layer framework that takes these large-scale POIs, which we call Super-POIs, into account. The framework is first divided into Outer Model and Inner Model and then combined by an Embedded GRASP-VNS Algorithm based on an embedding strategy. For Outer Model, we apply Greedy Randomized Adaptive Search Procedure (GRASP) for route construction and Variable Neighborhood Search (VNS) for local search. During the outer route construction process, Super-POI is first treated as a "meta node" and in the late period, inner route is revised dynamically to adapt to the outer route where DFS-based Tree Search with Pruning is applied. Furthermore, we consider a special case in the Super-POI and modify the solution of Chinese Postman Problem to reduce the complexity. Finally, experiments based on real datasets demonstrate the effectiveness of our proposal.

Keywords: Tourist Trip Design Problem · Points of Interest
Chinese Postman Problem · Combinatorial optimization

1 Introduction

Tour planning is a challenging task for tourists. They need to gather massive information and design routes under realistic constraints. Nowadays tour recommendation aims to offer tour plan advice for tourists. Current studies usually focus on data analysis on POIs' attributes mined from social networks [3,4]. Then, Tourist Trip Design Problem (TTDP) [2] is proposed to design the route. In these works, each POI is treated just as a vertice, where detailed information is ignored and the scenery during the tour is rarely considered.

However, with higher demand in tour planning, new challenges are brought to current recommendations. POIs have different covering areas and it seems no longer appropriate to treat all the POIs, whatever a small restaurant or a huge scenic park, as equal. It would be ambiguous to recommend a one-day tour in Shanghai, China as shown Fig. 1(a) because what tourists actually prefer is a detailed route in the Bund rather than just reaching it. Hence, sightseeing routes inside the large-scale POIs, which we call them "Super-POIs" in this paper, are required to consider and we go through a toy example to illustrate it.

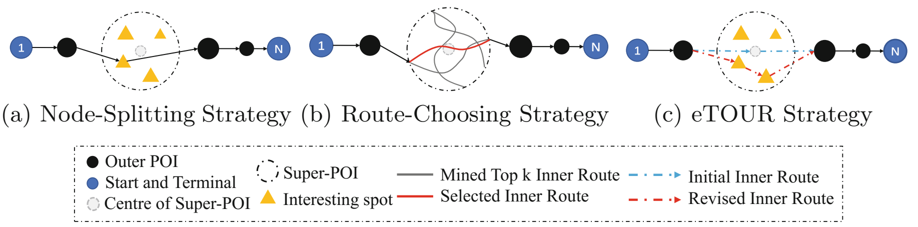


(a) The Bund in Shanghai (b) Yellowstone Park

Fig. 1. Inner route examples

Example 1: John is going on a long vacation in America and one of the POIs recommended is the Yellowstone Park. However, he would get disappointed if a rough tour was offered only telling him to visit it. With five entrances existing, the Yellowstone Park has a complicated road network and plenty of interesting spots inside it. Under this circumstance, John would be forced to select the entrances, interesting spots and scenic routes by himself. In contrary, if a detailed tour plan is given as noted in Fig. 1(b), John would go on this travel happily.

The above example shows the significance of inner route consideration in Super-POIs. Hence, we propose a two-layer framework, *Embedded Tour* (eTOUR) to better satisfy tourists' demands. We first divide it into *Outer Model* and *Inner Model*, considering the visiting sequence of POIs as well as inner routes inside the Super-POIs respectively. Then, to combine these two models, we compare three strategies. Firstly, simply splitting a Super-POI into several ones and merging them into the Outer Graph, as shown in Fig. 2(a), is unsuitable. Sometimes only a few Inner-POIs are selected in tour plans, resulting in a non-full content trip inside the Super-POI. Another strategy in Fig. 2(b) is first to obtain several "good" routes in each Super-POIs and then derives trips including choosing the best inner routes. However, in this case, each Super-POI is required to calculate the inner top k routes initially. Moreover, both two strategies will lead to a size growth in Outer Graph. Differently, in Fig. 2(c), Super-POI is first treated as a "meta-node" during the insertion-based route construction and in the late period, inner route is revised dynamically to adapt to the outer route. This is because two models are dependent with each other. In addition, only selected Super-POIs are supposed to be considered. Hence, we choose the third strategy.



(a) Node-Splitting Strategy (b) Route-Choosing Strategy (c) eTOUR Strategy

Fig. 2. Combination strategies

To achieve this strategy, an *Embedded GRASP-VNS Algorithm* is correspondingly introduced. Moreover, two methods are developed for Inner Model. Finally, experiments based on real datasets demonstrate the effectiveness of our proposal. In the rest of this paper, related work is discussed in Sect. 2. Section 3 describes the framework and Sects. 4, 5 present the algorithms. Section 6 discusses the experiments while conclusions are drawn in Sect. 7.

2 Related Work

Tour Recommendation: Framework of current tour recommendations contains two sections, i.e. offline and online section. Offline focuses on the acquirements of POI information and users' profiles based on photos and contexts shared on the social networks [6,7]. With the attributes of POIs, online section is to select POIs and design routes. A "filter-first, tour-second" framework was proposed in [3]. With pre-selected mandatory POIs, collaborative filtering is applied to identify a subset of optional POIs. Then, trip design is solved via an Iterated Tabu Search algorithm. [1] proposed a global k-means clustering algorithm and POIs are replaced by clusters to decrease the complexity.

Tourist Trip Design Problem: The basic model of TTDP is Orienteering Problem (OP). Its extensions are developed in a various way, taking more realistic constraints into account such as Team OP and OPTW [2]. Among kinds of extensions, TOPTW is the suitable basic problem in tour planning. Vansteenwegen et al. proposed an iterated local search heuristic in [10], which consists of an insertion and a shake step. MC-TOP-MTW [8], taking multi-constraints and multiple time windows into account, is achieved by GRASP and ILS, while in [5] Lin et al. proposed a simulated annealing (SA) algorithm for it.

3 A Two-Layer Framework for Tour Recommendation

The eTOUR framework is shown in Fig. 3. We divide our model into Outer Model and Inner Model, which respectively recommends outer routes and inner routes inside the large-scale POIs.

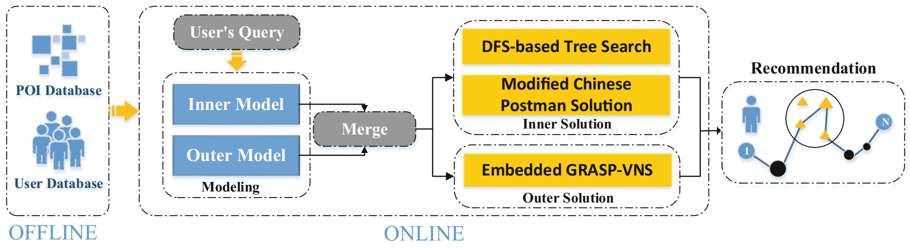


Fig. 3. eTOUR framework overview

Definition 1 (Outer-POI). *Point of Interest that tourists may choose. For each Outer-POI* V_i^O, *it has several properties: visit duration* d_i^O, *popularity* Pop_i^O, *budget cost* b_i^O, *time windows* $TW_i^O = [Open_i, Close_i]$ *and category* C_i^O.

Outer Model aims at the design of POIs' visiting sequences. **Super-POI** is a special kind of Outer-POI with a large area covered. We treat each POI as a vertice and convert an Outer Graph $G^O = (V^O, E^O)$. Note that the transport time t_{ij} between each pair of vertices are known by offline section. To recommend a trip plan for user, the profit of each Outer-POI is first predicted. The profit reflects user's likelihood to visit the POI and is often measured by the combination of POI's popularity and user's preference. Then, with user's query for the trip includes the start and terminal POI, maximum time budget and cost budget, the route planning is referred to *Team Orienteering Problem with Time Windows* (TOPTW) [10]. Its goal is to obtain a route that maximizes the total collected profits which represents user's satisfaction to the route plan.

Definition 2 (Inner-POI). *Important spots inside the Super-POI. It has two types namely interesting spots and entrances or exports. Interesting spot in Super-POI* V_i^O *has properties: visit duration* d_j^i, *popularity* Pop_j^i, *profit* p_j^i *and category* C_j^i *while the visit durations and profits of entrances or exports are set to zero.*

Definition 3 (Scenic-Path). *Beautiful path between Inner-POIs* V_j^i *and* V_k^i, *which has properties: travel duration* t_{jk}^i, *popularity and category.*

Inner Model focuses on the inner route design in the Super-POI. Similarly, the profits of both Inner-POI p_j^i and Scenic-Path s_{jk}^i are first calculated. In this problem, the goal is to obtain an optimal route that maximizes the collected profits both in vertices and edges under the constraints of maximal time T_{max}^i and entrance selection. Note that vertices can be reached more than once but the profit and duration time are only collected at the first reach.

4 An Embedded GRASP-VNS Algorithm

An Embedded GRASP-VNS Algorithm in Algorithm 1 is proposed to combine the Inner and Outer Model. As for Outer Model, there are two steps, i.e., route construction and local search. The route construction is usually insertion-based, which means that route is constructed by inserting the most suitable vertice one by one until no feasible vertice exists. After this, local search is applied to escape from a local optimal solution. During the construction of Outer Model, Super-POIs are first viewed as "meta-nodes" and they can be inserted as the common POI does. In late period, the inner routes of the selected Super-POIs are considered and revised intermittently to adapt to the constructing routes.

GRASP: Insertion-Based Construction. Greedy Randomized Adaptive Search Procedure is often applied for construction [3]. At each insertion iteration, promising vertices are picked into a Candidate List according to Eq. (1) and then randomly select one to insert, which guarantees the route to be greedily and randomly combined. Here, the conception of slack is detailed discussed in [10].

$$Ratio_i = \frac{p_i^2}{Slack_{avg}^i}; \quad Slack_{avg}^i = \frac{\sum_{V_j^O \in R} Slack_j^i}{n} \quad (n = |V_j^O| \in Route) \tag{1}$$

To determine when to adjust the inner route, we define two criterions in Eq. (2). Tightness is measured by the average slackness of the route R while Neighbor Uncertainty signifies the proportion of wait time in Super-POI's neighbors among the total wait time, denoting the change possibility of its neighbors.

$$Tightness = \frac{Slack_{avg}}{T_{max}^O} \leq \alpha; \; Neighbor\ Uncertainty = \frac{W_i + W_{latter(i)}}{\sum_j W_j} \leq \beta \tag{2}$$

VNS: Local Search. Four neighborhood structures [9] are applied, (1) 2-Opt: remove and replace two arcs in a sub-route. (2) Replace: remove one vertice and replace it with an unvisited one. (3) 2-Opt*: remove and replace two arcs between two sub-routes in a solution. (4) Insert: insert one available vertice.

5 Solution of Inner Model

5.1 DFS-Based Tree Search with Pruning

Due to the small size of Inner Graph, DFS-based Tree Search is applied for exact solutions and several pruning methods are proposed. The DFS-Based tree search systematically constructs path begin with the starting node. Current node will deeper search for its connected nodes as childnodes and backtrack to its parentnode if the time constraint breaks or the pruning rule meets.

Rule 1. Due to the multi-allowed reach at each node, blindly searching back and forth between two or several nodes or rounding in cycle is only time consuming with no profit collected, which could be skipped.

Rule 2. Further search can stop if the left time in current searching path less than the Quick-Out Time. Quick-Out Time is defined as the least transport time from present node to the terminal nodes regardless of node visit and is apparently the shortest path to the terminal node, which can be easily derived by Dijkstra Algorithm in $O(|V^i|log|V^i| + |E^i|)$.

Algorithm 1: Embedded GRASP-VNS

```
1  while $RouteNumber \leq MaxNumber$ do
2      repeat
3          Select the top $K$ highest ratio node $\rightarrow$ Candidate List;
4          CurrentNode $\leftarrow$ RandomlyChoose(Candidate List);
5          Insert(CurrentNode);
6          foreach Super-POI inserted in Route do
7              if Revision Condition meets then revise inner route;
8      until no feasible insertion available;
9      RouteSet $\leftarrow$ Route;
10 repeat
11     foreach Route in RouteSet do
12         Local Search: 2-Opt; Replace; 2-Opt*; Insert;
13         Revise inner route with probability $p_r$;
14     Perturbation;
15 until no further improved solution;
16 output BestRoute;
```

5.2 Modified Chinese Postman Solution

Here, we discuss a special case of Inner Model. Under many circumstances, Super-POIs usually have one entrance and the routes traveled by users are similar. To simplify this condition, we define a *Key Sub-Graph* $H^i \subseteq G^i$, which is a connected graph with one entrance and assumed to be mandatorily visited. Moreover, rest of the interesting spots and edges in G^i is set to be optional.

To traverse the Key Sub-Graph H^i, the Chinese Postman Problem (CPP) is referred. In our problem, a modified method of CPP is developed. The fundamental method to solve CPP is to convert the graph into an *Euler Graph* by adding the least-weighted edges. These edges are derived by calculating the minimum weighted matching of the graph constructed by Singular Points and their pairwise shortest paths. Note that Singular Points refer to the vertice V that $d(V) = 2k - 1, k \in \mathcal{N}^*$. The only step we have modified is to derive the shortest path from Graph G^i rather than the Key Sub-Graph H^i. This is motivated by adding a possible lower-weighted edge between two Singular Points and making the optional paths promising to be added. The algorithm can be solved in polynomial time with $O(|V^i|^2 log|V^i| + |V^i||E^i| + |V^i|^3 + |E^i|^2)$.

6 Experiments and Results

Tourism Real Dataset. we crawled POIs information on Baidu Map[1], which includes 82 POIs in Beijing and 50 POIs in Hangzhou. Among these POIs, there are five Super-POIs in Beijing dataset and three in Hangzhou dataset.

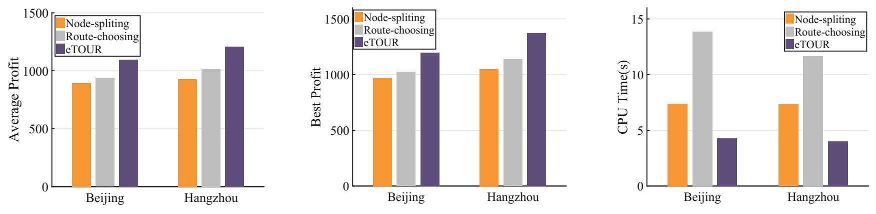
[1] http://lbsyun.baidu.com/.

The attributes of POIs, such as time windows, cost, visit duration location and popularity, can be obtained from crawler data. Besides, we only consider POIs' popularity for profit value while ignoring the user preference discussed in Sect. 3, which will not influence the demonstration of our strategy and algorithm.

The experiments are tested on a Intel Core i5 server with 3.20 GHZ processor and 8 GB RAM. The programs are coded in Python.

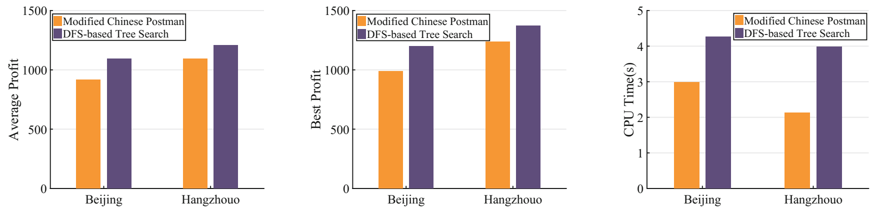
Strategy for Combination of Inner and Outer Model. Here we make comparisons between Node-splitting, Route-choosing and eTOUR Strategy as discussed in Fig. 2. As is shown in Fig. 4, eTOUR Strategy has an apparent advantage over two other strategies on CPU time and is slightly superior in profit, especially when the graph scales out. This is for the reason that the inner route in eTOUR Strategy is calculated during the outer route construction, meanwhile, the outer route will be slightly adjusted according to the latest inner route, which contributes to be closer to the optimum value. In addition, with only selected Super-POIs required to be considered, the time cost is reduced markedly. Moreover, Fig. 4(a) and (c) show that Node-splitting Strategy costs less CPU time than Route-choosing Strategy while the latter gains slightly better profit value. This is because Route-choosing strategy selects more valuable routes, yet it costs much more CPU time.

Performance Under Different Solutions of Inner Model. As shown in Fig. 5(a) and (b), Modified Chinese Postman method is a little inferior to DFS-based Tree Search method in both the average profit value and the best profit



(a) AvgProfit (b) BestProfit (c) CPU Time

Fig. 4. Strategy comparison for combination of two models



(a) AvgProfit (b) BestProfit (c) CPU Time

Fig. 5. Performance under different solutions of Inner Model

value. However, as is depicted in Fig. 5(c), DFS-based Tree Search method costs much more CPU time than Modified Chinese Postman method due to the reason that the latter owns the polynomial complexity.

7 Conclusion

In this paper, we introduce eTOUR, a novel two-layer framework for tour recommendation, taking Super-POIs into account. We believe that the framework can absorb interest of the current recommendations and the application of CPP for a special case may inspire the further data mining of Interesting Patterns.

Acknowledgements. This work is supported by the program of International S&T Cooperation (2016YFE0100300), the China 973 project (2014CB340303), the National Natural Science Foundation of China (Grant number 61472252, 61672353), the Shanghai Science and Technology Fund (Grant number 17510740 200), and CCF-Tencent Open Research Fund (RAGR20170114).

References

1. Alghamdi, H., Zhu, S., El Saddik, A.: E-tourism: mobile dynamic trip planner. In: IEEE International Symposium on Multimedia (ISM), pp. 185–188 (2016)
2. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G.: A survey on algorithmic approaches for solving tourist trip design problems. J. Heuristics **20**(3), 291–328 (2014)
3. Kotiloglu, S., Lappas, T., Pelechrinis, K., Repoussis, P.: Personalized multi-period tour recommendations. Tour. Manag. **62**, 76–88 (2017)
4. Lim, K.H., Chan, J., Leckie, C., Karunasekera, S.: Personalized tour recommendation based on user interests and points of interest visit durations. In: International Joint Conferences on Artificial Intelligence (IJCAI), pp. 1778–1784 (2015)
5. Lin, S.W., Vincent, F.Y.: A simulated annealing heuristic for the multiconstraint team orienteering problem with multiple time windows. Appl. Soft Comput. **37**, 632–642 (2015)
6. Lu, X., Wang, C., Yang, J.M., Pang, Y., Zhang, L.: Photo2Trip: generating travel routes from geo-tagged photos for trip planning. In: ACM International Conference on Multimedia (ACM MM), pp. 143–152 (2010)
7. Majid, A., Chen, L., Mirza, H.T., Hussain, I., Chen, G.: A system for mining interesting tourist locations and travel sequences from public geo-tagged photos. Data Knowl. Eng. **95**, 66–86 (2015)
8. Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.: The multiconstraint team orienteering problem with multiple time windows. Transp. Sci. **47**(1), 53–63 (2013)
9. Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F.: Heuristics for the multi-period orienteering problem with multiple time windows. Comput. Oper. Res. **37**(2), 351–367 (2010)
10. Vansteenwegen, P., Souffriau, W., Berghe, G.V., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Comput. Oper. Res. **36**(12), 3281–3290 (2009)

Service Management

Hierarchical Recursive Resource Sharing for Containerized Applications

Young Jin Kim[1], Young Choon Lee[2], Hyuck Han[3], and Sooyong Kang[1](✉)

[1] Department of Computer Science, Hanyang University, Seoul, Korea
{play1490,sykang}@hanyang.ac.kr

[2] Department of Computing, Macquarie University, Sydney, Australia
young.lee@mq.edu.au

[3] Department of Computer Science, Dongduk Women's University, Seoul, Korea
hhyuck96@dongduk.ac.kr

Abstract. Applications are increasingly containerized using techniques, such as LXC and Docker. Scientific workflow applications are no exception. In this paper, we address the problem of resource contention between concurrently running *containerized* scientific workflows. To this end, we design and implement Hierarchical Recursive Resource Sharing (HRRS), which structures multiple concurrent containers in a hierarchy that automatically and dynamically regulates their resource consumption based on their level/tier in the hierarchy. The hierarchy is recursively updated as the top-tier container completes its execution with the second-tier container becoming the top-tier container inheriting the resource consumption priority. We have evaluated the performance of HRRS using multiple large-scale scientific workflows containerized by Docker. The experimental results show the significant reduction of resource contention as evident in performance improvement of 49%, 160% and 18% compared with sequential execution, concurrent execution with fair resource share and execution with submission interval, respectively.

Keywords: Resource sharing · Containers
Containerized scientific workflows

1 Introduction

First, it was virtual machines (VMs). Advances in x86 VM techniques, such as VMware [2] and Xen [4] have revolutionized the way computing resources are used, with many promises of cloud computing including elasticity and scalability. Now, it is lightweight containers, such as LXC [1] and Docker [9] containers. The co-location of these containers is also actively sought for resource efficiency. This results in the well-known performance isolation issue, as in VM consolidation, due to resource contention between co-located containers.

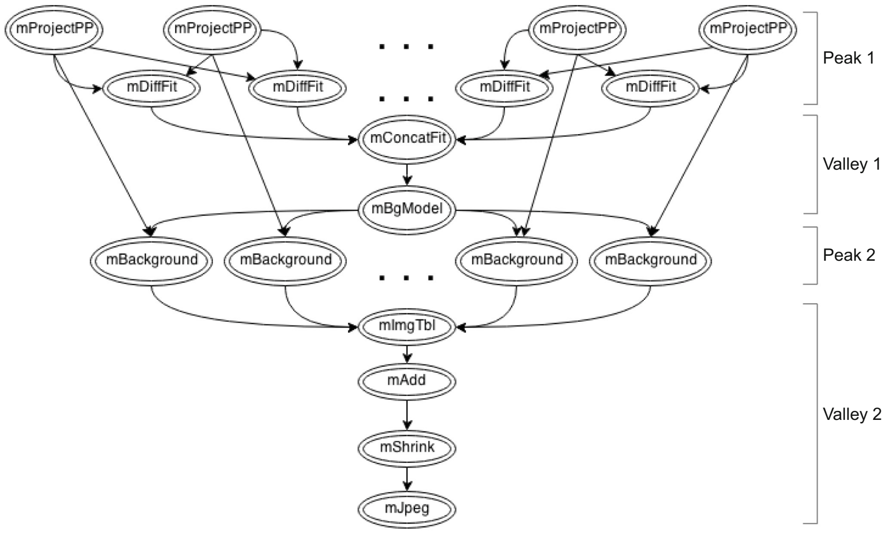
Like many other applications, scientific workflows have been increasingly containerized for efficient management and better performance. However, due to their large-scale nature with high resource demands, the realization of such advantages is difficult to achieve when resource efficiency/utilization needs to be explicitly taken into account. In particular, scientific workflows often deal with large amounts of data leading to their inherent resource consumption pattern of "read I/O - CPU - write I/O" showing resource consumption peaks and valleys (see Fig. 1(b)). While the concurrent execution of scientific workflows can increase resource utilization particularly exploiting those load valleys, the excessive resource contention during resource consumption peaks negates such a utilization benefit resulting in significant performance degradation.

In this paper, we address the problem of resource contention between concurrently running containerized scientific workflows. Scientists often run multiple instances of a particular workflow (or workflow ensemble) for the extensive scientific analysis with different data sets. For example, with the Montage astronomical image mosaic engine [13] (Fig. 1), a set of small image mosaics are needed to produce a large image mosaic; the Galactic Plane workflow ensemble [8] consists of 17 workflows, each of which contains 900 sub-workflows. Clearly, there is a dilemma between resource utilization and performance.

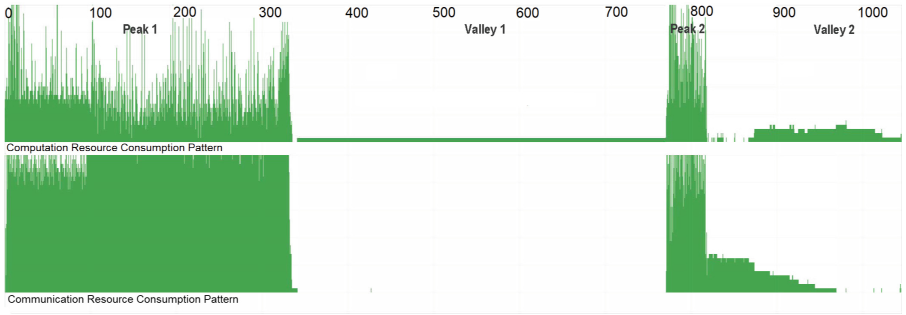
Motivated by the fact that scientific workflows are resource-intensive applications in both CPU and I/O showing a certain resource consumption pattern with several execution phases (see Fig. 1(b)), we propose to concurrently run multiple instances of a containerized scientific workflow application on a *single*, *large* physical machine, with their resource consumption alternating for efficient resource use (or simply resource efficiency). Note that we have chosen to use a single, large physical machine to avoid excessive communication between tasks of a particular workflow. To this end, we develop *Hierarchical Recursive Resource Sharing* (HRRS) as a novel resource sharing technique. HRRS enables multiple containers to be tiered in a hierarchical fashion, for the dynamic and automatic regulation of their resource consumption. In particular, the hierarchy dictates the resource consumption priority. The hierarchy is recursively updated as the top-tier container completes its execution with the second-tier container becoming the top-tier container inheriting the resource consumption priority.

We implement HRRS using Docker containers and demonstrate its efficacy with multiple large Montage workflows, each of which deals with over 8.5K tasks and approximately 35GB of data footprint. The performance gain of HRRS by reducing resource contention has been presented in comparison with sequential execution, concurrent execution with fair resource share and execution with submission interval. The first two execution modes are two extreme ways of execution whereas the last one is a compromise by running containers (Montage workflows) with a certain time interval 'profiled' before, i.e., 260 s for the Montage workflow used in this study. Our experimental results show that HRRS outperforms these three modes by 49%, 160% and 18% on average and up to 72%, 411% and 50%, respectively.

We choose an astronomical image mosaic engine, Montage [13] (Fig. 1) as a representative example of scientific workflows due to the following reasons: (1) there are several execution phases showing peaks and valleys of resource



(a) Workflow structure represented by a directed acyclic graph (DAG). Nodes represent tasks and edges represent precedence/(data) dependencies.



(b) Resource consumption patterns. Top one shows CPU resource consumption pattern. Bottom one shows I/O resource consumption pattern. Corresponding tasks of load peaks and valleys are shown in Figure 1(a).

Fig. 1. Montage workflows.

consumption, (2) scientists in astronomy do have the need to execute multiple instances of a Montage workflow, (3) Montage workflows are both compute-intensive and data-intensive, and (4) the Montage source code and data is available to the general public (i.e., open source[1]). However, HRRS is not limited to Montage or scientific workflows. In fact, it can be applied to any applications that show load peaks and valleys and that have the need for concurrent execution.

[1] The source code and documentation are available at https://github.com/Caltech-IPAC/Montage.

The rest of this paper is organized as follows. Section 1 presents HRRS. Section 3 details the implementation of HRRS. Section 4 presents experimental results. We discuss related work in Sect. 5 followed by our conclusion in Sect. 6.

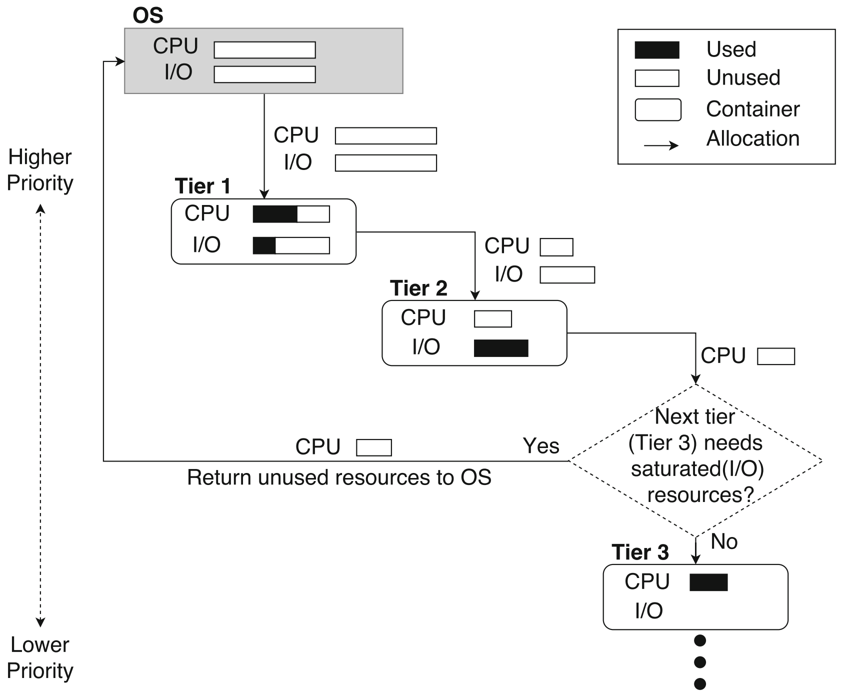


Fig. 2. HRRS conceptual diagram. Whichever resources remaining after use of the upper tiers are passed down (i.e., hierarchy) until either a particular resource is saturated or no more tiers to use the remaining resources; this takes place dynamically and automatically once HRRS structures containers in a hierarchy. In this figure, the I/O resource is saturated with the first two tiers. If the 3rd tier (more precisely, the application running in the 3rd tier) requires I/O resource, it cannot proceed its execution although there is some spare CPU resource. The spare CPU resource is then returned to the root (OS).

2 Hierarchical Recursive Resource Sharing

To achieve the minimal resource contention with concurrent execution of multiple applications/containers, HRRS arranges containers in a hierarchical structure (tiers, see Fig. 2). HRRS allocates resources (CPU and I/O) maximally to the container in the top tier (or simply the top-tier container) and minimally to containers in the rest of tiers. In this way, the top-tier container is privileged to use as much resources as it requires ensuring performance isolation. In the meantime, the second-tier container with the minimal resource allocation is able to use whatever amounts of resources left from the top tier due to load peaks and valleys. Although all tiers from the second tier have the same resource allocation weights (minimally allocated amounts of resources), the container hierarchy with dynamic resource allocation and resource weight inheritance makes their resource allocation weights relative. In particular, for a given tier, if its upper tier(s) leaves some unused resources, the current lower tier has privilege to use as much as it requires, as if it is the top tier, despite its minimal resource allocation. Also, this dynamic resource allocation takes place recursively as the top-tier container completes its execution making the second-tier become the top-tier inheriting resource allocation weights.

2.1 Dynamic Resource Allocation

In order to exploit resource consumption patterns of applications, resources need to be dynamically allocated. We ensure such dynamic resource allocation to be hierarchical and recursive. In particular, containers in the resource allocation hierarchy use resources as much as possible from top to bottom until remaining resources are not sufficient for a particular tier. The identification of resource demands for different tiers is shown in the following. Note that CPU and I/O resources are collectively dealt with.

The resource allocation of HRRS is divided into two cases, the maximal allocation to the top tier and the minimal allocation to the remaining lower tiers. The CPU resource allocation of the top tier is calculated as follows:

$$CPU_1 = min(CPU_{max}, Demand_{CPU_1}) \tag{1}$$

where $Demand_{CPU_1}$ is the amount of CPU demanded by the top tier container W_1. Equation 1 ensures the allocation of the minimum between the demanded amount of CPU resource and the maximum amount of CPU resource which can be provided by the system. After the resources are allocated to W_1, the amount of CPU resources available to the next tier is $CPU_{max} - CPU_1$. Therefore, the maximum amount of CPU available to the container W_n located in the n-th tier is expressed as:

$$CPU_{max_n} = CPU_{max} - \sum_{k=1}^{n-1} CPU_k \tag{2}$$

Then, the resources that can be allocated to W_n are calculated as follows.

$$CPU_n = min(CPU_{max_n}, Demand_{CPU_n}) \tag{3}$$

If the overall resource allocation is dictated by controlling only CPU resources, there might be a serious performance issue when concurrently executing (workflow) containers enter a peak I/O usage period. In particular, in case that the upper tier does not use 100% of the CPU, and yet, I/O is fully saturated, if we allocate resources only considering CPU resources, there is a chance that some remaining CPU resources are allocated to one or more lower tiers. Such

CPU resource allocation incurs some side effect that the container located in the lower tier requests I/O resources using its allocated CPU resources. In order to resolve this side effect, I/O resources are also explicitly taken into account by HRRS' dynamic resource allocation. In case of block I/O, the resources are taken from the upper tier according to the hierarchical structure. Therefore, resources can be allocated in the same manner as the CPU. Similar to Eq. 3, the block I/O resource that can be assigned to W_n can be expressed as follows.

$$IO_n = min(IO_{max_n}, Demand_{IO_n}) \tag{4}$$

Then, the system resource $Resource_n$ that can be allocated to W_n is expressed as follows.

$$Resource_n = <CPU_n, IO_n> \tag{5}$$

2.2 Recursive Weight Inheritance

When the top-tier container is terminated while other containers in the hierarchy are still running, the container in the second tier automatically becomes the top tier. HRRS enables the very next tier (to the terminating top tier) to "inherit" resource allocation weights. In fact, the actual weight *values* of the new top-tier remain as before. However, as the position of this tier is at the top of the hierarchy, its weights become relative maximum. Specifically, in any tier of the hierarchy, there is only one container and thus, its resource allocation weights are relative maximum. Such inheritance takes place recursively. Note that if the priority corresponding to the top tier is not inherited to the next lower tier by the very nature of the HRRS' hierarchical structure, the remaining containers work the same way as fair resource share in which all running containers are competing with each other.

3 Implementation

In this section, we detail the prototype implementation of HRRS for containerized scientific workflows. We use Docker [9] to containerize Montage scientific workflows. The actual execution of such workflow is dealt with by our workflow execution engine (DEWE v2[2] [15]). In particular, for each workflow we create a Docker container based on the Docker image file for DEWE v2 with a respective Montage workflow application and data.

We use a Linux kernel feature of `cgroups`[23] for resource control. In particular, CPU and block I/O resources are controlled with `cpu.share` and `blkio.weight` that enable to allocate proportional CPU time and I/O bandwidth to the (control) groups, respectively. `cgroups` is also well integrated with Docker.

[2] The source code and tutorial of DEWE v2 are available from https://github.com/qyjohn/DEWE.v2.

The hierarchy of tiers in HRRS is constructed by setting each Docker container (tier) its corresponding priority in the hierarchy. Since a set of containers are created for running multiple instances of a particular Montage workflow, the hierarchy is constructed in the order of their creation, i.e., time stamps. The actual hierarchy is constructed using the `--cgroups-parent` option of Docker, which determines the location of cgroups directory. The parent of the first container is the root directory of `cgroups` and parents of the following containers are dictated by the hierarchy using `cgroups-parent`. For example, for the first tier container (`container_1`) and the second tier container (`container_2`),their values of `--cgroups-parent` are `/root_cgroup/` and `/root_cgroup/container_1`, respectively.

The sharing values (resource allocation weights) for occupying the resources, such as `cpu.share` and `blkio.weight` can be inherited from the upper tier to lower tier. In particular, we set (relative) sharing values of CPU and I/O to 1024 and 1000, respectively, for the top tier and 2 and 10, respectively, for the rest of tiers. These extreme values ensure the top tier takes the resources as much as it needs and the sharing values keep on being allocated to the bottom tier recursively; and, this works exactly the same way of Eqs. 3 and 4.

4 Experiments

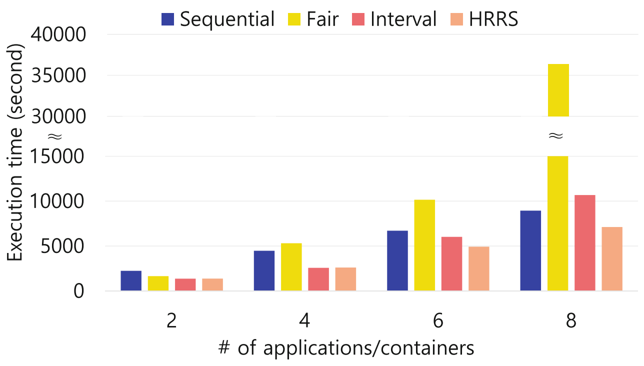
In this section, we present our evaluation results in comparison with three execution modes. The results are presented in three respects: CPU utilization, read I/O throughput and write I/O throughput.

4.1 Experimental Setup

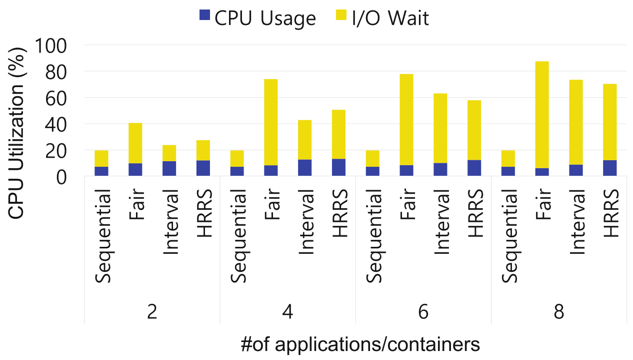
We have used Montage astronomical image mosaic engine. In particular, 6.0 degree Montage workflow is used. A 6.0 degree Montage workflow consists of 8,586 tasks, 1,444 input files with a total size of 4.0 GB, and 22,850 intermediate files with a total size of 35 GB. We conducted our experiments on a 32-core Intel Xeon (E5-2650 v2) machine with 128 GB of RAM and 1 TB of 7200 RPM HDD. Docker version 17.12.0-ce is used in the Linux kernel version of 4.4.0-83 generic. We run one application per container; hence, hereafter we use application and container interchangeably. The numbers of containers used in our experiments are 2, 4, 6 and 8.

The execution modes compared with HRRS are as follows.

- Sequential: containers run one at a time.
- Fair (resource share): all containers run simultaneously using default resource schedulers of Linux (Completely Fair Scheduler or CFS for CPU and Completely Fair Queuing or CFQ for I/O).
- Interval [15]: containers are run in a certain time interval (i.e., 260 s in this study) identified through the prior profiling, i.e., the elapsed time between the beginning of first load peak and its end.



(a) Execution Time



(b) CPU resource usage including I/O wait

Fig. 3. Execution time and resource usage with respect to different numbers of 6.0 degree Montage workflows. I/O_Wait in (b) represents the waiting time for the response from I/O devices.

4.2 Results

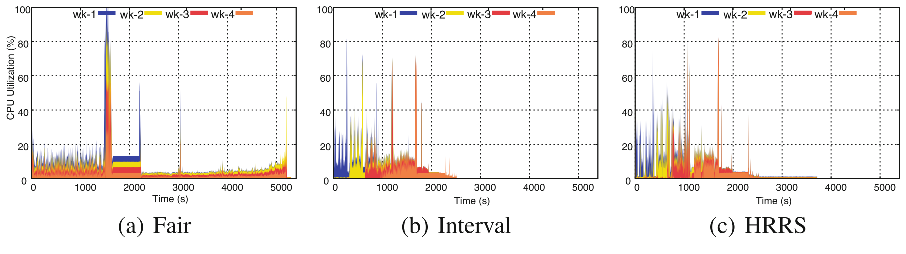
We first present the comparison of overall performance between four different execution modes including our own HRRS (Fig. 3). Results are shown in terms of execution time (Fig. 3(a)) and CPU utilization including I/O wait (Fig. 3(b)), respectively. Detailed experimental results for 4, 6 and 8 containers are then presented (Figs. 5, 6, 7, 8, 9 and 10). Note that due to space limit, we show condensed figures in 3D for I/O resource consumption of 4 and 6 containers. Besides, we have left out detailed results for 2 containers due to insignificance.

4.3 Overall Performance

The effect of different execution modes is primarily reflected on the (total) execution time of given workflows. The execution times of Sequential can be seen as a baseline. The performance of Fair is worst when the number of containers becomes more than two as concurrent containers compete for the same resource at the same (or very similar) time. Such resource contention becomes excessive when running 8 containers concurrently as shown in Fig. 3(a); that is, the execution time is nearly three times longer than that of even Sequential.

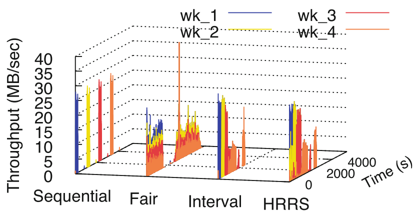
Interval and HRRS apparently mitigate the resource contention by alternating load peaks and valleys. While HRRS constantly shows shortest execution times, Interval only gives shorter execution times over Sequential and Fair for up to 6 containers. This performance difference is due to the nature of resource sharing, i.e., static and dynamic for Interval and HRRS, respectively. In particular, the worse performance with 8 concurrent containers in Interval is due to primarily accumulated resource contention. When running multiple containers (even of the same application), unless the periods of load peaks and valleys are identical and the sum of resource consumption of peaks and valleys is less than the maximum available resources, a certain degree of resource contention is unavoidable. As can be seen in resource consumption pattern of Montage in Fig. 1(b), peaks and valleys cannot be perfectly alternated resulting in some delay; and, such delay accumulates. As a result, the interval profiled is no longer effective.

In the meantime, HRRS minimizes such delay accumulation by its priority hierarchy. The resource consumption–at any given point in time–between containers is dynamically and automatically ensured not to exceed the maximum available resources in terms of both CPU and I/O. This can be noticed by average CPU consumption in Fig. 3(b). In particular, the higher average CPU consumption of Fair and Interval indicates more resource contention. Note that the reduced performance gap between sequential execution and HRRS with 8 concurrent containers is due to lack of memory. In particular, HRRS at any given time deals with several containers exceeding the memory available. As a result, lower tiers use swap space of disk; and, this adds execution time overheads. This issue can be resolved by vertical scaling.

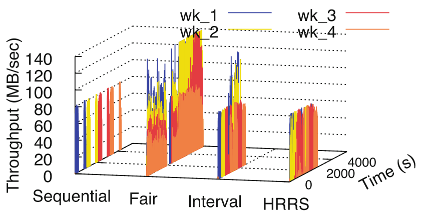


(a) Fair (b) Interval (c) HRRS

Fig. 4. CPU usage patterns (excluding I/O wait) of four 6.0 degree Montage workflow w.r.t. different execution modes.



(a) Read I/O



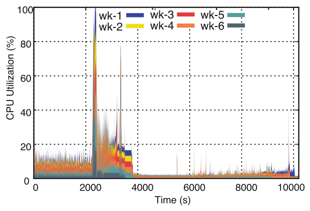
(b) Write I/O

Fig. 5. Resource usage patterns of four 6.0 degree Montage workflow w.r.t. different execution modes.

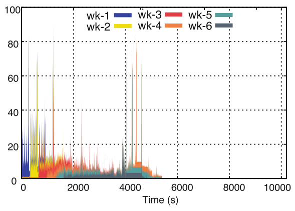
4.4 Results for 4 Containers

Figures 4 and 5 show the resource consumption patterns of execution of four containers. When the number of containers is 4, the performance of Fair is worse than that of sequential with 5,412 s and 4,472 s, respectively. Looking at the average resource utilization, in the case of Fair, although the total CPU utilization rises to approximately 74%, the actual CPU utilization except I/O wait reduced to 8.4%. This means less CPU utilization to actually handle tasks, but more CPU resources are used to wait for I/O because of a large number of concurrent I/O requests. On the other hand, in the case of Interval and HRRS, 2,600 and 2,568 s are recorded, respectively. Total CPU usage increased about twice as much as that of running two containers. The actual CPU utilization rates except I/O wait, were 12.77% and 14.26%, respectively.

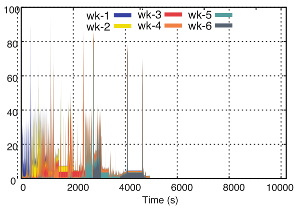
The accumulated delay phenomenon of Interval appears between the third container and the fourth container (Fig. 4(b)). In particular, the resource consumption of third and forth containers conflicts each other during the period of `mProjectPP` and `mDiff`. However, in the case with four containers, the conflicted period is very short and only two containers were overlapped in nearly the end of the entire execution. The accumulation of such contention does not significantly affect the overall performance of Interval. Thus, HRRS (Fig. 4(c)) and Interval (Fig. 4(b)) show similar performance when four containers are running, as shown in Fig. 3(a).



(a) Fair

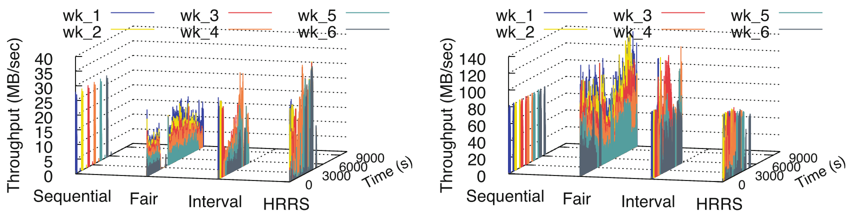


(b) Interval



(c) HRRS

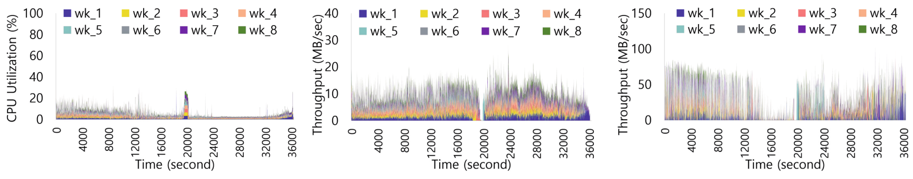
Fig. 6. CPU usage patterns (excluding I/O wait) of six 6.0 degree Montage workflow w.r.t. different execution modes.



(a) Read I/O (b) Write I/O

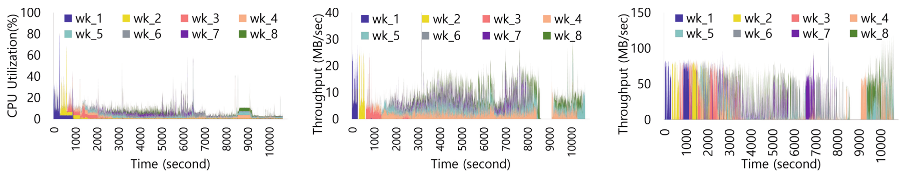
Fig. 7. Resource usage patterns of six 6.0 degree Montage workflows w.r.t. different execution modes.

Figure 5(a) describes read I/O patterns. Both Interval and HRRS can be seen that read I/O operations occurred in the period of `mBackground∼mJPEG` (see Fig. 1) as in the case of Fair. In Sequential execution mode, since the read operations during `mBackground∼mJPEG` are processed in memory, the actual I/O does not occur. However, when four containers simultaneously run, the read operations request the actual I/O to the I/O devices because of insufficient memory. In the case of Fair, all containers are processing the workflow of the same phase trying to occupy memory at the same time. In the cases of Interval and HRRS, the memory occupancy time of each container is different. Therefore, operating system has more time to deal with managing memory resulting in much less I/O operations than Fair.



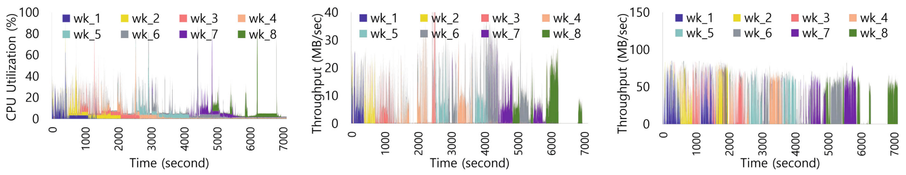
(a) CPU (excluding I/O wait) (b) Read I/O (c) Write I/O

Fig. 8. Resource usage patterns of eight 6.0 degree Montage workflows using Fair share.



(a) CPU (excluding I/O wait) (b) Read I/O (c) Write I/O

Fig. 9. Resource usage patterns of eight 6.0 degree Montage workflows using Interval.



(a) CPU (excluding I/O wait) (b) Read I/O (c) Write I/O

Fig. 10. Resource usage patterns of eight 6.0 degree Montage workflows using HRRS.

4.5 Results for 6 Containers

In the case with six containers, the I/O contention problem of Interval becomes more apparent. The total CPU utilization of Interval is increased by 20%, but the actual CPU utilization was 2.5% lower than the case of 4 containers. This means that the proportion of I/O wait has increased. Figure 6 shows the CPU usage pattern of different execution modes. In the case of Interval, the first four containers work relatively well as seen in previous experiments, but from the fifth container, the interval is far less effective. Figures 7(a) and 7(b) show I/O contention dramatically increases with Interval as containers' execution phases coincide from fourth container and onward (wk_4, wk_5 and wk_6). In particular, execution phases of different containers are gradually not well alternating making resource consumption patterns similar to those of Fair at some point, resulting in performance degradation.

Such an issue is not observed with HRRS. Although the actual CPU usage was reduced by 1%, the overall CPU utilization was increased by 7%. This means that the proportion of I/O wait did not increase significantly compared to other execution modes. HRRS has no significant difference in resource utilization pattern between the six containers. HRRS ensures the top tier consumes as much as resources and passes only those left to lower tiers. As a result, resource contention is constantly avoided. This not only contributes to good performance but also increases the actual CPU utilization.

4.6 Results for 8 Containers

Figures 8, 9 and 10 show resource usage patterns of eight containers using Fair, Interval and HRRS, respectively. As the number of containers increases, CPU resource usage of Fair and Interval emerge to a similar pattern, i.e., CPU resource usage patterns of the last four containers. In the meantime, HRRS manages resources based on the hierarchy regardless of the number of containers. In particular, since resources remaining from the upper tiers are allocated to the lower tiers, containers in the lower tiers can acquire resources in the valley phase of the upper tiers, so even if the number of containers is increased to 8, resource contention is much avoided.

5 Related Work

Efficient resource management has been studied for different purposes and applications including containerized scientific workflows [7,11,14,15,17,19,21,22,24]. Two key objectives in these studies are maximizing resource utilization and minimizing resource contention.

5.1 Resource Sharing for Maximization of Utilization

A majority of previous works on resource management in the cloud environment focus on how to fairly allocate the certain resources to a VM in the hypervisor or scheduler level [7,14]. The authors in [7] studied proportional sharing through the absolute assurance to guarantee the fairness of CPU usage in the Database-as-a-Service system. This technique works well for CPU sharing but it does not solve problems such as I/O contention. The authors in [14] developed a framework to apply priority to I/O for scheduling, and the authors in [22] studied problems arising when using high performance storage devices such as SSD. A high-performance storage device has multiple channels, but since the Linux I/O scheduler has a single queue, fairness is not guaranteed when I/O requests are made in various applications. To solve this problem, the authors in [22] implemented a virtual multi-channel. These previous works focus primarily on single resource type remaining resource contention with multiple resources types in our study still a challenge.

On the other hand, the work in [11] identified the need for resource allocation and allowed users to share resources fairly through the share value. The authors in [17] proposed the resource allocation technique that provides the minimum resources in order to satisfy SLA (service level agreement) calculated based on resource monitoring in the multi-tenant situation. However, when users use the same resource simultaneously, the above resource sharing techniques share resources according to the max-min fairness method, so the resource contentions occur in the workflow ensemble executions which the resource usage patterns of jobs are the same.

5.2 Resource Shaping for Minimization of Contention

Resource consumption shaping was proposed as an extension to network-traffic shaping for data center utilization [12]. In our previous work [19], we studied resource shaping at local task/core level and proposed a technique called *Local Resource Shaper* (LRS). This technique simply uses two tiers, active tier and passive tier for shaping CPU consumption for MapReduce jobs.

There were attempts to maximize resource utilization by profiling jobs in advance to find the resources bottleneck [21] and Cake [24] uses a two-level scheduling scheme to dynamically adjusts the level of concurrency based on measured resource contention (device latency). In order to fully utilize one type of resource, they tend to face the underutilization of other resources.

5.3 Workflow Scheduling

Scientific workflow applications have been increasingly deployed and executed in clouds to take advantage of resource abundance and cost efficiency [3,5,6,18,20]. More recently, scientific workflows have also been containerized using Linux containers, particularly for more efficient application management [10,25]. As these applications spawn a large number of precedence-constrained tasks, their scheduling faces several challenges in clouds, such as optimizing resource provisioning and minimizing costs of their execution. In [15], we studied the large-scale execution of multiple scientific workflows. We showed that running multiple workflows with a certain time interval helps reduce resource contention. However, not only the profiling of workload is a prerequisite to find the "right" interval, but also the interval can become inadequate as the number of applications increases. In this paper, we have eliminated the need for interval; instead, we have studied the method of recursively allocating resources from upper tiers to lower tiers through hierarchical structure.

The authors in [16] showed that the total execution time of scientific applications can vary depending on the resource consumption patterns of the co-running applications in three types of computing platforms, supercomputers, grids and clouds. The use of resource consumption patterns in [16] is similar to our approach of using peak and valley load periods. However, this work focuses on performance by co-scheduling tasks of applications matching different capacities of resources of three computing platforms.

6 Conclusion

In this paper, we have presented a new resource sharing technique, called HRRS for containerized scientific workflow applications. These applications are becoming increasingly large in terms of the number of tasks and resources requirements of CPU and I/O. The scale of the execution of such applications becomes even larger with the need for running multiple instances of them. The optimization of resource utilization plays a crucial role. However, we identified the maximization of resource utilization by simply consolidating workloads/containers often makes a detrimental impact on performance due to excessive resource contention. Our HRRS technique significantly overcomes such resource contention by structuring multiple containers in a hierarchy with recursive resource weight inheritance. In particular, HRRS enables multiple containers to dynamically use resources (both CPU and I/O) by alternating load peaks and valleys. We have demonstrated that the automation of such alternating without fine control of resource usage is possible for containerized applications of Montage astronomy workflows. Our experimental results confirmed this claim with HRRS' performance improvement sourced from the reduction of resource contention.

Acknowledgement. This work was supported by the ICT R&D program of MSIP/I-ITP (12221-14-1005, Software Platform for ICT Equipment). Dr. Young Choon Lee's work was supported by the AWS Cloud Credits for Research program.

References

1. Linux Containers (LXC). https://linuxcontainers.org/. Accessed 2 July 2018
2. VMware. https://www.vmware.com/. Accessed 25 July 2018
3. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
4. Barham, P., et al.: Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP), pp. 164–177 (2003)
5. Chen, W., Lee, Y.C., Fekete, A., Zomaya, A.Y.: Adaptive multiple-workflow scheduling with task rearrangement. J. Supercomput. **71**(4), 1297–1317 (2015)
6. Chopra, N., Singh, S.: Deadline and cost based workflow scheduling in hybrid cloud. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 840–846 (2013)
7. Das, S., Narasayya, V.R., Li, F., Syamala, M.: CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service. Proc. VLDB Endow. **7**(1), 37–48 (2013)
8. Deelman, E., Juve, G., Malawski, M., Nabrzyski, J.: Hosted science: managing computational workflows in the cloud. Parallel Process. Lett. **23**(02) (2013)
9. Docker Inc.: Docker Container, https://www.docker.com/. Accessed 25 July 2018
10. Gerlach, W., Tang, W., Wilke, A., Olson, D., Meyer, F.: Container orchestration for scientific workflows, pp. 377–378 (2015)
11. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: NSDI, vol. 11, p. 24 (2011)
12. Hamilton, J.: Internet-scale service efficiency. In: LADIS (2008). Keynote talk
13. Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Prince, T.A., Williams, R.: Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. Int. J. Comput. Sci. Eng. **4**(2), 73–87 (2009)
14. Jain, N., Lakshmi, J.: PriDyn: enabling differentiated I/O services in cloud using dynamic priorities. IEEE Trans. Serv. Comput. **8**(2), 212–224 (2015)
15. Jiang, Q., Lee, Y.C., Zomaya, A.Y.: Executing large scale scientific workflow ensembles in public clouds. In: 2015 44th International Conference on Parallel Processing (ICPP), pp. 520–529 (2015)
16. Kim, S., Hwang, E., Yoo, T.K., Kim, J.S., Hwang, S., Choi, Y.R.: Platform and co-runner affinities for many-task applications in distributed computing platforms. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 667–676 (2015)
17. Krebs, R., Spinner, S., Ahmed, N., Kounev, S.: Resource usage control in multi-tenant applications. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 122–131 (2014)
18. Lin, C., Lu, S.: Scheduling scientific workflows elastically for cloud computing. In: 2011 IEEE 4th International Conference on Cloud Computing (CLOUD), pp. 746–747, July 2011
19. Lu, P., Lee, Y.C., Gramoli, V., Leslie, L.M., Zomaya, A.Y.: Local resource shaper for mapreduce. In: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 483–490 (2014)

20. Maciej, M.: Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press
21. Polo, J., et al.: Resource-aware adaptive scheduling for mapreduce clusters. In: ACM/IFIP/USENIX Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware (Middleware), pp. 187–207 (2011)
22. Tan, H., Li, C., He, Z., Li, K., Hwang, K.: VMCD: a virtual multi-channel disk I/O scheduling method for virtual machines. IEEE Trans. Serv. Comput. **9**(6), 982–995 (2016)
23. Heo, T.: cgroups-v2. https://www.kernel.org/doc/Documentation/cgroup-v2.txt. Accessed 25 July 2018
24. Wang, A., Venkataraman, S., Alspaugh, S., Katz, R., Stoica, I.: Cake: enabling high-level SLOs on shared storage systems. In: ACM International Symposium on Cloud Computing (SoCC) (2012)
25. Zheng, C., Thain, D.: Integrating containers into workflows: a case study using makeflow, work queue, and docker. In: Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, VTDC@HPDC 2015, Portland, Oregon, USA, 15 June 2015, pp. 31–38 (2015)

A Fuzzy-Based Auto-scaler for Web Applications in Cloud Computing Environments

Bingfeng Liu[1], Rajkumar Buyya[1], and Adel Nadjaran Toosi[2](✉)

[1] School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
bingfengl@student.unimelb.edu.au, rbuyya@unimelb.edu.au

[2] Faculty of Information Technology, Monash University, Melbourne, Australia
adel.n.toosi@monash.edu

Abstract. Cloud computing provided the elasticity for its users allowing them to add or remove virtual machines depending on the load of their web applications. However, there is still no ideal auto-scaler which is both easy to use and sufficiently accurate to make web applications resilient under the dynamic load. The threshold-based auto-scaling approaches are among the most popular reactive auto-scaling strategies due to their high learnability and usability. However, the static threshold would become undesirable once the workload becomes highly dynamic and unpredictable. In this paper, we propose a novel fuzzy logic based approach that automatically and adaptively adjusts thresholds and cluster size for a web application. The proposed auto-scaler aims at reducing resource consumption without violation of Service Level Agreement (SLA). The performance evaluation is conducted with the real-life Wikipedia traces in the Amazon Web Services cloud platform. Experimental results demonstrate that our reactive auto-scaler efficiently reduces cloud resources usage and minimizes the SLA violations.

1 Introduction

The pay-as-you-go, elasticity, and on-demand nature of cloud resources are among the main features exhorting web service providers to host their application in clouds nowadays. Cloud users can rapidly launch Virtual Machines (VMs) for a particular time frame and only pay for their usage. In other words, they do not need to spend a significant amount of time and money to buy the actual hardware and pay maintenance fees [2,5]. Since the rate of web application requests varies with time, it is often hard to determine the right amount of needed cloud resources [13]. Manually watching the load of the web-application is both tedious and resource-wasting led to the birth of auto-scaling. An auto-scaler automatically removes or adds the right amount of resources to optimize the performance and cost of the web-application.

The auto-scaling methods fall into two main categories: *reactive* and *proactive* approaches [16,22]. The reactive approaches check if certain metrics (e.g., CPU

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 797–811, 2018.
https://doi.org/10.1007/978-3-030-03596-9_57

utilization) exceed a threshold and provision VMs to meet the demand. The reactive methods only use recently observed metrics to understand the current state of the cluster and plan scaling decisions. The proactive approaches involve modeling and predicting the load of the web application based on the patterns of the historical data using machine learning techniques [16,22]. However, proactive approaches are difficult to tune and often do not perform well for highly dynamic unforeseen workloads. The proactive methods need to continuously update their model to capture changes in the load patterns to predict future where the growth of the model complexity might become the burden of cluster resources.

Due to the simplicity and the adequate performance, most companies and organizations still prefer to use reactive approaches [16,22]. In line with this view, we also focus on more commonly accepted reactive approaches in this paper. However, the problem of reactive approach is that the threshold does not change according to dynamic workloads and cluster state. Therefore, the naïve static threshold-based approaches often have oscillation problem where the auto-scaler frequently keeps changing the number of VMs back and forth to adjust the cluster size [22]. In this paper, we propose a reactive auto-scaling approach that aims at building dynamic thresholds to overcome these issues. Our approach dynamically adjusts thresholds based on the current cluster size and load.

To this end, the fuzzy logic looks very intuitive to use and unlike machine learning, it does not need historical data and long training time to build the model. One can set up fuzzy linguistic rules with metrics that are critical to satisfying the Service Level Agreement (SLA). For example, a rule can be as simple as "IF *cpu load* IS *low* THEN *cluster size* IS *small*". Two fuzzy engines are proposed in this paper. The first one uses request response time and cluster size to output the appropriate upper threshold value for the auto-scaler. If the current response time exceeds the dynamic upper threshold, then the second fuzzy engine with cluster size and CPU load as inputs comes into play to determine the cluster size avoiding the violation of the response time constraint required by the SLA.

The evaluation of the proposed auto-scaler is performed via a Wikijector agent [3] replaying the real-life requests from Wikipedia traces to stress the cluster and monitoring how auto-scaler adjusts the cluster size to handle this stress. The performance of the proposed approach is compared to both the Amazon Web Services (AWS)'s auto-scaler and its emulated version. The results demonstrate that the proposed auto-scaler significantly reduces cost and SLA violations.

2 Background

Cloud providers like AWS offer built-in auto-scaling services to their users for dynamic auto-scaling of resources used by applications [17]. AWS offers its auto-scaling service via *auto-scaling group* and uses *Cloud Watch* to monitor user-defined metrics [18]. The user can set alarms on metrics to trigger scaling policies once they are over certain thresholds. The AWS auto-scaler similar to many other approaches uses static thresholds and static steps for adding or removing VMs.

Most of the auto-scalers (including our proposed method) adopts *Monitoring*, *Analyzing*, *Planning* and *Execution* (MAPE) loop to adjust the size of the cluster to fulfill the SLA requirement [12,16]. *Monitoring* is not only necessary for the decision making of the auto-scaler but also important for evaluating and improving its performance by logging all the relevant metrics data for offline analysis. In this work, *Ganglia* [14] is used for the monitoring purpose. The *Analyzing* phase is for making the gathered metrics meaningful and to pass them as the inputs to the auto-scaler to trigger the scaling policies at the right time. The *Planning* phase is a challenging phase in MAPE loop – there is no commonly agreed way to know how many VMs are needed to avoid over-provisioning or under-provisioning. The final step which is called *Execution* is responsible to perform the actual scaling-in or scaling-out processes according to the decision of the new cluster size made in the Planning phase via the cloud provider's APIs.

3 Related Work

There are many reactive auto-scaling approaches proposed in the literature. Lim et al. [11] focused on using the averaged CPU utilization of VMs to determine the number of VMs needed to be removed or added. They use a pair of threshold boundaries to make a range but not just a single limit, and then varied the lower bound of the thresholds to make sure the auto-scaler does not switch the number of VMs back immediately after changing the values of the thresholds. Hasan et al. [8] proposed a novel static threshold technique to prevent the oscillation problem for a short period of fluctuating load. The authors first choose static upper and lower bound and then select extra sub-upper and sub-lower bound in between. If the current metrics exceed any of the thresholds for a specific pre-set duration, the system starts a scaling-out or scaling-in the cluster.

The oscillation problem is a common issue in auto-scaling, and many solutions have been proposed to address it in the literature. The oscillation problem occurs when request load changes slightly and triggers the scaling-out process to add a new VM into the cluster. However, the newly added VM causes a load drop, and the new cluster size becomes too powerful for the current load which in turn triggers a scale-in process in a short period. Unlike [8,11], our proposed auto-scaler adaptively updates the upper-threshold based on the cluster size. The main idea is that a larger cluster has more tolerance to small changes in the load and upper threshold is set to a higher value.

Frey et al. [7] developed a fuzzy logic auto-scaler to improve Quality of Service (QoS). The result of the fuzzy auto-scaler shows the technique is effective in reducing the response time. Their method uses inputs such as 'high prediction' (request load) and the 'slope of the response time'. The high prediction input helps the cluster perform scaling earlier and the slope of response time helps to add multiple VMs to reduce the response time. The high prediction input in [7] is formed based on the knowledge of the application's history load pattern whereas the proposed auto-scaler does not assume a priori knowledge of the application.

Lama et al. [10] proposed an adaptive fuzzy system using machine learning without offline learning. The neural network dynamically generates fuzzy rules, membership functions and input parameters with online learning algorithms.

Contrary to our approach that focuses on adapting thresholds, they focus on constructing the fuzzy control structure and adapting control parameters using online learning approaches. In their approach, fuzzy engine has to adapt itself to changes in the load that takes time and could seriously affect the small web applications that are growing popularity [16,22]. Jamshidi et al. [9] also developed an adaptive Fuzzy logic based auto-scaler with Fuzzy Q-learning method to modify the fuzzy rules at run time so that defining fuzzy rules are not longer a manual process. Arabnejad et al. [1] enhanced the output (actions) of the fuzzy controllers with reinforcement learning. The reinforcement learning uses the current state of the system, and randomly tries out different actions to set the actions of the fuzzy controllers. It takes a random approach to explore all the possible actions until the model cannot improve the overall performance. Müller et al. [15] built a governance platform to satisfy multiple users' QoS requirements where users are routed to different provisioning levels depends on their SLAs.

Kubernetes is a cluster manager for containerized applications (e.g. applications run in docker). Kubernetes currently (at the time of writing this paper) supports horizontal auto-scaling for pods, a group of containers deployed together on the same host. The auto-scaling algorithm is used to keep CPU utilization at a target level. The algorithm determines the number of pods by dividing the sum of CPU utilization in the pods by the target CPU utilization. In cloud environment, users provision cloud resource in form of of VMs. Thus, our work focuses on auto-scaling of VMs. However, our method can be used along with container cluster management for the management of web application micro services. Baresi et al. [4] enhanced their framework to be able to deploy multi-tier applications with Docker containers. The framework performs resource management with control theory to satisfy SLA (response time) and use cloud resources (CPU cores) effectively.

In web applications, the correlation of CPU utilization and response time might not be strong. The trigger should be driven by the response time since it is possible that the SLA is well satisfied while the CPU utilization triggers unnecessary scaling. The service-time-related metrics of the application are better than the metrics which only show the state of the VMs [16]. This is also why [7,8,10,11] tend to use response time to benchmark their auto-scaler.

Contrary to our approach, most of the auto-scaling studies only focus on the *Analysis* part of the MAPE loop; they do not mention much about the planning phase and how many VMs should be added or removed if the threshold is triggered (including [7,8,10,11]). The time taken to start new VM is usually around several minutes [12], so naïvely launching a single VM once at a time might not be feasible to solve the under-provisioning issue [16].

4 Fuzzy Auto-scaler

In this section, we propose our fuzzy logic-based approach for auto-scaling of web applications. First, we identify inputs and metrics that are relevant to the auto-scaling. Then, we discuss the importance of dynamically adapting the upper threshold. Finally, we propose our algorithm to set the right size for the cluster.

Table 1. Request details for Fig. 1

Request	Url	Response body size (Byte)	Request type
1	index.php?action=raw& title=Cliff_Clavin	6867	Text
2	index.php/Main_Page	13397	Text
3	index.php/Bufotenin	30930	Text
4	index.php/List_of_dog_breeds	65521	Text
5	skins/common/ajax.js?99	5320	Javascrpit

4.1 Input Selection

The most important aim of autoscaling is to maintain the response time below a certain level to provide a satisfactory experience to users. Therefore, the use of the response time as an input to the auto-scaler is highly desired. The response time alone does not provide us information regarding how many more VMs are needed to avoid SLA violations. The naïve approach is to launch a small static number of VMs to see if the response time is lowered to the desired range. However, the time of booting up and configuring the newly launched VMs is often too long that would lead to SLA violation. The long booting time of VMs forces the auto-scaler to make the best decision with a minimal number of trials.

CPU utilization can be used to determine the cluster size. However, the problem with CPU utilization is that it does not reflect the actual overall load of the VMs accurately. For example, consider a web-application like Mediawiki, which is I/O intensive rather than CPU intensive. The use of CPU is low during the I/O processes hence a delusion of the low CPU utilization happens, while the load of the VM would be high. In addition, the maximum CPU utilization of a single core VM only goes up to 100%, so it is hardly useful to determine how many more VMs are needed.

Service rate, the rate at which requests are processed, also sounds relevant to determine the cluster size. The tools like 'Jmeter'[1] or 'Ab'[2] can be used to flood the VMs to profile the maximum number of requests that can be processed by a VM. This number can be used to set up the optimal number of required VMs. However, the service rate is highly dependent on the complexity of the request's response body. Table 1 shows a group of 5 Mediawiki requests with various response body sizes and request types. Figure 1 shows that the larger response body size (Table 1), the smaller service rate (Request 1 to 4). However, the size of the response body is not the only factor which could affect the service rate. The request 5 has similar response body size to request 1 but the maximum number of processed requests per second is much higher. The reason is that Request 5 is a JavaScript file, so it needs lower computations and I/O operations to produce the response and hence has lower complexity than request 1. It is hard to define the complexity of the request since the content of the page varies greatly. Therefore, the use of service rate to determine the cluster size is not effective if the request received by the VM has a huge difference to the requests used for profiling.

[1] http://jmeter.apache.org/.
[2] https://httpd.apache.org/docs/2.4/programs/ab.html.

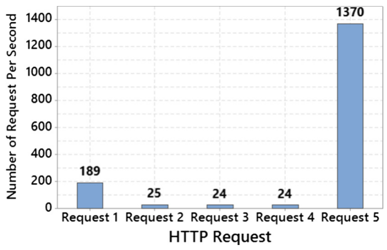


Fig. 1. Jmeter load test with different requests

4.1.1 Metrics Correlations with Response Time

We conduct a load test to see how the changes in the metrics are correlated. Wikijector is used to send requests from Wikipedia trace files to the load balancer instance. We apply a simple scaling out rule that only adds one new VM each time the response time is over the static upper threshold. According to the test, Table 2 shows Pearson and Spearman Rho correlation value of each metric to request response time. Results demonstrate that the CPU load has the highest Pearson and Spearman Rho correlation value to the response time. Thus it could be a potentially useful metric for determining the cluster size. While CPU utilization reflects the CPU usage, the CPU load metric can give us the actual demand of the computer's resources (not only CPU). Unlike CPU utilization, CPU load can go beyond 100%. The analogy between CPU utilization and load is the high way traffic. CPU utilization is how often the freeway has cars running on it, whereas CPU load is how many cars are both running on and waiting to enter the freeway [21].

Table 2. Metrics correlations with response time (P-value represents the significance of the correlation)

Metric	Pearson value	p-value	Spearman value	p-value
CPU load last minute	0.42	0.00	0.92	0.00
CPU System	0.11	0.12	0.80	0.00
CPU User	0.08	0.30	0.80	0.00
CPU IO Wait	0.26	0.00	−0.08	0.25
Memory cached	−0.24	0.00	−0.25	0.00
Number of request per second	−0.28	0.00	0.27	0.00

4.2 Dynamic Upper Threshold

The main idea behind the fuzzy rules for the dynamic upper threshold is that when the cluster size is large (e.g. 100 web-servers running), then a moderate change of the load will only fluctuate the response time in a small amount so the fuzzy engine should output a higher upper threshold of response time. Whereas, when there is only one running instance, the cluster is very likely to violate the SLA (large fluctuation) with even a small amount of request load increase. In this case, the threshold should be more sensitive (lower). Therefore, the inputs of the fuzzy engine are chosen as the current normalized *response time*, a ratio to the maximum allowed response time in SLA, and the normalized *cluster size*, a ratio to the maximum cluster size.

4.3 Dynamic Cluster Size

In an ideal situation, a fully utilized single CPU core VM has CPU load at 100%, and all CPU load beyond 100% means the VM is overloaded. Note that, 100% CPU load only means that the VM is fully loaded, but it can still provide a decent response time. In a single core computer like *t2.micro*, the 100% of the CPU load indicates that the computational resource is fully utilized and if the value is 200%, it indicates that we need one more instance to handle the load.

The scaling-in approach used in the proposed auto-scaler is conservative but dynamic. We scan through all launched VMs and stop all the VMs which have CPU load below a certain threshold (e.g., 50%) instead of pre-defining the number of VMs to scale-in. Since, a single type of instance is used in our approach, thus the total CPU load of the cluster not overloading VMs is calculated by:

$$TotalCPULoadNotOverloading = MaxNo.ofInstance \times 100\% \tag{1}$$

The cluster stress value is the ratio of the average CPU overload value (CPU load beyond 100%) to the summation of the CPU load of a cluster without overloads (equal and below 100% CPU load) and is calculated as:

$$ClusterStress = \frac{AverageCPUoverload}{TotalCPULoadNotOverloading} \tag{2}$$

For example, if we have 2 VMs and they have CPU load of 150% and 200% respectively, the average CPU overload value will be (150+200)/2 - 100% = 75%. If we only allow a maximum of 5 VMs in the cluster, then we will have total CPU load without overloading VMs of 500% (5 * 100%). Therefore, the cluster stress is equal to 75/500 = 0.15. The cluster stress represents how huge the average CPU overload is with respect to the total CPU load that the cluster can accommodate without overloading VMs.

For scaling out, we use the cluster stress value and the normalized current cluster size as inputs to the fuzzy engine. We use these inputs to calculate the needed number of VMs. Since the user's budget is often limited, a maximum number of VMs is set for the cluster. The output of the fuzzy engine, *NewClusterPower*, is the ratio of the cluster size to the maximum value. Based on this ratio, the new number of VMs is calculated as:

$$\lceil NewClusterPower \times MaxNumberOfVirtualMachines \rceil \tag{3}$$

The correlation between CPU load metric and response time is not 100%, so the average CPU overload value cannot be fully trusted to perform scaling out. Therefore, the output of the cluster size fuzzy engine for scaling out should consider the current cluster size. The fuzzy engine acts conservatively by adding less number of VMs determined from the average CPU overload value when the cluster size is small and also when the cluster size is near the maximum.

4.4 The Proposed Auto-scaling Algorithm

Algorithm 1 shows the overview of how adaptive response time threshold and dynamic cluster size work together to perform auto-scaling. Two fuzzy engines are developed: *upperThresholdFuzzy* and *clusterSizeFuzzy*. Both of them use the Mamdani fuzzy inference system and the defuzzification method of the centroid of the area to produce the crisp output value. All the membership functions for mapping input values (all of them are from 0 to 1) to fuzzy sets are Gaussian membership functions as shown in Fig. 2. Sample fuzzy rules used in our fuzzy engines are specified in Tables 3 and 4.

The algorithm first gathers the current average response time ($RTime$), CPU Loads ($curLoad$) and the cluster size ($CSize$) from the load balancer instance (Line 3). After gathering all metrics needed, the current response time and the current cluster size are used by the *upperThresholdFuzzy* engine to set the upper-threshold (Line 4). If the current response time breaches the adaptive upper threshold twice consecutively (Lines 5 and 6), the *clusterSizeFuzzy* engine is called (Line 7) to determine the new cluster size. For each scaling out or in,

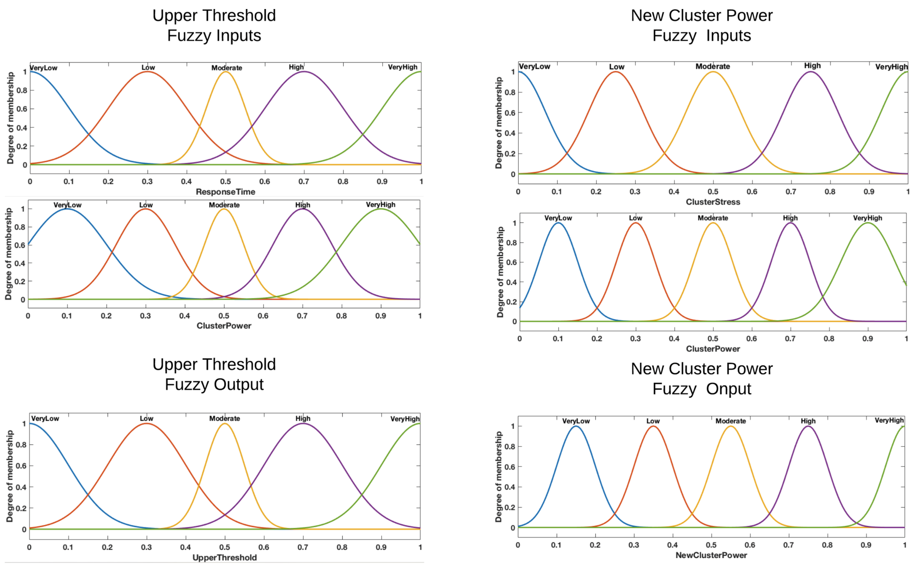


Fig. 2. Fuzzy engines for setting upper-threshold and adding new virtual machines

Table 3. Partial add virtual machines fuzzy rules

ClusterPower	ClusterStress	NewClusterPower
VeryLow	VeryLow	VeryLow
VeryLow	Low	Low
Low	VeryHigh	Moderate
Moderate	Moderate	High
High	High	VeryHigh
VeryHigh	VeryHigh	VeryHigh

Table 4. Partial upper threshold fuzzy rules

ClusterPower	ResponseTime	UpperThreshold
VeryLow	VeryLow	VeryHigh
VeryLow	Low	VeryLow
Low	Moderate	Low
Moderate	VeryHigh	VeryLow
High	High	High
VeryHigh	VeryHigh	High

the *warmUpTime* is set to a certain value (e.g., 30 s) which means we do not allow any scaling policies to be executed in this period. The reason for setting *warmUpTime* is that the newly launched VM normally has an unstable metrics due to the operating system's bootstrapping and all the configurations for the applications. For scaling in, we loop through each running VM and check if their CPU load is under *idleLoad* value, e.g. 50% (Line 17) and if the same VM CPU load is under *idleLoad* twice consecutively (Line 19), it will be stopped and the *warmUpTime* is reset. The scaling in/out loop is executed periodically, e.g. every 5 s (Line 27), that is because checking the metrics too frequently adds pressure to VMs, and often many metrics do not have high-resolution updates.

Algorithm 1. Fuzzy Auto-scaler

```
Require: idleLoad, checkTime, consecutiveOverThreshold, warmUpTime
 1: procedure StartScaling
 2:     while true do
 3:         RTime, CSize, curLoad ← AllLaunchedInstances.getStats()
 4:         $upperThreshold$ ← upperThresholdFuzzy(RTime, CSize)
 5:         if $RTime >$ upperThreshold then
 6:             if $consecutiveOverThreshold == true$ and $warmUpTime <= 0$ then
 7:                 newClusterSize ← clusterSizeFuzzy(curLoad,CSize)
 8:                 scalingUp(newClusterSize)
 9:                 Set $warmUpTime$
10:                 $consecutiveOverThreshold \leftarrow false$
11:             else
12:                 $consecutiveOverThreshold \leftarrow true$
13:         else
14:             $consecutiveOverThreshold \leftarrow false$
15:         if warmUpTime <= 0 then
16:             for Instance in AllLaunchedInstances do
17:                 if Instance.load < idleLoad then
18:                     $Instance.consecutiveBelowThreshold += 1$
19:                     if $consecutiveBelowThreshold == 2$ then
20:                         $Instance.stop()$
21:                         Set $warmUpTime$
22:             if $warmUpTime != 0$ then
23:                 setConsecutiveBelowThresholdToZero(AllLaunchedInstances)
24:         else
25:             setConsecutiveBelowThresholdToZero(AllLaunchedInstances)
26:         sleep(checkTime)
27:         $warmUpTime -= checkTime$
```

5 System Prototype

The instances used in this system are all *t2.micro* since it offers the free-tier policy (no costs of using VMs) and also keeps the system homogeneous, which lowers the system overheads for evaluating the fuzzy auto-scaler (Fig. 3).

The HAproxy instance (load balancer instance) receives all the HTTP requests from the users first; then it evenly distributes them to each Mediawiki-server instance. The HAproxy instance also hosts the master node of the Ganglia monitoring system which is used to gather all the instances' current performance metrics (e.g. CPU load). Since all the requests go through the HAproxy instance first, the HTTP request related metrics like request response time could be retrieved from the HAproxy. The fuzzy auto-scaler communicates to the HAproxy instance to gather metrics of all the AWS t2.micro instances and processes them locally to decide on whether to perform scaling process and to determine the suitable size of the current Mediawiki-server cluster needed to satisfy the SLA. The orchestration of the Mediawiki web-application cluster is controlled by the fuzzy auto-scaler instance which uses Python Boto3 (AWS SDK for Python [19]) library to manipulate the AWS cloud resources. The Fuzzy auto-scaler instance starts or stops VMs based on the metrics retrieved from the HAproxy instance. The automation process of configuring different instances are done with Ansible [6] tool which is also executed in a Fuzzy auto-scaler instance.

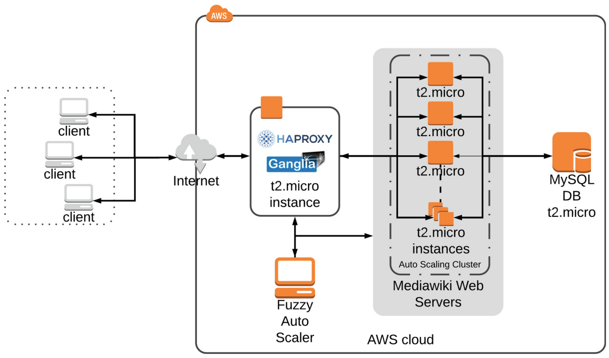


Fig. 3. System architecture diagram

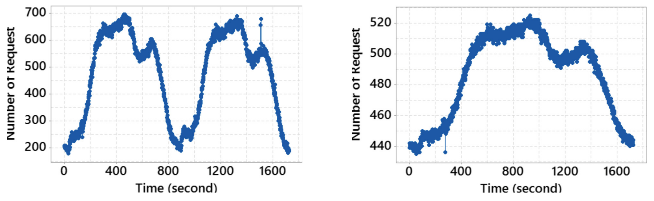
6 Performance Evaluation

6.1 Experimental Setup

Each experiment is carried out for 30 min and is repeated for three times. We report the best result obtained by each auto-scaler. The AWS t2.micro instances used for the experiments have only 30 CPU credits with 1 point per minute for the full power CPU usage. Once the CPU credits are exhausted the VM will be restricted to a baseline (10% CPU utilization) [20]. The trace files used in the experiments to evaluate the auto-scaler are all produced from the Wikipedia workload trace file. The 'shape' of the original trace is preserved while the time dimension is scaled down to 30 min.

The AWS emulated auto-scaler is a copied version of the AWS auto-scaling policy which uses Ganglia instead of AWS CloudWatch for monitoring. The reason for adding it is that the AWS native auto-scaler does not offer fine grain control and monitoring data for validating scaling policies.

In experiments, we have a maximum limit of 10 t2.micro instances, and the desired response time is set to 200ms for the SLA. For the AWS native and emulated auto-scalers, the upper threshold is 100 ms (static), and the fuzzy auto-scaler uses the adaptive upper-threshold from the fuzzy engine. Since AWS native auto-scaler cannot read individual VM's metric to stop a VM, we set a lower threshold of 70% CPU load (average of all launched VMs) for both AWS native and emulated auto-scalers, while the fuzzy auto-scaler has a lower threshold of 50% for each VM. We also deliberately use a cluster size of two VMs at the beginning of the experiment to show the differences of each auto-scalers. Thus, there will be many SLA violations at the beginning.



(a) Two full day trace of Wikipedia (b) The one day trace of Wikipedia

Fig. 4. The trace files used in the experiments, both of them are 30 min long

6.2 Experimental Results

The trace file used in experiments (Fig. 4a) is extracted from the two day Wikipedia trace and is scaled down into 30 min. Figure 5 shows the CDF (Cumulative Distribution Function) of the average response time below 200ms for all auto-scalers. The fuzzy auto-scaler preforms about **8.94%** better than the AWS emulated auto-scaler (adding one VM each scaling out) and **16.28%** better than the AWS native auto-scaler. The cost of each auto-scaler can be calculated from the area under the step-curve in 'Number of Instance vs Time' scatter graph. The cost of the AWS native one is the lowest since it often under-provisions resources. The cost of the fuzzy auto-scaler is similar to the AWS emulated auto-scaler, but the fuzzy auto-scaler provides better results in terms of keeping the response time under 200 ms.

The number of instances versus time scatter graphs show that the fuzzy auto-scaler performs a lower number of scaling out actions to reach the desired cluster size, whereas AWS native auto-scaler and emulated auto-scaler with adding one

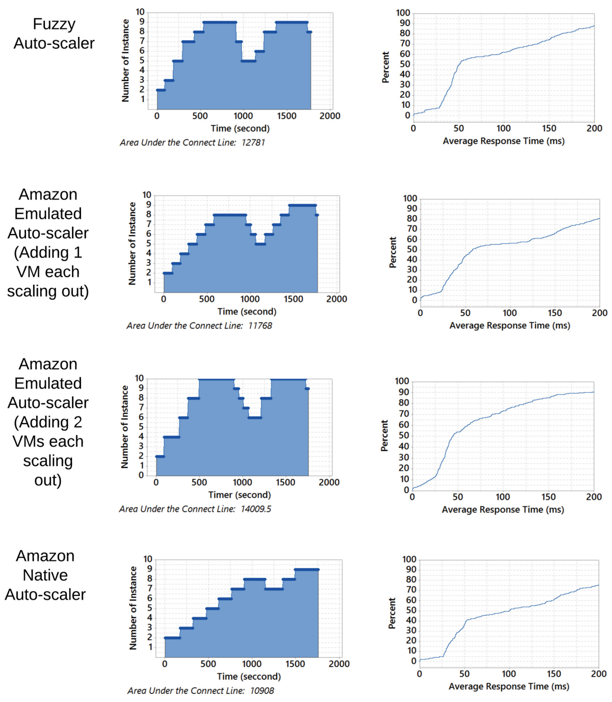


Fig. 5. Resource usage and average response time for different auto-scalers

VM require more time to do the same. The amazon emulated auto-scaler with adding two VMs options is the fastest in this regards. However, it causes the unnecessary cost to provide the same performance. Therefore, adding multiple VMs during the scaling-out is very important to process the web request load. However, the lack of prediction ability of the fuzzy auto-scaler leads in several trials to gradually reach the desired cluster size. The initial cluster size is small, so it can quickly get flooded with requests and metrics raise up to the maximum. Hence using metric values is not reliable to determine the actual request load. Thus, the method of measuring the current request load needs further investigations.

Table 5 shows results summary. Compared to AWS native auto-scaler, AWS emulated auto-scalers with adding one and two VM(s) each scaling out and fuzzy auto-scaler generate 7.88%, 28.43% and 17.17% higher cost respectively. However, with 1% increase in SLA satisfaction, the fuzzy auto-scaler only requires 1.05% (17.17/16.28) increase in cost while AWS emulated auto-sacalers with adding one and two VM(s) each scaling out need 1.17% (7.88/6.74) and 1.46% (28.43/19.53) increase in cloud resource usage respectively. The fuzzy auto-scaler use resources more efficiently to provide better SLA satisfaction. The AWS native auto-scaler and AWS emulated auto-scaler adding one VM each scaling out have overall lower cost than fuzzy auto-scaler. However, they cause more SLA violations by under-provisioning resources.

Table 5. Experimental results summary

Auto-scaler	% of requests with response time under 200 ms (Improvement to AWS native)	The area under the graph or cost (Improvement to AWS native)
Fuzzy auto scaler	87.97 (16.28%)	12781.00 (17.17%)
AWS emulated (adding 1 VM)	80.75 (6.74%)	11768.00 (7.88%)
AWS emulated (adding 2 VMs)	90.43 (19.53%)	14009.50 (28.43%)
AWS native	75.65 (0%)	10908.00 (0%)

6.3 Dynamic Scaling-Out Evaluation

Figure 5 shows the downside of using a static number to perform scaling out. The AWS emulated auto-scaler (adding 2 VMs each scaling out) reached the maximum number of 10 VMs whereas in the same experiment only the maximum of 9 VMs used by fuzzy auto-scaler without violating the upper threshold. This happens due to the lack of ability to dynamically adjusting the added number of VMs in the cluster size. Even though AWS emulated auto-scaler (adding 2 VMs each scaling out) has a better result of not violating the SLA (90.43%), but it also uses the resource less efficiently compared to the fuzzy auto-scaler in order to reach this result. With respect to the AWS native auto-scaler, AWS emulated auto-scaler (adding 2 VMs each scaling out) needs **1.46%** more resources to achieve 1% increase of the response time under 200 ms while fuzzy auto-scaler only needs **1.05%**.

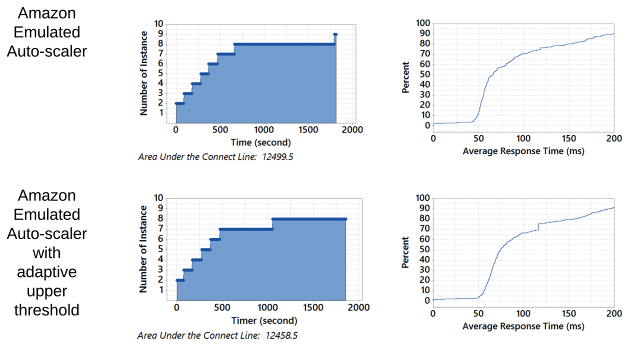


Fig. 6. The adaptive upper threshold experiment result

6.4 Adaptive Upper-Threshold Evaluation

In order to evaluate the impact of an adaptive upper threshold, we use Amazon emulated Auto-scaler with and without the adaptive upper threshold fuzzy engine. The static threshold is set to 100 ms for the one without the fuzzy engine. The change in the request rate of the trace shown in Fig. 4a is considerably quick which decreases the chance of observation of adaptive upper-threshold benefits. Thus, a 30 min version of the one day trace file is made (Fig. 4b).

As shown in Fig. 6, the adaptive upper threshold keeps the cluster at the size of 7 longer than non-adaptive threshold AWS emulated auto-scaler that leads to **0.33%** cost saving. The percentage of response time under 200 ms is close in this case while AWS emulated with adaptive threshold has **91.74%**, and the non-adaptive upper threshold one has **90.14%** which shows **1.78%** improvement. Due to the restriction of the experiment time, the cost saving is not clearly observable, but we expect that the dynamic adaptive upper threshold utilizes cloud resources more efficiently in a long run since small load fluctuation is typical in a long-running web application.

It is worth mentioning that the load prediction can enhance the number of trials to reach the desired number of virtual machines quicker when encountering the seasonal loads. The experiment results also shows that the response time is decreased while adding more Mediawiki instances so that in our experiments the single MySql database instance in not a bottleneck for the Mediawiki cluster performance.

7 Conclusions and Future Work

In summary, a dynamic upper-threshold auto-scaler for the web application is proposed in this paper. We designed and presented two fuzzy engines to dynamically adjust upper threshold and cluster size respectively. The fuzzy logic approach increases the percentage of the service rate within the SLA target by 16.28% in comparison to AWS native auto-scaler. The experimental results suggest that threshold and the added number of VMs should be dynamically selected since the static values are not always right choices for all cluster sizes.

The creation of the optimal auto-scaler is far from trivial. In the future, we will explore techniques to determine the best candidate VMs for scaling-in. Scaling-in is riskier than scaling-out, as over-provisioning only costs money, but under-provisioning means losing customers. The auto-scaling of session-based web applications, where users have sticky sessions needs more attention since it requires session migration. It will be interesting to perform a more in-depth investigation of the correlation between proposed metrics which allows us to design more efficient auto-scalers. In the future, we will also try to implement different auto-scaling methods and benchmark them against our fuzzy auto-scaler with different workloads.

References

1. Arabnejad, H., et al.: A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: Proceedings of 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 64–73. IEEE (2017)
2. Armbrust, M., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)

3. van Baaren, E.-J.: Wikibench: a distributed, Wikipedia based web application benchmark. Master's thesis, VU University Amsterdam (2009)
4. Baresi, L., et al.: A discrete-time feedback controller for containerized cloud applications. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 217–228. ACM (2016)
5. Buyya, R., et al.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
6. DeHaan, M.: Ansible (2018). https://www.ansible.com/. Accessed 28 Feb 2018
7. Frey, S., et al.: Cloud QoS scaling by fuzzy logic. In: Proceedings of 2014 IEEE International Conference on Cloud Engineering (IC2E), pp. 343–348. IEEE (2014)
8. Hasan, M.Z., et al.: Integrated and autonomic cloud resource scaling. In: Proceedings of 2012 IEEE Network Operations and Management Symposium (NOMS). IEEE. 2012, pp. 1327–1334
9. Jamshidi, P., Pahl, C., Mendonça, N.C.: Managing uncertainty in autonomic cloud elasticity controllers. IEEE Cloud Comput. **3**, 50–60 (2016)
10. Lama, P., Zhou, X.: Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In: Proceedings of 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 151–160. IEEE (2010)
11. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: Proceedings of the 7th International Conference on Autonomic Computing, pp. 1–10. ACM (2010)
12. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
13. Lorido-Botrán, T., Miguel-Alonso, J., Lozano, J.A.: Autoscaling techniques for elastic applications in cloud environments. Technical report EHU-KAT-IK-09-12. Department of Computer Architecture and Technology, University of Basque Country (2012)
14. Massie, M., et al.: Monitoring with Ganglia: Tracking Dynamic Host and Application Metrics at Scale. O'Reilly Media, Inc., Sebastopol (2012)
15. Müller, C., et al.: An elasticity-aware governance platform for cloud service delivery. In: Proceedings of 2016 IEEE International Conference on Services Computing (SCC), pp. 74–81. IEEE (2016)
16. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling web applications in clouds: a taxonomy and survey. ACM Comput. Surv. **51**(4), 73:1–73:33 (2018)
17. Amazon Web Services. Amazon Web Services (AWS) - Cloud Computing Services (2018). https://aws.amazon.com/. Accessed 28 Feb 2018
18. Amazon Web Services. AWS Auto Scaling (2018). https://aws.amazon.com/autoscaling/. Accessed 28 Feb 2018
19. Amazon Web Services. AWS SDK for Python (Boto3) (2018). https://aws.amazon.com/sdk-for-python/. Accessed 28 Feb 2018
20. AmazonWeb Services. CPU Credits and Baseline Performance (2018). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-credits-baseline-concepts.html. Accessed 28 Feb 2018
21. Walker, R.: Examining load average. Linux J. **2006**(152), 5 (2006)
22. Yazdanov, L.: Towards auto-scaling in the cloud: online resource allocation techniques. Ph.D. thesis. Dissertation, Technische Universität Dresden, Dresden (2016)

Runtime Monitoring in Continuous Deployment by Differencing Execution Behavior Model

Monika Gupta[1(✉)], Atri Mandal[3], Gargi Dasgupta[3], and Alexander Serebrenik[2]

[1] IBM Research, New Delhi, India
gupmonik@in.ibm.com
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
a.serebrenik@tue.nl
[3] IBM Research, Bengaluru, India
{atri.mandal,gaargidasgupta}@in.ibm.com

Abstract. Continuous deployment techniques support rapid deployment of new software versions. Usually a new version is deployed on a limited scale, its behavior is monitored and compared against the previously deployed version and either the deployment of the new version is broadened, or one reverts to the previous version. The existing monitoring approaches, however, do not capture the differences in the execution behavior between the new and the previously deployed versions.

We propose an approach to automatically discover execution behavior models for the deployed and the new version using the execution logs. Differences between the two models are identified and enriched such that spurious differences, e.g., due to logging statement modifications, are mitigated. The remaining differences are visualized as cohesive diff regions within the discovered behavior model, allowing one to effectively analyze them for, e.g., anomaly detection and release decision making.

To evaluate the proposed approach, we conducted case study on Nutch, an open source application, and an industrial application. We discovered the execution behavior models for the two versions of applications and identified the diff regions between them. By analyzing the regions, we detected bugs introduced in the new versions of these applications. The bugs have been reported and later fixed by the developers, thus, confirming the effectiveness of our approach.

Keywords: Continuous deployment · DevOps · Execution logs
Runtime flow graph · Release decision · Visualization

1 Introduction

Increasing speed of the changing priorities of customers causes many companies to adopt continuous deployment [1,8,15,23]. A continuous deployment model is crucial for service delivery business as it ensures that software services are always in a releasable state, and changes are incremental. To ensure high quality release in continuous deployment, the upcoming release is staged in production environment using such strategies as blue-green deployment [13], dark launches [9], canary release [13,26] and shadow testing [23], and its performance is monitored [3,13] to quickly identify whether it is misbehaving [23,26].

Vast amount of data is logged during the execution of the new and previously deployed software versions. Existing monitoring systems keep track of suspicious events in logs (e.g., errors, warning messages, stack traces) and raise alerts. However, such systems do not leverage the unstructured data captured in the execution logs to efficiently derive and compare the dynamic behavior of the new and the previously deployed versions in a holistic manner.

In this work, we present a novel approach to automatically detect discrepancies in the fast evolving applications adopting continuous deployment. This is achieved by identifying the differences in the behavior model of the previously deployed and new version, derived by mining the execution logs.

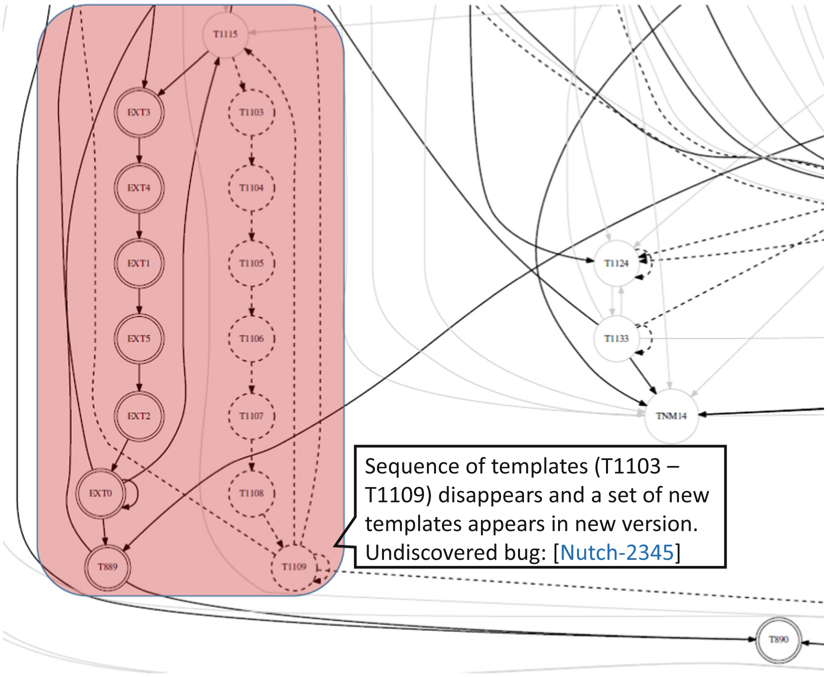

T1115
EXT3
EXT4
EXT1
EXT5
EXT2
EXT0
T889
T1103
T1104
T1105
T1106
T1107
T1108
T1109
T1124
T1133
TNM14
T890
Sequence of templates (T1103 – T1109) disappears and a set of new templates appears in new version. Undiscovered bug: [Nutch-2345]

Fig. 1. Differences between the execution behavior models of two versions of Nutch. Vertices added in the new version are encircled twice, added edges–bold, deleted vertices and edges–dashed. The analysis of differences allowed us to discover a bug that we reported as NUTCH-2345. The bug was fixed by the Nutch developers.

2 Motivating Example: A Bug in Nutch

As part of an issue *[NUTCH-1934]*[1] the class *Fetcher* counting ca. 1600 lines of code is refactored to improve modularity. We took the version before and after refactoring to identify differences between the two versions. We used Nutch to crawl a set of URLs thus generating the execution logs for both the versions. We map the generated execution logs to templates derived from the Nutch source code using string matching. A subset of log lines was not mapped to any source code template (that is, from third party library) and clustered using a combination of approximate and weighted edit distance clustering. Execution behavior model is discovered automatically for each of the versions using the respective templatized execution logs. Each vertex in the model corresponds to a unique template. Using our automated approach, many diff regions are detected between the two discovered models.

Figure 1 presents one of the diff regions, i.e., deletion of a set of vertices $T1103$–$T1109$ (represented as dashed) from class *Fetcher.java* and addition of new vertices $EXT0$–$EXT5$ (double circled) from apparently third party library (prefixed with EXT). We manually investigated this diff region and found that the code fragment corresponding to templates $T1103$–$T1109$ has been moved from *Fetcher.java* to *FetchItemQueue.java*[2]. Inspecting *FetchItemQueue.java* we found that *FetchItemQueue**s*** is used as logger instead of *FetchItemQueue*. Consequently, the log messages from *FetchItemQueue* had a wrong class name, and thus were not mapped to the corresponding source code logging statement and treated as log statements from third party library ($EXT0$–$EXT5$).

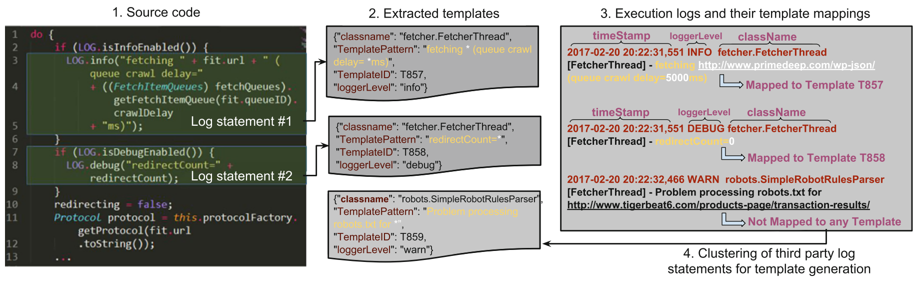
This issue was introduced in Nutch 1.11 and fixed after we reported it[3] in Nutch 1.13. Using our approach, the issue would have been detected in the version 1.11 itself. This highlights the potential of our approach for discovering anomalies by analyzing automatically identified diff regions.

3 Proposed Approach

The proposed approach takes executions logs and source code for the deployed and new version as starting points. Since our approach is targeted towards continuous deployment, access to both these artifacts can be assumed. The approach leverages execution logs without instrumenting the code because instrumentation overhead is not possible in a fast evolving production software [31]. Nevertheless, execution paths are successfully captured from the existing logs because in practice, sufficient logging is done to facilitate runtime monitoring [6,16].

Our approach consists of three broad phases: *template mining* that maps each line in execution logs to a unique template (Sect. 3.1), *execution behavior model mining* that derives execution behavior models from the templatized logs and refines the model using multimodal approach (Sect. 3.2), and *analysis of*

[1] https://issues.apache.org/jira/browse/NUTCH-1934.
[2] http://svn.apache.org/viewvc?view=revision&revision=1678281.
[3] https://issues.apache.org/jira/browse/NUTCH-2345.



```
do {
    if (LOG.isInfoEnabled()) {
      LOG.info("fetching " + fit.url + " (
          queue crawl delay="
          + ((FetchItemQueues) fetchQueues).
            getFetchItemQueue(fit.queueID).
            crawlDelay
          + "ms)");
    }
    if (LOG.isDebugEnabled()) {
      LOG.debug("redirectCount=" +
          redirectCount);
    }
    redirecting = false;
    Protocol protocol = this.protocolFactory.
        getProtocol(fit.url
        .toString());
    ...
```

```
{"classname": "fetcher.FetcherThread",
"TemplatePattern": "fetching * (queue crawl delay= *ms)",
"TemplateID": T857,
"loggerLevel": "info"}

{"classname": "fetcher.FetcherThread",
"TemplatePattern": "redirectCount=*",
"TemplateID": T858,
"loggerLevel": "debug"}

{"classname": "robots.SimpleRobotRulesParser",
"TemplatePattern": "Problem processing robots.txt for *",
"TemplateID": T859,
"loggerLevel": "warn"}
```

```
2017-02-20 20:22:31,551 INFO  fetcher.FetcherThread
[FetcherThread] - fetching http://www.primedeep.com/wp-json/
(queue crawl delay=5000ms)

2017-02-20 20:22:31,551 DEBUG fetcher.FetcherThread
[FetcherThread] - redirectCount=0

2017-02-20 20:22:32,466 WARN  robots.SimpleRobotRulesParser
[FetcherThread] - Problem processing robots.txt for
http://www.tigerbeat6.com/products-page/transaction-results/
```

Fig. 2. Given the source code (1) templates are extracted (2), and log lines are mapped to them (3). Log lines from external libraries are clustered to create new templates (4).

the model differences to identify the differences between the execution behavior models and classify them into cohesive diff regions (Sect. 3.3).

3.1 Template Mining

A template is an abstraction of a logging statement in the source code consisting of a fixed part and variable part (denoting parameters) [2,18]. Due to presence of parameters templates often manifest themselves as different log messages. Thus, identifying the templates from the execution log messages has inherent challenges [20]. If no source code is available, templates can be inferred by clustering log messages [20,27]. However, often log messages from different logging statements are clustered together, resulting in inaccurate templates. Since we have access to source code, we extract templates using regular expressions (cf. Fig. 2).

Derive Templates from the Source Code: In this step the print statements are identified from the source code along with the class name and severity level (e.g., INFO, WARN and DEBUG) [6]. We search for the logging statements in the source code using regular expressions with some enhancements to identify ternary print statements and ignore commented logging statements in the source code. As shown in Fig. 2, logging statement is parsed and represented as a regular expression which is then assigned a unique template id. Class name and severity level are also stored as additional information to disambiguate templates which have identical invariant pattern but appear in different classes of the code.

While the complete source code is used to extract templates for the deployed source code version, to extract the templates for the new version we only analyze the *diff* between the two source code revisions as indeed, continuous deployment encourages incremental changes. Not only is the extraction more efficient, this also ensures that the unchanged templates between the two versions are represented by the same template ID. The main shortcoming of *diff* is that if a logging statement is modified, it is represented in the *diff* as a combination of addition and deletion, that will be interpreted as addition of a new template and deletion of the old template. Thus, two execution behavior models will appear different for the templates which are actually the same. Since modification of logging statement is frequent [6,16], we address this shortcoming using a novel multimodal approach for template merging and model refinement (Sect. 3.2).

Templatize Log Messages: In this step, template id is assigned to each log line appearing in the execution logs, by matching with templates obtained from the previous step. To reduce the search space for the match, class name and severity level (if included as part of the log messages) are used as additional matching parameters (cf. Fig. 2). While regular expression matching can find the matching template, log lines matching multiple templates, templates with no fixed part and log lines generated by the third party libraries require special treatment. If a *log line matches more than one template* it is mapped to the most specific template, i.e., the template with the largest fixed part. If there is *one logging statement in a class without constant part* then all the unmapped log lines from that class with same logging level are mapped to it[4]. Finally, *log lines from external sources* such as third party libraries for which we do not have access to source code cannot be templatized as explained above. These log lines are clustered using a combination of approximate clustering [20] and weighted edit distance similarity [10]. Each cluster generated after the refinement is represented as a template and is assigned a unique template ID. Thereafter, non-templatized log lines are matched with the templates derived from clustering step such that all the log lines are assigned a unique template id.

3.2 Mining Execution Behavior Model Using Multimodal Approach

Execution Behavior Model (EBM) is a graphical representation of the templatized execution logs capturing the relationship between the templates. Each vertex in the model corresponds to a unique template and the edges represent the flow relationship between the templates. Since template represents a logging statement from the code, EBM captures a subset of possible code flows.

Accuracy of identified diff regions directly depends on the accuracy of the EBM mining which in turn depends on the accuracy of the template mining. As discussed in Sect. 3.1, the execution logs are templatized with high precision using source code. However, for log lines being generated from third party libraries we had to resort to the clustering based technique which has inherent limitations. This limits the template mining accuracy and consequently the accuracy of EBM mining. This is even more apparent in the new version because only a limited amount of logs is available, which is a hindrance to accurate mining [20]. Further, inconsistency in the templates because of the modified log statements in source code being recorded as new templates leads to many spurious differences between the compared models thus, making the *diff* analysis practically less effective. To overcome this problem, we propose an iterative EBM refinement

[4] The case with multiple such statements is very rare and hence does not affect our approach.

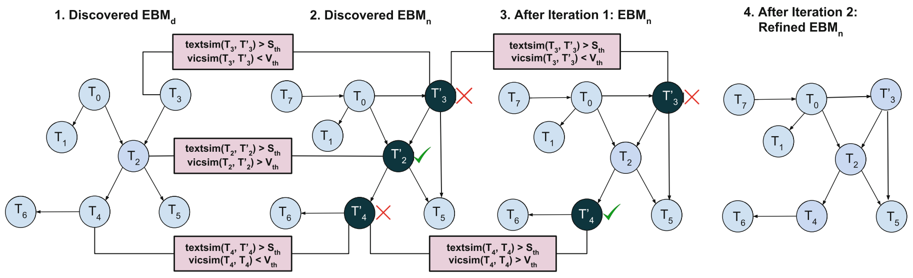

1. Discovered EBM$_d$
2. Discovered EBM$_n$
3. After Iteration 1: EBM$_n$
4. After Iteration 2: Refined EBM$_n$
textsim(T$_3$, T'$_3$) > S$_{th}$ vicsim(T$_3$, T'$_3$) < V$_{th}$
textsim(T$_2$, T'$_2$) > S$_{th}$ vicsim(T$_2$, T'$_2$) > V$_{th}$
textsim(T$_4$, T'$_4$) > S$_{th}$ vicsim(T$_4$, T$_4$) < V$_{th}$
textsim(T$_4$, T$_4$) > S$_{th}$ vicsim(T$_4$, T$_4$) > V$_{th}$
T$_0$ T$_1$ T$_2$ T$_3$ T$_4$ T$_5$ T$_6$ T$_7$ T'$_2$ T'$_3$ T'$_4$

Fig. 3. Iterative multimodal execution behavior model refinement

strategy using multimodal signals that is, *text* and *vicinity* (i.e. predecessors and successors in EBM) of the template.

Iterative Execution Behavior Model Refinement: We derive execution behavior model for the deployed, EBM_d and the new version, EBM_n using corresponding templatized execution logs. We compare EBM_d and EBM_n to identify the vertices which are present in EBM_n but not in EBM_d (that is, $\triangle T_{add}$) and vice-versa, i.e. $\triangle T_{del}$. It is possible that the vertex from $\triangle T_{add}$ set is actually same as the vertex from set $\triangle T_{del}$ but captured as different template as discussed above. We identify and resolve such cases using proposed multimodal approach thus, reducing the spurious *diff* and making the comparison more effective.

One of the multimodal signals that we use is *textual similarity* between the templates from $\triangle T_{del}$ and $\triangle T_{add}$. If there are m templates in $\triangle T_{del}$ and n templates in $\triangle T_{add}$ then similarity is calculated between $m \times n$ pairs. The pairs with textual similarity above a threshold are captured as *potential merge candidates*. We do not merge the templates simply based on text similarity because there can be two textually similar templates corresponding to different logging statements in the code. Hence, to improve the precision, we evaluate the similarity for one more modality, i.e., *vicinity similarity*, where vicinity is the set of predecessors and successors. If the vicinity similarity is above a threshold, the templates are marked as identical. Thresholds for textual similarity and vicinity similarity can be selected based on grid search and fine tuned to project requirements [20].

We continue the process iteratively with each step leading to a more refined EBM_n. With every iteration some of the vertices are marked as identical which in turn can change the value of vicinity similarity for other candidate pairs. We stop the iterations when no more candidate pairs can be merged and the EBM_n output of subsequent steps no longer changes.

Example 1. Consider the EBMs shown in Fig. 3. By comparing EBM_d and EBM_n, we observe that $\triangle T_{del} = \{T_2, T_3, T_4\}$ and $\triangle T_{add} = \{T_2', T_3', T_4'\}$. We calculate *text similarity* for all the nine pairs and find the *potentially similar candidate set*, $C = \{(T_2, T_2'), (T_3, T_3'), (T_4, T_4')\}$. *Vicinity similarity* is checked for all the candidates and in first iteration vicinity similarity is above the threshold only for one pair, (T_2, T_2') which is marked as identical and removed from C. In the next iteration, the remaining pairs from C are analyzed for the vicinity similarity which is found to be greater than the threshold for (T_4, T_4'), which is again marked as identical and removed from C. Only one pair, (T_3, T_3') is not marked as same because its vicinity similarity is below threshold even though the textual similarity is high. Consequently, *diff* set after EBM_n refinement is reduced to $\triangle T_{add} = \{T_3'\}$ and $\triangle T_{del} = \{T_3\}$.

3.3 Analyzing Differences Between Execution Behavior Models

Since EBMs are graphs identifying the differences between them can be seen as the graph isomorphism problem [14], known to be in NP. However, since we ensure the consistency in the template ID across the two models, the comparison of two models is simplified. The refined models are compared to identify the following differences: sets of vertices, $\triangle \mathit{diff}_v$ and edges, $\triangle \mathit{diff}_e$ which are added/deleted, as well as the set of vertices for which the relative frequency of outgoing transitions has changed $\triangle \mathit{diff}_{dist}$ in EBM_n when compared to EBM_d. For efficient follow-up analysis, we group the identified differences into cohesive regions such that the related differences are investigated as single unit.

Example 2. Deletion of T1103–T1109 and the corresponding edges, and addition of EXT0–EXT5 and the corresponding edges in Fig. 1 are grouped together.

Vertex Anchored Region: Intuitively, we would like to find the maximum point from which the difference in execution behaviors is observed and the minimum point up to which there are differences in the execution behavior. For the differences with same maximum point, it is highly likely that they are caused due to modification in same code, and, thus, should be investigated as a single unit.

A vertex, v_i is randomly selected from $\triangle \mathit{diff}_v$ as a seed to detect the region. We back traverse the graph till an unchanged ancestor (that is, vertex common between the two models) is detected along all the paths to v_i. All the vertices and edges along the path (including unchanged ancestor) are marked as part of the region. For all marked vertices, all the outgoing branches are traversed and marked till an unchanged child vertex (that is, vertex common between the two models) is detected. Unchanged child vertex is not included in the region because the boundary of region is defined till the last difference in the included path. Effectively, a region covering a set of vertices and edges is identified. The process repeats as long as there remain unmarked vertices in $\triangle \mathit{diff}_v$. At the end of this step, all vertices from $\triangle \mathit{diff}_v$ and *some* edges from $\triangle \mathit{diff}_e$ are marked as part of some region. We call these regions as *vertex anchored regions.*

Example 3. Consider Fig. 4 where $\triangle \mathit{diff}_v = \{T_0, T_2, T_3, T_4, T_6, T_7, T_8, T_{10}\}$ and $\triangle \mathit{diff}_e = \{(T_0, T_{11}), (T_1, T_0), (T_1, T_4), (T_1, T_3), (T_1, T_2), (T_1, T_6), (T_4, T_5), (T_3, T_5),$ $(T_2, T_5), (T_{11}, T_6), (T_6, T_7), (T_7, T_{11}), (T_{10}, T_{11}), (T_{10}, T_9), (T_9, T_{10}), (T_9, T_8), (T_8, T_9),$

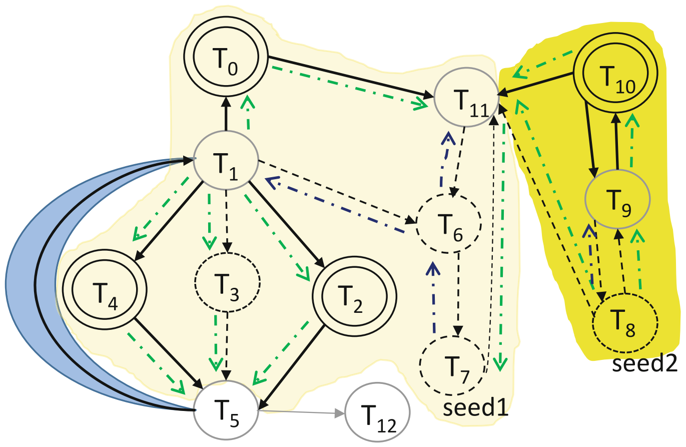


Fig. 4. There are two vertex anchored region (different shades of yellow) and one edge between the unchanged vertices (blue). Blue pointers correspond to backtracking, and green pointers depict forward tracking. (Color figure online)

$(T_8, T_{11}), (T_5, T_1)\}$. We choose T_7 as the first seed and back traverse its incoming path (blue pointers) up to maximum unchanged vertices, i.e., $\{T_1,\ T_{11}\}$, marking vertices $\{T_7, T_6, T_1, T_{11}\}$ and edges $\{(T_7, T_6), (T_6, T_1), (T_6, T_{11})\}$. Next, the outgoing branches are traversed (green pointers) till unchanged vertex is detected and the corresponding vertices are marked. As a result, the light yellow region is created consisting of $V_{r1} = \{T_0, T_2, T_3, T_4, T_6, T_7\}$ and $E_{r1} = \{(T_0, T_{11}), (T_1, T_0), (T_1, T_4), (T_1, T_3), (T_1, T_2), (T_1, T_6), (T_4, T_5), (T_3, T_5), (T_2, T_5), (T_{11}, T_6), (T_6, T_7), (T_7, T_{11}), (T_5, T_1)\}$ from $diff_v$ and $diff_e$ respectively. For the next iteration we choose T_8 as a seed from the set of uncovered vertices in $diff_v$ and repeat the process to identify another region. The second region becomes $V_{r2} = \{T_8, T_9, T_{10}\}$ and $E_{r2} = \{(T_{10}, T_{11}), (T_{10}, T_9), (T_9, T_{10}), (T_9, T_8), (T_8, T_9), (T_8, T_{11})\}$. Hence, all the vertices from $diff_v$ and a subset of $diff_e$ are grouped in one of the cohesive regions.

Edge Anchored Region: Not all edges from $\triangle diff_e$ belong to one of the vertex anchored regions. These are mainly the edges added/deleted between unchanged vertices and should be analyzed separately. We refer to each of these edges along with its vertices as an *edge anchored region.*

Example 4. After detecting the vertex anchored regions in Fig. 4 only one edge in $\triangle diff_e$ is unmarked. The only edge anchored region is hence $V_{r3} = \{T_1, T_5\}$ and $E_{r3} = \{(T_5, T_1)\}$.

Distribution Anchored Region: Apart from the above two cases of structural changes (addition or deletion of vertex or edge) in execution behavior model, we investigate the vertices common in both the versions of the model to detect the deviations in changes in the relative frequency of outgoing transitions. To capture the distribution change, for a given vertex v and its outgoing transitions common between the two models we compute $\frac{|f_d(i)-f_n(i)|}{f_d(i)}$, where $f_d(i)$ ($f_n(i)$) is a relative frequency of transition i in EBM_d (EBM_n) among the outgoing transitions of v common between the two models. If the metric value is above threshold for at least one transition from the vertex v, it is marked as *distribution anchored region*. Threshold needs to be decided manually based on the project requirements such that minor changes are discounted (that is, not considered as part of the differences) and major changes are marked in the differences.

4 Evaluation: Open Source and Proprietary Applications

We evaluated our approach on two different applications: (i) Nutch[5], an open source web crawler, and (ii) an industrial log analytics application. We have already shown some initial results on the Nutch project in Sect. 2 and discuss the other findings here. Also all the artifacts such as execution logs, templatized logs, execution behavior model and diff files are made publicly available for reproducibility of the results[6]. Details of the industrial application cannot be divulged for confidentiality reasons. We selected these applications primarily because of the availability of the source code and historical data on bugs and the corresponding fixes, as well as frequently occurring incremental changes in these applications. Execution logs for these projects were not available so we use a custom load-generator to generate logs for different source code versions.

Table 1. Properties for the two versions of Nutch application

Attribute	Nutch	
	Ver 1	Ver 2
Classes	415	420
Total LOC	67658	67891
Logging statements in src	1098	1097
Total lines in execution log (approx)	94137	125695
Total [Info, Debug]	19K,73K	26K,98K
Total [Error, Warn]	408,178	354,604
Vertices in model	106	104
Edges in model	328	310

[5] http://nutch.apache.org/.
[6] https://github.com/Mining-multiple-repos-data/Nutch-results.

4.1 Experimental Results for Nutch

Two Nutch versions were used: (i) before the commit for [NUTCH-1934], henceforth called version 1 (deployed/prod version) and (ii) after the commit for [NUTCH-1934], henceforth called version 2 (new version). This commit is considered a major change as a big class, *Fetcher.java* (ca. 1600 lines of code) is refactored into *six* classes. Table 1 presents details for two Nutch versions. We derive the templates from the source code for version 1, henceforth called $templates_{v1}$. To derive the templates for version 2, $templates_{v2}$, 46 templates are deleted, and 47 templates are added to $templates_{v1}$ in accordance with the code *diff* (here, *git-diff*) between the two code versions. We generate the execution logs for both the versions by crawling same URLs (that is, mimic prod) and observe that number of loglines generated for version 1 are less than that for version 2 (cf. Table 1). The execution logs for version 1 and version 2 are templatized using $templates_{v1}$ and $templates_{v2}$, respectively. Around 12% log lines are not templatized, and hence are clustered. 80 clusters are obtained. The clusters are further refined and grouped using weighted edit distance reducing their number to 26. Non-templatized log lines are matched with the templates generated after clustering and every line in the execution log is templatized for both the versions. We discover the execution behavior model (EBM) for both the versions, EBM_1 and EBM_2 and refine them using multimodal approach.

The behavior model shows that there are 53 added and 47 deleted vertices in EBM_2 as compared to EBM_1. For every pair of the added and deleted vertices, text similarity is calculated from the source code. The *text similarity* is found to be above a threshold (here, 0.8) for 36 out of 2491 pairs and the corresponding vicinity is compared in EBM_1 and EBM_2. *Vicinity similarity* is also found to be above a threshold (that is, 0.5) for all the 36 candidate pairs. Thus, these vertices are marked to be the same templates across the two EBMs. For better understanding, diff refinements file is made publicly available at the *link* (See footnote 6). As a result, all the templates which are captured as new template because of refactoring got mapped to the corresponding old templates reducing the number of differences significantly. Refined EBM_2 is compared with EBM_1 to identify and analyze the differences. The final refined model with diffs is made publicly available (See footnote 5).

We observe several differences which are grouped as cohesive regions using approach discussed in Sect. 3.3.

We identified one region which is explained in Sect. 2. In the additional region we observe (i) deletion of vertex corresponding to template "Using queue mode : byHost" (though present in source code of both the versions), and (ii) significant change in distribution of a vertex $T1135$ such that the edge $T1135 \rightarrow T1131$ traversed only twice in EBM_1 has been traversed 601 times in EBM_2. Both observations are related to *FetcherThread.java* which is investigated manually and a bug is identified in the way URLs are redirected. Instead of following the correct redirect link, the code was following the same link over and over again. After the maximum number of retries is exceeded further processing of the URL stopped with the message $T1131$ ("- redirect count exceeded *"), thus, increasing frequency of this edge traversal. This bug has already been reported as

NUTCH-2124[7] and attributed to patch commit we are analyzing. This validates the findings of our approach and highlights its usefulness. Therefore, using our approach we not only detect differences but also provide the context to derive actionable insights.

4.2 Experimental Results for the Industrial Application

The EBM generated automatically by our code is shown in Fig. 5 with annotations. Grey denotes the part which is common in EBM_d and EBM_n, dashed is the part which is present only in EBM_d but not in EBM_n, and bold edges/double encircled correspond to the part which is present in EBM_n but not in EBM_d. We selected two code revisions (referred to as $v1$ and $v2$) of the project such that it captures different kinds of code changes possible in software development cycle.

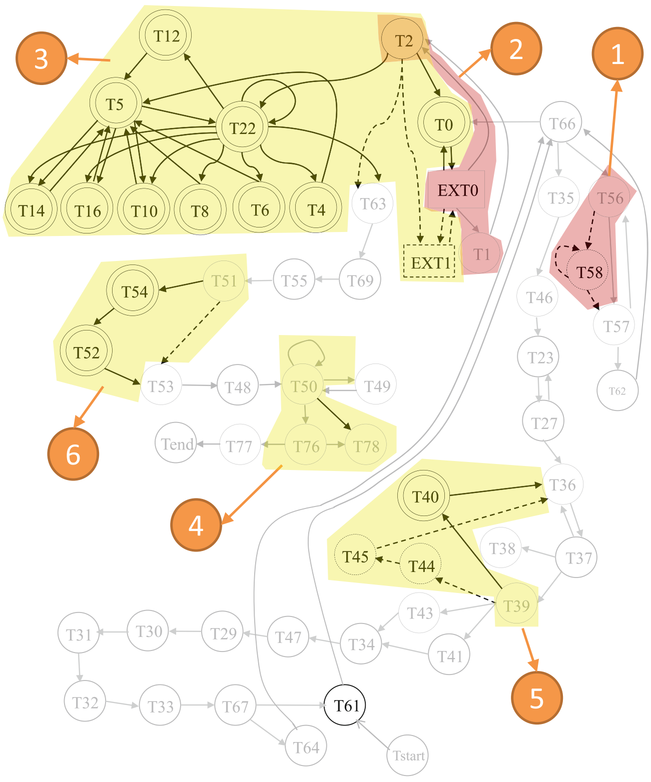


Fig. 5. Annotated execution behavior model highlighting the regions of *diff* for internal analytics application. Grey part is common in EBM_d and EBM_n, dashed is the part which is present only in EBM_d but not in EBM_n, and bold edges/double encircled are present in EBM_n but not in EBM_d

[7] https://issues.apache.org/jira/browse/NUTCH-2124.

As shown in the Fig. 5, our approach detected six different regions of change between the two revisions, which we explain below.

Region 1: $T58$ template is present in both the source code versions but is not observed in EBM_n. Manual inspection of the code and commit history reveals that this is actually a bug caused due to faulty regular expression match and hence one conditional statement is skipped completely.

Region 2: A shift in distribution between edges $(EXT0, T1)$ and $(EXT0, T2)$ that is, increase in transition to $T1$ by factor of 8. Manual inspection reveals that the cause of this anomaly is a wrong Boolean condition check, which caused the distribution to be flipped between two conditional statements.

Region 3: Many new nodes appeared in EBM_n because a new Java class is added (identified in manual investigation) which gets invoked in the new version, i.e., this is an evolutionary design change. Addition of $T0$ however is not exactly related to this change. It is from the class that invokes the new feature but was added in Region 3 alongside the new class because of its close proximity.

Region 4: It has only one change viz. the addition of edge $T50 \rightarrow T78$ and an accompanying decrease in the frequency of $T76 \rightarrow T78$. Manual investigation highlights that $T78$ corresponds to a new exception check added in the class containing $T50$ thus whatever is not caught at $T50$ level is caught at $T76$.

Region 5: Main change is addition of node $T40$ and disappearance of nodes $T44$ and $T45$. Both $T44$ and $T45$ are exception nodes which exist in both code revisions while $T40$ is a new node. On manual inspection, it is revealed that this change is actually a result of bug fix that is, for `ArrayOutOfBounds` exception. This validates that the bug fix is working as intended.

Region 6: Two new nodes appeared in EBM_n and investigation of revision history reveals that a new function was added with two prints which is invoked just after $T51$ in the code thus, an evolutionary change.

To summarize, our approach has successfully detected all seven regions of code change between the two code revisions. It coalesced two of the regions (in Region 3) but this does not affect the usability of our approach as these regions are in close proximity. Manual investigation of *diff* regions in EBM highlights regression bugs as well as validates the evolutionary changes.

5 Discussion

Based on the project requirements, the information from the proposed approach can be leveraged in different ways to help improve the continuous deployment process. It provides additional insights (not to replace the existing practices) for some of the use cases as discussed below:

- *Go/No-go during Release Decision Making:* Most software companies have the concept of a go/no-go meeting before a production release where product, development and operations managers get together to decide whether to go ahead with the release of newer version or not. Our approach provides a way to visualize differences between the code versions in a graphical way thus being easily consumable for decision-making.
- *Update Test Suite to Cover Modified Execution Flows:* The proposed approach identifies region of differences between two execution behavior models thus, merit for comprehensive review. Regression testing can be performed for the diff regions instead of testing the whole application thus making regression testing leaner and at the same time more reliable and effective.
- *Optimal Test Suite Coverage:* By looking at discovered execution flow graphs, it is possible to identify the code paths which are frequently taken during runtime, and tests can be designed intelligently. Hence ensure that more frequent paths are tested rigorously using sophisticated techniques.

The approach assumes the presence of an identifier (i.e. thread ID) to capture the trace for an execution. Since thread ID is often present in the execution logs [32], it is fair to make this assumption. When the identifier for an execution is not present, execution behavior model can be mined using other techniques [20].

To keep our approach language independent and light weight, we do not use static analysis techniques [19]. Also static analysis will not capture the complete reality of execution behavior which gets influenced by prod configuration.

We mine the differences between the execution behavior models of two versions. However, we do not associate the differences with the change type such as bug fixing or feature addition. This kind of classification will not only help in quick resolution of bugs but will also act as an additional check to see if all the release items have been properly taken care before signing off on deployment.

6 Threats to Validity

Threats to Construct Validity: It focuses on the relation between the theory behind the experiment and the observation [29]. Performance of the approach depends on the pervasiveness of logging hence, if less logging statements then it may not be possible to derive useful inferences. However, given that logs are primary source for problem diagnosis, sufficient logging statements are written in the software [6].

Threats to External Validity: External validity is concerned with the generalizability of the results to other settings [29]. We conducted experiments on one open source and one proprietary project to illustrate the effectiveness of the approach. However, both are Java based projects using Log4j library for logging thus, very similar in terms of logging practice. While the approach does not make any project specific assumptions, it is possible that the performance can vary for different project characteristics. Accuracy of multimodal approach depends on the thresholds and thus can vary across projects.

7 Related Work

Execution logs have been extensively studied in such contexts as anomaly detection [4,20], identification of software components [24], component behavior discovery [17], process mining [28], behavioral differencing [12], failure diagnosis [25], fault localization [30], invariant inference [5], and performance diagnosis [10,26]. In this section, we focus on automatic analysis of execution logs.

Goldstein *et al.* [12] analyze system logs, automatically infer Finite State Automata, and compare the inferred behavior to the expected behavior. However, they work on system logs with predefined states while we identify these states (templates) first. Moreover, they present the differences as independent units whereas we group them together rendering the representation more usable.

Cheng *et al.* [7] propose to extract the most discriminative subgraphs which contrast the program flow of correct and faulty execution. Fu *et al.* [10] derive a Finite State Automata to model the execution path of the system and learn it to detect anomalies in new log sequences. However, these are supervised approaches assuming the presence of ground truth for correct and faulty executions to learn a model. Nandi *et al.* [20] detect anomalies by mining the execution logs in distributed environment however, anomalies are detected within the same version, no differencing between the flow graphs of two versions.

Tarvo *et al.* [26] automatically compare the quality of the canary version with the deployed version using a set of performance metrics such as CPU utilization and logged errors however, do not detect the differences in execution flow which is crucial for finding discrepancies. A set of techniques compare multiple versions of an application. Ramanathan *et al.* [22] consider program execution in terms of memory reads and writes and detect the tests whose execution behavior is influenced by these changes. Ghanavati *et al.* [11] compare the behavior of two software versions under the same unit and integration tests. If a test fails in the new version, a set of suspicious code change is reported. This approach works best when comprehensive test suites are available which is not the case in considered Agile environment.

Beyond the specifics of the execution log analysis, our work can be positioned in the context of continuous deployment. Continuous deployment can be seen as an extension of continuous integration [33] and the following step on "the stairway to heaven", the typical evolution path for companies [21].

8 Conclusion and Future Work

We have presented an approach to efficiently highlight the differences in the execution behavior caused due to incremental changes in fast evolving applications. We automatically discover execution behavior model using multimodal approach for the deployed and the new version by mining the execution logs. The models are compared to automatically identify the differences which are presented as cohesive diff regions. Since we have used a graphical representation, we not only identify diff regions but also the context to facilitate in-depth analysis.

Our preliminary evaluation on the open source project Nutch and internal log analytics application illustrates the effectiveness of the approach. Using our approach, we were able to detect multiple bugs introduced in new version for both the applications. Following the analysis, we found that some of the detected bugs were already reported in their issue tracking system therefore, we reported the remaining ones which were later fixed by the developers.

As part of future work, we plan to evaluate the approach on several other applications. Also we plan to automatically classify identified *diff* regions as anomaly and drill down to the root cause commit(s) using revision history.

References

1. Adams, B., McIntosh, S.: Modern release engineering in a nutshell-why researchers should care. In: SANER, pp. 78–90 (2016)
2. Arnoldus, J., van den Brand, M.G.J., Serebrenik, A., Brunekreef, J.: Code Generation with Templates. Atlantis Press, Amsterdam (2012)
3. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective. Addison-Wesley Professional, Boston (2015)
4. Bertero, C., Roy, M., Sauvanaud, C., Trédan, G.: Experience report: log mining using natural language processing and application to anomaly detection. In: ISSRE, pp. 351–360 (2017)
5. Beschastnikh, I., Brun, Y., Schneider, S., Sloan, M., Ernst, M.D.: Leveraging existing instrumentation to automatically infer invariant-constrained models. In: ESEC/FSE, pp. 267–277. ACM (2011)
6. Chen, B., Jiang, Z.M.J.: Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation. EMSE **22**(1), 330–374 (2016)
7. Cheng, H., Lo, D., Zhou, Y., Wang, X., Yan, X.: Identifying bug signatures using discriminative graph mining. In: ISSTA, pp. 141–152. ACM (2009)
8. Claps, G.G., Svensson, R.B., Aurum, A.: On the journey to continuous deployment: technical and social challenges along the way. IST **57**, 21–31 (2015)
9. Feitelson, D.G., Frachtenberg, E., Beck, K.L.: Development and deployment at facebook. IEEE Internet Comput. **17**(4), 8–17 (2013)
10. Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: ICDM, pp. 149–158 (2009)
11. Ghanavati, M., Andrzejak, A., Dong, Z.: Scalable isolation of failure-inducing changes via version comparison. In: ISSRE Workshops, pp. 150–156 (2013)
12. Goldstein, M., Raz, D., Segall, I.: Experience report: log-based behavioral differencing. In: ISSRE, pp. 282–293 (2017)
13. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson Education, London (2010)
14. Kolovos, D.S., Di Ruscio, D., Pierantonio, A., Paige, R.F.: Different models for model matching: an analysis of approaches to support model differencing. In: ICSE Workshop on Comparison and Versioning of Software Models, pp. 1–6 (2009)
15. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. IEEE Softw. **32**(2), 64–72 (2015)
16. Li, S., Niu, X., Jia, Z., Wang, J., He, H., Wang, T.: Logtracker: learning log revision behaviors proactively from software evolution history. In: ICPC (2018)

17. Liu, C., van Dongen, B.F., Assy, N., van der Aalst, W.M.P.: Component behavior discovery from software execution data. In: ICPC (2018)
18. Messaoudi, S., Panichella, A., Bianculli, D., Briand, L., Sasnauskas, R.: A search-based approach for accurate identification of log message formats. In: ICPC (2018)
19. Muske, T., Serebrenik, A.: Survey of approaches for handling static analysis alarms. In: SCAM, pp. 157–166 (2016)
20. Nandi, A., Mandal, A., Atreja, S., Dasgupta, G.B., Bhattacharya, S.: Anomaly detection using program control flow graph mining from execution logs. In: KDD, pp. 215–224 (2016)
21. Olsson, H.H., Alahyari, H., Bosch, J.: Climbing the "stairway to heaven" - a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: Software Engineering and Advanced Applications, pp. 392–399 (2012)
22. Ramanathan, M.K., Grama, A., Jagannathan, S.: Sieve: a tool for automatically detecting variations across program versions. In: ASE, pp. 241–252 (2006)
23. Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., Stumm, M.: Continuous deployment at Facebook and OANDA. In: ICSE Companion, pp. 21–30 (2016)
24. Shatnawi, A., Shatnawi, H., Aymen Saied, M., Al Shara, Z., Sahraoui, H., Seriai, A.: Identifying software components from object-oriented APIs based on dynamic analysis. In: ICPC (2018)
25. Tan, J., Pan, X., Kavulya, S., Gandhi, R., Narasimhan, P.: SALSA: analyzing logs as StAte machines. In: USENIX Workshop on Analysis of System Logs, pp. 1–8 (2008)
26. Tarvo, A., Sweeney, P.F., Mitchell, N., Rajan, V., Arnold, M., Baldini, I.: CanaryAdvisor: a statistical-based tool for canary testing. In: ISSTA, pp. 418–422. ACM (2015)
27. Vaarandi, R., Pihelgas, M.: Logcluster–a data clustering and pattern mining algorithm for event logs. In: Network and Service Management, pp. 1–7. IEEE (2015)
28. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Petri Nets, pp. 368–387 (2008)
29. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
30. Wong, W.E., Debroy, V., Golden, R., Xu, X., Thuraisingham, B.: Effective software fault localization using an RBF neural network. IEEE Trans. Reliab. **61**(1), 149–169 (2012)
31. Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., Pasupathy, S.: SherLog: error diagnosis by connecting clues from run-time logs. In: ACM SIGARCH Computer Architecture News, vol. 38, pp. 143–154 (2010)
32. Yuan, D., Zheng, J., Park, S., Zhou, Y., Savage, S.: Improving software diagnosability via log enhancement. ACM Trans. Comput. Syst. (TOCS) **30**(1), 4:1–4:28 (2012)
33. Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., Vasilescu, B.: The impact of continuous integration on other software development practices: a large-scale empirical study. In: ASE, pp. 60–71 (2017)

Leveraging Computational Reuse for Cost- and QoS-Efficient Task Scheduling in Clouds

Chavit Denninnart[1](✉), Mohsen Amini Salehi[1](✉), Adel Nadjaran Toosi[2](✉), and Xiangbo Li[3](✉)

[1] School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA
{cxd9974,amini}@louisiana.edu

[2] Faculty of Information Technology, Monash University, Melbourne, VIC, Australia
adel.n.toosi@monash.edu

[3] Brightcove Inc., Scottsdale, AZ, USA
xli@brightcove.com

Abstract. Cloud-based computing systems could get oversubscribed due to budget constraints of cloud users which causes violation of Quality of Experience (QoE) metrics such as tasks' deadlines. We investigate an approach to achieve robustness against uncertain task arrival and oversubscription through smart reuse of computation while similar tasks are waiting for execution. Our motivation in this study is a cloud-based video streaming engine that processes video streaming tasks in an on-demand manner. We propose a mechanism to identify various types of "mergeable" tasks and determine when it is appropriate to aggregate tasks without affecting QoS of other tasks. Experiment shows that our mechanism can improve robustness of the system and also saves the overall time of using cloud services by more than 14%.

Keywords: Task aggregation · Oversubscription · Cloud computing Video stream processing · Task scheduling

1 Introduction

With Cloud and Edge Computing gaining more popularity as the back-end platform of many applications, the need for efficient use of these platforms is of paramount importance for individual users and businesses. A common practice to efficiently utilize cloud resources is to use a central queue of arriving tasks with a scheduler that allocates these tasks to a scalable pool of worker Virtual Machines (VMs). The tasks often have individual deadlines that failure to meet them compromises the Quality of Experience (QoE) expected by the end-users.

Although cloud providers supply virtually unlimited resources, users generally have budget constraints, thus, cannot lavishly acquire cloud resources (VMs) [2]. Such limitation raises the *oversubscription* problem, particularly, when there is a surge in the tasks arriving to the system. An oversubscribed system is defined as a system that is overwhelmed with arriving tasks to the extent that there is no way to meet the deadlines of all the tasks, thus, violating end-users' QoE.

A large body of research has been dedicated to alleviate the oversubscription problem. The approaches undertaken in these research works follow two main lines of thinking; First, *allocation-based approaches* that try to minimize the impact of oversubscription through efficient mapping (scheduling) of the tasks to the resources. Second, approaches based on *computational reuse* that avoid or alleviates the oversubscription through efficient caching of the computational results.

Although both of the aforementioned approaches are effective, they are limited in certain ways. The allocation-based approaches cannot entirely resolve the oversubscription because there is no such a solution according to the above-mentioned definition. In addition, many of the approaches are based on complex scheduling algorithms that impose extra overhead to the already oversubscribed system. The approaches based on computational reuse are also limited because they can only reuse the computations for tasks that are identical to the ones already completed and cached. In other words, if two tasks share part of their computation, it cannot be captured by current caching techniques.

In this research, we propose a mechanism based on computational reuse that aims at alleviating oversubscription by aggregating identical and similar tasks in the scheduling queue. Our mechanism makes the scheduling queue less busy and potentially lighten up the overhead of the scheduling process. It complements the existing scheduling-based and caching-based approaches but is not a replacement for them.

We define *mergeable* tasks as those tasks that are either identical or sharing part of their operation with other tasks. We need a mechanism to, first, detect different types of mergeable tasks and, second, eliminate the detected mergeable tasks from the scheduling queue without causing further deadline violations in the system.

Our motivational case study in this research is a video streaming engine that needs to process videos (*e.g.,* downsizing resolution or bit-rate) in the cloud before streaming them to viewers [1]. In this system, it is likely that viewers request same videos to be streamed, hence, creating similar tasks in the system especially when the system is oversubscribed. For example, two viewers who use similar display devices may request to stream the same video with the same or different specifications. The former case creates *identical* tasks in the system whereas the latter one creates *similar* tasks.

In this research, we develop an Admission Control component that is able to detect different levels of similarity between tasks. The system is aware of the tasks' deadlines and performs merging without introducing additional deadline violations. The task aggregation also results in efficient utilization of resources and enable more tasks to meet their deadlines. Therefore, both viewers and system providers can be benefited from the proposed mechanism. In summary, the **key contributions** of this research are as follows: **(A)** Proposing an efficient way of identifying potentially mergeable tasks; **(B)** Determining appropriateness and potential side-effects of merging tasks; **(C)** Analyzing the performance of the task aggregation mechanism on the viewers' QoE and time of deploying cloud resources (VMs).

Although we develop this mechanism in the context of video streaming, the idea of task aggregation and research findings of this work are valid for other domains.

2 Background for Merge-Aware Admission Control

While storing multiple versions of the same video to serve different types of display devices is a conventional practice, Cloud-based Video Streaming Engine (CVSE) [6] enables on-demand (*i.e.*, lazy) processing of video streams, particularly for rarely accessed video streams [3].

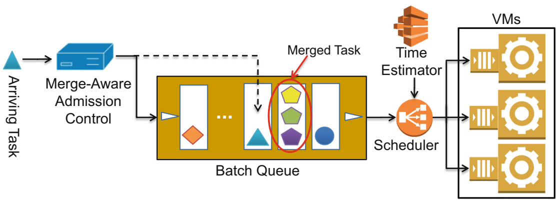


Fig. 1. Overview of Merge-Aware Admission Control for CVSE system

In the CVSE architecture, each task is a GOP (Group Of Picture) of the requested video stream. A GOP task request (hereafter, called task) includes the operation required along with the corresponding parameters bound to that request. Admission Control component, as shown in Fig. 1, sends the task to the batch queue (aka scheduling queue) where the task waits to be assigned by the scheduling policy [4] to one of multiple VMs' queues. Most of the scheduling policies are reliant on the Time Estimator component that is aware of the expected execution time of each task type (*e.g.*, different transcoding operations) on the cloud VMs [5]. Tasks get executed on the assigned VM and streamed to the viewer (More details about CVSE is in [5,6]).

In this paper, we develop our task aggregation mechanism inside Admission Control component of CVSE. For an arriving task, Admission Control recognizes if it is mergeable with the ones exist in the batch queue or local queues of the VMs. Then, the Admission Control decides the feasibility of merging (*i.e.*, if merging causes deadline violation for other tasks).

3 Task Similarity Detection

3.1 Categories of Mergeable Tasks

Mergeability of two given tasks can be explained based on the amount of computation the two tasks share. In particular, mergeability of two or more tasks can be achieved in the following levels:

(A) *Task-level:* This is when more than one instance of the same task exists in the scheduling queue. Therefore, this level is also known as *Identical tasks* and can achieve maximum computational reusability. As these tasks are identical, merging them consumes the same resources required for only one task, hence, reducing both cost and processing delay.
(B) *Operation-level:* This is when two or more tasks perform the same operation on the same data but with different configurations. In this level of merging, the two tasks can share part of their processing.
(C) *Data-level:* This is when two or more tasks perform different operations on the same data. This level of merging achieves the minimum reusability by saving only the time and processing overhead of loading data.

It is noteworthy that the aforementioned reusability levels are generic and can be further categorized depending on the context.

3.2 Detecting Similar Tasks

In this section, we provide a method to detect similar tasks. Although our solution carry out task aggregation using Admission Control component. We would like to note that, it is theoretically possible to carry out task merging in the scheduling queue, *i.e.,* after the task admission. In this case, to find mergeable tasks, we need to scan the entire queue and perform a pair-wise matching between the queued tasks. Practically, this approach is not efficient, because each time the queue is scanned, it imposes a significant number of redundant comparisons. Hence, we choose to perform task merging upon task arrival using the Admission Control component of the system.

Assuming there are n tasks in the queue, for each arriving task, a naïve mergeable task detection method has the overhead of performing n comparisons to find the mergeable tasks. To reduce the overhead, we propose a method that functions based on the hashing techniques. The general idea of the proposed method is to generate a hash key from the arriving task request signature (*e.g.,* GOP id, processing type, and their parameters). Then, the Admission Control finds mergeable tasks by searching for a matching key in the hash table of tasks exist in the scheduling queue.

The explained method can detect task-level mergeability. We need to expand it to detect operation- and data-level of task mergeabilities. To maximize the computational reusability, an arriving task is first verified against task-level mergeability. If there is no match in the task-level, then the method proceeds with checking the next levels of mergeability, namely operation-level and data-level, respectively. To achieve the multiple levels of mergeability, we create three hash-tables—each covers one level of mergeability. The hash-keys in each level are constructed from the tasks' characteristics that are relevant in deciding mergeability at that level. For instance, in video streaming case study, keys in the hash-table that verifies task-level mergeability are constructed from GOP id, processing type, and their parameters. While, keys in the hash-table that verifies operation-level mergeability are constructed from GOP id and processing type. Similarly, keys in the hash-table of data-level mergeability are constructed from GOP id.

Upon arrival of task *j*:

(1) if *j* merges with existing task *i* on Task-level similarity:
   - No update on hash-table is required

(2) if *j* merges with existing task *i* on Operation- or Data-level similarity:
   - Add an entry to each hash-table with hash-keys of task *j* and point them to merged task $i + j$

(3) if *j* matches with existing task *i* but the system chooses not to merge them:
   - Add an entry to each hash-table with hash-keys of task *j* and point them to task *j*

(4) if *j* does not match with any of the existing tasks:
   - Hash-keys of task *j* are added to the respective hash-tables

Upon task *j* completing execution (*i.e.,* dequeuing task *j*):

- Remove all entries pointing to task *j* from hash-tables

Fig. 2. The procedure to update hash-tables upon arrival or completion of tasks

Each entry of the hash-tables includes a hash-key and a pointer to the corresponding task. Entries of the three hash-tables must be updated upon a task arrival and execution. The only exception is Task-level merging, which does not require updating the hash-tables. Figure 2 shows the procedure for updating the hash-tables for a given task j.

When the system merges task j with existing task i, the merged task, denoted as $i + j$, is essentially the object of task i that is augmented with request information (*e.g.,* processing parameters) of task j. In this case, as shown in Step (2) of this procedure, the system only adds an entry to each hash-table with hash-key of task j pointing to merged task $i+j$ as existing key for task i already pointed to task $i + j$. When task j is mergeable with existing task i, but the system decides to add task j to the batch queue without merging. In this case, task j has a higher likelihood of merging with other arriving tasks. The reason is that task i has not merged with task j and it does not merge with other arriving tasks. Hence, as shown in Step (3) of the procedure, the matching entry pointing to task i is redirected and points to task j. It is worth noting that if the arriving task does not match with any of the existing tasks, as shown in Step (4), its hash-keys must be generated and added to the respective hash-tables. Also, when a task completes its execution, its corresponding entries are removed from the hash-tables.

4 Identifying Merging Appropriateness

Imagine an arriving task merges into an existing task in the queue. If such merging is not a Task-level similarity, the execution time of merged task is increased compared to a task before merging. The increased execution time delays the execution of other tasks waiting behind in the queue which could result in deadline violations. Therefore it is critical to assess the impact of merging tasks before performing the merge.

Impact of merging can be assess based on additional deadline misses of tasks following merged tasks when merging occurred against without. Impact of merging assessor create virtual copies of scheduling queue in two scenarios: with merging occurred and without. It simulates the scheduling and estimates completion time of each task, then compares to its deadline. Merging is only carried out if it does not cause additional deadline violations than it would normally happen if the tasks are not merged.

The estimated completion time of task i on a given machine m, denoted as C_i^m and formally shown in Eq. 1, is calculated as the sum of the four following factors: (A) current time, denoted τ; (B) estimated remaining time to complete the currently executing task on machine m, denoted e_r^m; (C) sum of the estimated execution times of N tasks pending in machine queue m, ahead of task i. This is calculated as $\sum_{p=1}^{N}(\mu_p + 2\cdot \sigma_p)$; (D) estimated execution time of task i.

$$C_i^m = \tau + e_r^m + \sum_{p=1}^{N}(\mu_p + 2\cdot \sigma_p) + (\mu_i + 2\cdot \sigma_i) \tag{1}$$

5 Performance Evaluation

5.1 Experimental Setup

We implemented a prototype of CVSE with task aggregation mechanism equipped. It is designed to operate in different modes, namely real streaming mode and emulation mode that is used for testing purposes [6]. In this study, to examine various workloads, we used CVSE in the emulation mode. We evaluated the proposed mechanism using eight homogeneous VMs modeled after Amazon GPU (`g2.2xlarge`) VM.

The video repository we used for evaluation includes multiple replicas of a set of benchmark videos. Videos in the benchmarking set are diverse both in terms of the content types and length. The length of the videos in the benchmark varies in the range of [10, 600] seconds splitting into 10-110 Group Of Picture (GOP) segments. The benchmark videos are publicly available for reproducibility purposes at https://goo.gl/TE5iJ5. For each GOP of the benchmark videos, we obtained their processing times by executing each processing operation 30 times on Amazon GPU VM. The processing operations we benchmarked are: reducing resolution, changing codec, adjusting bit rate, and changing frame rate.

To evaluate the system under various workload intensities, we generate [2000, 3000] GOP processing tasks within a fixed time interval. We collect the deadline miss-rate (DMR) and makespan (*i.e.,* execution time to finish all tasks) of completing all tasks. We conducted each experiment 30 times, each time with random task arrival time and order. Mean and 95% confidence interval of the results are reported. We examined three queuing policies, namely FCFS (First-Come-First-Serve), EDF (Earliest Deadline First), and MU (Max Urgency). For each queuing policy, we studied no task merging versus task merging. In the experiments, all tasks must be completed, even if they miss their deadline.

5.2 Impact of Task Aggregation

Evaluating Makespan: In the first experiment, our goal is to see the impact of task merging on makespan. In fact, makespan implies the time cloud resources are deployed, which implies the cost incurred to execute all the tasks. We examine the system under various subscription levels (from 2000 to 3200 GOPs) arriving within the same time interval. As we can see in Fig. 3, our proposed merging mechanism saves between 4.40% and 14.33% in makespan. Execution time saving is more pronounced when the system is more oversubscribed. It is worth noting that makespan does not vary under different scheduling policies.

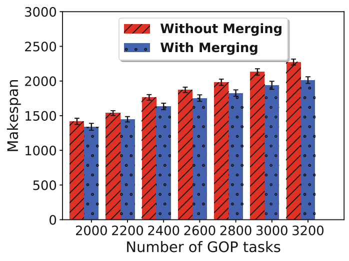
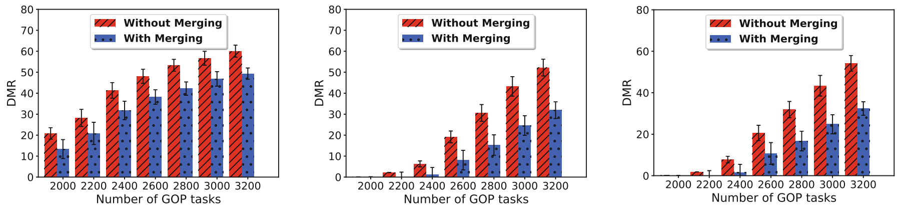


Fig. 3. Comparing the total time to complete tasks (*i.e.,* makespan) under varying number of arriving GOP tasks (horizontal axes) in two scenarios: without task merging, and with task merging.

Evaluating Deadline Miss Rate (DMR): In this experiment, our goal is to evaluate viewers' QoE. For that purpose, we measure the deadline miss rate resulted from no merging versus merging tasks under various oversubscribed levels and with different scheduling policies. As shown in Fig. 4, we observe that task aggregation significantly reduces deadline miss rate in all scheduling policies. We can see that the improvement in deadline miss rate of FCFS is less than EDF and MU scheduling policies. This is because FCFS by nature causes a larger average waiting time and does not schedule tasks by considering their deadline. Therefore, task merging mechanism performance, when combined with FCFS, is lower than other scheduling polices.

Comparing the results shown in Fig. 3 with those in Fig. 4 reveals that the difference in deadline miss rate is more dramatic than the makespan time. This is due to the fact that even small reduction in task completion time can cause the merged tasks meet their deadlines instead of missing that. We can conclude that the impact of task aggregation mechanism on viewers' QoE can become more remarkable when it is combined with efficient scheduling policies.



(a) DMR under FCFS Queue (b) DMR under EDF Queue (c) DMR under MU Queue

Fig. 4. Comparing the deadline miss-rate (DMR) under varying number of GOP tasks (horizontal axes) in two scenarios: without task merging, and with task merging. Subfigures (a), (b), and (c) show the DMR under FCFS, EDF, and MU queuing polices.

6 Related Works

Software-based computational reuse has been extensively researched and used. However, not many systems can merge and reuse tasks before tasks are actually executed and many of them tie very closely to one specific application. Below are some notable works in this area.

Popa *et al.* [8] presented modules to identify identical and similar tasks to cache partial results and reuse them on incremental computation specifically on Dryad platform context. They proposed two solutions: One solution automatically caches computational results. Another solution merges tasks based on programmer's defined merge function. Their first solution is a caching system while their second solution is similar to our work, but more specific to Dryad platform which does not have deadline and QoE to consider.

Paulo and Pereira *et al.* [7] developed a system to perform deduplication of high throughput data using $Bloomfilters$. $Bloomfilters$, while fast, have chances of giving false positive hash checking. Therefore they achieve lower overhead data duplicate detection than hash table approach we use, at the price of compromised accuracy.

7 Conclusion and Future Works

In this paper, we improve efficiency of the system in oversubscribed condition by merging arriving tasks with other (exact or similar) tasks. We dealt with two challenges: First, how to identify identical and similar tasks in an efficient manner? Second, how to perform merging without violating the deadline of other tasks in the system? To address the first challenge, we identified three main levels of similarity that tasks can be merged. Then, we developed a method to detect different levels of task similarity within a constant time complexity. To address the second challenge, we developed a method that determines impact of merging and only perform merge operations if other tasks' deadline are not affected. Experimental results demonstrate that the proposed system can reduce the overall execution time of tasks by more than 14%, hence, cloud VMs can be deployed for a shorter time. This benefit comes with improving QoE of the users. Although we implemented this system in the context of video streaming, the concept can be applied to other domains as long as we can define similarity levels in those domains. In the future, we plan to extend this work by exploring the impact of marginally compromising QoE, in favor of a remarkable cost-saving on the cloud resources.

Acknowledgments. This research was supported by the Louisiana Board of Regents under grant number LEQSF(2016-19)-RD-A-25.

References

1. Ahmad, I., Wei, X., Sun, Y., Zhang, Y.-Q.: Video transcoding: an overview of various techniques and research issues. IEEE Trans. Multimed. **7**(5), 793–804 (2005)
2. Bi, J., et al.: Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center. IEEE Trans. Autom. Sci. Eng. **14**(2), 1172–1184 (2017)
3. Darwich, M., Beyazit, E., Salehi, M.A., Bayoumi, M.: Cost efficient repository management for cloud-based on-demand video streaming. In: Proceedings of the 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 39–44, April 2017
4. Hosseini, M., Salehi, M.A., Gottumukkala, R.: Enabling interactive video stream prioritization for public safety monitoring through effective batch scheduling. In: Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications, HPCC 2017, December 2017
5. Li, X., Salehi, M.A., Bayoumi, M., Buyya, R.: CVSS: a cost-efficient and QoS-aware video streaming using cloud services. In: Proceedings of the 16th IEEE/ACM International Conference on Cluster Cloud and Grid Computing, CCGrid 2016, pp. 106–115, May 2016
6. Li, X., Salehi, M.A., Bayoumi, M., Tzeng, N.-F., Buyya, R.: Cost-efficient and robust on-demand video stream transcoding using heterogeneous cloud services. IEEE Trans. Parallel Distrib. Syst. (TPDS) **29**(3), 556–571 (2018)
7. Paulo, J., Pereira, J.: Distributed exact deduplication for primary storage infrastructures. In: Magoutis, K., Pietzuch, P. (eds.) DAIS 2014. LNCS, vol. 8460, pp. 52–66. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43352-2_5
8. Popa, L., Budiu, M., Yu, Y., Isard, M.: DryadInc: reusing work in large-scale computations. In: Proceedings of 1st USENIX workshop on Hot Topics in Cloud Computing, HotCloud 2009, June 2009

QKnober: A Knob-Based Fairness-Efficiency Scheduler for Cloud Computing with QoS Guarantees

Shanjiang Tang[1](✉), Ce Yu[1](✉), Chao Sun[1], Jian Xiao[1], and Yinglong Li[2]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
{tashj,yuce,sch,xiaojian}@tju.edu.cn

[2] School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China
liyinglong@ruc.edu.cn

Abstract. Fairness and efficiency are generally two important metrics for users in modern cloud computing. Due to the heterogeneous resource demands of CPU and memory for users' tasks, it cannot achieve the strict 100% fairness and the maximum efficiency at the same time. Quantitatively showing the fairness degradation/loss becomes essentially important in the design of any fairness-efficiency tradeoff scheduler. Existing fairness-efficiency schedulers (e.g., Tetris) can balance such a tradeoff elastically by relaxing fairness constraint for improved efficiency using the knob. However, their approaches are *insensitive* to the fairness degradation under different knobs, which makes several drawbacks. First, it cannot quantitatively tell how much *relaxed* fairness can be guaranteed (i.e., QoS of fairness guarantee) given a knob value. Second, it fails to meet several essential properties such as sharing incentive. To address these issues, we propose a new fairness-efficiency scheduler, *QKnober*, to balance the fairness and efficiency elastically and flexibly using a tunable fairness knob. QKnober is a *fairness-sensitive* scheduler that can maximize the system efficiency while guaranteeing the θ-soft fairness by modeling the whole allocation as a combination of *fairness-purpose* allocation and *efficiency-purpose* allocation. Moreover, QKnober satisfies fairness properties of sharing incentive, envy-freeness and pareto efficiency given a proper knob. We have implemented QKnober in YARN and evaluated it using real experiments. The results show that QKnober can achieve good performance and fairness.

1 Introduction

In the current era of 'big data', it has become typical to take existing large-scale data computing frameworks such as MapReduce [8] and Spark [26] for big data analytics in a cloud system consisting of many machines [16]. At any time, there are many users running their data-parallel applications on the cloud. Typically, users' submitted jobs often contain many tasks and their tasks tend to have *heterogeneous* resource requirements towards different resource types

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 837–853, 2018.
https://doi.org/10.1007/978-3-030-03596-9_60

(e.g., CPU and memory). For example, tasks of machine learning applications are CPU-intensive [10], whereas hash join and sort tasks of database queries are memory-intensive [6].

Fairness and efficiency are generally two critical metrics for both users and resource providers in cloud computing [12]. Being aware of heterogeneous resource demands of users' tasks, there is a need to consider multi-resource fairness that takes multiple resource types into account. By leveraging the game-theoretic definition, a *robust* multi-resource fair allocation is the one in which

– *all users in the shared system should perform no worse than that under an exclusively non-sharing partition of the system. (Sharing incentive)*
– *no user envies the allocations of any other users. (Envy freeness)*
– *no user can increase its resource allocation with harming at least one other user. (Pareto efficiency)*

Dominant Resource Fairness (DRF) is one of the most well-known multi-resource fair allocation policies [9] with the above three game-theoretic properties. It introduces the concept of dominant resource, referred to as the resource that is heavily used by a user. The fairness is achieved by equalizing the share of each user's dominant resource. Although there have since been a number of extensions [13,23], they draw little attention to the influence on system efficiency. Recent studies have shown that there is a tradeoff between fairness and efficiency in multi-resource allocation [10,11,22]. Guaranteeing the strict 100% fairness across users would produce inefficient resource allocations. Conversely, seeking for high system efficiency is often at the cost of compromised fairness. DRF and its extensions tend to over constrain the system for high fairness guarantee, resulting in resource allocations with low system efficiency.

Many existing fairness-efficiency schedulers seek to relax fairness (i.e., allowing some degree of unfairness) for efficiency improvement by employing knob-based heuristic algorithms [7,10,17,24]. Quantifying the fairness degradation/loss is essentially important for users in order to properly configure the knob value. Tetris [10] is the state-of-the-art knob-based tradeoff scheduler that allows users to balance fairness and efficiency flexibly by tuning the fairness knob in cloud computing. However, due to its *insensitiveness* of fairness degradation under different knobs, there are some shortcomings (see Sect. 3): (1). it cannot quantitatively show users how much *relaxed* fairness can be guaranteed given a fairness knob (i.e., QoS of fairness guarantee); (2). it fails to satisfy several fairness properties such as sharing incentive.

In this paper, we develop a new fairness-efficiency scheduler, *QKnober*, to allow users to balance fairness and efficiency flexibly with a knob factor $\rho \in [0, 1]$. Unlike the previous schedulers [7,10,24], QKnober is a *fairness-sensitive* scheduler that works on the *relaxed* fairness (i.e., *soft* fairness in Sect. 4.1), which refers to the maximum difference between the normalized shares of any two users. It is achieved by modeling the multi-resource allocation as a combination of *fairness-purpose* allocation and *efficiency-purpose* allocation (Sect. 4.1). Given a knob ρ, QKnober first performs the fairness-purpose allocation for the QoS of

θ-soft fairness guarantee (see Theorem 1 in Sect. 4.1) and then does the efficiency-purpose allocation for maximizing the system efficiency. We show that with a proper knob configuration, QKnober can ensure that each user in the shared system can get at least the amount of resources as that under the exclusively non-sharing partition of the system. It also can guarantee that every user prefers to its own allocation and no user envies the allocations of any other users. Furthermore, QKnober keeps that the system is fully utilized by ensuring that no user can get more resource allocation without decreasing the allocation of at least one user.

We have implemented QKnober in YARN [20]. We evaluated QKnober with testbed workloads in a Amazon EC2 cluster consisting of 60 nodes. Our results show that QKnober strikes a flexible balance between fairness and efficiency. There can be up to 57% performance improvement as we decrease the knob factor from one to zero for QKnober. Moreover, it outperforms its alternatives DRF and Tetris by 31.2% and 4.5% on average, respectively. Finally, we show that the scheduling overhead of QKnober is minor ($< 0.42\,\text{ms}$).

2 Desirable Allocation Properties

From the economic point of view, a *good* fair allocation policy in cloud computing system should provide the following essential game theoretic properties, including sharing incentive, envy-freeness, and pareto efficiency [9].

Sharing Incentive (SI): Resource sharing is an essential and effective approach to improve the system utilization and efficiency [15]. A good allocation policy should satisfy sharing incentive (SI) such that each user in the system performs at least as good as it would be under a statically equal split of the resources of the computing system. Otherwise, users would be more likely to divide the computing system equally and exclusively use their own partitions without sharing. Thus, to enable resource sharing possible and sustainable, it is a must requirement to satisfy sharing incentive [18].

Formally, let $\mathbf{U}_i = \langle u_{i,1}, \ldots, u_{i,m} \rangle$ be the resource allocation vector for user i. Let $N_i(\mathbf{U}_i)$ denote the number of tasks scheduled for user i under the resource allocation vector $\mathbf{U}_i$. An allocation policy is sharing incentive if it satisfies the following condition for each user $i \in [1, n]$,

$$N_i(\mathbf{U}_i) \geqslant N_i(\overline{\mathbf{U}}_i), \tag{1}$$

where $\overline{\mathbf{U}}_i = \langle \overline{u}_{i,1}, \ldots, \overline{u}_{i,m} \rangle$ represents the resource allocation vector for user i under the exclusively non-sharing partition of the computing system.

Envy-Freeness (EF): An allocation is envy-freeness (EF) if no user envies the allocation of other users associated with a desire to receive that same allocation. That is, every user prefers its own allocation to that of any other user. To provide EF, there is a need to ensure that every user cannot have more tasks scheduled by switching its allocation with any other user.

Given the resource allocation vector $\mathbf{U}_i$ for user i, an allocation policy satisfies EF if

$$N_i(\mathbf{U}_i) \geqslant N_i(\mathbf{U}_j), \tag{2}$$

for any two users $i, j \in [1, n]$.

Pareto Efficiency (PE): PE is another critical property that should be satisfied by a fair resource allocation policy [19]. It is essential for high resource utilization and efficiency. An allocation policy is PE if it is not possible for a user to get more tasks scheduled without decreasing the number of running tasks of at least one other user.

Let $\mathbf{U} = \langle \mathbf{U}_1, \ldots, \mathbf{U}_n \rangle$ be the resulting allocation for all users produced by a fair allocation policy. The allocation $\mathbf{U}$ is PE if it does not exist any feasible allocation $\breve{\mathbf{U}}$ satisfying the following two conditions at the same time, i.e., (1). $\forall i \in [1, n], N_i(\mathbf{U}_i) \leqslant N_i(\breve{\mathbf{U}}_i)$; (2). $\exists j \in [1, n], N_j(\mathbf{U}_j) < N_j(\breve{\mathbf{U}}_j)$.

3 Background and Motivation

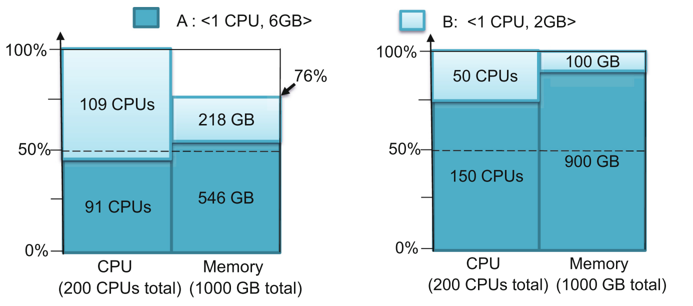
In this section, we motivate our work by reviewing and analyzing the limitations of existing schedulers.

Fairness* vs. *System Efficiency. In multi-resource allocation, the fairness and system utilization/efficiency highly depend on the workload characteristic and allocation ratio of the resource. If the users with memory-intensive workloads have small allocation ratio, it may result in low utilization for the memory resource due to insufficient requests. Similar case does also hold for other resources (e.g., CPU). On the contrary, maintaining high resource utilization for all resources often generate the allocations in a manner that starves some users, resulting in unfairness problem for users in the allocation. We next demonstrate these problems using examples of Dominant Resource Fairness (DRF) [9], which is a popular multi-resource allocation policy with many attractive merits (e.g., sharing incentive, envy-freeness and pareto-efficiency).

Example 1. Consider a computing system consisting of 200 CPUs and 1000 GB memory in total. It is shared by two users A and B equally with the task requirement of ⟨1 CPU, 6 GB ⟩ for A and ⟨ 1 CPU, 2 GB ⟩ for B, respectively.

The dominant resource for user A is memory because each task of A consumes 1/200 of the total CPUs and 6/1000 of the total memory, while the dominant resource for B is CPU. DRF achieves the fairness by equalizing the dominant resource shares (i.e., $546/1000 = 109/200$) for A and B, with the resulting allocation illustrated in Fig. 1(a). The memory utilization is only $\frac{(546+218)}{1000} \approx 76\%$. This is because DRF does not consider resource efficiency when making allocation decision. It only focuses on achieving the fairness among users, but does not deal with the impact of such adjustments on the system efficiency.

In fact, both CPU and memory resources in Example 1 can be fully utilized if the scheduler allocates ⟨150 CPUs, 900 GB⟩ to A and ⟨50 CPUs, 100 GB⟩ to



(a) DRF allocation. (b) Pure efficiency allocation.

Fig. 1. Allocation results with different policies for Example 1. The memory utilization is only 76% for DRF, whereas we can achieve 100% utilization for CPU and memory with pure efficiency policy.

B, as illustrated in Fig. 1(b). However, the dominant resource shares of A and B are no longer the same (i.e., $\frac{900}{1000} > \frac{50}{200}$), being *unfair* for B. It implies that there tends to be a tradeoff between fairness and system efficiency in resource allocation.

Flaws of the *State-of-the-Art* Tradeoff Scheduler for Cloud Computing. To balance the tradeoff between fairness and efficiency elastically and flexibly, many fairness-efficiency schedulers [10,17,22,24] take *knob-based* heuristics, which is *promised* as an effective approach in multi-resource allocation [10]. Wang et al. [22,24] studied the fairness-efficiency tradeoff in networking system by considering network packet processing and data transfer flow across different machines. EMRF [17] is a fairness-efficiency tradeoff scheduler for Coupled CPU-GPU architectures. In contrast, for cloud computing, Tetris [10] is the *state-of-the-art* knob-based scheduler. Specifically, in each resource allocation, it first sorts all tasks according to the DRF. Then, it searches the best task for efficiency among the runnable tasks belonging to the first $(1 - f)$ tasks in the sorted list, where $f \in [0, 1]$ is a knob provided by users in advance. It computes the *alignment score*, defined as the weighted dot product between the vector of machine's available resources and the task's peak resource demand, to the machine for each task, and the best task is picked with the largest alignment score. However, there are several flaws for Tetris as follows:

First, although Tetris allows users to relax fairness for efficiency improvement by tuning the fairness knob, it is *insensitive* to fairness degradation for different knobs during the allocation. Particularly, it cannot quantitatively show users how much *relaxed* fairness (i.e., *soft* fairness in Sect. 4.1) can be guaranteed (i.e., QoS of fairness guarantee) given a knob configuration. To explain it, let's revisit Example 1 by assuming that at each allocation, there are 500 tasks for A and B, respectively. We can see that the task (share: $\langle\frac{1}{200}, \frac{6}{1000}\rangle$) of A is more beneficial to the system utilization than that (share: $\langle\frac{1}{200}, \frac{2}{1000}\rangle$) of B according to the resource type difference of their tasks ($A : |\frac{1}{200} - \frac{6}{1000}| = \frac{1}{1000}$,

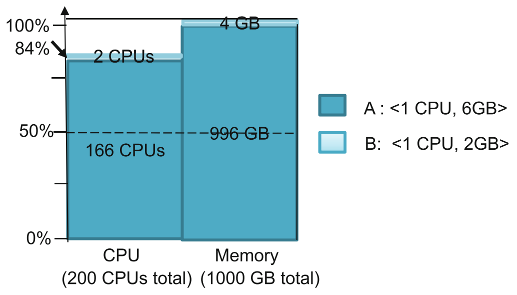


Fig. 2. The resulting allocation with Tetris for Example 1 when the knob f satisfies $0 \leqslant f < 0.5$. In this case, different value of knobs does not work for fairness improvement, indicating that Tetris is *insensitive* to fairness under different knobs.

$B : |\frac{1}{200} - \frac{2}{1000}| = \frac{3}{1000}$). At each allocation, Tetris first sorts all 1000 tasks of A and B according to DRF policy. Then, it tries to pick up a task from the first $(1-f)$ tasks in the sorted list that is most beneficial to the system utilization. When $0 \leqslant f < 0.5$, the task range that Tetris can choose at the second stage is $500 < (1-f) * 1000 \leqslant 1000$. In this case, Tetris always picks up preferred tasks from A rather than B until it cannot fulfilled, resulting in allocation as shown in Fig. 2. It shows that the knob of Tetris does not work for fairness improvement when $0 \leqslant f < 0.5$, i.e., Tetris is *insensitive* to fairness degradation under different knobs, implying that it cannot tell users how much relaxed fairness can be guaranteed under a knob setting. However, in practice, the QoS guarantee of different levels of fairness is very important for users under different knobs configurations.

Second, Tetris violates the sharing incentive property (See definition in Sect. 2). Let's take Example 1 as a counterexample to demonstrate it. Provided that $f = 0$, Tetris is purely for system efficiency optimization by picking the best task for efficiency among all tasks every time, resulting in the allocation as illustrated in Fig. 1(b). We can see that, B receives less resources (i.e., fewer tasks scheduled) in the sharing system than that (i.e., $\langle$100 CPUs, 200 GB$\rangle$) of exclusively using its partition of the system without sharing, violating the sharing incentive property.

Motivated by these, we seek to explore a new fairness-efficiency allocation policy that guarantees the soft fairness and satisfies all the desirable properties listed in Sect. 2.

4 Allocation Model and Scheduling Policy

In this section, we model the multi-resource allocation in cloud computing based on DRF, and propose a proposed fairness-efficiency scheduling policy called QKnober.

4.1 Multi-resource Allocation Model

1) Basic Setting. We start by defining some terms used in our model. Suppose that the computing system consists of m resource types (e.g., CPU, memory, disk) with the capacity of $\mathbf{R} = \langle r_1, \ldots, r_m \rangle$ shared by n users, where r_i denotes the total amount of resource i. For each user i, let w_i denote its share weight in the shared computing system and $\mathbf{D}_i = \langle d_{i,1}, \ldots, d_{i,m} \rangle$ be its *resource demand vector*, where $d_{i,j}$ denotes the amount of resource j required by a task of user i. We assume that each user has an infinite number of tasks to be scheduled, and all its tasks are divisible and with the same resource demand. We later discuss how these assumptions can be relaxed for practical usage in Sect. 5.

Given the allocation matrix $\mathbf{U} = \langle \mathbf{U}_1, \ldots, \mathbf{U}_n \rangle$ for all users, it is a *feasible* allocation if it satisfies that,

$$\sum_{i=1}^{n} u_{i,k} \leqslant r_k, \quad \forall k \in [1, m], \tag{3}$$

The maximum number of tasks $N_i(\mathbf{U}_i)$ (possible fractional) that user i can schedule under the resource allocation vector $\mathbf{U}_i$ is,

$$N_i(\mathbf{U}_i) = \min_{1 \leqslant k \leqslant m} \{u_{i,k}/d_{i,k}\}, \tag{4}$$

2) Allocation Model. An efficient resource allocation should never let a user get more resources than it actually needs in the computing system. We call such an allocation *non-wasteful*. Formally, an allocation $\mathbf{U}_i$ is *non-wasteful* if and only if it satisfies the following condition:

$$\mathbf{U}_i = N_i(\mathbf{U}_i) \cdot \mathbf{D}_i, \tag{5}$$

It is worthy mentioning that we can always convert an allocation to the *non-wasteful* allocation by transferring the redundant/unused resources of each user to other potential users without decreasing the number of tasks scheduled for that user. Without loss of generality, in the following discussions, we limit our focus on the non-wasteful allocation.

Scheduling tasks to the computing system is analogous to the multi-dimensional knapsack problem [5] by viewing the computing system as a knapsack and each task as a knapsack item. The weight of an item (or task) from user i is $\mathbf{D}_i$. In this work, since we are interested in the efficiency of resource allocation, the value of an item (or task) is the sum of the amount of different typed resources it required (normalized to the system capacity), i.e., $\sum_{k=1}^{m} d_{i,k}/r_k$. Let $\epsilon_i(\mathbf{U}_i)$ be the efficiency (i.e., knapsack cost value) of a feasible resource allocation $\mathbf{U}_i$ contributed by user i in the system. According to the knapsack problem, we have

$$\epsilon_i(\mathbf{U}_i) = N_i(\mathbf{U}_i) \cdot \sum_{k=1}^{m} d_{i,k}/r_k, \tag{6}$$

for a single user i. Then the efficiency $\epsilon(\mathbf{U})$ of a feasible allocation $\mathbf{U}$ for all users in the system can be calculated as

$$\epsilon(\mathbf{U}) = \sum_{i=1}^{n} \epsilon_i(\mathbf{U}_i) = \sum_{i=1}^{n} \{N_i(\mathbf{U}_i) \cdot \sum_{k=1}^{m} d_{i,k}/r_k\}. \tag{7}$$

Let s_i denote the share of dominant resource for user i in the computing system. According to Formula (5), we have

$$s_i = \max_{1 \leqslant k \leqslant m} u_{i,k}/r_k = N_i(\mathbf{U}_i) \cdot \max_{1 \leqslant k \leqslant m} d_{i,k}/r_k. \tag{8}$$

Formula (8) indicates that there is a proportional relationship between a user's dominant resource share and the number of tasks scheduled. The Dominant Resource Fairness (DRF) achieves the fairness by guaranteeing that the (weighted) shares of dominant resource across users are the same, i.e.,

$$\frac{s_1}{w_1} = \frac{s_2}{w_2} = \ldots = \frac{s_n}{w_n}. \tag{9}$$

Let s_i^{max} and $N_i(\mathbf{U}_i^{max})$ represent the maximum share of dominant resource and the corresponding number of tasks scheduled for user i under the DRF allocation. The DRF allocation can be viewed as progressive filling when all tasks are divisible [9]. The allocation terminates when at least one typed resource is fulfilled. In that case, we are unable to increase each user's dominant resource. That is, the dominant resource share and the corresponding number of tasks scheduled are maximized for each user under DRF. It thus holds,

$$\max_{1 \leqslant k \leqslant m} \{\sum_{i=1}^{n} \frac{u_{i,k}}{r_k}\} = \max_{1 \leqslant k \leqslant m} \{\sum_{i=1}^{n} \frac{N_i(\mathbf{U}_i) \cdot d_{i,k}}{r_k}\} = 1. \tag{10}$$

By computing $N_i(\mathbf{U}_i)$ with Formulas (8), (9) and (10), we can derive $N_i(\mathbf{U}_i^{max})$ as follows:

$$N_i(\mathbf{U}_i^{max}) = w_i / (\max_{1 \leqslant k \leqslant m} \{\frac{d_{i,k}}{r_k}\} \cdot \max_{1 \leqslant k \leqslant m} \{\frac{1}{r_k} \cdot \sum_{j=1}^{n} \frac{w_j \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m}\{\frac{d_{j,k'}}{r_{k'}}\}}\}).$$

According to Formula (8), we can get s_i^{max} as

$$s_i^{max} = w_i / \max_{1 \leqslant k \leqslant m} \{\frac{1}{r_k} \cdot \sum_{j=1}^{n} \frac{w_j \cdot d_{j,k}}{\max_{1 \leqslant k' \leqslant m}\{\frac{d_{j,k'}}{r_{k'}}\}}\}. \tag{11}$$

4.2 QKnober

Recall that the model in Sect. 4.1 is a strict 100% fairness allocation model. By altering the model slightly, we can develop a knob-based fairness-efficiency scheduler, *QKnober*, to allows users to balance fairness and system efficiency flexibly using a fairness knob.

The basic idea is as follows. Instead of strictly seeking for 100% fairness as DRF does, we can compromise fairness for increased allocation efficiency by tolerating some degree of fairness loss. Particularly, we classify the fairness into two types: *hard fairness* and *soft fairness*. The hard fairness refers to that the allocation shares of all users should be the same (i.e., Formula (9) should be guaranteed). In contrast, the soft fairness tolerates some degree (measured by θ) of unfairness across users. Formally, we define *θ-soft fairness* by changing Formula (9) as follows:

$$|\frac{s_i}{w_i} - \frac{s_j}{w_j}| \leqslant \theta, \forall i, j \in [1, n]. \tag{12}$$

Typically, DRF focuses on the hard fairness across users, limiting the allocation efficiency improvement. In contrast, QKnober, as a fairness-efficiency tradeoff scheduling policy, is interested in the soft fairness, which can leave some room for efficiency improvement. In the following, we describe our design of QKnober policy.

1) QKnober Design. The fairness-efficiency tradeoff allocation can be considered as an integration of two stages allocations: *fairness-purpose* allocation (i.e., purely for fairness optimization) and *efficiency-purpose* allocation (i.e., purely for efficiency optimization). For QKnober, it first does the fairness-purpose allocation with DRF to achieve the soft fairness guarantee. Next it turns to the efficiency-purpose allocation for efficiency maximization. Particularly, it offers users a knob $\rho \in [0, 1]$ to control and balance the two stages allocations flexibly. Let $\bar{s_i}$ and $s_i^{'}$ be the dominant resource shares of the resulting allocation for user i in the stage of fairness-purpose allocation and efficiency-purpose allocation, respectively. By combining the allocations of two stages, we get the final dominant resource share s_i for each user i as follows:

$$s_i = \bar{s_i} + s_i^{'}. \tag{13}$$

Fairness-purpose Allocation. In the stage of fairness-purpose allocation, instead of guaranteeing the hard (dominant resource) fairness of s_i^{max} for each user i, QKnober seeks to guarantee the soft fairness of $s_i^{max} \cdot \rho$ (i.e., $\bar{s_i} = s_i^{max} \cdot \rho$). According to Formula (13), we have

$$s_i = s_i^{max} \cdot \rho + s_i^{'}. \tag{14}$$

It leaves $\mathbf{R}^{'} = \langle r_1^{'}, \ldots, r_m^{'} \rangle$ idle resources for efficiency-purpose allocation, where $r_i^{'} = r_i - \sum_{j=1}^{n} \frac{s_j^{max} \cdot \rho \cdot d_{j,i}}{\max_{1 \leqslant k \leqslant m} d_{j,k/r_k}}$ according to Formula (8). The small value of ρ favors the efficiency optimization. In contrast, the large value of ρ can make the fairness-purpose allocation dominant, benefiting more for fairness optimization. Typically, QKnober reduces to DRF when $\rho = 1$.

Theorem 1. *QKnober is a θ-soft fairness policy where*

$$\theta = \max_{1\leqslant i\leqslant n}\{\frac{\max_{1\leqslant k\leqslant n} d_{i,k}/r_k}{w_i \cdot \max_{1\leqslant k\leqslant m}\{\frac{d_{i,k}}{r_k - \sum_{j=1}^{n}\frac{s_j^{max}\cdot\rho\cdot d_{j,k}}{\max_{1\leqslant k'\leqslant m} d_{j,k'}/r_{k'}}}\}}\}.$$

The proof of Theorem 1 can be found in Appendix A of Technique Report [14].

Efficiency-purpose Allocation. Theorem 1 shows that the fairness-purpose allocation of QKnober can guarantee *θ-soft fairness* across users. In the second stage, we perform the efficiency-purpose allocation with the remaining idle resource vector $\mathbf{R}^{'}$ so that its overall efficiency is maximized.

Formally, our work is to search a feasible allocation $\mathbf{U}^{'}$ such that Formula (7) is maximized. Particularly, for any two users i and j with the same normalized task demands (i.e., $\frac{\mathbf{D}_i}{|\mathbf{D}_i|} = \frac{\mathbf{D}_j}{|\mathbf{D}_j|}$), exchanging resources between them has no impact on efficiency but could affect fairness. In order for better fairness, we still keep Formula (9) holding for any two users satisfying $\frac{\mathbf{D}_i}{|\mathbf{D}_i|} = \frac{\mathbf{D}_j}{|\mathbf{D}_j|}$ by adding Formula (19). We can model the efficiency-purpose allocation as a linear programming optimization problem as follows:

$$\textbf{Maximize} \qquad \epsilon(\mathbf{U}^{'}) = \sum_{i=1}^{n}\{N_i(\mathbf{U}_i^{'}) \cdot \sum_{k=1}^{m} d_{i,k}/r_k\}. \tag{15}$$

subject to:

$$s_i^{'}/w_i = s_j^{'}/w_j. \quad (\mathbf{D}_i/|\mathbf{D}_i| = \mathbf{D}_j/|\mathbf{D}_j|, \forall i,j \in [1,n]). \tag{16}$$

and

$$\sum_{i=1}^{n}\{d_{i,k}\cdot N_i(\mathbf{U}_i^{'})\} \leqslant r_k - \sum_{j=1}^{n}\frac{s_j^{max}\cdot\rho\cdot d_{j,k}}{\max_{1\leqslant k'\leqslant m} d_{j,k'}/r_{k'}}. \tag{17}$$

for $\forall k \in [1,m]$. By resolving the linear program, the optimal (maximum) value of $\epsilon(\mathbf{U}^{'})$ can be obtained. Finally, the total system efficiency $\epsilon(\mathbf{U})$ can be computed by combining the allocation efficiencies in the two allocation phases.

To summarize, we have shown that QKnober is a knob-based fairness-efficiency scheduling policy that can maximize the system efficiency while guaranteeing the *θ-soft fairness* with the provided knob ρ. Particularly, different configurations of the fairness knob ρ can result in different soft fairness guarantees for QKnober (i..e, QKnober is *sensitive* to the fairness degradation under different knobs).

2) Properties Analysis of QKnober. We give an analysis of the three essential properties defined in Sect. 2 for QKnober.

Theorem 2. (Sharing Incentive): *The QKnober allocation policy is sharing incentive when*

$$\rho \geqslant (\max_{1\leqslant k\leqslant m}\{\frac{1}{r_k}\cdot\sum_{j=1}^{n}\frac{w_j\cdot d_{j,k}}{\max_{1\leqslant k'\leqslant m}\{\frac{d_{j,k'}}{r_{k'}}\}}\})/\sum_{j=1}^{n} w_j.$$

The proof of Theorem 2 can be found in Appendix B of Technique Report [14].

By properly configuring the knob ρ according to Theorem 2, QKnober can guarantee that each user can schedule at least as the number of tasks as that under exclusively using its own partition of the system resources with no sharing. Next, we show that QKnober is envy-freeness, namely, no user envies the allocation results of any other users under its allocation.

Theorem 3. ***(Envy Freeness):*** *Every user under the QKnober allocation prefers its own allocation to others.*

The proof of Theorem 3 can be found in Appendix C of Technique Report [14].

We next show that QKnober produces an efficient allocation under which no user can increase its allocation without decreasing that of other users.

Theorem 4. ***(Pareto Efficiency):*** *The QKnober allocation policy is pareto efficient.*

The proof of Theorem 4 is given in Appendix D of Technique Report [14].

5 Implementation of QKnober

In our former discussions of QKnober policy, there are several key assumptions that may not be the case in a real-world computing system. For practical application of QKnober, we need to relax these assumptions by considering complicated and challenging factors for real applications and computing system. In the following, we highlight these challenges and then give our solutions to address them in YARN. Detailed implementation of QKnober can be found in Appendix E of Technique Report [14].

C1:**Online Users with a Finite Number of Tasks.** In the previous discussions, it has assumed that there are an infinite number of tasks for each user at any time. However, in practice, the tasks of users are arriving over time, implying that the number of tasks per user is generally finite at a time.

Iterative QKnober Approach. We can address this problem through a small modification on QKnober as follows. First, we classify all users into two kinds: *active users* (i.e., with pending tasks) and *inactive users* (i.e., with no pending tasks). The system maintains the list of active users, where an inactive user becomes active whenever there arrives a pending task of it. The system performs QKnober allocation iteratively. In each allocation round, the system uses progressive filling approach to allocates resources to active users based on QKnober until one of them has all its pending tasks scheduled. After that, the active user becomes inactive and will not be considered in the following allocation. The system then starts a new allocation round and repeats the above allocation procedure until there is no active user or no sufficient idle resources that can be allocated.

C2:**Heterogeneous and Indivisible Tasks.** In QKnober allocation model, we have assumed that tasks are divisible and all the tasks of a user are homogeneous in their resource demands. However, in the real world, it may not be the case. First, the tasks demands of a user are most likely to be diverse (e.g., different demands between map and reduce tasks of a user's MapReduce job). Second, fractional tasks are often not supported and accepted by existing systems (e.g., MapReduce, Spark).

In QKnober, whether to perform fairness-purpose allocation or to do efficient-purpose allocation is determined by the maximum dominant resource share s_i^{max} and knob factor ρ (See Sect. 4.1). When the demands of all the tasks of a user are homogeneous, the maximum dominant resource share is fixed and can be estimated by Formula (11) for each user. However, in the heterogeneous case, it varies dynamically with the running and new arriving tasks. Moreover, estimating the maximum dominant resource share in such case is *NP-hard.*

Fitness-based Approximation Approach. We propose a heuristic approach based on the First-Fit algorithm [4] as follows. The algorithm first estimates the *current* average resource demand of tasks based on its running and pending tasks for each user. Then, it computes the maximum dominant resource share for each user by using its current average resource demand of tasks with Formula (11). However, in practice, there could be a large number of pending tasks at runtime. It indicates that picking all pending tasks might not reflect the *current* average resource demand of a user. To address it, we instead only consider a certain number of tasks that just fill the remaining idle space of the cloud system. We achieve and update it for current average resource demand by using the First-Fit algorithm dynamically. That is, we count the pending tasks in the queue order until the cloud system can be filled. Then the current average resource demand can be estimated based on the running tasks and those counted pending tasks.

C3:**Heterogeneous and Distributed Computing System.** The QKnober allocation model assumes the computing system as a single super-server, which however may not always be the case. A real-world computing system (e.g., Google production cluster, Amazon EC2) generally consists of many heterogeneous servers with different resource capacities connected via a high-speed network. In this case, scheduling tasks efficiently to the computing system is analogous to the *NP-hard* multi-dimensional knapsack problem [5] mentioned above.

Affinity-based Task Scheduling Approach. We develop a heuristic approach for efficiency-purpose allocation by defining *affinity* of a task relative to the system. That is, when there are some idle resources on a server, we first filter out the set of pending tasks that can be accommodated by that server. We then compute the affinity score for each of these tasks, as the dot product between the task's resource demand and the vector of that server's idle resources. The one with the highest affinity score among all these pending tasks is chosen for scheduling.

6 Experimental Evaluation

6.1 Experimental Setup

Hadoop Cluster. We have implemented QKnober in the version of YARN-2.4.0. We deploy the YARN framework in an Amazon EC2 cluster consisting of 60 Amazon EC2 t2.medium instances each with 2 virtual cores and 4 GB memory. We configure 1 instance as master, and the remaining 59 instances as slaves, each of which is configured with <2 virtual cores, 4 GB>.

Workloads. We run four data-parallel workloads: (1) *Facebook Workload:* It is based on the distribution of jobs sizes and inter-arrival time at Facebook provided by Zaharia et. al. [25]. The workload consists of 100 jobs. It is a mix of large number of small-sized jobs ($1 \sim 15$ tasks) and small number of large-sized jobs (e.g., 800 tasks[1]); (2) *Purdue Workload:* Seven benchmarks (e.g., WordCount, TeraSort, Grep, InvertedIndex, HistogramMovices, Sequence-Count, Self-join) are randomly chosen from Purdue MapReduce Benchmarks Suite [3], with 100G wikipedia data [2] as input data; (3) *Spark Workload:* It is a combination of six algorithms (e.g., PageRank, GaussianMixture, BinaryClassification, Kmeans and Alternating Least Squares (ALS)) using provided example benchmarks; (4) *TPC-H Workload:* To emulate continuous analytic query, such as analysis of users' behavior logs, we ran TPC-H benchmark queries on Hive [1]. 120 GB data are generated with provided data tools.

6.2 Testbed Experimental Results

This section first evaluates the fairness and efficiency of QKnober under different knob values. Then, we compare the performance of QKnober with its alternatives DRF and Tetris. Finally, the overhead evaluation of our QKnober system can be found in Appendix F of Technique Report [14].

Fairness vs. Efficiency We show in Sect. 4.2 that QKnober is an elastic knob-based tradeoff allocation policy that allows users to balance the fairness and efficiency flexibly. In this section, we evaluate the impact of different knob values on the fairness and efficiency with the mix of four workloads experimentally. Suppose that there are four users A, B, C, D with the weighted shares of $1 : 2 : 3 : 4$, each running Facebook, Purdue, Spark and TPC-H workloads on the YARN cluster, respectively. With QKnober policy, we can then maximize the system efficiency while guaranteeing the soft fairness. We define a term called *soft fairness degree* to quantify the soft fairness based on Formula (12). The smaller soft fairness degree indicates the better fairness, and vice versa.

Figure 3 presents the experimental results for QKnober policy under different knob configurations. We compute *speedup* based on the case when the knob is 1.0. The larger value indicates the better performance. It can be observed that

[1] We reduce the size of the largest jobs in [25] to have the workload fit our cluster size.

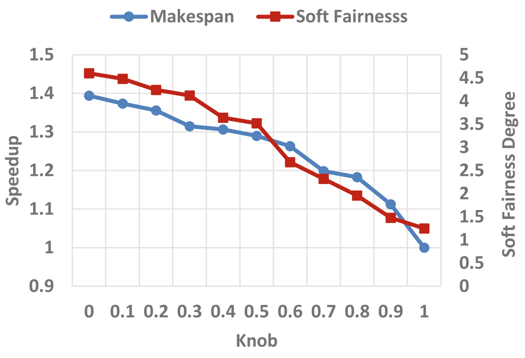
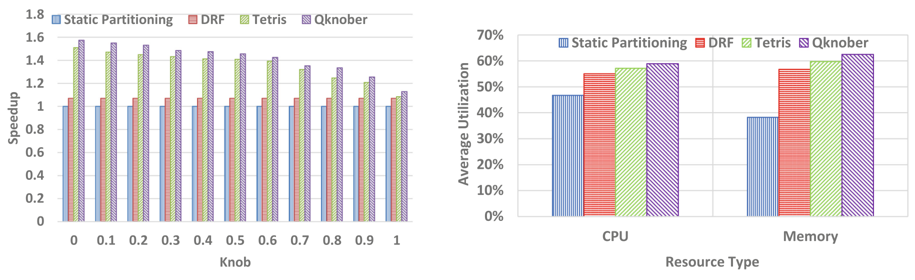


Fig. 3. The system efficiency and soft fairness for QKnober under different knobs, where the speedup is computed over the case of knob $\rho = 1$.

there is a strong tradeoff between fairness and efficiency. When the knob is small, it benefits the system efficiency but harms the fairness. In contrast, when we increase the knob value, the fairness can become better at the cost of system efficiency. It means that users can make their own tradeoff preference over the fairness and efficiency by tuning the knob value.



(a) Speedup (over Static Partitioning). (b) Average resource utilization.

Fig. 4. The comparison results of performance and resource utilization for Static Partitioning (i.e., non-sharing case), DRF, Tetris and QKnober under different knob configurations, where the speedup is computed over that of Static Partitioning.

Performance Evaluation. Figure 4(a) gives the performance results for Static Partitioning, DRF, Tetris and QKnober under different knob configurations, where the speedup is computed over that of Static Partitioning. Particularly, we implement the static partitioning policy by dividing the whole cluster resources (e.g., CPU and memory) into four isolated portions for four workloads according to their weights mentioned in Sect. 6.2, and let them run exclusively without sharing. We have the following observations:

First, resource sharing (e.g., DRF, Tetris and QKnober) performs better than non-sharing (e.g., Static Partitioning). For fairness-only policy DRF, there is about 10% performance improvement over Static Partitioning. In contrast, for fairness-efficiency policies like Tetris and QKnober, the improvement can be up to 57% as we decrease the knob factor ρ from 1.0 to 0.0. The performance gain is mainly due to the resource preemption of unused resources from overloaded users in the sharing case, making the resource utilization higher than the non-sharing case. As illustrated in Fig. 4(b), the resource utilizations for sharing policies (e.g., DRF, Tetris, QKnober) are higher than that of static partitioning. For example, the average cpu utilizations for DRF, Tetris and QKnober are $55\%, 57\%$ and 59%, respectively, higher than the static partitioning of 46%.

Second, QKnober outperforms other baseline allocation policies DRF and Tetris in all knob configurations. Particularly, the reason why QKnober is better than DRF even when the knob is 1.0 is due to its efficient affinity-based task placement in reducing the fragmentation of machines in multi-resource allocation, whereas DRF policy does not have such a concern and simply views all machines as a single super machine. Moreover, both Tetris and QKnober are knob-based fairness-efficiency allocation policies. The reason why QKnober performs better than Tetris is due to their different approaches in the efficiency-purpose allocation. Tetris takes heuristic bin packing approach, whereas QKnober adopts the optimal linear programming method. It makes QKnober achieve a higher resource utilization than Tetris as shown in Fig. 4(b).

7 Related Work

There is a general tradeoff between fairness and efficiency in multi-resource allocation, which has been studied by a lot of research works. Joe-Wong et al. [11] proposed a unifying *mathematical* framework to capture the tradeoff between fairness and efficiency, which are specified by two parameters for a given multi-resource allocation problem. Their work is just a theoretically analytic study and cannot be practically used to real systems such as Hadoop directly. In contrast, our proposed knob-based policy QKnober is practical. We have implemented it in Hadoop that allows users to balance the fairness-efficiency tradeoff flexibly by tuning the knob in the range of $[0, 1]$. Wang et al. [22,24] and Danna et al. [7] studied the fairness-efficiency tradeoff for packet processing consuming both CPU and link bandwidth by proposing a GPS-like fluid model. Tang et al. [17] considered Coupled CPU-GPU architecture by proposing a fairness-efficiency scheduler called EMRF through extending DRF. Wang et al. [21] proposed a bottleneck-aware allocation policy to balance fairness and efficiency for users in multi-tiered storage consisting of SSD and HDD. In contrast, we consider the job scheduling in cloud computing. Tetris [10] is the most closely related work to our work. It is a fairness-efficiency scheduler for cloud computing that balances the performance and fairness by leveraging alignment heuristics to efficiently pack tasks with heterogeneous resource demands to servers. However, it cannot provide us a soft fairness guarantee given a knob setting due to its *unawareness* of fairness degradation (Sect. 3) during its fairness-efficiency scheduling. Moreover, it doesn't satisfy sharing incentive property. In comparison, our proposed knob-based policy QKnober is fairness-sensitive, which maximizes the efficiency while guaranteeing the θ-soft fairness under a knob configuration (See Theorem 1). Additionally, it satisfies sharing incentive, envy freeness and pareto efficiency properties with a proper knob configuration.

8 Conclusion

This work studies the tradeoff between fairness and efficiency for users in a shared computing system. Quantifying the fairness degradation/loss is essential for users to better understand the tradeoff. We show that the knob-based approach is a promising solution to achieving the *flexible* and *elastic* tradeoff balance for users. However, existing knob-based fairness-efficiency schedulers are not aware of fairness degradation during its fairness-efficiency allocation, which either fail to guarantee the QoS of δ-fairness or violate desired properties in Sect. 2. To address it, we develop a new knob-based fairness-efficiency policy called QKnober. It is a fairness sensitive scheduler that allows users to balance the fairness and efficiency with a knob while guaranteeing δ-soft fairness. Typically, we provably show that it meets several desirable properties including sharing incentive, envy freeness and pareto efficiency with a proper knob setting. Finally, we implement QKnober in YARN and our real experiments show that it achieves promised initial results.

Acknowledgement. This work is sponsored by the National Natural Science Foundation of China (61602336, 61772544, U1731125) and Tianjin Natural Science Foundation (18JCZDJC30800).

References

1. Apache TPC-H benchmark on hive. https://issues.apache.org/jira/browse/HIVE-600
2. Puma datasets. http://web.ics.purdue.edu/fahmad/datasets.htm
3. Ahmad, F., Lee, S., Thottethodi, M., Vijaykumar, T.N.: PUMA: Purdue mapreduce benchmarks suite. ECE technical reports (2012)
4. Brent, R.P.: Efficient implementation of the first-fit strategy for dynamic storage allocation. ACM Trans. Program. Lang. Syst. **11**(3), 388–403 (1989)
5. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. J. Heuristics **4**(1), 63–86 (1998)
6. Dageville, B., Zait, M.: SQL memory management in Oracle9i. In: VLDB 2002, pp. 962–973. VLDB Endowment (2002)
7. Danna, E., Mandal, S., Singh, A.: A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In: INFOCOM 2012, pp. 846–854 (2012)
8. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)

9. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: NSDI 2011, Berkeley, CA, USA, pp. 323–336. USENIX Association (2011)
10. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. In: SIGCOMM 2014, pp. 455–466. ACM (2014)
11. Joe-Wong, C., Sen, S., Lan, T., Chiang, M.: Multiresource allocation: fairness-efficiency tradeoffs in a unifying framework. IEEE/ACM Trans. Netw. **21**(6), 1785–1798 (2013)
12. Niu, Z., Tang, S., He, B.: An adaptive efficiency-fairness meta-scheduler for data-intensive computing. IEEE Trans. Serv. Comput., p. 1 (2017)
13. Parkes, D.C., Procaccia, A.D., Shah, N.: Beyond dominant resource fairness: extensions, limitations, and indivisibilities. ACM Trans. Econ. Comput. **3**(1), 3:1–3:22 (2015)
14. Tang, S., Yu, C., Sun, C., Xiao, J., Li, Y.: QKnober: a knob-based fairness-efficiency scheduler for cloud computing with QoS guarantees. Technical report (2018). http://cs.tju.edu.cn/faculty/tangshanjiang/tr/QKnoberTR.pdf
15. Tang, S., Lee, B.S., He, B.: Fair resource allocation for data-intensive computing in the cloud. IEEE Trans. Serv. Comput. **11**(1), 20–33 (2018)
16. Tang, S., Niu, Z., He, B., Lee, B.S., Yu, C.: Long-term multi-resource fairness for pay-as-you use computing systems. IEEE Trans. Parallel Distrib. Syst. **29**(5), 1147–1160 (2018)
17. Tang, S., He, B.S., Zhang, S., Niu, Z.: Elastic multi-resource fairness: balancing fairness and efficiency in coupled CPU-GPU architectures. In: SC 2016, Piscataway, NJ, USA, pp. 75:1–75:12. IEEE Press (2016)
18. Tang, S., Lee, B.S., He, B., Liu, H.: Long-term resource fairness: towards economic fairness on pay-as-you-use computing systems. In: ICS 2014, New York, NY, USA, pp. 251–260. ACM (2014)
19. Varian, H.R.: Equity, envy, and efficiency. J. Econ. Theory **9**(1), 63–91 (1974)
20. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S.: Apache hadoop YARN: yet another resource negotiator. In: SOCC 2013, New York, NY, USA, pp. 5:1–5:16. ACM (2013)
21. Wang, H., Varman, P.: Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In: FAST 2014, Berkeley, CA, USA, pp. 229–242. USENIX Association (2014)
22. Wang, W., Feng, C., Li, B., Liang, B.: On the fairness-efficiency tradeoff for packet processing with multiple resources. In: CoNEXT 2014, New York, NY, USA, pp. 235–248. ACM (2014)
23. Wang, W., Li, B., Liang, B.: Dominant resource fairness in cloud computing systems with heterogeneous servers. In: 2014 Proceedings IEEE INFOCOM, pp. 583–591, April 2014
24. Wang, W., Ma, S., Li, B., Li, B.: Coflex: navigating the fairness-efficiency tradeoff for coflow scheduling. In: INFOCOM 2017
25. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: EuroSys 2010, New York, NY, USA, pp. 265–278. ACM (2010)
26. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Hot Cloud 2010, vol. 10, p. 10 (2010)

Energy-Efficient and Quality of Experience-Aware Resource Provisioning for Massively Multiplayer Online Games in the Cloud

Yongqiang Gao(✉), Lin Wang, Zhulong Xie, Wenhui Guo, and Jiantao Zhou

Inner Mongolia University, Hohhot, China
csgyq@outlook.com, csimuwl@outlook.com, csxzl@outlook.com, csgwh@outlook.com, cszjt@outlook.com

Abstract. Massively Multiplayer Online Games (MMOGs) routinely have millions of registered players and hundreds of thousands of active concurrent gamers. To guarantee quality of experience (QoE) to a highly variable number of concurrent players, MMOG infrastructure have converted nowadays into cloud computing paradigm. Many leading MMOG companies have begun to build increasing numbers of energy hungry data centers for running the MMOG services requested by the players. A main challenge for MMOG service providers is to find the best tradeoff between two contradictory aims: improving the QoE and reducing energy costs. In this paper, we propose a dynamic resource provisioning scheme for large-scale MMOG services implemented on top of cloud infrastructures which takes advantage of both virtual machine resizing and server consolidation to achieve energy efficiency and desired QoE requirements. Our experimental results indicate that, compared to an over-provisioning of infrastructural resources, our resource provisioning scheme can achieve up to 54.5% energy savings while providing the just-good-enough QoE to gamers under rapidly changing workloads.

Keywords: Cloud computing · MMOG · Dynamic resource provisioning

1 Introduction

Nowadays, Massively Multiplayer Online Games (MMOGs) has attracted a lot of attention from both academia and industry. A recent report [1] states that the number of MMOG players have grown to 20 million worldwide persons in 2010. A MMOG is a seamless virtual world where millions of world-wide participants play roles and interact with their surroundings via avatars, which are virtual roles. There are several types of MMOGs, including MMORPG (Massively Multiplayer online role-playing games, e.g., World of Warcraft), MMOFPS (Massively Multiplayer Online First Person Shooter, e.g., Firefall), MMORTS (Massively Multiplayer Online Real Time Strategy, e.g., Boom Beach) and so on.

Online Gaming used to be implemented as a client-server architecture, where a central game server is responsible to handle commands from players and returns state updates

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 854–869, 2018.
https://doi.org/10.1007/978-3-030-03596-9_61

about the game session to its connected clients. In order to provide good quality of experience to players, MMOG providers used to overly provision game servers in order to comply with the delay aware constraint. However, the static resource provisioning method would lead to resource waste during slower gaming times and increase MMOG providers' total operational costs. To address this problem, MMOG operators have changed the system architecture from client/server to cloud computing. In MMOG clouds, games are stored and run on remote cloud servers. After receiving a player request, the MMOG cloud platform will allocated a virtual machine (VM) to run the player's requests.

A main challenge for MMOG service providers is to find the best tradeoff between two contradictory aims: improving the quality of experience (QoE) and reducing the cost of energy consumption by MMOG infrastructures. On the one hand, they need to offer sufficient number of high-performance servers that enable high quality of experiences that lead to player satisfaction and loyalty. On the other hand, they have to reduce energy consumption as much as possible in order to reduce the total cost of ownership and increases the return on investment of cloud infrastructures. For example, leading MMOG companies operate over 10,000 servers and cost over $50 million per year [2].

In this paper, we propose a dynamic resource provisioning scheme that is able to automatically manage physical resources of a MMOG cloud infrastructure in such a way to minimize the energy cost of MMOG service providers while achieving just-good-enough QoE for game players. More specifically, the primary contributions of this paper are as follows:

- We consider the problem of dynamic resource provisioning from the perspective of MMOG operators with their own datacenter, and aim at cutting down the energy cost of MMOG operators, by combines the global resource allocation among physical servers with the QoE-driven dynamic resource distribution at physical server level.
- We formulate the problem into a constrained optimization problem and utilize the genetic algorithm to solve the problem. Our proposed optimization model considers multiple types of resources, including CPU, memory and network bandwidth, and the power consumption incurred due to migrating virtual machines.
- We conduct extensive experiments using real-world data to evaluate the effectiveness of our resource provisioning policy. Our experimental results show that, compared with other alternatives, our resource provisioning policy can save at least 9.8% of energy consumption, while providing even better QoE for game players.

The rest of this paper is structured as follows. The related works are summarized in Sect. 2. In Sect. 3, the system model is introduced. Our proposed genetic algorithm is presented in Sect. 4. In Sect. 5, the experimental evaluations and results are discussed. Finally, conclusions are made and future work is discussed in the last section.

2 Related Work

In recent years, extensive efforts have been put into the research of enhancing the performance of cloud gaming systems. Wang et al. [3] presented a load assignment solution for cloud-based distributed interactive applications in order to minimize the inter-action delay among clients. Choy et al. [4] proposed a hybrid edge-cloud architecture for reducing the processing delay on the server side. Chen et al. [5] developed a heuristic algorithm to solve the inter-player delay optimization problem that aims at minimizing this inter-player delay, while preserving good-enough absolute response delay experienced by players. These studies are different from our work because they focus solely on improving the performance of cloud gaming systems and thus cannot provide energy savings.

There is a huge body of work on how to provide a tradeoff between monetary costs and the user QoE for hosting cloud gaming in public clouds. Wang ea al. [6] proposed two practical online algorithms, one deterministic and another randomized, that dynamically combine the two instance to serve time-varying demands at minimum cost. Nae ea al. [7] proposed a service-level-agreement aware cost-effective model for hosting and operating MMOGs based on cloud-computing principles. Basiri et al. [8] addressed the resource provisioning problem for cloud gaming to minimize the operational costs of the cloud gaming while guaranteeing the QoE for users. In contrast to our approach, these approaches are not designed for the private cloud environment. That is, they have acquired resources (virtual machines) from public clouds to serve as rendering and game servers.

Some server provisioning issues for cloud gaming providers with their own datacenter have been studied recently. Wu et al. [9] presented an online control algorithm to perform intelligent request dispatching and server provisioning. The objective of their work is to reduce the provisioning cost of cloud gaming service providers while still ensuring the user QoE requirements. Hong et al. [10] studied the VM placement problems for maximizing the total net profit for service providers while maintaining just-good-enough gaming QoE, Lee et al. [11] proposed a zone-based server consolidation solution for MMORPGs, which exploits the unique spatial locality property of gamers' interactions, to reduce hardware investment and energy consumption while maintaining user-perceived service quality. In contrast, our scheme focuses on CPU, memory and network resources, and combines the global resource allocation among physical servers with the QoE-driven dynamic resource distribution at physical server level in order to guarantee the QoE for game players while minimizing power consumption.

3 Problem Formulation

3.1 System Overview

We consider a large-scale cloud infrastructure to support MMOG. A virtualized computing environment is assumed, where there are multiple MMOGs hosted in VMs. The MMOG clouds use two techniques to partition a virtual game world for parallelizing: zoning and replication. Zoning partitions the game world into adjacent areas to be handled independently by separate VMs. Replication copies the game area onto different VMs when the game area has a large number of avatars interacting with each other. The avatars served by a VM are called active entities. The avatars which are active in the other VMs are called shadow entities. The state of active entities and shadow entities are periodically synchronized across VMs. Multiple VMs can run on the same physical machine (PM).

The architecture of our resource provisioning scheme is shown in Fig. 1(a). The *Historical Database* module is employed to collect history data of game players and send collected data to the *Load Predictor*. Next, the *Load Predictor* module uses neural network to predict the number of players in the next control period by history data. The *Energy Optimizer* module is then updated using the predicted number of players hosted by each VM and a new instance of the optimization problem is constructed. The *reconfiguration* module utilizes a capacity model to estimate the resource needs of VMs and implements a genetic algorithm to solve the new optimization problem instance, generating a new resource provisioning scheme (i.e. how much resource is allocated to each VM, which servers must be active and which VMs can be placed on which server). The above optimization process is periodically performed.

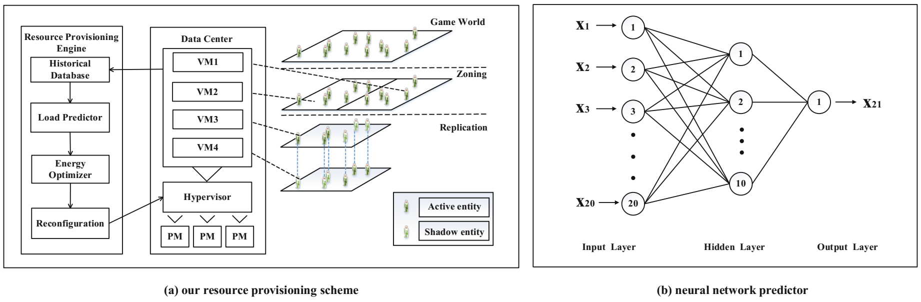


Fig. 1. The architecture of our resource provisioning scheme and neural network predictor

3.2 Load Prediction

The number of players hosted by a game area (VM) can change over time. It is difficult to determine the number of players by analytical means, therefore we employ a neural network prediction model to predict the number of players in the near future. Figure 1 (b) illustrates the architecture of our load predictor. The load predictor has two hidden layers and one output layer. We input the number of players from time slot 1 to time slot 20 and predict the number of players at time slot 21. The hidden layers has 10 neurons. Each layer's neuron performs calculations based on the following equations [12]:

$$k_j = g_1\left(\sum\nolimits_{i=1}^{20} x_i w_{i,j} + b_j\right) \quad \forall j \in \{1, 2, \ldots, 10\} \tag{1}$$

$$x_{21} = g_2\left(\sum\nolimits_{i=1}^{10} k_i w'_i + b'\right) \tag{2}$$

$$g_1(y) = \frac{1}{1 + e^{-y}} \tag{3}$$

$$g_2(y) = y \tag{4}$$

where $w_{i,j}$ and w'_i are the connection weights between hidden layer and output layer, $g_1(y)$ and $g_2(y)$ are is the activation function, and b_j and b' are biases in the hidden layer and the output layer, respectively. In addition, $x_i(i = 1, 2, ..., 20)$ and k_i are an input of the load predictor and the output layer, respectively. x_{21} is an output. The Levenberg-Marquardt algorithm is employed for the network training and the neural network predictor is trained only once in an offline fashion.

3.3 Delay Model

Interaction delay is the most critical user-perceived QoE metric for the MMOG cloud. We define the interaction delay as the lag between the time the client sends a player's command to the VM and the time the corresponding game frame is displayed to the player. Interaction delay mainly consists of network delay, processing delay and playout delay. Network delay is essentially the network round-trip time, which can be measured by tools such as Ping. Processing delay represents the time required by VM to receive and process a player's command, and to encode and packetize the corresponding game frame for the client. As for playout delay, it is the time for the client to receive, decode, and display the encoded frame. Because playout delay is usually constant and occur at the client side, we do not consider it in our model for the sake of brevity. Finally, we define the interaction delay $ID_{i,j}$ of a game player i connected to a VM j as the sum of processing delay $PD_{i,j}$ and network delay $ND_{i,j}$, which can be represented as below

$$ID_{i,j} = ND_{i,j} + PD_{i,j} \tag{5}$$

$$ND_{i,j} = D_{i,j}^{player-VM} + D_{i,j}^{VM-player} + 2D_{i,j}^{propagation} \tag{6}$$

$$D_{i,j}^{propagation} = S_{i,j}/R_l \tag{7}$$

$$D_{i,j}^{player-VM} = L_i/R_i^{ul} \tag{8}$$

$$D_{i,j}^{VM-player} = L_j/R_i^{dl} \tag{9}$$

In these expressions, R_i^{ul} and R_i^{dl} are the uplink and downlink data rate of player i, $S_{i,j}$ is the distance between user i and VM j, R_l is the speed of light, $D_{i,j}^{player-VM}$ is the transmission delay for player's data sent to VM, and $D_{i,j}^{VM-player}$ is the transmission delay for VM's data sent to the player. $D_{i,j}^{propagation}$ represents propagation delay between player i and VM j. L_i is the player's data-packet length, which is transmitted to VM, and L_j is the VM's data-packet length, which is transmitted to player.

In order to mathematically express the processing delay, each VM is modeled as a G/G/1 queuing system to deal with an arbitrary arrival distribution and service time distribution. The queuing discipline is assumed to be first-comefirst-serve (FCFS).

According to the queuing foundations, the average processing time of requests served at VM j can be calculated using the following equation

$$PD_{i,j} = \frac{\lambda_j \cdot \left(\delta_1^2 + \delta_2^2\right)}{2 \cdot \mu_j \cdot \left(\mu_j - \lambda_j\right)} \tag{10}$$

where λ_j is players arrival rates and μ_j is the average service time. δ_1^2 and δ_2^2 are the variance of service time and the variance of inter-arrival time, respectively.

3.4 VM Capacity Model

We propose in this subsection an analytical model for VM capacity in MMOG clouds. Our model considers three main types of resources used by VMs: CPU, memory, and network bandwidth.

Previous work [13] shows that CPU capacity is approximately proportional to its throughput. Thus, the CPU capacity R_j^{CPU} of VM j can be modeled as:

$$R_j^{CPU} = e \cdot t_j \tag{11}$$

where e is an experimentally determined model parameter and $t_j = 1/\mu_j$ represents CPU throughput.

Inspired by the linear capacity model in [14], we can define the memory capacity R_j^{mem} for a VM j as follow

$$R_j^{mem} = \sum\nolimits_{i=1}^{|v_j|} AE_i \cdot m_{cs} + BE_j \cdot m_{es} + m_{game} + m_{world} \tag{12}$$

where v_j is the set of VMs serving in the same game area as VM j, AE_i is the number of the avatars for VM i, BE_j is the number of the non-player characters for VM j, m_{cs} is the amount of memory needed to store the state of one avatar, m_{es} is the amount of memory needed to store the state of an non-player character, m_{game} is the amount of memory needed to run the actual game engine with no game world loaded and m_{world} is the amount of memory used for the game world being played.

Each VM has an incoming and outgoing network bandwidth capacity. As in [14], we define the outgoing network bandwidth capacity R_j^{out} as follows

$$R_j^{out} = \frac{AE_j \cdot d_{out} + \left(|v_j| - 1\right) \cdot \left(AE_j + BE_j\right) \cdot d_{updt}}{T_s} \tag{13}$$

where d_{out} represents the amount of data sent to a client, d_{updt} is the amount of data exchanged between VMs for updating a single entity state and T_s is the control time period.

Similarly, the incoming network bandwidth capacity R_j^{in} for a VM j is defined as [14]

$$R_j^{in} = \frac{AE_j \cdot d_{in} + \sum_{i=1,i\neq j}^{|v_j|} (AE_i + BE_i) \cdot d_{updt}}{T_s} \tag{14}$$

where d_{in} is the amount of data received from a client.

3.5 Power Model

The processor is one of the largest power consumers in today's servers. Recent study has shown that the server power consumption can be accurately described by a linear relationship between the power consumption and CPU utilization [15]. In order to save energy, servers are switched off when they are idle. Because today's server use very little power when in power-off mode, we neglect the power consumption during power-off mode. Finally, the server power consumption P_i can be modeled as

$$P_i = \begin{cases} (P_i^{max} - P_i^{min}) \cdot U_i^{CPU} + P_i^{min} & U_i^{CPU} > 0 \\ 0 & otherwise \end{cases} \tag{15}$$

where U_i^{CPU} is CPU utilization of server i, P_i^{min} and P_i^{max} are the average power consumption when the server i is idle and fully utilized, respectively.

Similar to previous work [16], we also consider the power consumption costs incurred when moving VMs. The power consumption $MP_j^{s,i}$ of migrating a VM j from a server s to a server i can be expressed as follows:

$$MP_j^{s,i} = v_{s,i} \cdot R_j^{mem} + z_{s,i} \tag{16}$$

where $v_{s,i}$ and are $z_{s,i}$ experimentally determined model parameters.

3.6 Optimization Model

Suppose that we are given N VMs $i \in NV$ that are to be placed on M physical servers $j \in NP$. Let CN_j be the number of players connected to VM j, ID^{th} be max tolerable interaction delay of players, $R_i^{CT}, R_i^{mT}, R_i^{oT}$, and R_i^{iT} are the resource threshold of CPU, memory, outgoing and incoming network bandwidth associated with each server respectively. The decision variable $x_{i,j}$ indicates if VM i is assigned to physical server j. Let $\overline{p_j}$ be an index of the server hosing VM j in the previous control time period and $\overline{x_{i,j}}$ be the values of the variables $x_{i,j}$ in the previous control time period. With the models defined above, we formulate the power optimization problem as:

$$min \quad \left[\sum_{i=1}^{M} P_i + \sum_{i=1}^{M} \sum_{j=1}^{N} \left[MP_j^{\overline{p_j},i} \cdot \max\left(0, x_{i,j} - \overline{x_{i,j}}\right)\right]\right] \tag{17}$$

$$s.t. \qquad \sum\nolimits_{j=1}^{N} x_{i,j} \cdot R_j^{CPU} \le R_i^{CT} \qquad \forall i \in NP \tag{18}$$

$$\sum\nolimits_{j=1}^{N} x_{i,j} \cdot R_j^{mem} \le R_i^{mT} \qquad \forall i \in NP \tag{19}$$

$$\sum\nolimits_{j=1}^{N} x_{i,j} \cdot R_j^{out} \le R_i^{oT} \qquad \forall i \in NP \tag{20}$$

$$\sum\nolimits_{j=1}^{N} x_{i,j} \cdot R_j^{in} \le R_i^{iT} \qquad \forall i \in NP \tag{21}$$

$$\sum\nolimits_{i=1}^{M} x_{i,j} = 1 \qquad \forall j \in NV \tag{22}$$

$$ID_{p,j} \le ID^{th} \qquad \forall j \in NV, \forall p \in CN_j \tag{23}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in NP, \forall j \in NV \tag{24}$$

The objective function in Eq. (17) minimizes the power consumed by the MMOG cloud. Equations (18), (19), (20) and (21) impose CPU, memory, outgoing and incoming network bandwidth constraints on each sever, respectively. Equation (22) ensures that each VM is assigned to only one of the servers. Equation (23) ensures the QoE requirement. Equation (24) defines the domain of the variables of the problem.

4 Genetic Algorithm Design and Analysis

Due to the NP-Hard nature of the optimization problem described above, it is not possible to develop an algorithm to find the best solutions in practically acceptable times. This section will show how to apply a genetic algorithm to efficiently search for good solutions in large solution spaces. The proposed genetic algorithm is mainly related to the following elements: (1) chromosome encoding; (2) crossover; (3) mutation; (4) fitness function; (5) selection strategy. Specific implementation of these elements results in distinct genetic algorithms with varying degrees of success.

4.1 Chromosome Encoding

A chromosome in the proposed genetic algorithm is composed of an N-by-M matrix. The columns of the matrix correspond to the different physical machines (PMs) and the rows correspond to the different VMs. The elements of the matrix represent genes. The value of a gene is 0 or 1, representing if a VM is assigned to a physical machine. Figure 2(a) shows an example VM placement and its corresponding chromosome.

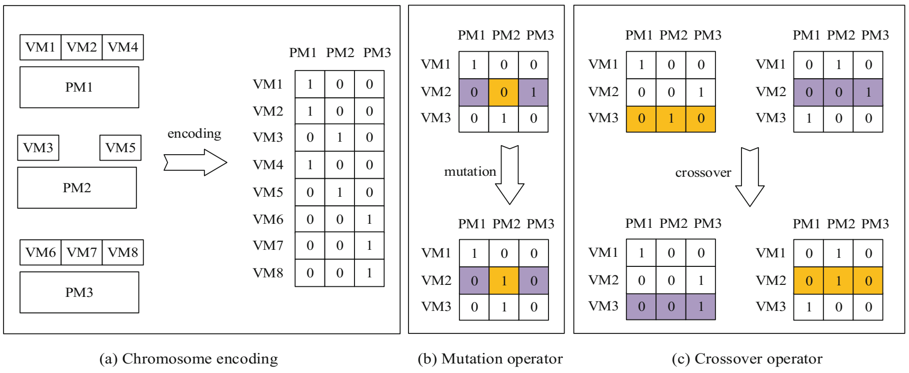


Fig. 2. Chromosome encoding, crossover and mutation operators for our genetic algorithm

4.2 Crossover

Crossover is a genetic operator that aims to combine the better characteristics among the preferred chromosomes. At present, various crossover methods have been developed for particular problems to provide effective implementation of genetic algorithms.

The proposed genetic algorithm adopts a single-point crossover operator, which is described in Fig. 2(c). The two parent chromosomes swap a row of genes randomly selected with each other, so that produce new chromosomes.

4.3 Mutation

The mutation operator is similar to hill-climbing method, where a small change is made to a current solution in order to explore its neighborhood solution in the search space. The idea behind mutation is to introduce some extra genetic variability into the population. In our work, the mutation operator firstly randomly selects a gene in the chromosome and then inverts the value of the chosen gene. Finally, the value of other genes in the same row is either 0 or 1 satisfying the condition that the sum of the value of all genes in the row is equal to 1. Figure 2(b) shows how the mutation operator operates.

4.4 Fitness Function

The role of the fitness function is to numerically measure the performance of the chromosome. For real-world applications of genetic algorithms, choosing the fitness function is the most important step. In this paper, the fitness *f(x)* of an individual *x* in the population of the proposed genetic algorithm is designed in Eq. (25) below:

$$f(x) = \begin{cases} E_{min}/E(x) & if\, x\, is\, fesible \\ E_{min}/E(x) + \infty & otherwise \end{cases} \tag{25}$$

Where *E* represents the objective function, *E(x)* is the power consumption of the current solution *x* and E_{min} represents the power consumption of the best solution in the previous population. The fitness function penalizes an infeasible solution, and make sure that the fitness value of any feasible solution is more than that of any infeasible solution and that the less power consumption and the greater the fitness value is.

4.5 Selection Strategy

The selection strategy copes with which of the chromosomes in the current population will be used to reproduce child in hopes that offspring will have even higher fitness. We employ rank-based roulette wheel selection as a selection mechanism. Rank-based selection strategies firstly sort individuals in the population according to their fitness and then computes selection probabilities according to their ranks rather than fitness values. Let *NS* be the number of solutions in each population in genetic algorithm. Then the selection probability, *P(x)* for individual *x* is define as:

$$P(x) = \frac{R(x)}{\sum_{i=1}^{NS} R(x)} \tag{26}$$

In Eq. (26), $R(x)$ represents the rank of individual x and can be scaled linearly using the following formula [17]

$$R(x) = 2 - SP + \left[2 \times (SP - 1) \times \frac{(L(x) - 1)}{(NS - 1)} \right] \tag{27}$$

Where *SP* is the selective pressure and limited to the range [1, 2]. *L(x)* represents the position of individual *x* in the sorted population.

4.6 The Description of the Proposed Genetic Algorithm

Based on the above definitions, the proposed genetic algorithm can be summarized in the following steps.

1. Randomly generate an initial population of *NS* chromosomes.
2. Select *NS/2* pairs of chromosomes from a current population according to the selection strategy.
3. Apply the crossover operators to each of the selected pairs in Step 2 to generate *NS* chromosomes with a predefined crossover probability *CP*.
4. Apply the mutation operators to each of the generated *NS* chromosomes with a predefined mutation probability *MP*.
5. Randomly remove one chromosome from the current population and add the best chromosome in the previous population to the current one.
6. If the maximum number *MI* of iterations is reached, stop this algorithm. Otherwise, return to Step 2.

5 Performance Evaluation

5.1 Experimental Environment

Based on open source software and commodity hardware, we have implemented a MMOG cloud that has 6 physical servers of two different types, including 3 IBM Flex System x220 servers, 3 IBM Flex System x440 servers. Their parameters are listed in Table 1. All the physical servers are connected through a 10 Gigabit Ethernet network. We adopt Xen Server as the virtualization software on physical servers and utilize OpenStack to create and manage VMs. There are 62 VMs running on 6 physical servers in the MMOG cloud. These VMs are divided into three finer categories: a proxy server for dispatching requests and a monitor server as well as 60 game servers. Xen's credit scheduler, Xen's balloon driver and weight-based proportional sharing [18] are used to dynamically adjust the allocation of CPU, memory and network resources needed by VMs, respectively. The proxy server and monitor server are both equipped with single Intel Xeon E5-4620 2.2 GHz core and 2 GB RAM. Network bandwidth between the proxy server and the player's computer is 100 Mbps, while network bandwidth between the proxy server and game servers is 10 Gbps.

Table 1. Specification of two types of server used in our testbed

Type	Processor	Memory	Boot time(s)	P_i^{max}(W)	P_i^{min}(W)
1	Two octa-core Intel Xeon E5-2420 1.9 GHz processors	16 GB	102	196	91
2	Two octa-core Intel Xeon E5-4620 2.2 GHz processors	32 GB	90	261	102

We employ the Ganglia Monitoring System [19] to monitor and collect the information of the MMOG cloud. Therefore, Ganglia Monitoring Daemon is installed on each game server to monitor its status, and Ganglia Meta Daemon is installed on the monitor server to collect the monitoring information. We select the BZFlag MMOFPS [20] and the Stendhal MMORPG [21] as the example applications for the evaluation of the proposed resource provisioning strategy. The BZFlag MMOFPS is deployed on 35 VMs and the Stendhal MMORPG is deployed on 25 VMs. We utilize the API provided by the KBEngine [22] game engine to implement an account generator that is responsible for creating a large number of players for the experimental game, and a proxy player machine that is responsible for generating the action requests and automatically sending them on behalf of the players to the game server. In addition, players can move across game maps dynamically. The power consumption of the cloud infrastructures is measured with a WattsUp Pro power meter, which has an accuracy of $\pm1\%$ of the measured value.

The workloads used in our experiments are generated based on the web traces from World of Warcraft (WoW) website [23]. We choose two kinds of 284-min traffic pattern from the traces and then scaled them to the range of the number of players that

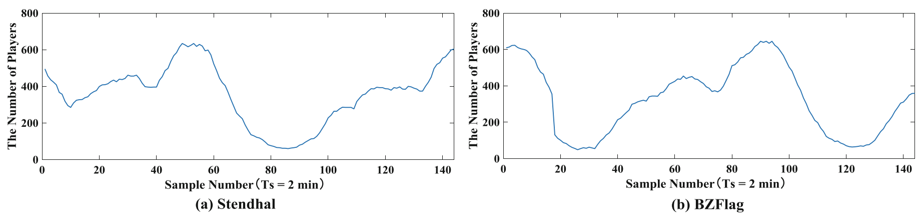


Fig. 3. Workload traces for two MMOG applications

our experimental setup can handle. Figure 3 shows these scaled workloads and the data are sampled every 2 min. The setting values for various parameters in the genetic algorithm have a direct effect on the algorithm performance. Appropriate parameter values were determined on the basis of preliminary computational experiments. The final parameter settings were determined to be *NS = 50, CP = 0.5, MP = 0.9* and *MI = 100.*

We choose a control interval T_s of two minutes. The number of control periods is 142. Every two minutes, our resource provisioning scheme controls VM resizing and the number of servers running in active and power-off modes according to system workload. When reconfiguring the MMOG cloud, the overhead of the physical server boot time must be considered since it greatly affects the performance during boot time. In this work, we use the double control periods (DCP) model proposed by Zheng et al. [24] to compensate the overhead. One control period is responsible for adjusting the number of active physical servers, and the other control period helps to switch on the additional servers in advance. Finally, in order to reduce the total migration time of all migrated VMs, we utilize simultaneous migrations to shorten the total migration time of all migrated VMs, that is, the MMOG cloud can perform multiple VM migrations simultaneously on different servers as long as these servers are not busy with other VM migrations.

5.2 Prediction Model Validation

In this section, we compare the neural network prediction against the well-known ARMA model. To evaluate the accuracy of the prediction models, we define two metrics, the mean absolute percentage error (*MAPE*) and the coefficient of determination $\left(\mathrm{R}^2\right)$, which is calculated as follows:

$$MAPE = \frac{1}{n}\sum\nolimits_{i=1}^{n}\left|\frac{\gamma_i - \widehat{\gamma}_i}{\gamma_i}\right| \tag{28}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(\gamma_i - \widehat{\gamma}_i\right)^2}{\sum_{i=1}^{n}\left(\gamma_i - \widehat{\gamma}\right)^2} \tag{29}$$

where n is the total number of samples, γ_i and $\widehat{\gamma}_i$ are the real value and the model-predicted value, and $\bar{\gamma}$ is the sample mean of γ_i. We run our neural network predictor

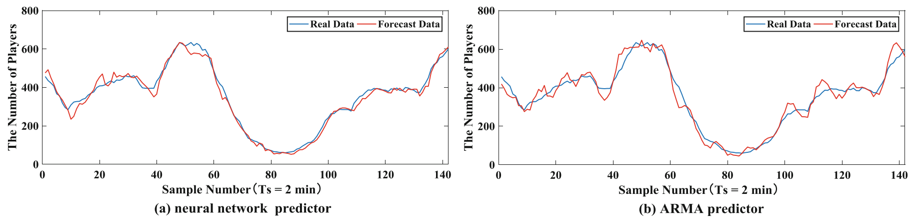


Fig. 4. Workload prediction using ARMA model and neural network model

Table 2. Predictive accuracy of the ARMA predictor and the neural network predictor.

Prediction method	MAPE	R^2
ARMA predictor	0.0893	0.9539
Neural network predictor	0.0583	0.9771

and the ARMA predictor on MATLAB and use the traffic pattern for the Stendhal MMORPG as the input data of two predictors. The predicted results are shown in Fig. 4 and Table 2. The result shows that our neural network predictor is significantly better than the ARMA model and further validates the effectiveness of the proposed neural network predictor.

5.3 Evaluating Effectiveness

To evaluate the effectiveness of our resource management scheme, we compared the power consumption and performance for our policy and for two other policies including a Linux performance policy, and a power and delay aware placement policy (QDH). The performance policy is a standard Linux baseline policy that sets the CPU speed to the highest available frequency all the time. The performance policy provides no VM consolidation and reproduces the behavior of a typical MMOG cloud. The QDH policy utilizes a heuristic proposed by Hong et al. [10] to dynamically consolidate VMs in order to minimize power consumption while maintaining just-good-enough gaming QoE. We conduct a set of experiments using two MMOG applications deployed in 60 VMs. The VM placement for the performance policy is statically configured such that interaction delay targets can be met under the peak workload. The interaction delay target for two MMOG applications are set to 0.1 s and 0.2 s, respectively.

The interaction delay results for three schemes are shown in Fig. 5(a) and (b). The horizontal straight line represents the interaction delay targets. As expected, the performance scheme meets desired QoE targets at any time. In comparison, our policy and QDH policy maintain substantially interaction delay below a predefined threshold. However, compared to the QDH policy, our approach is able to provide better assurance of interaction delay targets in the face of dynamically changing workloads. The reason is that we develop more comprehensive system models which take account of three types of resources and heterogeneous server types. The better performance of our

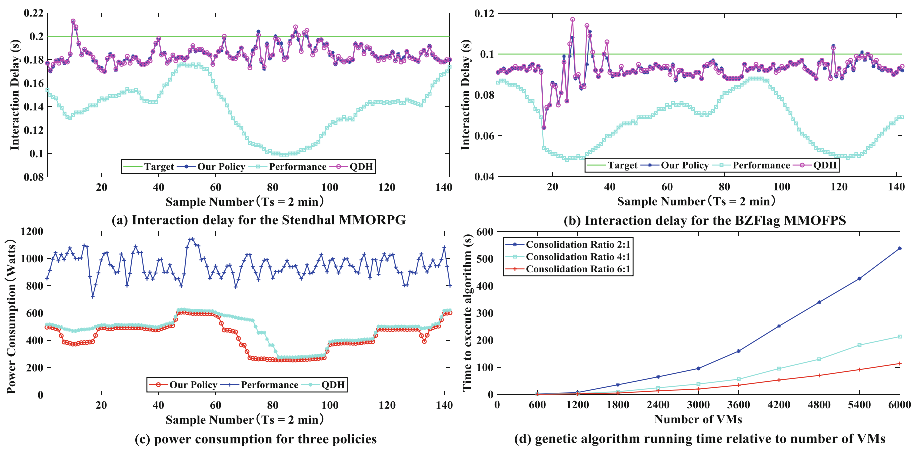


Fig. 5. The results of effectiveness and scalability evaluation

Table 3. Comparison of three resource provisioning policies

Approach	Energy consumption	QoE degradation
Performance policy	4.4564 kWh	0%
QDH policy	2.2464 kWh	5.63%
Our policy	2.0249 kWh	3.17%

policy compared to the QDH policy is also confirmed by a smaller value of QoE degradation shown in Table 3. The QoE degradation is defined as a ratio of the number of players missing their interaction delay targets to the total number of players served over the experiment.

Figure 5(c) and Table 3 show the power consumption and the energy consumption during a 284-min period for three policies, respectively. As can be seen from Table 3, the energy consumption reduction from our policy to the performance policy is about 54.5%. This is because, in the case of low loads, VMs are consolidated using live migration, hence a smaller number of the physical servers are running while other servers are turned off, thus gives us significant energy saving. Comparing our policy with the QDH policy, the results indicate an energy reduction of about 9.8%. This small difference is mainly obtained by finding better solution for VM placement problem.

In summary, compared to the performance scheme, our policy provides significant energy savings with sacrificing a little performance. Compared to the QDH policy, our policy provides better performance and saves more energy.

5.4 Evaluating Scalability

The last set of experiments is used to examine whether the proposed resource provisioning is scalable for large size of the MMOG cloud with thousands of VMs. The scalability of our resource provisioning scheme is determined by the computational complexity of the genetic algorithm for VM placement problem. Since the consolidation ratio, which is defined as the average number of VMs hosted per server, is an important factor which affects the time to solve the VM placement problem, we considered three type of consolidation ratio for this experiment: 2:1, 4:1 and 6:1. The execution time of the genetic algorithm was measured on a 3.8 GHz Intel Core i7 machine. Figure 5(d) shows the computation time curve for three consolidation ratio. From the figure we can see that the execution time of the genetic algorithm increases as the consolidation ratio decreases. It is because that, when the consolidation ratio decreases, the number of physical servers accommodating VMs rises up, making the placement problem harder. In addition, we can see that the genetic algorithm takes less than 10 min to solve the difficult placement problem of up to 6000 VMs. Therefore, our resource provisioning scheme can be suitable for large-scale MMOG clouds.

6 Conclusion

In this paper, we formulate the problem of dynamic resource provisioning from the perspective of cloud gaming providers with their own datacenter into a constrained optimization problem and utilize the genetic algorithm to solve the problem. Our experimental results show that, compared with other alternatives, our resource provisioning policy provides significant energy savings while achieving just-good-enough QoE for gamers. As continuity of this work, we propose to use the social structure of the players in online games to predict the number of players.

Acknowledgments. This work was supported by the National Natural Science Foundation of China under Grant No. 61662052, Inner Mongolia Science and Technology Innovation Team of Cloud Computing and Software Engineering and Inner Mongolia Application Technology Research and Development Funding Project.

References

1. Wu, J.: The world of MMORPG: a tale of two regions. Strategy analytics report (2010)
2. Song, S., Ryu, K.D., Da Silva, D.: Blue Eyes: scalable and reliable system management for cloud computing. In: IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8. IEEE (2009)
3. Wang, H., Shea, R., Ma, X., Wang, F., Liu, J.: On design and performance of cloud-based distributed interactive applications. In: 22nd International Conference on Network Protocols, pp. 37–46. IEEE (2014)
4. Choy, S., Wong, B., Simon, G., Rosenberg, C.: A hybrid edge-cloud architecture for reducing on-demand gaming latency. Multimed. Syst. **20**(5), 503–519 (2014)
5. Chen, Y., Liu, J., Cui, Y.: Inter-player delay optimization in multiplayer cloud gaming. In: 9th International Conference on Cloud Computing, pp. 702–709. IEEE (2016)
6. Wang, W., Li, B., Liang, B.: To reserve or not to reserve: optimal online multi-instance acquisition in IaaS clouds. In: 10th International Conference on Autonomic Computing, pp. 13–22. USENIX (2013)

7. Nae, V., Prodan, R., Iosup, A.: SLA-based operations of massively multiplayer online games in clouds. Multimed. Syst. **20**(5), 521–544 (2014)
8. Basiri, M., Rasoolzadegan, A.: Delay-aware resource provisioning for cost-efficient cloud gaming. IEEE Trans. Circ. Syst. Video Technol. **28**(4), 972–983 (2018)
9. Wu, D., Xue, Z., He, J.: iCloudAccess: cost-effective streaming of video games from the cloud with low latency. IEEE Trans. Circ. Syst. Video Technol. **24**(8), 1405–1416 (2014)
10. Hong, H.J., Chen, D.Y., Huang, C.Y., Chen, K.T., Hsu, C.H.: Placing virtual machines to optimize cloud gaming experience. IEEE Trans. Cloud Comput. **3**(1), 42–53 (2015)
11. Lee, Y.T., Chen, K.T.: Is server consolidation beneficial to MMORPG? A case study of World of Warcraft. In: 3rd International Conference on Cloud Computing, pp. 435–442. IEEE, July 2010
12. Azoff, E.M.: Neural Network Time Series Forecasting of Financial Markets. Wiley, Hoboken (1994)
13. Subramanian, C., Vasan, A., Sivasubramaniam, A.: Reducing data center power with server consolidation: approximation and evaluation. In: International Conference on High Performance Computing, pp. 1–10. IEEE (2010)
14. Nae, V., Iosup, A., Prodan, R.: Dynamic resource provisioning in massively multiplayer online games. IEEE Trans. Parallel Distrib. Syst. **22**(3), 380–395 (2011)
15. Rivoire, S., Ranganathan, P., Kozyrakis, C.: A comparison of high-level full-system power models. In: Workshop on Power Aware Computing and Systems, pp. 1–5. USENIX (2008)
16. Sharifi, M., Salimi, H., Najafzadeh, M.: Power-efficient distributed scheduling of virtual machines using workload-aware consolidation techniques. J. Supercomput. **61**(1), 46–66 (2012)
17. Razali, N.M., Geraghty, J.: Genetic algorithm performance with different selection strategies in solving TSP. In: World Congress on Engineering, pp. 1134–1139. IAENG (2011)
18. Hong, C.H., Lee, K., Park, H., Yoo, C.: ANCS: achieving QoS through dynamic allocation of network resources in virtualized clouds. Sci. Program. **2016**, 1–10 (2016)
19. Ganglia Monitor System. http://ganglia.sourceforge.net/. Accessed 06 June 2018
20. BZFlag. https://www.bzflag.org/. Accessed 06 June 2018
21. Stendhal. https://stendhalgame.org/ Accessed 06 June 2018
22. KBengine. https://github.com/kbengine/kbengine. Accessed 06 June 2018
23. World of Warcraft. http://www.worldofwarcraft.com/. Accessed 06 June 2018
24. Zheng, X., Cai, Y.: Optimal server provisioning and frequency adjustment in server clusters. In: International Conference on Parallel Processing Workshops, pp. 504–511. IEEE (2010)

A Cost-Effective Deadline-Constrained Scheduling Strategy for a Hyperparameter Optimization Workflow for Machine Learning Algorithms

Yan Yao, Jian Cao(✉), and Zitai Ma

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{yaoyan,cao-jian}@sjtu.edu.cn, mazitai@outlook.com

Abstract. As a method of data analysis that automates analytical model building, machine learning is becoming increasingly popular. In most machine learning algorithms, hyperparameter optimization or tuning is a necessary step. Unfortunately, the process of hyperparameter optimization is usually computationally expensive and time-consuming. Currently, machine learning is becoming a service so that cost and time should be considered when a machine learning service is provided. In this paper, we propose a scheduling approach to satisfy two contradictory targets, i.e., cost and time, when models corresponding to multiple settings of hyperparameters need to be tried. In this approach, the execution time of the model with specific settings of the hyperparameters can be predicted first. Then we generate an optimized workflow instance model, which consists of multiple parallel branches and each branch sequentially executes multiple models on a server. Based on the number partitioning algorithm, the branches are organized in such a way that they have a similar execution time and can be completed almost at the same time. Through experiments on different machine learning algorithms, it demonstrated that this approach meets the deadline and reduce the cost at the same time.

Keywords: Machine learning
Hyperparameter optimization workflow · Resource scheduling

1 Introduction

Big data analytics is an emerging technology, providing better insights from huge and heterogeneous data. Within the field of data analytics, supervised machine learning algorithms need a significant amount of time to be trained and evaluated on the data set while unsupervised machine learning algorithms need time to execute on the data set. Regardless as to whether they are supervised or unsupervised algorithms, most involve a set of configuration variables (a.k.a. *hyperparameters*) that must be configured prior to training and (or) execution.

Settings for the hyperparameters do not only affect the performance of the training process, but more importantly, the quality (e.g. prediction accuracy) of the machine learning algorithms. Unfortunately, tuning hyperparameters is usually computationally expensive and time-consuming. Currently, machine learning is becoming a service and the need to speed up hyperparameter optimization while the cost can also be reduced is crucial.

One way by which to organize the hyperparameter optimization process is sequential execution, i.e., multiple models defined by different settings of hyperparameters are executed sequentially on a server. Obviously, this approach is very time-consuming and time usually increases exponentially with the number of hyperparameters (seen Fig. 1(a)). Another option is parallel execution. As the multiple models to be evaluated are independent of each other, the process of hyperparameter optimization can take advantage of the distributed computing platform (e.g., cloud computing) so that the overall execution time can be reduced. An extreme configuration is each model is assigned to a server for execution which leads to the shortest overall execution time. However, this will incur a heavy cost in terms of computational resources (Fig. 1(b)).

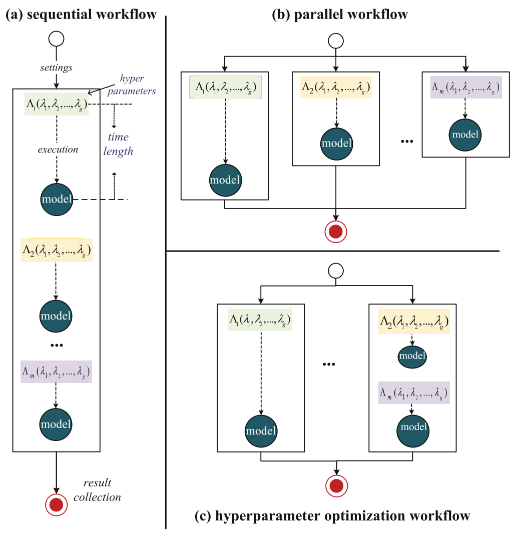

(a) sequential workflow
(b) parallel workflow
(c) hyperparameter optimization workflow
settings
hyper parameters
execution
time length
result collection
model
$\Lambda_1(\lambda_1,\lambda_2,...,\lambda_g)$
$\Lambda_2(\lambda_1,\lambda_2,...,\lambda_g)$
$\Lambda_m(\lambda_1,\lambda_2,...,\lambda_g)$

Fig. 1. Sequential, completely parallel and our optimized workflow instance model

In this paper, we propose a workflow scheduling approach for hyperparameter optimization with the objective of minimizing computational resources while meeting the deadline constraint at the same time. Central to this approach is an algorithm to generate an optimized workflow instance model, considering the fact that with different settings of the hyperparameters' values the execution time of corresponding machine learning models is very different. Thus, multiple models with a short execution time can share the same server while a server will be allocated to those models that need a long execution time exclusively, as shown in Fig. 1(c). This arrangement can save computational resources and also ensure the deadline is met. For this idea to work, we need to address two problems. One is given the hyperparameter settings, we need to predict the execution time of its corresponding machine learning model. The second problem is how to organize the execution of these models into different parallel branches of the workflow instance in a cost effective way while the deadline requirements are met.

Therefore, the main contributions of this paper are two-fold as follows:

– *Execution time prediction for machine learning models under different hyperparamenter settings*: In order to predict the execution time of machine learning models under different settings, we propose a regression-based approach which is based on the time complexity analysis of this model.
– *Generate an optimized workflow instance model for hyperparamater optimization*: We transform this problem into a number partitioning problem [7]. After an initial workflow instance model is created, it will be restructured based on deadline constraints using a multi-way number partitioning algorithm.

The rest of the paper is organized as follows. Section 2 summarizes the related work on hyperparameter optimization and the number partitioning problem. Section 3 formulates the scheduling problem which minimizes the execution cost with given deadline constraint. How to generate the workflow instance model for hyperparameter optimization is discussed in Sect. 4. Section 5 presents the experiment results on three machine learning models. Finally, conclusions are drawn in Sect. 6.

2 Related Work

Hyperparameter Optimization in Machine Learning. Many algorithms have been used for hyperparameters search, including grid search [1], random search [2], Bayesian optimization [9] and the swarm optimization algorithm [4]. It can be very time-consuming when working with a large number of hyperparameters. The focus of our paper is not a hyperparameter search strategy but trying to optimize the execution process after multiple settings have been decided by a dedicated search strategy.

Number Partitioning Problem. In this paper, we transform scheduling for hyperparameter optimization workflow as a number partitioning problem, which is one of the simplest yet NP-complete problems in combinational optimization [7]. A variety of methods have been proposed for number partitioning, such as heuristic-based methods (e.g., greedy heuristic, breadth first search, Karmarkar-Karp heuristic (a.k.a KK heuristics)), metaheuristic-based methods (e.g., tabu search, simulated annealing) and so on. The best polynomial time heuristics known to date is the differencing method of the KK heuristics [6].

Workflow Scheduling. Workflow scheduling is the problem of mapping inter-dependent tasks to a set of suitable resources. Since task scheduling is a well-known NP-complete problem [5], many heuristic methods have been proposed for a general workflow model [8]. Our study is on how to schedule a workflow model with a special structure, i.e., it has multiple homogenous machine learning tasks of different execution time.

3 Problem Formulation

The framework of our hyperparameter optimization workflow scheduling is illustrated in Fig. 2. Given a machine learning algorithm having n hyperparameters, hyperparameter tuning algorithms will give a set of alternative settings. Suppose there is g settings having execution time $t_i (1 \leq i \leq g)$. Assume the deadline for the hyperparameter optimization process is D. Our target is to schedule the hyperparameter optimization process in the way that can minimize the computational resources needed under the deadline constraint.

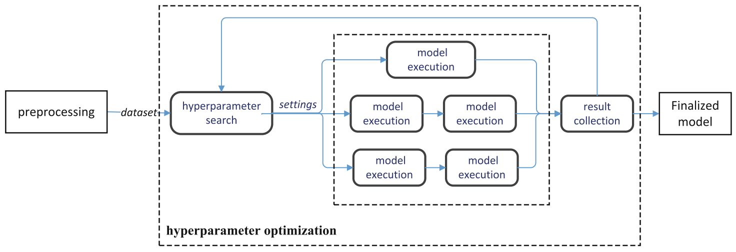


Fig. 2. Hyperparameter optimization workflow scheduling process

As these g alternative models can be executed independently of each other, the execution of each model can be viewed as a parallel task (a parallel branch in the workflow instance model). We want to reduce the branch number from g to κ by concatenating them and minimize the longest execution time of these κ branches at the same time. This problem can be viewed as a number partitioning problem, which is defined as:

Given a set $T = \{t_1, t_2, ..., t_g\}$ *and an integer* D*, a solution can be found to divide* T *into* κ *subsets* $T = \{T_1, T_2, ..., T_\kappa\}$*, with* $sum(T_i) < D$,$(1 \leq i \leq \kappa)$. *This solution should minimize the largest* $sum(T_i)$, $(1 \leq i \leq \kappa)$. The requirement for all κ subsets is that:

$$\forall_i, \quad sum(T_i) \leq D, \quad 1 \leq i \leq \kappa \tag{1}$$

$$\forall_{i,j}, \quad T_i \cap T_j = \emptyset \tag{2}$$

$$T_1 \cup T_2 \cup \cdots \cup T_\kappa = T \tag{3}$$

By applying the method to solve the number partitioning problem, we can ensure each branch of the hyperparameter optimization workflow model has a similar total execution time. Then each branch can be assigned to a server with the same configuration. Therefore, the computational cost of the hyperparameter optimization workflow can be measured by a metrics defined as $Cost = \kappa \cdot max_{1 \leq i \leq \kappa} sum(T_i)$. Since the upper bound of T_i is D, the upper bound of $Cost$ is $\kappa \cdot D$. We now discuss how to solve this problem and the details of our algorithm are given in the next section.

4 The Scheduling Strategy for the Hyperparameter Optimization Workflow

Step.1: Parallelizing the Process of Hyperparameter Optimization. The first step is to generate an initial hyperparameter optimization workflow instance model, in which the execution of a model corresponding to a setting of hyperparamters is a task. As these tasks can be executed independently of each other, therefore, in this initial workflow instance model, a parallel branch is created for each task. The total number of tasks (parallel branches) is $g = |\boldsymbol{\Lambda}|$ and the branch set is $B = \{b_1, b_2, ..., b_g\}$.

Step.2: Machine Learning Model Execution Time Prediction. The second step is to predict the execution time of g tasks ($T = t_1, t_2, ..., t_g$). There is a linear relationship between the execution time and the expression of its hyperparameters (Eq. (4)) on a platform, which can be learned by a linear fitting process from historical execution data on the same platform.

$$T_{alg} = \alpha C + \beta \tag{4}$$

Step.3: Workflow Instance Model Restructuring Now, we apply the number partitioning algorithm to restructure the original workflow instance model. For convenience sake, D and T are both divided by their greatest common divisor to obtain the new relative deadline D' and relative model execution time set $T' = \{t'_1, t'_2, ..., t'_g\}$. What we want to do is combine the tasks of model execution in T' into κ groups, and each group of tasks can be completed one by one before D'.

As presented in Algorithm 1, an initial value $\lceil \frac{sum(T')}{D'} \rceil$ for κ_0 is input to the algorithm (line 5). Then a Multi-way Karmarkar-Karp algorithm is applied to restructure the model. If the largest subset sum in the solution is larger than D', this means that the current model cannot meet the user's deadline constraint. Then, we increase an instance to a resource pool, and κ plus one, the solution is regenerated. This process is repeated until a valid solution is found.

4.1 Complexity Analysis

The algorithm involves the following steps:

1. Initial Workflow Instance Model Generation. In this step, the time complexity is $O(g)$, where g is the quantity of the parallel branches after instantiation.

2. Time Prediction for the Task. This step also needs $O(g)$ time. The estimated time is calculated by the linear equation. Each calculation can be done in $O(1)$ time.
3. Workflow Instance Model Restructuring. This step needs $O(g)$ for the heuristic algorithm to reduce one tuple in every iteration. In total, there are g tuples and then the model is improved with the different instance number k cost $O(g)|\kappa|$.

Overall, this algorithm can give an approximate solution in $O(g)$ time.

Algorithm 1. Workflow Instance Model Restructuring Algorithm

```
Require: $T'$: Set of relative time,$D'$: relative deadline
Ensure: $\delta$: the difference between the maximum and the minimum sum of subset $Scheduling\_plan$:
    the initial solution
 1: $terminate \leftarrow false$
 2: if $D' < \max_{t_i \in T'} t_i$ then
 3:     return $error$:no suitable plan
 4: end if
 5: $\kappa \leftarrow \frac{sum(T')}{D'} + 1$
 6: $Scheduling\_plan \leftarrow \emptyset$
 7: $\delta \leftarrow \infty$
 8: while $terminate != true$ do
 9:     $temp\_\delta, temp\_plan \leftarrow$MWKK$(T', \kappa)$
10:     $temp\_max \leftarrow \max_{subset \in tempplan} sum(subset)$
11:     if $temp\_max > D'$ then
12:         $\kappa \leftarrow \kappa + 1$
13:     else
14:         $terminate \leftarrow true$
15:         $initial\_plan \leftarrow temp\_plan$
16:         $\delta \leftarrow temp\_\delta$
17:     end if
18: end while
19: return $\delta, Scheduling\_plan$
```

5 Experiments

In order to prove the effectiveness of our proposed approach for a hyperparameter search, we take three machine learning algorithms, i.e., kNN [3], k-means [10] and CNN [11], as examples in our experiments. Without loss of generality, a server configuration with a CPU Intel core i5-6500 and memory of 8G is used in this experiment. In real applications, the server is selected based on the total budget and deadline.

5.1 Compared Strategies

We use three strategies as baselines:

1. Basic Scheduling (BS) Strategy: This is based on the initial hyperparameter optimization workflow instance model, i.e., each model corresponding to a setting is allocated to a branch, which is assigned to a separate server.

2. Greedy Algorithm (GA)-based Strategy: In this strategy, the execution time of models corresponding to the different settings of hyperparameters are first sorted in descending order. Then, the elements are taken from the sorted list, and are placed into the subset with the smallest sum of elements.
3. Karmarkar-Karp Algorithm (KK)-based Strategy: Here we use a multi-way Karmarkar-Karp algorithm to restructure the workflow instance model.

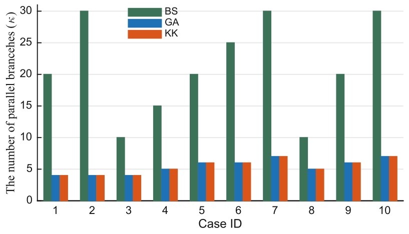
5.2 Experiment Results

After deadline D is assigned to the hyperparameter setting list L, the scheduling strategies generate the concrete workflow instance model. The model contains κ parallel branches and the task list of the ith parallel is S_i $(1 \leq i \leq \kappa)$. The relative deadline of the first three test cases is set to $sum(P')/3$, and the others are set to $sum(P')/4$, where $sum(P')$ is the sum of all the relative execution time of tasks.

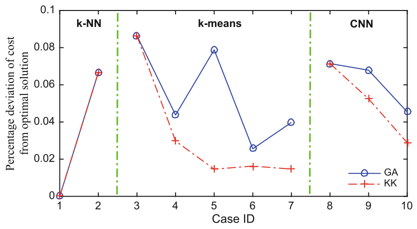
Table 1. Relative cost with different scheduling strategies

Case ID	*Algoritms*	BS		GA		KK		*OPT*
		κ	*Cost*	κ	*Cost*	κ	*Cost*	*Cost*
1	k-NN	20	20	4	20	4	20	20
2	k-NN	30	30	4	32	4	32	30
3	k-means	10	2000	4	1132	4	1132	1042
4	k-means	15	3090	5	1860	5	1835	1782
5	k-means	20	4160	6	3024	6	2844	2803
6	k-means	25	5200	6	2646	6	2622	2580
7	k-means	30	6240	7	3192	7	3115	3070
8	*CNN*	10	4350	5	2790	5	2790	2604
9	*CNN*	20	8980	6	5256	6	5180	4922
10	*CNN*	30	13350	7	8015	7	7888	7667

Table 1 shows the experiment results. The optimal solution (OPT) assumes that the total time can be divided into branches equally. However, most of time we can not divide these equally since we have a series of inseparable tasks. Therefore, the last column of Table 1 represent the upper bounds. Figure 3(a) shows the number of branches generated by three algorithms. Algorithms with restructuring strategies (GA, KK) are much less than full parallel (BS), leading to lower cost. Figure 3(b) shows that how close of the cost of GA and KK algorithms to the cost of optimal solution. We find the KK algorithm is better than BS and GA. Moreover, it is very close to the optimal solution in the last column of Table 1.



(a) the number of parallel branches.



(b) percentage deviation of cost from optimal solution.

Fig. 3. Result comparision

6 Conclusions

In this paper, we described how to generate the hyperparameter optimization workflow model to ensure the cost under deadline constraints when multiple settings are specified. Specifically, we can predict the execution time of each machine learning algorithm according to its hyperparameter values based on computation complexity analysis. Then a number partitioning algorithm is adopted to arrange the parallel branches in the workflow model. Moreover, we introduced a heuristic method to solve this problem and conducted experiments with different models.

Through the experiments, we can see that our model has stronger stability in different workflow models and reduce much more cost than BS, KK, GA and the other strategies. This demonstrates that our model has the advantage of reducing cost while ensuring that deadline requirements can be met.

Acknowledgments. This work is partially supported by National Key Research and Development Plan (No. 2018YFB1003800), China National Science Foundation (Granted Number 61472253, 61772334), and Cross Research Fund of Biomedical Engineering of Shanghai Jiaotong University (YG2015MS61).

References

1. Bergstra, J., Bardenet, R., Bengio, Y., Kegl, B.: Algorithms for hyper-parameter optimization, vol. 24, pp. 2546–2554 (2011)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(1), 281–305 (2012)
3. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
4. Guo, X.C., Yang, J., Wu, C., Wang, C.Y., Liang, Y.: A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. Neurocomputing **71**, 3211–3215 (2008)
5. Hartmanis, J.: Computers and intractability: a guide to the theory of NP-completeness (Michael R. Garey and David S. Johnson). Siam Rev. **24**(1), 90–91 (1982)

6. Karmarker, N., Karp, R.M.: The differencing method of set partitioning. Technical report. UCB/CSD-83-113, EECS Department, University of California, Berkeley (1983). http://www2.eecs.berkeley.edu/Pubs/TechRpts/1983/6353.html
7. Moffitt, M.: Search strategies for optimal multi-way number partitioning, pp. 623–629, August 2013
8. Rodriguez, M.A., Buyya, R.: A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. Concurr. Comput.: Pract. Exp. **29**(8) (2017). https://doi.org/10.1002/cpe.4041
9. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Neural Information Processing Systems, pp. 2951–2959 (2012)
10. Wu, X., et al.: Top 10 algorithms in data mining. Knowl. Inf. Syst. **14**(1), 1–37 (2007)
11. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53

Transparently Capturing Execution Path of Service/Job Request Processing

Yong Yang[1(✉)], Long Wang[2], Jing Gu[1], and Ying Li[1,3(✉)]

[1] School of Software and Microelectronics, Peking University, Beijing, China
{yang.yong,gu.jing,li.ying}@pku.edu.cn
[2] IBM Watson, Cambridge, USA
wanglo@us.ibm.com
[3] National Engineering Center of Software Engineering, Peking University, Beijing, China

Abstract. Distributed platforms are widely deployed to provide services in various trades. With the increasing scale and complexity of these distributed platforms, it is becoming more and more challenging to understand and diagnose a service request's processing in a distributed platform, as even one simple service request may traverse numerous heterogeneous components across multiple hosts. Thus, it is highly demanded to capture the complete end-to-end execution path of service requests among all involved components accurately. This paper presents REPTrace, a generic methodology for capturing the complete request execution path (REP) in a transparent fashion. We propose principles for identifying causal relationships among events for a comprehensive list of execution scenarios, and stitch all events to generate complete request execution paths based on library/system calls tracing and network labelling. The experiments on different distributed platforms with different workloads show that REPTrace transparently captures the accurate request execution path with reasonable latency and negligible network overhead.

Keywords: End-to-end tracing · Service request · Distributed system
Request execution path

1 Introduction

Computing workloads are increasingly being placed onto distributed platforms like cloud environments such as OpenStack and SoftLayer and job-processing environments such as Spark and Hadoop. Such distributed platforms are composed of numerous services and components in multiple levels, and support many applications on top of the platforms. These services, components and applications may have been developed by different teams or vendors using different programming languages. This situation brings about difficulty in understanding and diagnosing the processing of service requests and job requests, because the processing of a single service request or job request typically traverses a number of components across a set of machines.

In these platforms, traditional ways to monitor and trace a single program within a machine, e.g. strace, a debugging tool in Unix-like systems, are insufficient, because

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 879–887, 2018.
https://doi.org/10.1007/978-3-030-03596-9_63

they are unable to reveal what all the components do to process the service/job request, and hence they provide quite limited insights for developers and administrators to understand and diagnose a distributed platform. It is highly demanded by platform providers, e.g. Amazon, IBM who provide cloud platforms, and service providers, e.g. owners of those services registered onto service marketplace, (and certain service users/job submitters), to capture the complete end-to-end execution path of the service/job request processing among all involved components for individual requests. Ideally, with this complete request execution path available, we would be able to obtain a holistic view of the behavior of the distributed platforms during processing an individual service/job with in-depth details. This holistic view of the behavior with in-depth details can boost the understanding and diagnosis of service/job request processing, especially when source code of the distributed platforms is unavailable.

A Request Execution Path (REP as acronym) is defined as the complete path of processing a specific service/job request. Specifically, the REP (1) covers all of the executions of the given distributed platform's components in processing a specific request, including all of the instruction executions that participate in processing the particular request within those components' processes and threads, as well as the communications among these processes and threads during the processing; (2) identifies all these executions, and links them together according to the accurate causal relationships among them to form an integral unseparated view of the distributed platform's processing of this request. In this paper, we try to capture the REP without source code availability.

A number of technologies in current literature try to tackle a similar problem (end-to-end tracing), such as Google's Dapper [3], Twitter's Zipkin [9], and Facebook's Canopy [1]. They require source code availability and instrumentation into the source code, or require expertise knowledge of specific middleware, or only deal with vendors' custom libraries. Specifically, Stardust [8] requires manual modification of the distributed storage system. Pinpoint [5], Dapper [3], Pivot [2], Magpie [6] and Canopy [1] deal with specific middleware or vendor custom libraries only. HTrace [10] requires instrumentation to the system to enable tracing. Certain hardware and application support is demanded for X-trace [7] to work. vPath [4] made a thrust to provide a generic solution of tracing distributed systems/applications by monitoring thread and network activities. But vPath assumes certain communication styles and thread patterns, which do not hold in many modern distributed systems.

In this paper, we propose Request Execution Path Trace (REPTrace), a generic methodology that captures complete execution paths of service/job request processing of distributed platforms in a transparent way. By "transparency", we mean the REP is captured, for different distributed platforms without source code availability or code instrumentation. REPTrace intercepts runtime events at the system level (either within a physical machine, VM, or container), such as common library calls or system calls (intercepting library calls does not require root privilege, though intercepting OS system calls requires), and constructs the request execution path by identifying the causal relationships for four categories of events: thread events, process events, network communications events and control flow execution events. Principles and algorithms are proposed for identifying the causal relationships among the events (within the same category or across categories). Then REPTrace stitches all the events of one request's processing together using the identified causal relationships, and produces the complete request execution path. No source code or instrumentation to the distributed applications/platforms is required. No specific communication styles or thread patterns are assumed. Instead, we i) perform comprehensive analysis of execution scenarios when distributed systems process service requests and ii) leverage system-level tracing of library/system calls and labelling of network messages, to make our solution generic for distributed applications/platforms.

In summary, this paper makes the following contributions:

- We analyze execution scenarios during request processing and identify event causal relationships in all the execution scenarios. This comprehensive analysis is the basis for REPTrace.
- We propose REPTrace, a generic methodology for complete execution path capturing without source code availability, and implement REPTrace on Linux systems. Based on the comprehensive analysis, no specific communication styles or thread patterns are assumed in REPTrace. As far as we know, there is no prior art of tracing that tries to address the comprehensive communication/execution scenarios as we handle.
- We evaluate REPTrace with extensive experiments on different distributed platforms. The experiments show that REPTrace captures more complete request execution paths and handles more comprehensive scenarios than prior art (REPTrace generates only one path for one request's processing, while vPath generates more than one hundred path fragments as it does not handle many execution scenarios).

2 Overview of Request Execution Path

In this section, we first conduct a comprehensive analysis of the execution scenarios during the request processing, and then define the granularity of REP and corresponding event categories. Based on the granularity and event categories, we describe the REP representation. In the end, we describe the high-level principles to identify causal relationships among events.

2.1 Analysis of Execution Scenarios

The following execution scenarios may be involved during the request processing. This is a comprehensive list of scenarios we encountered when implementing and applying tracing tools to real-world distributed platforms, which we believe covers most, if not all, of the common execution scenarios in request execution path.

(a) Sequential execution of instructions within a thread, according to the control flow order;
(b) The current thread creates another thread and passes the handling of the request to the new thread. The current thread may stop processing (e.g. sleep), or continue processing this request;

(c) The current process forks a process, and passes the handling of the request to the new process; The current process may stop or continue processing this request;
(d) The current process/thread sends a message to another process/thread, which may be on the same machine or a different machine. Then the latter process/thread begins the processing of the request. Network communication and other similar mechanisms, like pipe, are covered in this scenario; The current process/thread may stop or continue processing this request;
(e) The current process/thread synchronizes the processing of request with another existing process/thread using certain IPC (Inter-Process Communication) mechanism such as process wait, thread join, signal, lock/unlock, semaphore, etc.;
(f) The current process/thread saves the request (or its intermediate state) in a message queue. Then a different process/thread picks up the request (or the intermediate state) from the message queue and begins processing;
(g) The current process/thread passes the handling of the request to another existing process/thread using shared memory, shared variables, or mapped device.

The scenarios *a* ~ *e* are about the execution of requests according to manipulations or interactions of system control objects, i.e. thread, process, signal, etc. The scenarios *f* and *g* are about transitions of request processing according to values of certain memory locations (or program variables). The scenarios *a* ~ *e* are control dependencies of request execution path and the scenarios *f* ~ *g* are data dependencies of request execution path.

2.2 Event Category

Granularity of REP. We select a mixture of a subset of system calls and a subset of library calls as the granularity of REP. The selected library/system calls are widely used and associated with the manipulations of thread/process and network communication. As a result of our granularity selection, the following four categories of events are dealt with in our current REPTrace design and implementation. In this paper, an event refers to the system/library call activity, which is intercepted and emitted by REPTrace.

Thread Event Category (TE). Events in this category denote the manipulations associated with thread objects as well as interactions between threads. These events include: library calls of thread creation, thread termination, thread joining.

Process Event Category (PE). Events in this category denote the manipulations associated with process objects as well as interactions between processes. These events include: library calls of process fork, process exec, process wait and their variants.

Network Communication Event Category (NCE). Events in this category denote the manipulations associated with network communications. These events can be classified into two types, network message sending and network message receiving, which are respectively referred as *send* and *recv* hereafter.

Control Flow Execution Event Category (CFEE). One event of this category denotes a thread's continuous instruction execution segment without other categories of events above. These execution segments are formed as the result of other categories' events splitting the program execution. This category of events is defined to only help people consider/understand the request processing as a complete path of defined events.

2.3 Causal Relationship Identification

Here we describe the principles of how the causal relationships among events are identified for the enumerated execution scenarios above. We classify all the causal relationships into two categories: inter-node causal relationship and intra-node causal relationship. Inter-node causal relationship covers only scenario *d*. Intra-node relationship covers the rest of all the scenarios.

Here we use CFEE event type in following rules to strictly define the causal relationship identification; this event type stitches all events together so that the entire request processing is a complete path of events. After the entire REP is formed, the events of the CFEE type are deleted (i.e., the parent event of a CFEE event *e* becomes the parent of *e*'s child events) from the final representation of the causal relationship to facilitate presentation and processing.

i. **Scenario a** consists of a sequence of events within the same thread. An event (event *A*) that is immediately before the next event (event *B*) along the execution sequence is event B's parent.
ii. **Scenario b** involves a thread creation. Suppose thread *A* invokes *pthread_create()* to create another thread *B*. Then the CFEE event immediately before *pthread_create()* is the parent of the *pthread_create()* event, which is the parent of the thread *B*'s starting CFEE event.
iii. **Scenario c** involves a process creation. The causal relationship identification between involved events is similar to the identification algorithm for **Scenario b**.
iv. **Scenario d** involves a network communication operation, or a *send-recv* pair. For a *send* event, we identify the corresponding *recv* event by inserting a unique ID (called DATA_ID) into the front of each transmitted message, and then extract this ID from the received message of the *recv* event. The *send* and *recv* events with the same DATA ID have the causal relationship with the *send* event being the parent of the *recv* event. In this scenario, we consider the *send* event as the only parent of the *recv* event, discarding the parent event for *recv* from **Scenario a**.
v. **Scenario e** involves system-level IPC mechanisms such as process wait, thread join, signal, etc. For each of these mechanisms we define its specific causality identification method. For example, *waitpid()* blocks the calling process *A* (the current process) and waits for the specified process *B* to terminate or change state (stopped or resumed or signals). Then process *A*'s execution CFEE event immediately after the *waitpid()* in the control flow order is the child of the last CFEE event of process *B* before *B*'s termination or state change and the *waitpid()* event. In this way, one event has two parent events. Similarly, specific identifications of event causal relationships associated with thread join, signals, lock/unlock and other IPC mechanisms are defined correspondingly based on their semantics in manipulating execution behavior.

vi. **Scenario f and scenario g** involve data dependencies. Generic accurate analysis of data dependencies along individual traces transparently is a difficult problem to tackle. This paper mainly addresses the construction of request execution path using control dependencies rather than data dependencies, as handling data dependencies is a radically different topic than handling control dependencies. So we do not address all data dependencies in these scenarios, but only identify those data dependency causal relationships in which data writing and reading events can be correlated by specific IDs like Job ID of Hadoop, message ID or session ID.

After identifying all the causal relationships of events belonging to a request, we represent the request execution path as a graph, specifically a directed acyclic graph (DAG), in which each vertex refers to an event and each edge refers to a causal relationship between two events. Figure 1 (a) shows an example of a request execution path.

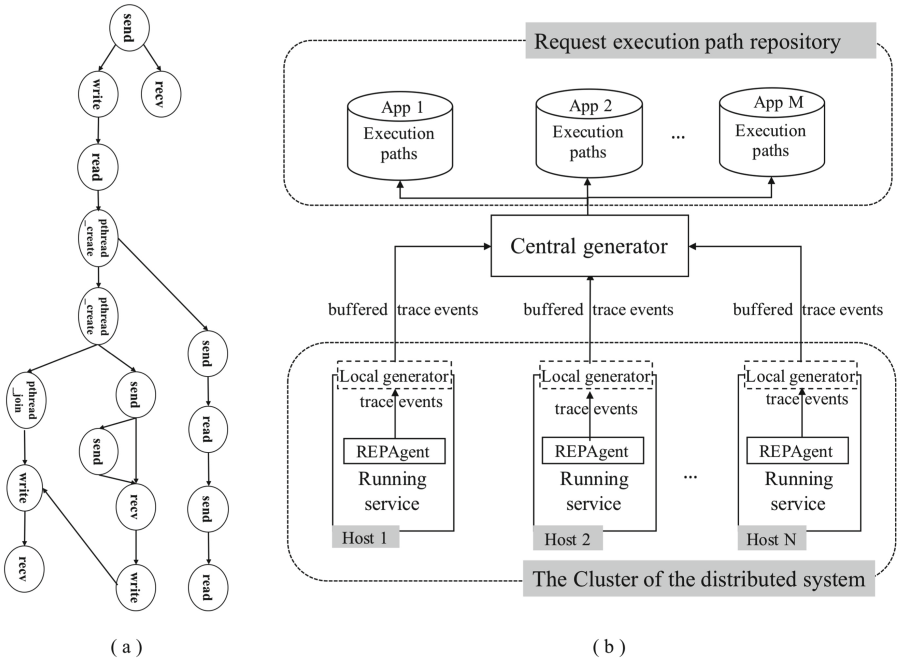


Fig. 1. (a) An example of a request execution path (b) system architecture of REPTrace

3 System Design

REPTrace consists of a REPAgent (and an optional Local Generator) on each host of the traced distributed system, as well as a Central Generator (see Fig. 1 (b)).

REPAgent intercepts library/system calls of the host, and emits trace events to the Central Generator. A trace event maps to an intercepted library/system call event. The Central Generator collects these trace events and runs the causal relationship identification in an asynchronous mode (so the impact to the request processing is minimized).

Then Central Generator stores the complete REP of every request in the repository. If further performance impact minimization is required, an optional Local Generator can buffer REPAgent's events to avoid network latency and overhead incurred by the communication to the Central Generator. Note that TCP message fragmentation needs to be handled before the causal identification is performed in the Central Generator. Due to page limitation, here we do not discuss details of these components.

4 Experiment and Evaluation

Experimental Setup. We implement REPTrace on Linux distributions. The source code and the full list of intercepted library/system calls can be found on the project website [11]. The experiments were conducted on Hadoop (a combination of Yarn, MapReduce and HDFS), Spark, a Tensorflow-based application and Angel (Tencent's open source large-scale machine learning system).

Overhead Study. The latency and network traffic overhead incurred by REPTrace on different systems with different workloads are measured in our experiments. The experimental results show 4.5% latency and 1.8% network traffic overhead by average. Due to the page limitation, we do not present details of the overhead measurement here.

Completeness of the Generated Request Execution Path. Asingle job of WordCount application was executed on Hadoop with a 500 M input file and REPTrace captured the trace events during the job execution. The generated request execution path is to be used for purposes of understanding and diagnosing the processing of service requests and job requests. So, the generated REP must be complete (i.e. all causal relationships of all the events along the request execution are identified and stitched properly in the result path) and correct (i.e. the identified relationships between events reflect the real causal relationships of them along the request execution).

REPTrace produces 72853 trace events for a single job's processing in our experiment. The processing of a single job request is a holistic session. So, all events in this session should be linked into a single execution path if the generated REP is complete (those events associated with heartbeat during the job execution are also regarded as part of the session, as they are integral part of the Hadoop system).

REPTrace stitches the 72853 trace events into one request execution path. As comparison, the prior state-of-the-art solution, vPath, produces 28908 trace events and links these events into more than one hundred execution path fragments, because vPath does not cover all types of events in execution and its assumption of threading patterns does not hold for most execution scenarios in Hadoop (we got the source code of vPath from the vPath author and used it as our baseline when evaluating the completeness), such as the widely used Hadoop RPC (Remote Procedure Call) mechanism.

Correctness of the Generated Request Execution Path. We performed thorough manual inspection of the REP generated in the experiment and consulted the relevant source code carefully. We did not find any error in those inspected causal relationships of the result execution path, i.e. all inspected ones reflect the real causal relationships as expected and as specified in Sect. 2.3.

The REPTrace has been applied for anomaly detection and system understanding. Due to the page limitation, we do not present details of leveraging the generated REP for anomaly detection or system understanding here. These contents are available per request.

5 Conclusion

In this paper, we propose REPTrace, a generic end-to-end tracing methodology, which is able to automatically generate the complete execution path of service/job requests for a variety of distributed platforms in a transparent way. We analyzed the possible request execution scenarios in distributed platforms and corresponding causal relationships, and illustrated how REPTrace captures the accurate causal relationships in these scenarios. We conducted experiments to evaluate the correctness and completeness of the request execution path produced by REPTrace, and compared the results with the existing state-of-the-art solution. Experiments show that REPTrace produces correct execution path which is much more complete than prior art (i.e. we cover much more execution scenarios), with small performance overhead (4.5% latency and 1.8% network traffic overhead).

REPTrace has the limitation of failing to handle general data-dependency causal relationship when specific IDs like job ID, message ID, request ID, are unavailable. Handling this limitation is one of our future work. Moreover, we have applied the generated REP for anomaly detection and system understanding, which is not presented here due to page constraints.

References

1. Kaldor, J., Mace, J., Bejda, M., et al.: Canopy: an end-to-end performance tracing and analysis system. In: ACM Symposium on Operating Systems Principles (2017)
2. Mace, J., Roelke, R., Fonseca, R.: Pivot tracing: dynamic causal monitoring for distributed systems. In: Proceedings of the 25th Symposium on Operating Systems Principles, ACM, pp. 378–393 (2015)
3. Sigelman, B.H., Barroso, L.A., Burrows, M., et al.: Dapper, a large-scale distributed systems tracing infrastructure Technical report, Google, Inc., (2010)
4. Tak, B.C., Tang, C., Zhang, C., et al.: VPath: precise discovery of request processing paths from black-box observations of thread and network activities. In: USENIX Annual Technical Conference (2009)
5. Chen, M.Y., Kiciman, E., Fratkin, E., et al.: Pinpoint: problem determination in large, dynamic internet services. In: 2002 Proceedings International Conference on IEEE Dependable Systems and Networks, 2002 DSN, pp. 595–604 (2002)
6. Barham, P., Isaacs, R., Mortier, R., et al.: Magpie: online modelling and performance-aware systems. In: HotOS, pp. 85–90 (2003)

7. Fonseca, R., Porter, G., Katz R.H, et al.: X-trace: a pervasive network tracing framework. In: Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, p. 20 (2007)
8. Thereska, E., Salmon, B., Strunk, J., et al.: Stardust: tracking activity in a distributed storage system. In: ACM SIGMETRICS Performance Evaluation Review, ACM, vol. 34, no. 1, pp. 3–14 (2006)
9. Zipkin. https://zipkin.io/
10. HTrace. http://htrace.incubator.apache.org/
11. REPTrace. https://github.com/kliosvseyy/REPTrace

Author Index