Min Luo
Liang-Jie Zhang (Eds.)

LNCS 10967

Cloud Computing – CLOUD 2018

11th International Conference
Held as Part of the Services Conference Federation, SCF 2018
Seattle, WA, USA, June 25–30, 2018
Proceedings

Springer

Lecture Notes in Computer Science 10967

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

More information about this series at http://www.springer.com/series/7409

Min Luo · Liang-Jie Zhang (Eds.)

Cloud Computing – CLOUD 2018

11th International Conference
Held as Part of the Services Conference Federation, SCF 2018
Seattle, WA, USA, June 25–30, 2018
Proceedings

Springer

Editors
Min Luo
Huawei Technologies CO., Ltd
Shenzhen
China

Liang-Jie Zhang
Kingdee International Software Group CO. Ltd
Shenzhen
China

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-94294-0 ISBN 978-3-319-94295-7 (eBook)
https://doi.org/10.1007/978-3-319-94295-7

Library of Congress Control Number: 2018947340

LNCS Sublibrary: SL3 – Information Systems and Applications, incl. Internet/Web, and HCI

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume presents the accepted papers for the 2018 International Conference on Cloud Computing (CLOUD 2018), held in Seattle, USA, during June 25–30, 2018. The International Conference on Cloud Computing (CLOUD) has been a prime international forum for both researchers and industry practitioners to exchange the latest fundamental advances in the state of the art and practice of cloud computing, identify emerging research topics, and define the future of cloud computing. All topics regarding cloud computing align with the theme of CLOUD. We celebrated the 2018 edition of the gathering by striving to advance the largest international professional forum on cloud computing.

For this conference, each paper was reviewed by at least three independent members of the international Program Committee. After carefully evaluating their originality and quality, we accepted 29 papers.

We are pleased to thank the authors whose submissions and participation made this conference possible. We also want to express our thanks to the Program Committee members, for their dedication in helping to organize the conference and reviewing the submissions. We owe special thanks to the keynote speakers for their impressive speeches. We would like to thank Chengzhong Xu, Xianghan Zheng, Dongjin Yu, Mr. Ben Goldshlag, and Pelin Angin, who provided continuous support for this conference.

Finally, we would like to thank Rossi Kamal, Tolga Ovatman, Fadi Al-Turjman, Mohamed Nabeel Yoosuf, Bedir Tekinerdogan, and Jing Zeng for their excellent work in organizing this conference.

May 2018

Min Luo
Liang-Jie Zhang

Conference Committees

General Chair

Chengzhong Xu — Chinese Academy of Science (Shenzhen), China

Program Chair

Min Luo — Huawei, USA

Short Paper Track Chair

Dongjin Yu — Hangzhou Dianzi University, China

Publicity Chair

Pelin Angin — Middle East Technical University, Turkey

Application and Industry Chair

Ben Goldshlag — Goldman Sachs, USA

Program Co-chair

Xianghan Zheng — Fuzhou university, China

Cloud Reliability Track Chair

Rossi Kamal — Royal University of Dhaka, Bangladesh

Cloud Performance Track Chair

Tolga Ovatman — Istanbul Technical University, Turkey

Cloud IoT Services Track Chair

Phu Phung — University of Dayton, USA

Cloud Networking Track Chair

Fadi Al-Turjman — METU Northern Cyprus Campus, Northern Cyprus

Cloud Software Engineering Track Chair

Mohammad Hammoud — Carnegie Mellon University in Qatar, Qatar

Cloud Data Analytics Track Chair

Mohamed Nabeel Yoosuf — Qatar Computing Research Institute, Qatar

Cloud Infrastructure and Management Track Chair

Bedir Tekinerdogan — Wageningen University, The Netherlands

Services Conference Federation (SCF 2018)

General Chairs

Calton Pu — Georgia Tech, USA
Wu Chou — Vice President-Artificial Intelligence and Software at Essenlix Corporation, USA

Program Chair

Liang-Jie Zhang — Kingdee International Software Group Co., Ltd, China

Finance Chair

Min Luo — Huawei, USA

Panel Chair

Stephan Reiff-Marganiec — University of Leicester, UK

Tutorial Chair

Carlos A. Fonseca — IBM T.J. Watson Research Center, USA

Industry Exhibit and International Affairs Chair

Zhixiong Chen — Mercy College, USA

Operations Committee

Huan Chen (Chair) — Kingdee Inc., China
Jing Zeng — Tsinghua University, China

Cheng Li	Tsinghua University, China
Yishuang Ning	Tsinghua University, China
Sheng He	Tsinghua University, China

Steering Committee

Calton Pu	Georgia Tech, USA
Liang-Jie Zhang (Chair)	Kingdee International Software Group Co., Ltd., China

CLOUD 2018 Program Committee

Haopeng Chen	Shanghai Jiao Tong University, China
Steve Crago	University of Southern California, USA
Alfredo Cuzzocrea	University of Trieste, Italy
Roberto Di Pietro	University of Rome, Italy
J. E. (Joao Eduardo)	Ferreira University of Sao Paulo, Brazil
Chris Gniady	University of Arizona, USA
Daniel Grosu	Wayne State University, USA
Waldemar Hummer	IBM T.J. Watson Research Center, USA
Marty Humphrey	University of Virginia, USA
Gueyoung Jung	AT&T Labs, USA
Nagarajan Kandasamy	Drexel University, USA
Yasuhiko Kanemasa	Fujitsu Laboratories Ltd., Japan
Wubin Li	Ericsson Research, Canada
Li Li	Essenlix Corp, USA
Shih-chun Lin	North Carolina State University, USA
Jiaxiang Lin	Fujian Agriculture and Forest University, China
Lili Lin	Fujian Agriculture and Forest University, China
Xumin Liu	Rochester Institute of Technology, USA
Brahim Medjahed	University of Michigan, USA
Ningfang Mi	Northeastern University, USA
Shaolei Ren	Florida International University, USA
Norbert Ritter	University of Hamburg, Germany
Han Rui	Chinese Academy of Sciences, China
Rizos Sakellariou	University of Manchester, Britain
Upendra Sharma	IBM T.J. Watson Research Center, USA
Jun Shen	University of Wollongong, Australia
Evgenia Smirni	College of William and Mary, USA
Anna Squiciarini	Penn State University, USA
Zaogan Su	Fujian Agriculture and Forest University, China
Stefan Tai	Berlin University, Germany
Shu Tao	IBM T. J. Watson Research Center, USA
Qingyang Wang	Louisiana State University, USA
Pengcheng Xiong	NEC Labs, USA
Yang Yang	Fuzhou University, China
Changcai Yang	Fujian Agriculture and Forest University, China

Ayong Ye	Fujian Normal University, China
I-Ling Yen	University of Texas at Dallas, USA
Qi Yu	Rochester Institute of Technology, USA
Liang-Jie Zhang	Kingdee International Software Group, China
Ming Zhao	Arizona State University, USA
Xianghan Zheng	Fuzhou University, China

Contents

Research Track: Cloud Resource Management

Research Track: Cloud Management

Application and Industry Track: Cloud Service System

Research Track: Cloud Schedule

A Vector-Scheduling Approach for Running Many-Task Applications in the Cloud

Brian Peterson, Yalda Fazlalizadeh, Gerald Baumgartner(✉), and Qingyang Wang

Division of Computer Science and Engineering, School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA
{brian,yalda,gb,qywang}@csc.lsu.edu

Abstract. The performance variation of cloud resources makes it difficult to run certain scientific applications in the cloud because of their unique synchronization and communication requirements. We propose a decentralized scheduling approach for many-task applications that assigns individual tasks to cloud nodes based on periodic performance measurements of the cloud resources. In this paper, we present a vector-based scheduling algorithm that assigns tasks to nodes based on measuring the compute performance and the queue length of those nodes. Our experiments with a set of tasks in CloudLab show that the application proceeds in three distinct phases: flooding the cloud nodes with tasks, a steady state in which all nodes are busy, and the end game in which the remaining tasks are executed on the fastest nodes. We present heuristics for these three phases and demonstrate with measurements in CloudLab that they result in a reduction of the overall execution time of the many-task application.

1 Introduction

Many scientific fields require massive computational power and resources that might exceed those available on a single local supercomputer. Historically, there have been two drastically different approaches for harnessing the combined resources of a distributed collection of machines: traditional computational supercomputer grids and large-scale desktop-based master-worker grids. Over the last decade, a commercial alternative has become available with cloud computing. For many smaller scientific applications, using cloud computing resources might be cheaper than maintaining a local supercomputer.

Cloud computing has been very successful for some types of applications, especially for applications that do not require frequent communication between different cloud nodes, such as MapReduce [8] or graph-parallel [24] algorithms. However, for applications with fine-grained parallelism, such as the NAS MPI benchmarks or atmospheric monitoring programs, it shows less than satisfactory performance [10,23]. The main reasons are that the performance of cloud

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 3–19, 2018.
https://doi.org/10.1007/978-3-319-94295-7_1

nodes is often not predictable enough, especially with virtualization, and that the communication latency is typically worse than that of a cluster.

For applications that can be broken into a set of smaller tasks, however, it may be possible to match the performance requirements of a task with the performance characteristics of (a subset of) the cloud nodes. For grids and supercomputers, this *many-task computing* approach [19] has proven very effective. E.g., Rajbhandari et al. [20] structured the computation of a quantum chemistry tensor contraction equation as a task graph with dependencies and were able to scale the computation to over 250,000 cores. They dynamically schedule tasks on groups of processors and use work-stealing for load balancing.

Because of the heterogeneity resulting from virtualization, a task scheduler would have to keep track of the performance characteristics of all cloud nodes and communication links. For large systems and for tasks involving fine-grained parallelism, this would become prohibitively expensive and result in a central task scheduler becoming a bottleneck. We, therefore, argue for the need to employ a decentralized scheduling algorithm.

IBM's Air Traffic Control (ATC) algorithm [2] attempts to solve this problem by arranging cloud nodes in groups and by letting the leader of each group (the air traffic controller) direct the compute tasks (aircraft) from a central job queue to their destination worker nodes that will then execute the task. While this approach distributes the load of maintaining performance information for worker nodes, the central job queue is still a potential bottleneck and the air traffic controllers may not have enough information about the computational requirements and communication patterns of the individual tasks. The Organic Grid [5,6] is a fully decentralized approach in which nodes forward tasks to their neighbors in an overlay network. Based on performance measurements, the overlay network is dynamically restructured to improve the overall performance. Since it was designed as a desktop grid with more heterogeneity than in the cloud and with potentially unreliable communication links, however, its algorithms are unnecessarily complex for the cloud and would result in too much overhead.

In this paper, we propose a light-weight fully decentralized approach in which the tasks decide on which nodes to run based on performance information advertised by the nodes. Similar to the Organic Grid, nodes advertise their performance to their neighbors on an overlay network and task migrate along the edges of that same overlay network. By letting the application decide which nodes to run on, this decision can be based on knowledge about the computational requirements and communication patterns that may not be available to a centralized scheduler or a system-based scheduler like the ATC algorithm.

We use a vector-based model to describe node information. By plotting the nodes in an n-dimensional space, the coefficient vector and the dot product can be used to find the node with characteristics most similar to the vector. Therefore, it is possible to think of the vector as the direction in which work should flow. Trivially, any scheduler should move work from a high queue length node to a low queue length node. However, it may also be desirable to take into account a measurement of node performance, so that preference is given

to run tasks on the fastest nodes. A central component of our experiments is to determine an optimal vector to minimize the time for computing jobs on a decentralized network of nodes.

The rest of this paper is organized as follows. Section 2 summarizes related decentralized systems and motivates our approach. In Sect. 3, we present the design of our vector-based task scheduling framework. Section 4 illustrates the effectiveness of our scheduling framework using experiments. Finally, Sect. 5 discusses related work and Sect. 6 concludes the paper.

2 Motivation

Prior work has shown the possibility of using a mix of decentralized strategies for work scheduling. The Organic Grid used a hierarchical strategy to move high performing nodes closer to a source of work. The ATC uses controller nodes to organize groups of workers, and controllers compete to get work for their worker nodes. We have simulated these algorithms and identified the comparative benefits and drawbacks of each approach [17,18].

The Organic Grid's organization helps move work to high performing nodes more quickly, however it builds an overlay network centered around a single work source. In a large cloud system, we may not always have a single source for work, so we have to consider how to build a network that takes performance into account when scheduling, but does not rely on a single point source of all work. We want to take the idea of measuring performance and using that information to drive scheduling. We experimented with simulating the Organic Grid, and one trend we noticed was that when work was added in to a new location, the overlay network that was built was destroyed and recreated around the new insertion point. Because in the original Organic Grid, the mobile agents that reorganized the network were associated with individual jobs, the knowledge gained in the previous session was lost and unavailable to the next job to use the system.

The ATC algorithm contains a degree of mid-level centralization, nodes are categorized into worker and controllers. Controllers organize a group of workers, and take on groups of tasks for workers. Our simulations show that this can improve performance. Centralization allows the benefit of full utilization. A controller knows how many workers it has, and is able to pull a job that will most completely utilize the nodes. A fully decentralized solution will not know exactly how to make an ideally sized job at each scheduling step. However the downside to this solution in our simulations was the communications burden that was placed on the controller nodes.

We choose the vector scheduling approach for our experiments, since it is fully decentralized and lightweight, i.e., it does not have the potential communication bottlenecks of a centralized scheduler or the ATC controller nodes and it does not have the high overhead of the Organic Grid.

In a decentralized network, any choice to move work has to be done based on a limited amount of information available at a certain point in the network. We organize our experimental network as a graph of worker nodes. In our initial

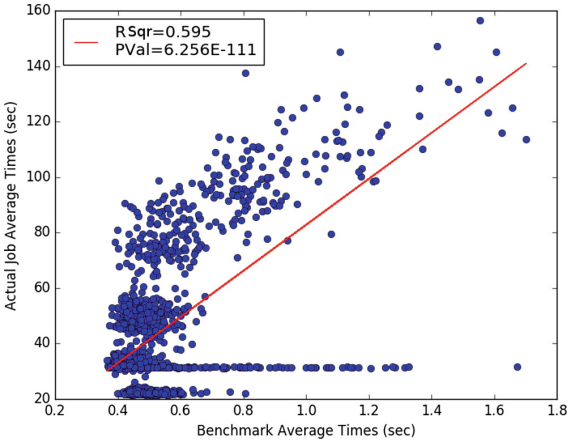


Fig. 1. Average benchmark performance vs. average measured job performance for performance and queue test

experiments, this graph is generated randomly. Each node periodically advertises its performance and status to its neighbors on the graph. Initially, we choose the queue length and the performance on the previous task as the information to advertise. When a set of jobs exist at a node that node will have to make a decision based on its view of the network about whether to leave those jobs to be completed locally or to move them to adjacent nodes in the graph.

Additionally, we need to measure the usefulness of the data we are gathering. If we measure performance, i.e., the speed at which independent tasks are completed, is our measurement going to correlate to actually completing jobs more quickly? While we had several benchmarks available to us in the form of the NAS benchmark set, which has been used in prior work to test the effectiveness of cloud computing platforms [10], how relevant are these benchmarks to actual computing tasks? While we were not able to completely answer this question before we started work, with hindsight we can measure this fairly accurately. Figure 1 shows an imperfect correlation between the job completion times and the measured performance times from the data of specifically the Queue and Performance mixed test that we discuss more in our experimental section. This test utilized the performance data to dictate where jobs were scheduled.

3 Vector-Based Job Scheduling Framework

Vector-based scheduling is the idea that there is a direction in which work should move through a network of nodes. We theorize that if the nodes can be placed in an n-dimensional space based on n measurable attributes, the best direction for work to move can be defined as a vector in that space. Therefore, we measure attributes about nodes at a given time, such as their current workload, or their performance on a given benchmark. After we describe each node's position, we

Table 1. Node information

Node	Measured		Normalized	
	Queue length	Benchmark (sec)	Queue length	Benchmark (sec)
A	7	0.8	1	1
B	1	0.1	−0.71	−1
C	0	0.2	−1	−0.71
D	1	0.6	−0.71	0.43

can compare those positions to an ideal vector using the dot product operation in order to determine which nodes best fit the direction defined by that vector. By collecting the dot product values of a group of nodes, we can create a weighted job distribution from any individual node to any set of neighbors. Using this methodology, we can make distributed scheduling decisions based on recent information at any point in the network of nodes.

As an example, suppose we measure only node performance and queue length. Node A enters a redistribution cycle, as a result of having a job surplus. Its neighbors are nodes B, C, and D. A has the information about itself and its neighbors as seen in the left part of Table 1. E.g., A has the largest number of jobs in the queue and it took 0.8 s for A to complete the benchmark run. As long as the computation cost is greater than the cost of movement, moving jobs towards idle nodes will be a good scheduling strategy We also hope to show that moving jobs towards higher performing nodes will be beneficial, and to what degree this is the case. B in this scenario is the highest performing node, and C is the node with the fewest jobs in its queue.

We standardize the measurements to distinguish between comparatively good characteristics and to evenly load the nodes. However, it can have the disadvantage of magnifying small differences in performance. Note that this example occurs for one scheduling choice at one point in time. Therefore, we will transform the set $\{7, 1, 0, 1\}$ into the range [-1, 1]. If another scheduling decision is made at a later time, the existing data at that point will be transformed independently from this decision. The transformed data is shown in the right two columns of Table 1.

An additional factor for determining where to move jobs is the cost of movement. The communication cost of moving a job from one location to another may vary from job to job depending on the memory requirements of a job. Additionally, in a decentralized network of independent decision makers, there is a danger of a thrashing, where jobs are repeatedly shifted around between the same set of nodes. We have implemented two policies to mitigate this problem, although it may be counterproductive to completely eliminate it.

The first and simplest rule to make a decision is to provided a minimum queue length below which a node will not attempt to move jobs. For our initial experiments that queue length is 1. The second possibility, which is more in line with our experimental methodology, is to provide a stasis vector as an experimental

parameter. This vector offsets the metrics of the work source node to make it a more likely candidate for work redistribution. This stasis vector is not large enough to override a significant difference in measured node quality, but should be large enough to keep insignificant changes in measured node quality from prompting wasteful rescheduling. Proving whether a stasis vector is useful, and what characteristics it should have will be a focus of the experimental stage.

4 Experimental Results

4.1 Experimental Setup

Organizing our experiments has allowed us to develop a cloud experiment management system. Cloud management systems, such as those used by CloudLab, have sophisticated tools to manage allocated resources. However, after resource allocation we often end up communicating with them via ssh. Even if we deploy our system in a Docker container, it is necessary to do some higher-level work to deploy the containers, perform the initial startup, and, crucially, to mutually identify cloud resources to one another. A general problem, of which we only attack a small part, is how to make cloud resources as automatically responsive as possible. When programming for either HPC or cloud, there are questions of scale, appropriate resource utilization, and ideal levels of parallelization, which might completely or at least approximately be automatically solvable, but still quite often they need to be configured in a tedious and manual manner. Also ideally, we want cloud resources to be as transparent as possible, but this requires intelligence to automatically allocate and schedule as much work as possible.

Building an experiment also requires describing the network that will execute the developing work. Although we consider a decentralized system that will be applicable to very large scale networks, we are somewhat restricted to the actual hardware available in our labs or on CloudLab. Therefore, most of the experiments concern how we can schedule work on heterogeneous decentralized networks, and what measurements and approaches produce measurably useful results. We can characterize a set of interconnected computing resources as a connected graph, but to determine what characteristics that graph should possess, we use the Erdős-Rényi algorithm for graph generation [9]. We define a probability that any two nodes in the graph of computing platforms are connected, and generate different graphs for different experiments. We examine any generated graph and ensure that it is connected by adding extra links between low ranked vertices of disconnected portions.

To perform our experiment, we first allocate machines and then start Docker containers on each machine. One container functions as an experimental controller, while the rest are experimental nodes. The controller is started first and is provided with information that defines the experiment. It knows how many nodes will participate in the experiment, how long the experiment should take, and what configuration information to give to the experimental nodes. The controller builds the graph that defines node-neighbor relationships, and contains

Table 2. CloudLab Clemson hardware

Name	CPU	RAM
c8220	2x Intel E5-2660 v2 10 core processor	256 GB
c6320	2x Intel E5-2583 v2 14 core processors	256 GB

the vector(s) to use in driving vector-based scheduling. This node starts listening for connection requests from other nodes. When nodes start up, they are provided with the controller's location, and contact the controller to start the experiment.

Once enough nodes have contacted the controller, it will provide each with configuration information, potentially a set of starting jobs, and a time to start executing, which will be the start time of the experiment. At that time, the nodes will begin processing work as well as communicating with their neighbors. Communication will contain information about the node's benchmark results and current work queue, as well as jobs to move from one node to another. Nodes will log all communication as well as all work they start and complete. The logs also contain time information from the perspective of the node on which the log was written. After the experiment time has elapsed, the nodes are terminated and their logs are collected. These logs form the basis of the experimental data presented in later sections.

Our first proof of concept tests concern using benchmark results to schedule independent jobs. For these tests, nodes will independently measure their own performance using a benchmark test, and then report this information to their neighbors. Additionally, we ensured that the nodes have variant performance characteristics by providing different nodes with more or fewer CPUs. This creates what we consider to be a more realistic cloud system resource limitation than our prior simulation work that produced variant performance characteristics artificially [18]. However, a suboptimal usage of those limited resources compounds a negative result across several processes that share both the same CPU and the same poor strategy. For consistent measuring and grouping, we allocated nodes such that those listening on specific ports have specific performance characteristics. More detailed hardware information is available in Table 2.

We measured the performance of the allocated cloud nodes for our tasks. The time per task averaged over 25 runs ranged from about 25s for the faster node to over 100 for the slowest node. For the faster nodes, the standard deviation was very small, for the slower nodes, the standard deviation increased with a 30s standard deviation for the slowest node.

4.2 Initial Cloud Vector Scheduling Results

The first test we performed indicated that there were essentially three phases of the scheduling problem for a decentralized scheduler with one large application to be completed by multiple nodes. By large we mean sized to completely utilize

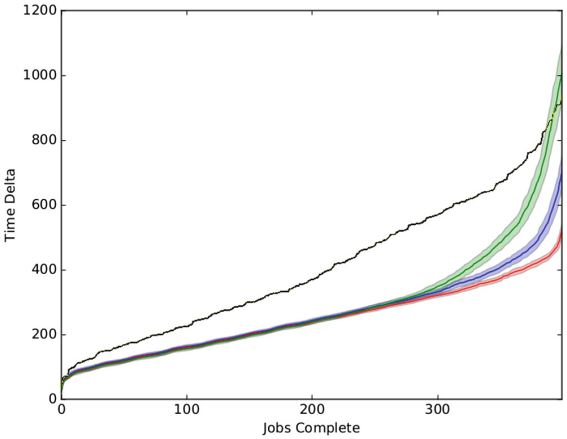


Fig. 2. Initial vector scheduling comparison showing the three scheduling phases (Color figure online)

the available computing resources for the majority of the total computation time. Phase One occurs when work moves from an extreme concentration at a single entry point to be distributed evenly throughout the network. Phase Two occurs while there is enough work to keep every computational node busy. Phase Three happens when there is still work to do, but not enough to provide work for each node. All three phases are visible in Fig. 2. The red scheduling approach is superior, but of red, blue, and green none clearly diverge until after 75% or 300 tasks are completed. This figure, as well as several more, show a job divided into 400 tasks that are gradually completed by a set of decentralized computing resources. Each line represents a single scheduling strategy, each of which was run through more than 20 experimental runs, allowing to create a 95% confidence interval, indicated by a cone around the line representing the average value. Approaches with an overall lower completion time, indicated by a lower position on the graph, are superior scheduling strategies. Note that the first and third phase each take a small portion of time, indicated by the curves at the beginning and end of each strategy.

Initial testing showed that for phases One and Two, most vector-based approaches are indistinguishable in performance, except for some extremely suboptimal methods that do not strongly weigh node queue lengths, such as the black line in Fig. 2. Phase One produced poor results originally, however this was not alleviated by using different vector scheduling strategies, but by increasing the frequency of rescheduling actions. This allowed work to spread through the network more quickly, allowing us to reach Phase Two work saturation more quickly. Phase Three proved more difficult to optimize, and the location at which the advantage of more vector-based strategies became clearer.

We also conducted experiments to determine the usefulness of our benchmark system. Initially, this was a promising route, as seen in Fig. 3. All our experiments

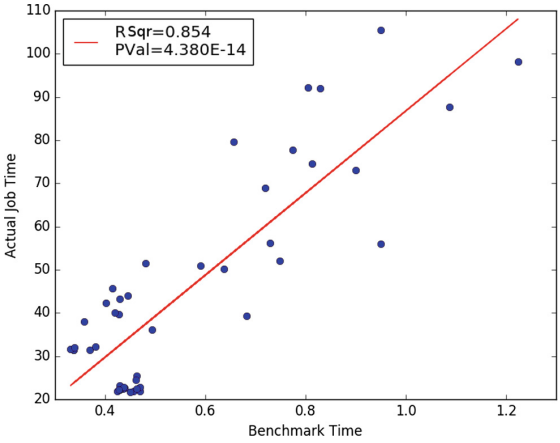


Fig. 3. Correlation between initial average benchmark and average job performance

have borne out the benchmark as a useful metric to drive scheduling at certain points, although later tests on a larger number of more diverse hardware have shown less of a relationship between a node's average benchmark and average performance, as seen in Fig. 1. The benchmark is a subset of the NAS benchmarks the Workstation class.

4.3 Cloud Vector Scheduling Experimental Results

We have demonstrated the use of benchmark values and node queue lengths as metrics to determine where work should move across computing nodes. We now examine our use of vectors to weigh the value of these metrics across several test cases on a larger platform with more performance and hardware diversity. Our first experiments are done using two different vectors, *flow vector* and *stasis vector*. The flow vector defines the characteristics of an ideal destination node, to which all available nodes' characteristics can be compared, to judge how many jobs they should be allocated. Therefore, if the flow vector defines a queue length of zero as a desirable characteristic, the nodes with the smallest queue length are considered the ideal targets for work to migrate towards. The stasis vector is meant to be a guard against unnecessary job movement and is used to prejudice scheduler towards jobs remaining on the node that they already occupy. However, the experimental results did not show that using a stasis vector was beneficial.

As mentioned previously, we utilized different strategies to optimize each of three phases in the initial experiments. Speeding up the distribution of work throughout a network is absolutely crucial to improving network performance in the first phase. The adjustment we make to do this is to allow a node that exhausted its queue to request additional rescheduling tasks from its neighbors. To avoid thrashing when the network ran low on work, we only allow a node to make one such request within a limited time frame of 10 s, and no work should be requested after a certain point in the experiment.

While this achieves the desired result of improved initial performance, the overall performance of the network suffers. Even though we attempt to avoid flooding the network with rescheduling requests at the end, the additional tax on the resources of the poor performers causes their last one or two jobs to take as much as three times as long as their already slower completion times. Recall that poor performance in this context is caused by sharing a CPU, therefore, a poor choice for a scheduling strategy on multiple nodes sharing the same CPU have a strong negative influence on the performance of each individual node. We argue that this is a valuable artifact of this experimental system, as shared resources are a potential problem with low-cost cloud systems. We eliminate this problem by creating a simpler strategy that only allows rapid rescheduling operations in response to the introduction of new work into the system.

The final phase of the network, when there is no longer enough work to satisfy every node in the system, requires a different approach. In our initial experiments, we had two goals. First, work must be moved away from low performing nodes and onto higher performing nodes. Second, any other burden on the network while the last jobs are completed should be avoided. Accomplishing the second goal involves turning off the functionality that allows nodes to request work once the initial job set has spread throughout the network. In these experiments we simply turned off the functionality after a certain amount of time in early Phase Two to stop allowing nodes to request more work. For future work however, we will look at a more sophisticated way to make this choice.

Flow-vector-based scheduling was most beneficial in optimizing the final phase. For most of the duration of an experiment, any rescheduling based on relative queue length is more or less indistinguishable. As long as it does not impose a large computational burden it will spread work throughout the network, saturating the available computational resources. However, just before some nodes become idle, it is important to move work preferentially to high performing nodes, so that the last few straggling jobs are given to nodes that will complete the jobs faster. To do this we change from a flow vector of $(-1.0, -0.3)$, which prefers a shorter queue, and considers a better performance less relevant, to a vector of $(-0.7, -0.5)$, which puts more emphasis on performance.

This methodology is compared to a method that simply uses the vector $(-1.0, 0.0)$ throughout the run in Fig. 4. Note that both of these scheduling approaches contained the optimizations to Phase One, and that neither of them produced any visible difference during the second phase. This demonstrates that the performance metric in the initial vector makes little difference until the network is no longer saturated with work. This is illustrated in Figs. 5 and 6. Note that in the queue-length-only setup there are some jobs on low performing nodes that push back the completion time of the entire system.

The fact that performance metrics are only useful near the end of the entire application, in Phase Three, leads to another possible improvement. Our initial data showed that the benchmark was a reasonable predictor of a node's job completion speed, see Fig. 3. With more data as a result of completed experiments, we can better analyze the value of the CG benchmark. The data in Fig. 1 shows

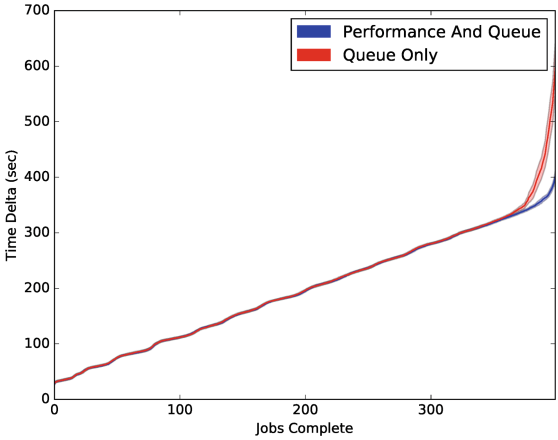


Fig. 4. Job completion time comparison between balanced queue and performance-based scheduling (blue) and queue-based scheduling (red) (Color figure online)

a much weaker relationship between average benchmark and average performance. Regardless, experimental results have shown the most recent benchmark to be useful when driving scheduling decisions as seen in Fig. 4. Additionally, this data shows the relationship between the average benchmark performance measurement, and the average job completion time. The relationship is not as consistent on the more diverse hardware used for the larger scale tests as it was on the original smaller scale setup. The measurement that is used for scheduling is not the overall average benchmark measurement but the most recent one that a node has taken. This distinction is valuable, because we suggest the most recently taken benchmark most accurately measures the node's expected performance at that time, considering that other usage of the node's hardware is a likely cause of performance degradation.

Note that in Fig. 1 the group that is spread along the x-axis (benchmark values), but are all consistently low on the y-axis (actual job performance). These measurements are all from nodes with a port in the 13000 s. The 13000 range is one of the subsets meant to be high performers, which was born out by the actual job completion rates, but not by the average benchmark measurements. Nodes in a given port range are executed on a subset of the hardware with similar characteristics, which are meant to provide either advantages or disadvantages in performance. The error bars in this figure are simply one standard deviation, indicating that the nodes with worse performance also had more highly deviating performance. We use standard deviation here, as opposed to a 95% confidence interval, as we are more interested in seeing the result of the performance modifications on the experiments that were run, and less interested in drawing conclusions about them as samples from a larger population.

Even if the average benchmark value is not useful, the most recent performance measure may be, as it did produce the more advantageous experimental

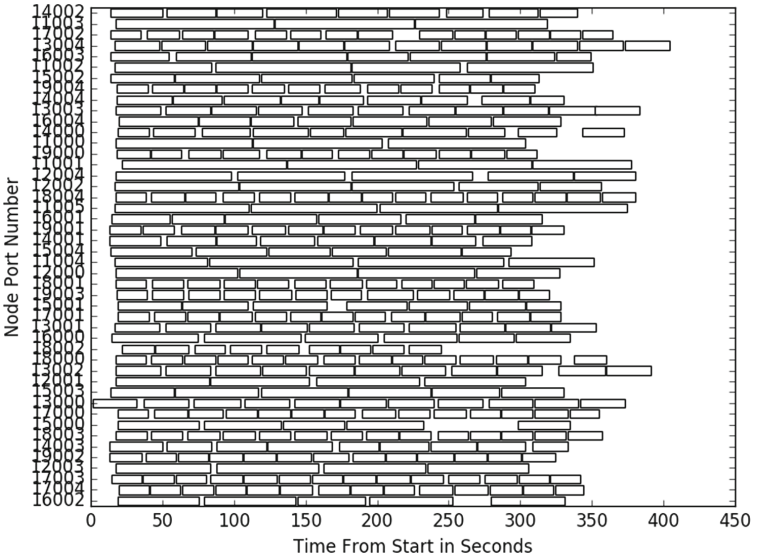


Fig. 5. Performance and queue-based scheduling for individual nodes in a single experiment

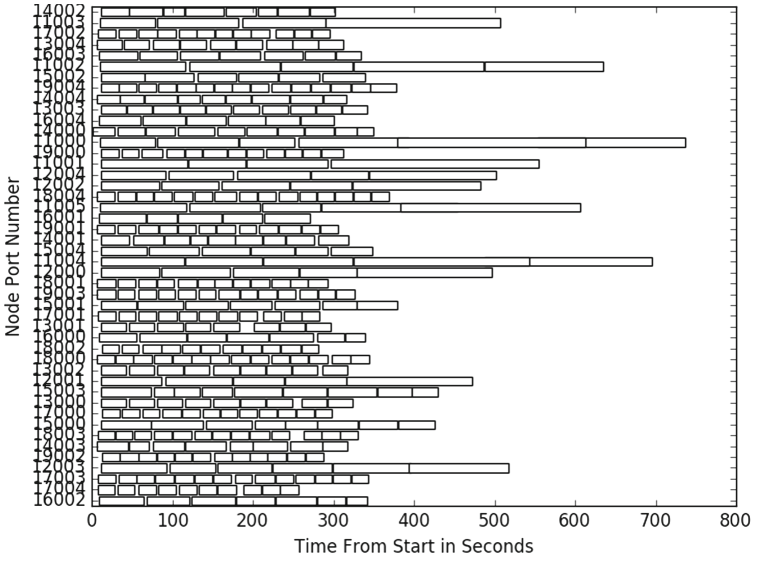


Fig. 6. Queue-based scheduling for individual nodes in a single experiment

results presented originally in Fig. 4. As a result of both the deficiencies in benchmark selection, and the result that a performance measurement is only useful near the end of work scheduling, a more simple and obvious methodology can be implemented for problems whose tasks are of consistent or predictable sizes. We considered using historical data at the beginning of the project, but initially rejected the approach because we would need some stop-gap measure to utilize before historical data was generated. A benchmark-based methodology seemed to be a better general solution, and may still be applicable given more relevant benchmarks. However, given that more than half of the work can be completed before implementing performance based scheduling, we can utilize the partial historical data available when some work has been completed to drive performance based scheduling, instead of using a benchmark. This result is shown in Fig. 7. The final phase curve is even less evident in these results, indicating that this strategy is superior.

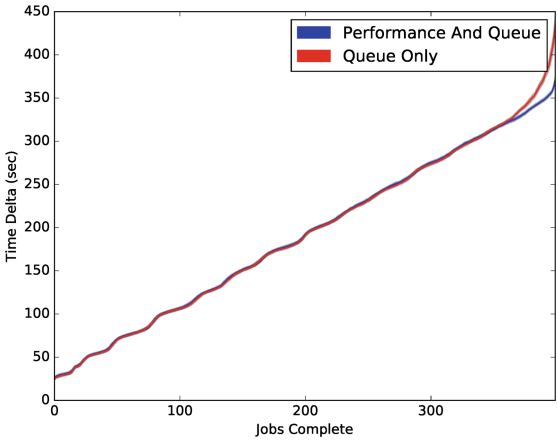


Fig. 7. Historical performance-based scheduling comparison

5 Related Work

Research on traditional grid scheduling has focused on algorithms for determining an optimal computation schedule based on the assumption that sufficiently detailed and up-to-date knowledge of the systems state is available to a single entity (the metascheduler) [1,11,21]. While this approach results in a very efficient utilization of the resources, it does not scale to large numbers of machines, since maintaining a global view of the system becomes prohibitively expensive. Variations in resource availability and potentially unreliable networks might even make it impossible.

A number of large-scale desktop grid systems have been based on variants of the master/workers model [4,7,14,16]. The fact that SETI@home had scaled to over 5 million nodes and that some of these systems have resulted in commercial enterprises shows the level of technical maturity reached by the technology. However, the obtainable computing power is constrained by the performance of the master, especially for data-intensive applications. Since networks cannot be assumed to be reliable, large desktop grids are designed for independent task applications with relatively long-running individual tasks.

Cloud computing has been very successful for several types of applications, especially for applications that do not require frequent communication between different cloud nodes, such as MapReduce [8] or graph-parallel [24] algorithms. However, for applications with fine-grained parallelism, such as the NAS MPI benchmarks or atmospheric monitoring programs, it shows less than satisfactory performance [10,23]. The main reasons are that the performance of cloud nodes is often not predictable enough, especially with virtualization, and that the communication latency is typically worse than that of a cluster [23]. While it is possible to rent dedicated clusters from cloud providers, they are significantly more expensive per compute hour than virtual machines (VMs). With virtualization, however, the user may not know whether a pair of VMs run on the same physical machine, are in the same rack, or are on different ends of the warehouse.

A related problem is the management of resources in a data center. Tso et al. [22] survey resource management strategies for data centers. Luo et al. [15] and Gutierrez-Estevez and Luo [12] propose a fine-grained centralized resource scheduler that takes application needs into consideration. By contrast, our approach uses existing VMs, measures their performance and lets the application decide which tasks to run on which nodes. However, it is not clear that a centralized scheduler would be fine-grained enough for large many-task and high-performance computing applications without creating a bottleneck.

6 Conclusion

We have proposed a decentralized scheduling approach for many-task applications that assigns individual tasks to cloud nodes in order to achieve both fast job execution time and high resource efficiency. We have presented a vector-based scheduling algorithm that assigns tasks to nodes based on measuring the compute performance and the queue length of those nodes. Our experiments with a set of tasks in CloudLab show that by applying our vector-based scheduling algorithm the application proceeds in three distinct phases: flooding the cloud nodes with tasks, a steady state in which all nodes are busy, and the final phase in which the remaining tasks are executed on the fastest nodes. We have also presented heuristics for these three phases and have demonstrated with measurements in CloudLab that they result in a reduction of the overall execution time of the many-task application. Our measurements have shown that initially it is more important to use queue-length as a metric to flood all the nodes, while in the last phase performance becomes a more important metric.

The disadvantage of a decentralized scheduling approach is that it does not guarantee perfect resource utilization, as shown by the gaps in Fig. 5. The main advantage, however, is that there is no central scheduler that could become a communication bottleneck.

The goal of this work is to allow large-scale many-task applications to be executed efficiently in the cloud. The measurements we have presented represent the first step toward this goal. For future work, we plan to use quantum chemistry simulations as a target application [3,13]. Using the approach from Rajbhandari et al. [20], this will involve scheduling a task graph with individual tasks of varying size, many of which requiring multiple nodes for a distributed tensor contraction. For good performance, it is critical that all nodes participating in a distributed contraction (or matrix multiplication) have similar performance characteristics and communication latency between them. This will require periodically running measurement probes that test the performance of each node, the communication latency with neighbors, and possibly other values. For our vector scheduling approach, we will then add communication latency as an additional dimension and allow scheduling a single task on a group of nodes with similar performance and low latency between them. A task graph with dependencies will likely result in more than three distinct phases and, therefore, require different heuristics.

Fine-tuning the performance of such applications in the cloud will require finding the best structure of the overlay network (probably based on performance measurements instead of being randomly generated), finding the most effective measurement probes and measurement frequency, and running large-scale experiments.

Acknowledgment. This research was partially supported by CloudLab (NSF Award #1419199), by the National Science Foundation under grant CNS-1566443, and by the Louisiana Board of Regents under grant LEQSF(2015-18)-RD-A-11.

References

1. Abramson, D., Giddy, J., Kotler, L.: High performance parametric modeling with Nimrod/G: killer application for the global grid? In: Proceedings of International Parallel and Distributed Processing Symposium, pp. 520–528, May 2000
2. Barsness, E., Darrington, D., Lucas, R., Santosuosso, J.: Distributed job scheduling in a multi-nodal environment. US Patent 8,645,745, 4 February 2014. http://www.google.com/patents/US8645745
3. Baumgartner, G., Auer, A., Bernholdt, D., Bibireata, A., Choppella, V., Cociorva, D., Gao, X., Harrison, R., Hirata, S., Amoorthy, S.K., Krishnan, S., Lam, C., Lu, Q., Nooijen, M., Pitzer, R., Ramanujam, J., Sadayappan, P., Sibiryakov, A.: Synthesis of high-performance parallel programs for a class of AB initio quantum chemistry models. Proc. IEEE **93**(2), 276–292 (2005)
4. Buaklee, D., Tracy, G., Vernon, M.K., Wright, S.: Near-optimal adaptive control of a large Grid application. In: Proceedings of International Conference on Supercomputing, pp. 315–326, June 2002

5. Chakravarti, A.J., Baumgartner, G., Lauria, M.: The Organic Grid: self-organizing computation on a peer-to-peer network. IEEE Trans. Syst. Man Cybern. Part A **35**(3), 373–384 (2005)
6. Chakravarti, A.J., Baumgartner, G., Lauria, M.: Self-organizing scheduling on the Organic Grid. Intl. J. High-Perf. Comput. Appl. **20**(1), 115–130 (2006)
7. Chien, A.A., Calder, B., Elbert, S., Bhatia, K.: Entropia: architecture and performance of an enterprise desktop grid system. J. Parallel Distrib. Comput. **63**(5), 597–610 (2003)
8. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
9. Erdos, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**(1), 17–60 (1960)
10. Evangelinos, C., Hill, C.N.: Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. In: 1st Workshop on Cloud Computing and its Applications (CCA) (2008)
11. Grimshaw, A.S., Wulf, W.A.: The Legion vision of a worldwide virtual computer. Commun. ACM **40**(1), 39–45 (1997)
12. Gutierrez-Estevez, D.M., Luo, M.: Multi-resource schedulable unit for adaptive application-driven unified resource management in data centers. In: 2015 International Telecommunication Networks and Applications Conference (ITNAC), November 2015
13. Hartono, A., Lu, Q., Henretty, T., Krishnamoorthy, S., Zhang, H., Baumgartner, G., Bernholdt, D.E., Nooijen, M., Pitzer, R.M., Ramanujam, J., Sadayappan, P.: Performance optimization of tensor contraction expressions for many-body methods in quantum chemistry. J. Phys. Chem. **113**(45), 12715–12723 (2009)
14. Litzkow, M., Livny, M., Mutka, M.: Condor – a hunter of idle workstations. In: Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104–111, June 1988
15. Luo, M., Li, L., Chou, W.: ADARM: an application-driven adaptive resource management framework for data centers. In: 2017 IEEE International Conference on AI and Mobile Services (AIMS), June 2017
16. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D.A., Freund, R.F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the 8th Heterogeneous Computing Workshop, pp. 30–44, April 1999
17. Peterson, B.: Decentralized scheduling for many-task applications in the hybrid cloud. Ph.D. thesis, Louisiana State University, Baton Rouge, LA, May 2017
18. Peterson, B., Baumgartner, G., Wang, Q.: A decentralized scheduling framework for many-task scientific computing in a hybrid cloud. Serv. Trans. Cloud Comput. **5**(1), 1–13 (2017)
19. Raicu, I., Foster, I.T., Zhao, Y.: Many-task computing for grids and supercomputers. In: Proceedings of the 2008 Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS 2008), Austin, TX, pp. 1–11. IEEE November 2008
20. Rajbhandari, S., Nikam, A., Lai, P.W., Stock, K., Krishnamoorthy, S., Sadayappan, P.: A communication-optimal framework for contracting distributed tensors. In: Proceedings of SC 2014, the International Conference on High Performance Computing, Networking, Storage, and Analysis, New Orleans, LA, 16–21 November 2014 (2014)
21. Taylor, I., Shields, M., Wang, I.: 1 - resource management of Triana P2P services. In: Nabrzyski, J., Schopf, J.M., Węglarz, J. (eds.) Grid Resource Management. Springer, Boston (2003)

22. Tso, F.P., Jouet, S., Pezaros, D.P.: Network and server resource management strategies for data centre infrastructures: a survey. Comput. Netw. **106**(4), 209–225 (2016)
23. Walker, E.: Benchmarking Amazon EC2 for high-performance scientific computing. In: LOGIN, vol. 33, no. 5, pp. 18–23 (2008)
24. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: GraphX: a resilient distributed graph system on Spark. In: First International Workshop on Graph Data Management Experiences and Systems, GRADES 2013, pp. 2:1–2:6. ACM, New York (2013)

Mitigating Multi-tenant Interference in Continuous Mobile Offloading

Zhou Fang[1(✉)], Mulong Luo[2], Tong Yu[3], Ole J. Mengshoel[3], Mani B. Srivastava[4], and Rajesh K. Gupta[1]

[1] University of California San Diego, San Diego, USA
zhoufang@ucsd.edu
[2] Cornell University, Ithaca, USA
[3] Carnegie Mellon University, Pittsburgh, USA
[4] University of California, Los Angeles, Los Angeles, USA

Abstract. Offloading computation to resource-rich servers is effective in improving application performance on resource constrained mobile devices. Despite a rich body of research on mobile offloading frameworks, most previous works are evaluated in a single-tenant setting, *i.e.*, a server is assigned to a single client. In this paper we consider that multiple clients offload various continuous mobile sensing applications with end-to-end delay constraints, to a cluster of machines as the server. Contention for shared computing resources on a server can unfortunately result in delays and application malfunctions. We present a two-phase *Plan-Schedule* approach to mitigate multi-tenant resource contention, thus to reduce offloading delays. The planning phase predicts future workloads from all clients, estimates contention, and devises offloading schedule to remove or reduce contention. The scheduling phase dispatches arriving offloaded workloads to the server machine that minimizes contention, according to the running workloads on each machine. We implement the methods into *ATOMS* (Accurate Timing prediction and Offloading for Mobile Systems), a framework that adopts prediction of workload computing times, estimation of network delays, and mobile-server clock synchronization techniques. Using several mobile vision applications, we evaluate ATOMS under diverse configurations and prove its effectiveness.

1 Introduction

Problem Background: Recent advances in mobile computing have made many interesting vision and cognition applications feasible. For example, cognitive assistance [1] and augmented reality [2] applications process continuous streams of image data to provide new capabilities on mobile platforms. However, advances in computing power on embedded devices do not satisfy such growing needs. To extend mobile devices with richer computing resources, offloading computation to remote servers has been introduced [1,3–5]. The servers can be either deployed in low-latency and high-bandwidth local clusters that provide timely offloading

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 20–36, 2018.
https://doi.org/10.1007/978-3-319-94295-7_2

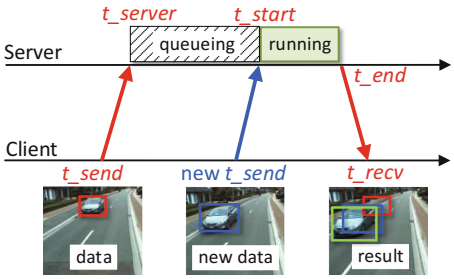


Fig. 1. ATOMS predicts processor contention and adjusts the offloading time (*t_send*) to avoid contention (queueing). (Color figure online)

services, as envisioned by the cloudlet [4], or the cloud that provides best-effort services.

A low end-to-end (E2E) delay on a sub-second scale is critical for many vision and cognition applications. For example, it ensures seamless interactivity for mobile applications [1], and a low sensing-to-actuation delay for robotic systems [6]. Among previous works on reducing offloading delays [1,10], a simple single-tenant setting that one client is assigned to one server is usually used to evaluate prototypes. However, in a practical scenario that servers handle tasks from many clients running diverse applications, contention on the shared server resources may raise up E2E delays and degrade application performance. Unfortunately, this essential issue of multi-tenancy is still untapped in these works.

While cloud schedulers have been well engineered to handle a wide range of jobs, new challenges arise in handling offloaded mobile workloads. First, there are stringent limits on server utilizations for conventional low latency web services [7]. However, computer vision and cognitive workloads are much more compute-intensive, which results in a large infrastructure cost to keep utilization levels low. Indeed, it is even not feasible for cloudlet servers that are much less resourceful than cloud. Second, there are many works on scheduling of batch data processing tasks with time-based Service-Level-Objectives (SLOs) [8,11,12]. However, these methods are inadequate in handling mobile workloads that desire sub-second E2E delays, compared to data processing tasks with minutes makespans and deadlines to hours.

Our Approach: This paper presents *ATOMS*, a mobile offloading framework that maintains low delays even under a high server utilization. Motivated by low-latency mobile applications, ATOMS consider a cloudlet setting where mobile clients connect to servers via high-bandwidth Wi-Fi networks, as in [1,4]. On the basis of load-aware scheduling, ATOMS controls future task offloading times in a client-server closed loop, to remove processor contention on servers. See Fig. 1, a client offloads an object detection task to a multi-tenant server. Due to processor contention, it may be queued before running. By predicting processor contention, the server notifies the mobile client to postpone offloading.

(a) Processor Contention (b) ATOMS

Fig. 2. 4 clients offload DNN object detection tasks to a 8-core server with periods 2 s to 5 s. The time slots of offloaded tasks are plotted and the load line (red) gives the total number of concurrent tasks. The dashed curve gives the CPU usage (cores) of the server. (Color figure online)

The postponed offloaded task is processed without queueing, and thus provides a more recent scene (blue box) that better localizes the moving car (green box).

Accordingly, we propose the *Plan-Schedule* strategy: (i) in the ***planning*** phase, ATOMS predicts time slots of future tasks from all clients, detects contention, coordinates tasks, and informs clients about new offloading times; (ii) in the ***scheduling*** phase, for each arriving task, ATOMS selects the machine that has minimal estimated processor contention to execute it. Figure 2 illustrates the effectiveness of ATOMS for removing processor contention. Figure 2a shows the time slots of periodically offloaded tasks. The load lines (red) give the total number of concurrent tasks, and contention (queueing) takes place when the load exceeds 1. Figure 2b shows that contention is almost eliminated because of the dynamic load prediction and the task coordination by ATOMS.

The challenge of deciding the right offloading times is that the server and the clients form an *asynchronous distributed system*. For scheduling activities, the uncertainties of wireless network delays and clock offsets must be carefully considered in the timeline of each task. ATOMS leverages accurate estimations on bounds of delays and offsets to handle the uncertainties. Variabilities of task computing times put additional uncertainties on the offloading timing, which are estimated by time series prediction. The predictability relies upon the correlation of continuous sensor data from cameras.

In addition, to ensure a high usability in diverse operating conditions, ATOMS includes the following features: (i) the support for heterogeneous server machines and applications with different levels of parallelism; (ii) the client-provided SLOs that control the offloading interval deviations from the desired period, which are caused by dynamic task coordination activities; (iii) the deployment of applications in containers, which are more efficient than virtual machines (VMs), to hide the complexities of programming languages and dependencies. ATOMS can be deployed in cloud environments and mobile networks as well, where removing resource contention is more challenging due to higher network uncertainties and network bandwidth issues. We analyze these cases by experiments using simulated LTE network delays.

This paper makes three contributions: (i) a novel Plan-Schedule scheme that coordinates future offloaded tasks to remove resource contention, on top of load-aware scheduling; (ii) a framework that accurately estimates and controls the timing of offloading tasks through computing time prediction, network latency estimation and clock synchronization; and (iii) methods to predict processor usage and detect multi-tenant contention on distributed container-based servers.

The rest of this paper is organized as follows. We discuss the related work in Sect. 2, and describe the applications and the performance metrics in Sect. 3. In Sect. 4 we explain the offloading workflow and the plan-schedule algorithms. Then we detail the system implementations in Sect. 5. Experimental results are analyzed in Sect. 6. In Sect. 7 we summarize this paper.

2 Related Work

Mobile Offloading: Many previous works are on reducing E2E delays in mobile offloading frameworks [1,9,10]. Gabriel [1] deploys cognitive engines in a nearby cloudlet that is only one wireless hop away to minimize network delays. Timecard [9] controls the user-perceived delays by adapting server-side processing times, based on measured upstream delays and estimated downstream delays. Glimpse [10] hides network delays of continuous object detection tasks by tracking objects on the mobile side, based on stale results from the server. This paper studies the fundamental issue of resource contention on multi-tenant mobile offloading servers, however, not yet considered by the previous works.

Cloud Schedulers: Workload scheduling in cloud computing has already been intensely studied. These systems leverage rich information, for example, estimates and measurements on resource demands and running times, to reserve and allocate resources, and reorder tasks in queue [8,11,12]. Because data processing tasks have much larger time scales of makespan and deadline, usually ranging from minutes to hours, these methods are inadequate in handling real-time mobile offloading tasks that desires sub-second delays.

Real-Time Schedulers: Real-time (RT) schedulers in [13–15] are designed for low latency and periodical tasks on multi-processor systems. However, these schedulers do not work in the scenario of mobile offloading. First, the RT schedulers can not handle network delays and uncertainties. Second, the RT schedulers are designed to minimize deadline miss rates, whereas our goal is to minimize E2E delays. In addition, the RT schedulers use worst-case computing times in scheduling. It results in an undesired low utilization for applications with highly varying computing times. As a novel approach for the mobile scenarios, ATOMS makes dynamic predictions and coordinations for incoming offloaded tasks, using estimated task computing times and network delays.

3 Mobile Workloads

3.1 Applications

Table 1 describes the vision and cognitive applications used for testing our work. They all require low E2E delays: FaceDetect and ObjectDetect lose trackability as delay increases; FeatureMatch can be used in robotics and autonomous systems to retrieve depth information for which timely response is indispensable. In another aspect, the three applications present differences in parallelism and variability of computing time. We use the differences to explore the design of a general and highly usable offloading framework.

Table 1. Test applications

Application	Functionalities	Time	Parallelism
Face detection	Haar feature cascade classifiers [16] in OpenCV [17]	Variable	Single-threaded
Feature matching	Detects interest points in left and right frames from a binocular camera, extracts and matches SURF [18] features	Variable	Feature extraction on two threads, then matching on one thread
Object detection	Localizes objects and labels each with a likeliness score using a DNN (YOLO [19])	Constant	Uses all cores of a CPU in parallel

3.2 Performance Metrics

We denote a mobile client as C_i with an ID i. The offloading server is a distributed system composed of resource-rich machines. An offloading request sent by C_i to the server is denoted as task T_j^i, where the task index j is a monotonically increasing sequence number. We ignore the superscript for simplicity when discussing only one client. Figure 1 shows the life cycle of an offloaded task. T_j is sent by a client at t_send_j. It arrives at a server at t_server_j. After queueing, the server starts to process it at t_start_j and finishes at $t\_end_j = t\_start_j + d\_compute_j$, where $d\_compute_j$ is the computing time.[1] The client receives the result back at t_recv_j. T_j uses T_paral_j cores in parallel.

We evaluate a task using two primary performance metrics of continuous mobile sensing applications [20]. **E2E delay** is calculated as $d\_delay_j = t\_recv_j - t\_send_j$. It comprises upstream network delay d_up_j, queueing delay d_queue_j, computing time $d\_compute_j$ and downstream delay d_down_j. ATOMS reduces d_delay_j by minimizing d_queue_j. **Offloading interval** represents

[1] A symbol starting with "$t\_$" is a timestamp and "$d\_$" is a duration of time.

the time span between successive offloaded tasks of a client, calculated as $d\_interval_j = t\_send_j - t\_send_{j-1}$. Clients offload tasks periodically and are free to adjust offloading periods. Applications can thus tune offloading period for energy consumption and performance trade-off. Ideally any interval is equal to d_period_i, the current period of client C_i. In ATOMS, however, the interval becomes non-constant due to task coordination. We desire stable sensing and offloading activities, so smaller interval jitters are preferred, given by $d\_jitter_j^i = d\_interval_j^i - d\_period_i$.

4 Framework Design

As shown in Fig. 3, ATOMS is composed of one *master* server and multiple *worker* servers. The master communicates with clients and dispatches tasks to workers for execution. It is responsible for planning and scheduling tasks.

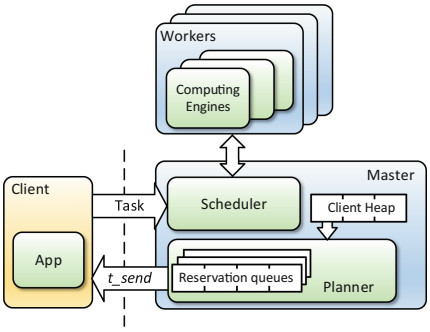


Fig. 3. The architecture of the ATOMS framework.

4.1 Worker and Computing Engine

We first describe how to deploy applications on workers. A worker machine hosts one or more *computing engines*. Each computing engine runs an offloading application encapsulated in a container. Our implementation adopts *Docker* containers.[2] We use Docker's resource APIs to set processor share, limit and affinity, as well as memory limit for each container. We focus on CPUs as the computing resource in this paper. The total number of CPU cores of worker W_k is W_cpu_k. The support for GPUs lies in our future work.

A worker can have multiple engines for the same application in order to fully exploit multi-core CPUs, or host different types of engines to share the machine by multiple applications. In this case, the total workloads of all engines on a worker may exceed the limit of processor resource (W_cpu). Accordingly,

[2] Docker: https://www.docker.com/.

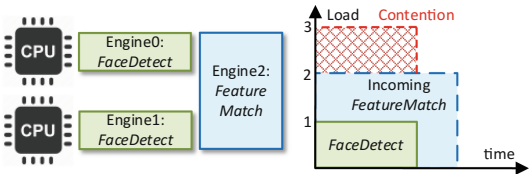


Fig. 4. A dual-core worker machine have two engines of FaceDetect ($T\_paral = 1$) and one engine of FeatureMatch ($T\_paral = 2$). The plot on right gives an example of processor contention.

we classify workers into two types: *reserved worker* and *shared worker*. On a **reserved worker**, the sum of processor usages of engines never exceed W_cpu. Therefore whenever there is a free computing engine, it is guaranteed that dispatching a task to it does not induce any processor contention. Unlike a reserved worker, the total workloads on a **shared worker** may exceed W_cpu. See Fig. 4, a dual-core machine hosts two FaceDetect engines ($T\_paral = 1$) and one FeatureMatch engine ($T\_paral = 2$). Both applications are able to fully utilize the dual-core processor. When there is a running FaceDetect task, an incoming FeatureMatch task will cause processor contention. *Load-aware scheduling* described in Sect. 4.4 is used for shared workers.

Workers measure the computing time $d\_compute$ of each task and returns it to the master along with the computation result. The measurements are used to predict $d\_compute$ for future tasks (Sect. 5.1). A straightforward method is measuring the start and end timestamps of a task, and calculating the difference ($d\_compute_{ts}$). However, it is vulnerable to processor sharing that happens on shared workers. We instead get $d\_compute$ by measuring *CPU time* ($d\_cputime$) consumed by the engine container during the computation.

4.2 Master and Offloading Workflow

In addition to the basic send-compute-receive offloading workflow, ATOMS has three more steps: *creating reservation*, *planning*, and *scheduling*.

Reservation: When the master starts the planning phase of task T_j, it creates a new reservation $R_j = (t\_r\_start_j,\ t\_r\_end_j,\ T\_paral_j)$, where $t\_r\_start_j$ and $t\_r\_end_j$ are the start and end times respectively, and T_paral_j is the demanded cores. As shown in Fig. 5, given the lower and upper bounds of upstream network delay ($d\_up_j^{low}$, $d\_up_j^{up}$) estimated by the master, as well as the predicted computing time ($d\_compute'_j$), the span of reservation is calculated as $t\_r\_start_j = t\_send_j + d\_up_j^{low}$ and $t\_r\_end_j = t\_send_j + d\_up_j^{up} + d\_compute'_j$. The time slot of R_j contains the uncertainty of the time when T_j arrives at the server (t_server_j), and the time consumed by computation. Provided that the predictions on network delays and computing times are correct, the future task will be within the reserved time slot.

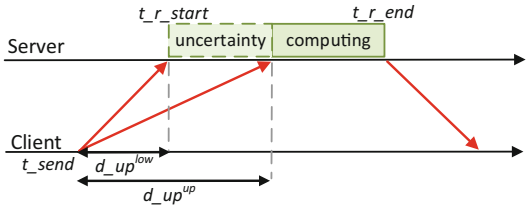


Fig. 5. Processor reservation for a future offloaded task includes the uncertainty of arriving time at the server and its computing time.

Planning: The planning phase runs before the real offloading. It coordinates future tasks of all clients to ensure that the total amount of all reservations never exceeds the limit of total processor resources of all workers. Client C_i registers at the master to initialize the offloading process. The master assigns it a future timestamp t_send_0 indicating when to send the first task. The master creates a reservation for task T_j and plans it when $t_{now} = t\_r\_start_j - d\_future$ where t_{now} is the master's current clock time, and d_future is a parameter for how far after t_{now} that the planning phase covers. The planner predicts and coordinates future tasks that start before $t_{now} + d\_future$.

T^i_{next} is the next task of client C_i to plan. The master plans future tasks in ascending order of start time $t\_r\_start^i_{next}$. For a new task to be planned with the earliest $t\_r\_start^i_{next}$, the planner creates a new reservation R^i_{next}. The planner takes R_{next} as input. It detects resource contention, and reduces that by adjusting the sending times of both the new task and a few planned tasks. We defer the details of planning to Sect. 4.3. d_inform_i is a parameter of C_i for how early the master should inform the client about the adjusted task sending time. A reservation R^i_j remains adjustable until $t_{now} = t\_send^i_j$ - d_inform_i. The planner then removes R_j and notifies the client. Upon receiving t_send_j, the client sets a timer to offload T_j.

Scheduling: The client offloads T_j to the master when the timer at t_send_j timeouts. After receiving the task, using the information of currently running tasks on each worker, the scheduler selects the worker that induces the least processor contention. The master dispatches it to the worker and gets back the result. We give the details in Sect. 4.4.

4.3 Planning Algorithms

The planning algorithm decides the adjustments to sending times of future tasks from each client. An optimal algorithm minimizes jitters of offloading intervals, while ensuring that the total processor usage is within the limit, and SLOs on offloading intervals are satisfied. Instead of solving this complex optimization problem numerically, we adopt a heuristic and feedback-control approach that adjusts future tasks in a fixed window from $t_{now} + d\_inform$ to $t_{now} + d\_future$. Our approach is able to improve the accuracy of computing time prediction by

using a small predicting window (see Sect. 5.1), and naturally handle changes of client number and periods.

The planner buffers reservations in *reservation queues*. A reservation queue stands for a portion of processor resource in the cluster. A queue Q_k has a resource limit Q_cpu_k with cores as the unit, used for contention detection. The sum of Q_cpu of all queues is equal to the total cores in the cluster. Each computing engine is assigned to a reservation queue. The parallelism of a reservation T_paral is determined by the processor limit of computing engines. For example, T_paral of a fine-parallelized task is different for an engine on a dual-core worker ($T\_paral = 2$) and one on a quad-core worker ($T\_paral = 4$).

Contention Detection: When the planner receives a new reservation R_{new}, it first selects a queue to place it in. It iterates over all queues, for Q_k, calculates the needed amount of time (Δ) to adjust R_{new}, and the total processor usage (Θ_k) of Q_k during the time slot of R_{new}. The planner selects the queue with the minimal Δ. In doing so, it checks whether the total load on Q_k after adding R_{new} exceeds the limit Q_cpu. If so, the algorithm calculates Δ: the contention can be eliminated after postponing R_{new} by Δ. Otherwise $\Delta = 0$. We give an example in Fig. 6 that a new reservation R_0^2 is being inserted into a queue. The black line in the lower plot is the total load. Contention arises after adding R_0^2. It can be removed by postponing R_0^2 to the end time of R_1^1. Δ is thus obtained.

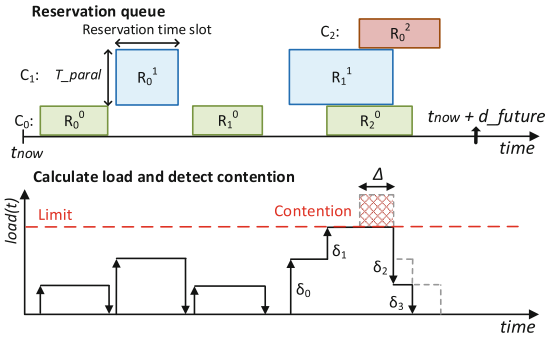


Fig. 6. An example of detecting processor contention and calculating required reservation adjustment. The top plot shows a reservation queue and the bottom plot shows the calculated total load *load(t)*.

If two or more planning queues have the same Δ, *e.g.*, several queues are contention-free ($\Delta = 0$), the planner calculates the processor usage Θ during the time slot of R_{new}: $\Theta = \int_{t\_r\_start_{new}}^{t\_r\_end_{new}} load(t)dt$. We consider two strategies. The *Best-Fit* strategy selects Q that has the highest Θ, which packs reservations as tightly as possible and leaves the least margin of processor resources on the queue. The other strategy is *Worst-Fit* that, in contrast, selects the queue with the lowest Θ. We study their difference through evaluations in Sect. 6.3.

SLOs: In the next *coordination* step, rather than simply postponing R_{new} by Δ, the planner moves ahead a few planned reservations as well, to reduce the duration to postpone R_{new}. The coordination process takes Δ as input and adjusts reservations according to *cost* (R_cost), a metric on how far the measured offloading interval $d\_interval$ deviates from the client's SLOs (a list of desired percentiles of $d\_interval$). For example, a client with period 1 s may require a lower bound $d\_slo^{10\%} > 0.9$ s and an upper bound $d\_slo^{90\%} < 1.1$ s.

The master calculates R_cost when it plans a new task, using the measured percentiles of interval ($d\_interval^p$). For the new reservation (R_{new}) to be postponed, the cost R_cost^+ is obtained from the upper bounds: $R\_cost^+ = \max(\max_{p \in \cup^+}(d\_interval^p - d\_slo^p),\ 0)$ where p is a percentile and $\cup^+$ is the set of percentiles that have upper bounds in the SLOs. R_cost^+ is the maximal interval deviation from the SLOs. For tasks to be moved ahead, deviation from lower bounds ($\cup^-$) are used instead to get the cost R_cost^-. The cost is a weight between two clients to decide the adjustment on each. SLOs with tight bounds on $d\_interval$ make the client less affected during the coordination process.

4.4 Scheduling Algorithms

The ATOMS scheduler dispatches tasks arriving at the master to the most suitable worker machine that minimizes processor contention. The scheduler keeps a global FIFO task queue for buffer tasks when all computing engines are busy. For each shared worker, there is a local FIFO queue for each application that it serves. When a task arrives, the scheduler first searches for available computing engines on any reserved workers. It dispatches the task if one is found and the scheduling process ends. If there is no reserved worker, or no engine is free, the scheduler checks shared workers that are able to run the application. It selects the best worker based on processor contention Φ and usage Θ. The task is then dispatched to a free engine on the selected shared worker. If no engine is free on the worker, the task is put into the worker's local task queue.

Here we detail the policy to select a shared worker. The scheduler uses estimated end time t_end' of all running tasks on each worker, obtained by predicted computing time $d\_compute'$. To schedule T_{new}, it calculates the load $load(t)$ on each worker using t_end', including T_{new}. The resource contention Φ_k on worker W_k is calculated by $\Phi_k = \int_{t_{now}}^{t\_end'_{new}} \max(load(t) - W\_cpu_k,\ 0)dt$. The worker with the smallest Φ is selected to run T_{new}. For workers with identical Φ, similar to the planning algorithm, we use processor usage Θ as the selection metric. We compare the two selection methods, Best-Fit and Worst-Fit, through evaluations in Sect. 6.

5 Implementation

In this section we present the implementation of computing time prediction, network delay estimation and clock synchronization.

5.1 Prediction on Computing Time

Accurate prediction of computing time is essential for resource reservation. Underestimation leads to failure in detecting resource contention, and overestimation causes larger interval jitters. We use *upper bound estimation* for applications with a low variability of computing times, and *time series prediction* for applications with a high variability. Given that T_n is the last completed task of client C_i, instead of just predicting T_{n+1} (the next task to run), ATOMS needs to predict T_{next} (the next task to plan, $next > n$). $N_{predict} = next - n$ gives how many values it needs to predict since the last sample. It is decided by the parameter d_future (Sect. 4.2) and the period of the client, calculated as $\lceil d\_future/d\_period_i \rceil$.

Upper Bound Estimation. The first method estimates the upper bound of samples using a TCP retransmission timeout estimation algorithm [21]. We denote the value to predict as y. The estimator keeps a smoothed estimation $y^s \leftarrow (1-\alpha) \cdot y^s + \alpha \cdot y_i$.) and a variation $y^{var} \leftarrow (1-\beta) \cdot y^{var} + \beta \cdot |y^s - y_i|$. The upper bound y^{up} is given by $y^{up} = y^s + \kappa \cdot y^{var}$, where α, β and κ are parameters. This method outputs y^{up} as the prediction of $d\_compute$ for T_{next}. This lightweight method is adequate for applications with low computing time variability, such as ObjectDetect. It tends to overestimate for applications with highly varying computing times because it uses upper bound as the prediction.

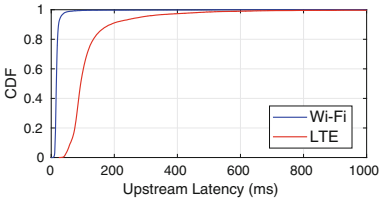
Time Series Linear Regression. In the autoregressive model for time series prediction problems, the value y_n at index n is assumed to be a weighted sum of previous samples in a moving window with size k. That is, $y_n = b + w_1 y_{n-1} + \cdots + w_k y_{n-k} + \epsilon_n$, where y_{n-i} is the ith sample before the nth, w_i is the corresponding coefficient and ϵ_n is the noise term. We use this model to predict y_n. The inputs (y_{n-1} to y_{n-k}) are the previous k samples measured by workers. We use a recursive approach to predict the $N_{predict}$th sample after y_{n-1}: to predict y_{i+1}, the predicted y_i is used as the last sample. This approach is flexible to predict arbitrary future samples, however, as $N_{predict}$ increases, the accuracy degrades because the prediction error is accumulated. The predictor keeps a model for each client which is trained either online or offline.

5.2 Estimation on Upstream Latency

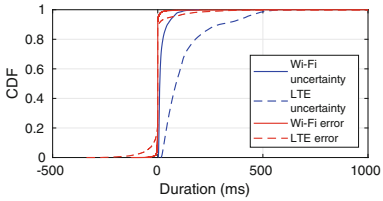
As discussed in Sect. 4.2, because network delays d_{up} may have large fluctuations, we use the lower and upper bounds (d_up^{low}, d_up^{up}) instead of the exact value in the reservation. The TCP retransmission timeout estimator [21] described in Sect. 5.1 is used to estimate network delay bounds. We use subtraction instead of addition to obtain the lower bound. The estimator has a non-zero *error* when a new sample of d_{up} falls out of the bounds, calculated as its deviation from the nearest bound. The error is positive if d_{up} exceeds d_up^{up}, and negative if it is smaller than d_up^{low}. The estimation uncertainty is given by d_up^{up} - d_up^{low}.

Because the uncertainty is included in task reservation, a larger uncertainty overclaims the reservation time slot, which causes higher interval jitters and lower processor utilizations.

We measure d_{up} of offloading 640×480 frames with sizes from 21 KB to 64 KB, using Wi-Fi networks. To explore networks with higher delays and uncertainties, we obtain simulated delays of Verizon LTE networks using the Mahimahi tool [22]. The CDFs of network latencies are plotted in Fig. 7a. We demonstrate the estimator performance (error and uncertainty) in Fig. 7b. Results show that the estimation uncertainty for Wi-Fi networks is small, and it is very large for LTE (maximal value is 2.6 s). We demonstrate how errors and uncertainties influence offloading performance through experiments in Sect. 6.



(a) Upstream latency.



(b) Uncertainty and error.

Fig. 7. Upstream latency of Wi-Fi and LTE networks, with uncertainty and error of estimation. The maximal latency is 1.1 s for Wi-Fi and 3.2 s for LTE. The parameters of estimator are $\alpha = 0.125$, $\beta = 0.125$, $\kappa = 1$.

5.3 Clock Synchronization

We seek a general solution for clock synchronization without patching the OS of mobile clients. The ATOMS master is synchronized to the global time using NTP. Because time service now is ubiquitous on mobile devices, we require clients to be *coarsely* synchronized to the global time. We do *fine* clock synchronization as follows. Client sends out a NTP synchronization request to the master each time it receives an offloading result to avoid the wake-up delay [9]. To eliminate the influence of packet delay spikes, the client buffers N_{ntp} responses and runs a modified NTP algorithm [23]. It applies clock filter, selection, clustering and combining algorithms to N_{ntp} responses and outputs a robust estimate on clock offset. It also outputs the bounds of the offset. ATOMS uses the clock offset to synchronize timestamps between clients and the master, and uses the bounds in all timestamp calculations to consider the remaining clock uncertainties.

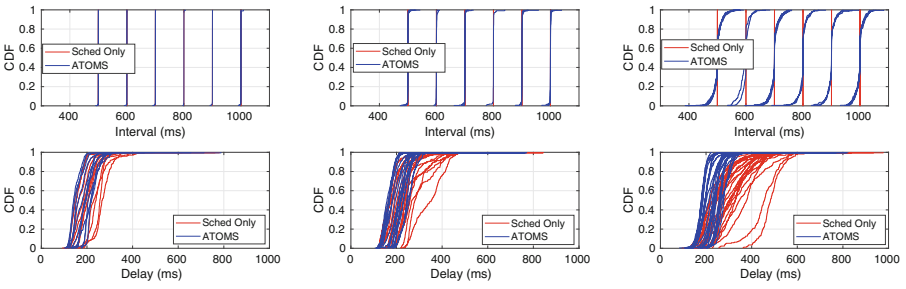
6 Evaluation

6.1 Experiment Setup

Baselines: We compare ATOMS with baseline schedulers to prove its effectiveness for improving offloading performance. The baseline schedulers use

load-aware scheduling (Sect. 4.4), but instead of using dynamic offloading time coordination in ATOMS, they use conventional task queueing and reordering approaches: (i) *Scheduling Only*: it minimizes the longest task queueing time; (ii) *Earliest-start-time-first*: it prioritizes the task with the smallest start time at client (*t_send*), which experiences the longest lag until now; (iii) *Longest-E2E-delay-first*: it prioritizes the task with the longest estimated E2E delay, including measured upstream and queueing delays, and the estimated computing time. Methods (ii) and (iii) are evaluated in the experiments using LTE networks (Sect. 6.2) where they perform differently from (i) due to larger upstream delays.

Testbed: We simulate a camera feed to conduct reproducible experiments. Each client selects which frame to offload from a video stream based on the current time and the frame rate. We use three public video datasets as the camera input: the Jiku datasets [24] for FaceDetect; the UrbanScan datasets [25] for FeatureMatch application, and multi-camera pedestrians videos [26] for ObjectDetect. We resize the frames to 640×480 in all the tests. Each test runs for 5 minutes. The evaluations are conducted on AWS EC2. The master runs on a c4.xlarge instance (4 vCPUs, 7.5 GB memory). Each worker machine is a c4.2xlarge instance (8 vCPUs, 15 GB memory). We emulate clients on c4.2xlarge instances. Pre-collected network upstream latencies (as described in Sect. 5.2) are replayed at each client to emulate the wireless networks. The prediction for FaceDetect and FeatureMatch uses offline linear regression, and the upper bound estimator is used for ObjectDetect. The network delay estimator setting is the same as in Fig. 7. We set d_inform (Sect. 4.2) to 300 ms for all clients.



(a) 12 clients. (b) 18 clients. (c) 24 clients.

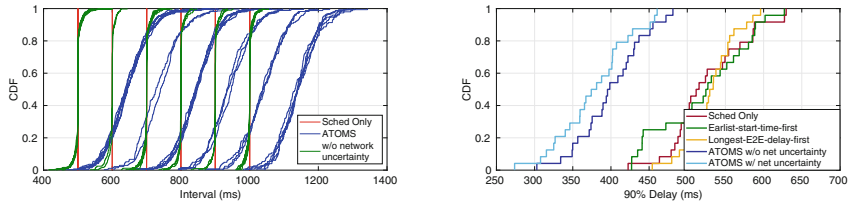
Fig. 8. Offloading performance (CDF of each client) of Scheduling Only and ATOMS running FaceDetect using Wi-Fi. The average CPU utilization is 37% in (a), 56% in (b) and 82% in (c).

6.2 Maintaining Low Delays Under High Utilization

We set 12 to 24 clients running FaceDetect with periods from 0.5 s to 1.0 s, using Wi-Fi networks. $d\_future = 2$ s is used in planning. We use one worker machine

(8 vCPUs) hosting 8 FaceDetect engines. The planner has a reservation queue for each engine with $Q\_cpu = 1$. See Fig. 8, with more clients, the interference becomes more intensive and the Sched Only scheme suffers from increasingly longer E2E delays. ATOMS is able to maintain low E2E delays even when the total CPU utilization is over 80%. Using the case with 24 clients as example, the 90% percentile of E2E delays is reduced by 34% in average for all clients, and the maximum reduction is 49%. The interval plots (top) show that offloading interval jitters increase in ATOMS, caused by task coordination.

LTE Networks: To investigate how ATOMS performs under larger network delays and variances, we run the test with 24 clients using LTE delay data. As discussed in Sect. 5.2, the reservations are longer in this case due to higher uncertainties of task arriving time. As a result, the total reservations may exceed the processor capability. See Fig. 9a, the planner has to postpone all reservations to allocate them, all clients hence have severely dragged intervals (blue lines). To serve more clients, we remove the uncertainty from task reservation (as in Fig. 5), and then the offloading intervals can be maintained (green lines in Fig. 9a). We show the CDFs of 90% percentiles E2E delays of 24 clients in Fig. 9b. Delays increase without including network uncertainties in reservations, but ATOMS still presents reasonable improvement: 90% percentile of delays is decreased by 24% in average and by 30% as the maximum among all clients. Figure 9b gives the performance of the reordering-based schedulers described in Sect. 6.1: Earliest-start-time-first scheduler and Longest-E2E-delay-first scheduler. The result shows that these schedulers perform similarly to Sched Only, and ATMOS achieves better performance.



(a) Offloading interval. (b) E2E delay.

Fig. 9. (a) Offloading interval running FaceDetect using LTE with 24 clients. The average CPU utilization is 83% for Sched Only, 59% for ATOMS, and 81% for ATOMS without network uncertainty. (b) CDFs (over all 24 clients) of 90% percentiles of E2E delay running FaceDetect using LTE networks. (Color figure online)

6.3 Shared Worker

Contention mitigation is more complex for shared workers. In the evaluations, we set up 4 ObjectDetect clients with periods 2 s and 3 s, and 16 FeatureMatch

clients with periods 2 s, 2.5 s, 3 s and 3.5 s. $d\_future = 6$ s is used in the planner. We use 4 shared workers (c4.2xlarge), and each hosts 4 FeatureMatch engines and 1 ObjectDetect engine.

Planning Schemes: We compare three schemes of planning: (i) a global reservation queue ($Q\_cpu = 32$) is used for 4 workers; (ii) 4 reservation queues ($Q\_cpu = 8$) are used and Best-Fit is used to select queue; (iii) 4 queues are used with Worst-Fit selection. Load-aware scheduling with Worst-Fit worker selection is used. The CDFs of interval (top) and E2E delay (bottom) of all clients are given in Fig. 10. It shows that Worst-Fit adjusts tasks more aggressively and causes the largest interval jitter. It allocates FeatureMatch tasks (low parallelism) more evenly to all reservation queues. Resource contention is more likely to take place when ObjectDetect (high parallelism) is planned, so more adjustments are made. The advantage of Worst-Fit is the improved delay performance. See the delay plots in Fig. 10, Worst-Fit evidently performs better for the 4 ObjectDetect clients: the worst E2E delay of the 4 clients is 786 ms for Worst-Fit, 1111 ms for Best-Fit and 1136 ms for Global. The delay performance of FeatureMatch is similar for the three schemes.

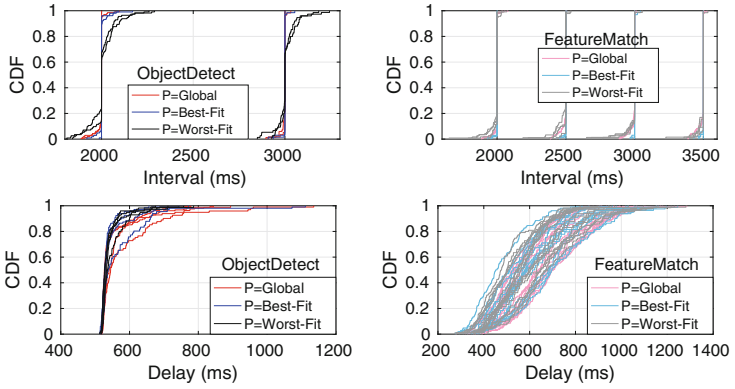


Fig. 10. Interval and E2E delay of 4 ObjectDetect and 16 FeatureMatch clients, using different planning schemes. The average CPU utilization is 36%.

Scheduling Schemes: Figure 11 shows the E2E delays using different scheduling schemes: (i) a simple scheduler that selects the first available engine; (ii) a load-aware scheduler with Best-Fit worker usage selection; (iii) a load-aware scheduler with Worst-Fit selection. The planner uses 4 reservation queues with Worst-Fit selection. For the simple scheduling, ObjectDetect tasks that can be parallelized on all 8 cores are more likely to be influenced by contention. FeatureMatch requires 2 cores at most and can get enough processors more easily. Best-Fit performs the best for ObjectDetect, whereas it degrades dramatically for FeatureMatch clients. The reason is that the scheduler tries to pack incoming

tasks as tightly as possible on workers. As a consequence, it leaves enough space to schedule highly parallel ObjectDetect tasks. However, due to the errors of computing time prediction and network estimation, there is a higher possibility of contention for the tightly placed FeatureMatch tasks. The Worst-Fit method has the best performance for FeatureMatch tasks and still maintains reasonably low delays for ObjectDetect. Therefore it is the most suitable approach in this case. Figure 12 compares the 90% E2E delay of all clients between Scheduling Only and ATOMS (Worst-Fit scheduling). In average, ATOMS reduces the 90% percentile E2E delay by 49% for the ObjectDetect clients, and by 20% for the FeatureMatch clients.

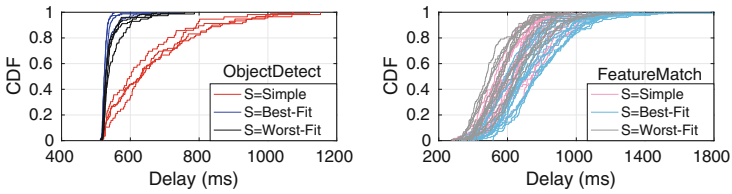


Fig. 11. E2E delay of ObjectDetect and FeatureMatch using different scheduling schemes. The average CPU utilization is 36% in all cases.

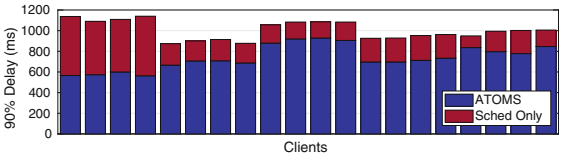


Fig. 12. 90% percentiles of E2E delay of ObjectDetect (bar 1 to 4) and FeatureMatch (bar 5 to 20) clients. The CPU utilization is 40% for Sched Only and 36% for ATOMS.

7 Conclusions

We present ATOMS, an offloading framework that ensures low E2E delays by reducing multi-tenant interference on servers. ATOMS predicts the time slots of future offloaded tasks, and coordinates them to mitigate processor contention on servers. It selects the best server machine to run each arriving task to minimize contention, based on real-time workloads on each machine. The realization of ATOMS is achieved by key system designs in computing time prediction, network latency estimation, distributed processor resource management and client-server clock synchronization. Our experiments and emulations prove the effectiveness of ATOMS in improving E2E delay for applications with various degrees of parallelism and computing time variability.

References

1. Ha, K., et al.: Towards wearable cognitive assistance. In: MobiSys (2014)
2. Jain, P., et al.: OverLay: practical mobile augmented reality. In: MobiSys (2015)
3. Cuervo, E., et al.: MAUI: making smartphones last longer with code offload. In: MobiSys (2010)
4. Satyanarayanan, M., et al.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
5. Han, S., et al.: MCDNN: an approximation-based execution framework for deep stream processing under resource constraints. In: MobiSys (2016)
6. Salmerón-Garcia, J., et al.: A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing. IEEE Trans. Autom. Sci. Eng. **12**(2), 444–454 (2015)
7. Meisner, D., et al.: PowerNap: eliminating server idle power. In: ASPLOS (2009)
8. Jyothi, S.A., et al.: Morpheus: towards automated SLOs for enterprise clusters. In: OSDI (2016)
9. Ravindranath, L., et al.: Timecard: controlling user-perceived delays in server-based mobile applications. In: SOSP (2013)
10. Chen, T.Y.H., et al.: Glimpse: continuous, real-time object recognition on mobile devices. In: SenSys (2015)
11. Tumanov, A., et al.: TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: EuroSys (2016)
12. Rasley, J., et al.: Efficient queue management for cluster scheduling. In: EuroSys (2016)
13. Rajkumar, R., et al.: Resource kernels: a resource-centric approach to real-time systems. In: SPIE/ACM Conference on Multimedia Computing and Networking (1998)
14. Brandenburg, B.B., Anderson, J.H.: On the implementation of global real-time schedulers. In: RTSS (2009)
15. Saifullah, A., et al.: Multi-core real-time scheduling for generalized parallel task models. In: RTSS (2011)
16. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR (2001)
17. Bradski, G.: The OpenCV library. Dr. Dobb's J. Softw. Tools **120**, 122–125 (2000)
18. Bay, H., et al.: Speeded-up robust features (SURF). Comput. Vis. Image Underst. **110**(3), 346–359 (2008)
19. Redmon, J., et al.: You only look once: unified, real-time object detection. In: CVPR (2016)
20. Ju, Y., et al.: SymPhoney: a coordinated sensing flow execution engine for concurrent mobile sensing applications. In: SenSys (2012)
21. Jacobson, V.: Congestion avoidance and control. In: SIGCOMM (1988)
22. Netravali, R., et al.: Mahimahi: a lightweight toolkit for reproducible web measurement. In: SIGCOMM (2014)
23. Mills, D., et al.: Network time protocol version 4: protocol and algorithms specification. RFC 5905 (Proposed Standard), June 2010
24. Saini, M., et al.: The Jiku mobile video dataset. In: MMSys (2013)
25. Raposo, C., Antunes, M., Barreto, J.P.: Piecewise-planar StereoScan: structure and motion from plane primitives. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8690, pp. 48–63. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10605-2_4
26. Fleuret, F., et al.: Multicamera people tracking with a probabilistic occupancy map. IEEE Trans. Pattern Anal. Mach. Intell. **30**(2), 267–282 (2008)

Dynamic Selecting Approach for Multi-cloud Providers

Juliana Carvalho[1,2,3(✉)], Dario Vieira[2], and Fernando Trinta[1]

[1] Federal University of Ceará (UFC), Fortaleza, Brazil
julianaoc@gmail.com, fernando.trinta@dc.ufc.br
[2] EFREI, Paris, France
dario.vieira@efrei.fr
[3] Federal University of Piauí (UFPI), Picos, Brazil

Abstract. Since multi-cloud can be used both to mitigate vendor lock-in and take advantages of cloud computing, an application can be deployed to multiple cloud providers that best meet the user and application needs. For this, it is necessary to select the cloud providers that will host an application. The selection process is a complex task due to the fact that each application part has its constraints, but mainly due to the existence of many cloud providers', each of them with its specific characteristics. In this article, we propose a cloud providers selection process to host applications based on microservices, in which each microservice must be hosted by the provider that best meet the user and microservice requirements. We use applications based on microservices because they can be independently deployed and scalable. In addition, we use Simple Additive Weighting method for ranking the candidate cloud providers then we select candidate providers amongst them to host each microservice by mapping the selection process to multi-choice knapsack problem. Besides that, the microservices are analyzed individually, except for the cost constraint. The selection process examines the cost constraint by observing all application microservices. Our approach still differs from others described in the literature because it selects multiple cloud providers to host the application microservices, which one for each application microservice. In this article, we also evaluated the proposed method through experiments and the result shows the viability of our approach. Finally, we point out future directions for the cloud providers selection process.

1 Introduction

Cloud computing has become a popular IT service delivery model in recent years. It brings several benefits, but also some challenges, such as vendor lock-in. Vendor lock-in arises as a consequence because the user application is dependent of a single cloud provider technology, which makes it impossible to automatically migrate the applications between cloud providers. Multiple clouds enable applications to take advantage of the best features of different components offered by several cloud providers, and mitigate vendor lock-in [1–3]. Therefore, software

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 37–51, 2018.
https://doi.org/10.1007/978-3-319-94295-7_3

architects can use multiple clouds to deploy the distributed application and take every possible advantages of cloud computing.

In spite of the possibility to take full advantage of cloud computing, it is necessary to select cloud providers that can best serve the application and the user. Choosing clouds is a complex task because there are several providers, each of them offering multiple services with the same functionalities, but different capabilities. In order to allow the software architects to choose the application requirements and architecture without needing to worry about the application deployment in multiple cloud, in [4], we propose PacificClouds, which is a new architecture for addressing multi-cloud interoperability. PacificClouds architecture is designed to deploy and manage applications based on microservices. We understand that microservices are a set of autonomous, independent, self-contained services, in which each service has a single goal, is loosely coupled, and interact to build a distributed application as in [4].

PacificClouds needs to select the cloud providers to host the application's microservices before the application deployment. In Sect. 2, we described the cloud selection process, which is based on the user and microservice requirements, considering the capabilities of the available cloud providers. The selection process verifies among available cloud providers which best meet the constraints of the user and the microservice, observing that each microservice service has different functionalities and constraints. PacificClouds must deploy an application microservice in the cloud provider that best meets its needs. Therefore, we realize that an application has several microservices and a microservice needs to use several cloud services. Also, there are many providers available and each of them offers several services with the same functionalities, but with different capacities.

In Sect. 4, we describe and compare some previous research works, which lead the selection process, and among them, some deal with cloud services selection in the same provider, such as [5–7], while some, [8], address services selection in cloud federation, and others such as [9] select services for microservices distributed in multiple providers. To the best of our knowledge only our work discusses cloud providers selection for deploying application microservices.

In order to lead with multiple cloud selection process, we propose an approach, which intends to be used by PacificClouds to deploy application microservices. The significant contributions of this work are as follows:

1. The clouds selection process to deploy application microservices is innovative, because our proposal selects multiple cloud providers to host the application microservices, which one for each microservice of an application. It is based on user and microservice's requirements.
2. We propose a formal description for the clouds selection process in Subsect. 2.1, which describes how to select each cloud service for each microservice. In addition, we present an algorithm for our approach in Subsect. 2.2.
3. The clouds selection uses Simple Additive Weighting (SAW) for ranking the cloud providers candidates [10], in which the user requirements are considered, and each of them has its priority (Subsect. 2.2).

4. We map our cloud providers selection process to multi-choice knapsack problem and develop using dynamic programming. We describe the mapping in Subsect. 2.2.
5. We performed an evaluation of the selection process and the outcome shows the feasibility of our approach. We describe the evaluation in Sect. 3.

2 The Proposed Approach

In this section, we propose a cloud providers selection model under multiple requirements to host an application based on microservices. We select the cloud providers checking a set candidate services in each cloud provider to optimize the application requirements. We consider that every candidate service meets all microservices requirements. Our model depicts the providers selection process in three steps: first, the software architect should send the application microservices and their requirements to PacificClouds; next, a discovery service should send the capabilities of cloud providers to PacificClouds; and, finally, PacificClouds should select one cloud provider for each application microservice throughout two phases: phase one is for selecting all cloud providers that meet the user requirements of all microservices of an application and phase two is for selecting a cloud provider to host each microservice of an application from those selected clouds in the first phase.

According to mentioned above, in the following subsection, we formalize the service model, which we use in cloud providers selection. Next, we describe the cloud providers selection in detail. In addition, we present an algorithm for proposed multiple cloud selection process.

2.1 Service Model Formalization

We can notice the existence of several cloud services with the same functionalities but different capabilities. We can also observe several cloud providers that offer multiple similarly featured services, but have different capabilities. Thus, software architects can request various service's capabilities. Even though we chose to assess application response time (execution time + delay), cloud availability and application execution cost, other requirements may be included in our model.

According to the providers selection process previously described, we propose:

- Definition 1: **cloud service model** - as S(S.rt,S.a,S.c), in which S.rt, S.a and S.c stand for response time, availability and cost respectively as in [6].
- Definition 2: **cloud services class** - as SC = $S_1, S_2, \ldots, S_o$, in which S_1, S_2,..., S_o are services of the same provider with same functionalities but different capabilities.
- Definition 3: **services provider model** - as SP = $SC_1, SC_2, \ldots, SC_p$, in which $SC_1, SC_2, \ldots, SC_p$ are services classes.
- Definition 4: **cloud provider set** - as CP = $SP_1, SP_2, \ldots, SP_q$, in which $SP_1, SP_2, \ldots, SP_q$ are services providers.

- Definition 5: **microservice model** - as MS = $S_1, S_2, \ldots, S_r$, in which S_1, $S_2, \ldots, S_r$ are cloud services indispensable to execute a microservice.
- Definition 6: **application model** - as AP = $MS_1, MS_2, \ldots, MS_t$, which $MS_1, MS_2, \ldots, MS_t$ are microservices.

All definitions described for the cloud selection process must follow the model of cloud service (Definition 1). Example: Definition X is modelled as X (X.rt, X.a, X.c). The software architect who wants to deploy an application in multi-cloud via PacificClouds must specify the user requirements thresholds for performance, availability and cost. Also, he must set the priority for each of the requirements. Multi-cloud selection process aims to select the providers that satisfy the user requirements and optimize the user's specified objectives, as follows:

1. **Availability Requirement:** the cloud availability for an application must meet at least the user-defined threshold, so that each application microservice meets the same threshold. We define MinAvbty as the user-defined minimum availability threshold. In (1), we define AP.a as the availability of the application, which is assigned the lowest availability value among their microservices and it must be greater than or equal to MinAvbty. In addition, in (2), $MS_i.a$ represents the availability of microservice i, which is assigned the lowest availability value among their services and it must be greater than or equal to MinAvbty. The cloud availability for a service is represented by $S_{ij}.a$ in (2), which is the service j of the microservice i. An application has t as the microservices maximum, and a microservice has r as the services maximum.

$$AP.a = \min_{1 \leqslant i \leqslant t} (MS_i.a) \geqslant MinAvbty, \forall MS_i \mid MS_i \in AP \tag{1}$$

$$MS_i.a = \min_{1 \leqslant j \leqslant r} S_{ij}.a \geqslant MinAvbty, \forall S_{ij} \mid S_{ij} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \tag{2}$$

2. **Response Time Requirement:** the response time for an application must meet the user-defined threshold, so that each application microservice meets the same threshold. We define MaxRT as the user-defined maximum response time threshold, and MaxRT is the maximum execution time threshold (MaxExecTime) plus the maximum delay threshold (MaxDelay) defined by the user as in (3). In (4), we define AP.rt as the response time of the application, which is assigned the highest response time value among their microservices and it must be less than or equal to MaxRT. In addition, in (5), $MS_i.rt$ represents the response time of microservice i, which is assigned the highest response time value among their services and it must be less than or equal to MaxRT. The response time for a service is represented by $S_{ij}.rt$ in (5), which is the service j of the microservice i. An application has t as the microservices maximum, and a microservice has r as the services maximum.

$$MaxRT = MaxExecTime + MaxDelay \tag{3}$$

$$AP.rt = \max_{1 \leqslant i \leqslant t} (MS_i.rt) \leqslant MaxRT, \forall MS_i \mid MS_i \in AP \tag{4}$$

$$MS_i.rt = \max_{1 \leqslant j \leqslant r} (S_{ij}.rt) \leqslant MaxRT, \forall S_{ij} \mid S_{ij} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \tag{5}$$

3. **Cost Requirement:** the application execution the cost should not be higher than Budget, which is cost threshold defined by the user. In (6), we define AP.c as the application execution cost, which is assigned the sum cost of all their microservices and it must be less than or equal to Budget. An application has t as the microservices maximum.

$$AP.c = \sum_{i=1}^{t} MS_i.c \leqslant Budget, \forall MS_i \mid MS_i \in AP \tag{6}$$

2.2 Cloud Providers Selection

In this subsection, we detail the proposed cloud providers selection process based on service model described above. The proposed process has two levels: the first level determines the candidate providers of all application microservices, and the second level selects the providers to deploy all application microservices.

First Level - Candidate Providers Determination. The candidate providers determination to deploy each microservice of an application is made by observing user requirements of each service. For this, we first rank the services in each provider that meets the user requirements based on Simple Additive Weighting (SAW). Next, we discover all candidate service combinations from all providers that meet the user requirements each microservice of an application. In addition, we calculate the score of each combination from all providers that meet user requirements of all microservices. This process consists of the three stages which will be described next.

First Stage. We select the services of each provider that meet all user requirements, which results in a candidate services set in each provider for each microservice of an application. We use the Simple Additive Weighting (SAW) technique to rank the providers that meet each microservice requirements, as in [7]. It has two phases:

– **Scaling Phase** - in this phase, the user requirements are numbered from 1 to 3, with 1 = availability, 2 = response time, 3 = cost. A matrix $R = (R_{ij}; 1 \leqslant i \leqslant n; 1 \leqslant j \leqslant 3)$ is built by merging the requirement vectors of all candidate services. These candidate services refer to the same service of the microservice. We must make the whole process for all services of the microservice. Each row R_i corresponds to a cloud service S_{ij} and each column R_j corresponds to a requirement. For this, there are two types of criteria:

- **Negative:** the higher the value, the lower the quality.

$$V_{ij} = \begin{cases} \frac{R_j^{Max} - R_{ij}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{7}$$

- **Positive:** the higher the value, the higher the quality.

$$V_{ij} = \begin{cases} \frac{R_{ij} - R_j^{Min}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{8}$$

where
$R_j^{Max} = Max(R_{ij}), R_j^{Min} = Min(R_{ij}), 1 \leqslant i \leqslant n$

– **Weighting Phase** - The overall requirements score is computed for each candidate cloud service, using the Eq. 9:

$$Score(S_i) = \sum_{j=1}^{3} (V_{ij} * W_j) \mid W_j \in [0,1], \sum_{j=1}^{3} W_j = 1 \tag{9}$$

Second Stage. We must discover all services combinations from a provider to compose a microservice. For this, we must make the combination among the candidate services of services requested by a microservice that are offered by the same provider. Equation (10) defines SSP_{ik} as the set of candidate combinations of the provider k for each microservice i. Also, in (10), $comb_{ikj}$ indicates the combination j of provider k for microservice i. A combination $comb_{ikj}$ is $(S_{i_1}, ..., S_{i_r})$, in which S_{i_n} indicates the candidate service i_n for service n in microservice i. A microservice has r as the service maximum, a candidate provider has m as the combination maximum, and a microservice has q the candidate providers maximum.

$$SSP_{ik} = \{comb_{ik1}, ..., comb_{ikm}\} \tag{10}$$

Third Stage. We must calculate the score and cost for each combination in SSP_{ik} as shown by (11) and (12). In (11), we define $comb_{ikj}.score$ as the average scores of each combination services. In (12), we define $comb_{ikj}.c$ as the sum cost of all combination services.

$$comb_{ikj}.score = \frac{Score(S_{i_1}) + Score(S_{i_2}) + ... + Score(S_{i_r})}{r} \tag{11}$$

$$comb_{ikj}.c = S_{i_1}.c + S_{i_2}.c + ... + S_{i_r}.c \tag{12}$$

At the end of the First Level, we have a set of all candidate combinations in all available providers for all microservices of an application (SAMS) and each of them has its score. Equation (13) defines SMS_i as the set of candidate providers for microservice i, in which SSP_{ik} indicates the set of candidate combinations

of provider k for microservice i. In (14), we define SAMS as the set of candidate providers for each microservice of an application, and an application has t as the microservices maximum.

$$SMS_i = \{SSP_{ik} \mid k \in [1, q]\} \tag{13}$$

$$SAMS = \{SMS_1, ..., SMS_t\} \tag{14}$$

Second Level - Cloud Providers Selection to Deploy Microservices. We must select the cloud providers from SAMS (First Level) to host each microservice of an application. For this, we must make the combination among the candidate combinations SAMS, in which each combination has one candidate of each microservice and the set of these combinations is called SAPP (15). Also, in (15), $comb_{ik_ij k_i}$ indicates combination j of candidate provider k for microservice i. Afterwards, (16), we define $SAPP_l.c$ as the execution cost of each combination in SAPP and verify if it is less than or equal to the application execution cost, and exclude it otherwise. Next, we calculate the score of every item in SAPP. A candidate combination score ($SAPP_l.score$) is a value among $]0, 1]$, and it is the score sum of each item combination, which is shown in (17).

$$SAPP = \{(comb_{1k_1j_{k_1}}, ..., comb_{tk_tj_{k_t}}) \mid comb_{ik_ij k_i} \in SSP_{ik}, SSP_{ik} \in SMS_i, SMS_i \in SAMS, 1 \leqslant i \leqslant t, 1 \leqslant k_i \leqslant q_i, 1 \leqslant j_{k_i} \leqslant w_{k_i}\} \tag{15}$$

$$SAPP_l.c = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.c \tag{16}$$

$$SAPP_l.score = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.score \tag{17}$$

The selection process described above is a combinatorial optimization problem and one of the most studied problems in combinatorial optimization is the knapsack problem. Hence, we map the selection process up to multi-choice knapsack problem. The application budget represents the maximum knapsack weight and each item in SAPP has its cost, i.e., its weight. Each SAPP item represents the set of services that are necessary for each application microservice. Thus, the microservices represent the classes, because each item of SAPP must contain set of services for each microservice. Our selection problem must maximize the user requirements through the score of each item in SAPP. The item score represents the profit, (18) illustrates the objective function to maximize the score, which is subject to (19), (20) and (21) for obeying total combination cost. In these equations, $comb_{ikj}$ is candidate combination j of provider k for microservice i,

SMS_i is a set of candidate providers for microservice i, as defined in formula 13, and SSP_{ik} is a set of combinations of provider k for each microservice i, as defined in formula 10.

$$max(\sum_{i=1}^{t} \sum_{k \in SMS_i} \sum_{j \in SSP_{ik}} comb_{ikj}.score * Comb_{ikj}) \tag{18}$$

$$\sum_{i=1}^{t} \sum_{k \in SMS_i} \sum_{j \in SSP_{ik}} comb_{ikj}.c * comb_{ikj} \leqslant AP.c \tag{19}$$

$$\sum_{k \in SMS_i} SSP_{ik} = 1, \forall 1 \leqslant i \leqslant t \tag{20}$$

$$\sum_{j \in SSP_{ik}} comb_{ikj} = 1, \forall 1 \leqslant i \leqslant t, k \in [1, q] \tag{21}$$

2.3 Algorithm for Cloud Providers Selection

Next, we present Algorithm 1 for the selection process of our approach. Whereby is possible to observe all selection process level described in this section. Algorithm 1 has as input the set of application requirements and the set of the capabilities of the providers and as output set of the providers to host each application microservice. We describe an overall Algorithm 1, as follow.

1. First, we discover the provider services that meet all the requirements of a microservice (line 4). Then, we rank them by calculating SAW Scaling Phase (line 6) and SAW Score Phase for each service (line 7). Next, we combine the services (line 8) so that each combination has all the services required by the microservice. The process is repeated on all available providers;
2. Next, we calculate the score (line 13) and the cost (line 14) of each combination. The entire process between lines 2 and 13 is repeated for all the application microservices, resulting in a set of candidate combinations for each microservice (SAMS), each of them has its score and cost;
3. Finally, from line 21 to 29, we selected a combination to host a microservice among the candidate combinations of the microservice in SAMS. For this, we create a matrix (Sol) using lines for every possible combination cost value (from 1 to AP.c) and columns for the cost of each existing combinations. Matrix Sol is filled with the score values according to the microservice to which the combination belongs, in order to obtain the combinations with the highest score that, when combined, do not exceed the budget value of the application (AP.c), according to the mapping made for the multi-choice backpack problem as described in (15), (16), (17), (18), (19), (20), and (21).

Algorithm 1. The Proposed Approach

input : Set of the application requirements ap
Set of the capabilities of the providers cp
output : Set of Providers prvdsMs to host each microservices of an application

```
1  foreach ms of the ap do
2      ssp ← initialize with empty set;
3      foreach sp of the cp do
4          candServ ← discoveryCandServ (ms,sp);
5          if (notempty(candServ)) then
6              matReq ← sawScalingPh (candServ) ;                    // formulas 7 and 8
7              candServSC ← sawScorePh (candServ,matReq,ap.weight) ;  // formula 9
8              ssp ← ssp + (sp.number,combMsSp (candServSC)) ;        // formula 10
9          end
10     end
11     sms ← initialize with empty set;
12     foreach comb of the ssp do
13         combSC ← calculateCombSC (comb) ;                          // formula 11
14         combCost ← calculateCombCost (comb) ;                      // formula 12
15         sms ← sms + (comb,combSC,combCost);
16     end
17     sams ← sams + (ms.number,sms);
18 end
   // The algorithm between lines 19 and 29 indicates the cloud providers selection using dynamic
   programming for mapping of the multi-choice knapsack problem.;
19 sol ← initialize the matrix with values 0;
20 last ← 1;
21 foreach sms of the sams do
22     for j ← last to sizeof(sms) − 1 do
23         for cost ← 1 to ap.c do
24             if (combCost[j] > cost) then sol(cost, j) ← sol(cost, j − 1);
25             else if (ms[j].number == 1) then sol(cost, j) ← combSC[j];
26             else if (ms[j].number ≠ ms[j − 1].number) then
               sol(cost, j) ← max(maxLine(sol(cost − combCost[j]), ms[j −
               1].number) + combSC[j], maxLine(sol(cost), ms[j − 1].number));
27             else sol(cost, j, ms[j].number) ←
               max(maxLine(sol(cost − combCost[j]), ms[j].number − 1) +
               combSC[j], maxLine(sol(cost), ms[j].number − 1));
28         end
29         last ← sizeof(sms);
30     end
31 end
```

3 Evaluation

We have developed a tool and set up scenarios to evaluate our approach. In this section, we describe how the tool was implemented, the scenarios configuration, the experiments, and the outcomes.

3.1 Tool Description

The tool was implemented using python 3.7 and we used the JSON format as input and output. The tool has two JSON files as input: one contains the service providers capabilities and the other has the application requirements. Each provider capabilities are organized by service classes, in which each class has its services. We consider three classes of services: compute, storage and database. Each service must contain its functional and non-functional capabilities. The nonfunctional capabilities we consider in our approach are availability, execution time, delay and cost, as described in Sect. 2.

Application information involves name, minimum availability, maximum response time, maximum budget, weight and priority of each user requirement and the microservices information.

The tool returns a JSON file, which contains the providers that will deploy the microservices, as well as the services that will be used and the providers cost.

As previously mentioned, we map our multi-cloud selection process to the multi-choice knapsack problem, which is a highly researched combinatorial optimization problem. The multi-choice knapsack problem can be developed through some techniques, such as dynamic programming and greedy. We use dynamic programming in our tool.

3.2 Setting Scenarios

For tool evaluation purposes, we randomly configure six sets of providers as in [5, 6, 11], which differ from one another by the number of providers and each service capabilities. For this, we first randomly configured the set with 5 providers. Next, we configure the other set of providers from (22). In (22), $SPrvds_x$ is a set of providers, in which x is the number of providers in SPrvds, and $SPrvds_{x-5}$ is the set of providers configured with the highest number of providers up to that moment according to the proposed scenario, and x must be higher than or equal to 5. If x is equal 5 indicates that is the first set of the scenario then x == 0 is the empty set. For example, if x == 15 then the set of 15 providers is based on the set of 10 providers previously configured. Each provider has three services classes: compute, storage and database, and each class has six services. The amount of providers in each set is a multiple value of 5, in which the first set has 5 providers and the last 30.

$$SPrvds_x = SPrvds_{(x-5)} + SPrvds_5 \tag{22}$$

We have also configured requirements for microservice-based applications. We configured 4 applications: the first one (APP1) has 3 microservices, the second (APP2) 4, the third (APP3) 5, and the last one (APP4) has 6 microservices. Each microservice contains 3 services of each class: compute, storage and database. Availability, response time, and cost requirements vary depending on the type of assessment to be performed.

3.3 Experiments and Outcomes Description

We evaluate our approach's feasibility by validating our tool's behavior. For this, we performed five experiments, in which we used the sets of providers and the applications requirements described in Subsect. 3.2. Each experiment was performed 30 times to reach the normal distribution [12].

Figure 1 illustrates the outcomes of the five experiments. In each graph, we present the provider's selection average execution time in milliseconds (ms) on the y-axis, the rows represent each application described in the subsection, and the x-axis shows each of the experiments. The first four experiments were performed using a set of 5 providers. The experiments and their respective outcomes are described afterwards.

1. **Availability requirement:** we configured each application with 5 different values for availability. Starting at 90%, indicating that all provider services must have this minimum value to meet application requirements, and ending at 98%, indicating that only cloud services with values between 98 and 100 meet this application requirement, which is in agreement with (1) and (2). The response time and cost requirements were set with values that can be met by most providers. Figure 1(a) shows a graph with the experiment outcomes. The graph illustrates the availability requirement values on the x-axis, the average execution time for providers selection on the y axis, and the rows represent each application. We can observe that the average time decreases with the increase of the availability constraint value in every application. This outcome is the expected one, because the increase of the availability requirement decreases both the number of services that meet the microservice requirements and the number of services that must be selected in our tool.
2. **Response Time requirement:** we configured each application with 5 different response time values. This value indicates that only providers services with values less than or equal to the requested one meet the application response time requirement, which is in accordance to (3), (4) and (5). Thus, the lower the value the less the services will meet the response time requirement. The availability and budget requirements were set with values that can be met by most providers. Figure 1(b) outcomes represent the expected one, because the graph shows the average time increases with the increase of the response time requirement level in every application.
3. **Cost requirement:** in this experiment, we set the availability and response time requirements with values that can be met by most providers, and the cost requirements were set from the minimum to the maximum cloud services

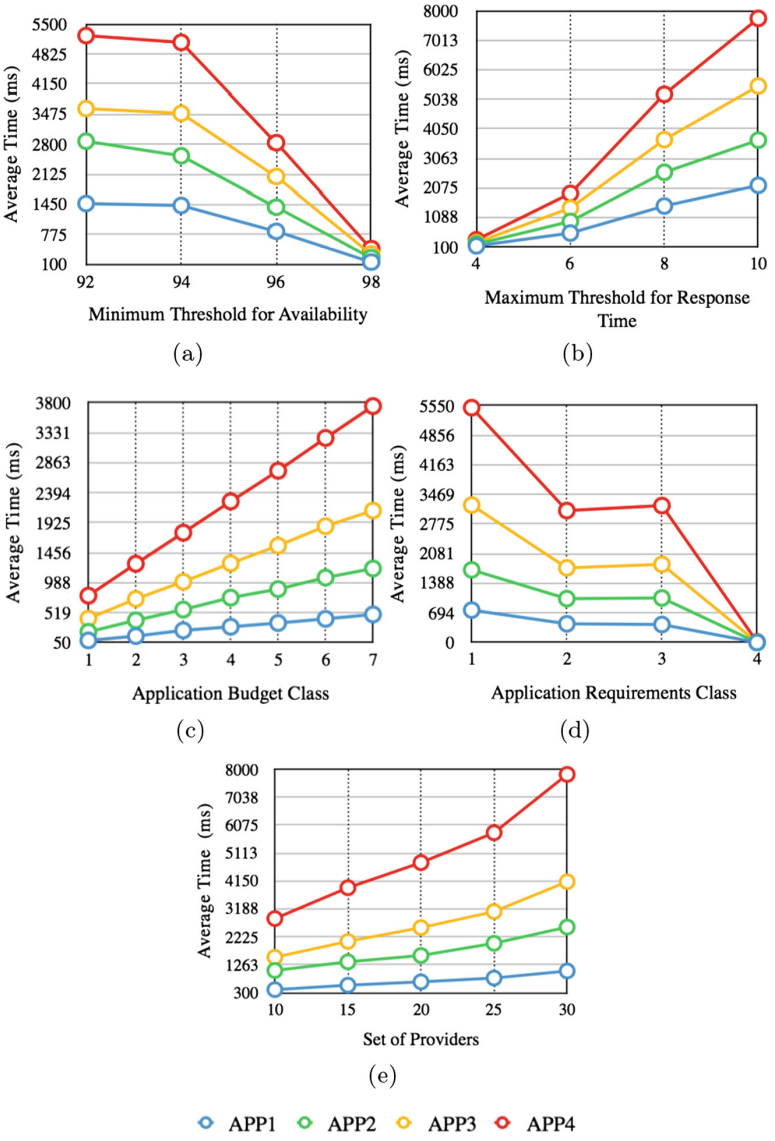


Fig. 1. Cloud providers selection execution time (a) application availability; (b) application response time; (c) application budget; (d) application user requirement class; and (e) set of providers.

values. Thus, the values in the x-axis represent budget classes, and each class has different budget values for each application, but all applications have the same value per service in each class. The budget value per class is different in each application, but the applications have the same value per service in each

class. Also, class 1 has the lowest value per service, class 2 has a value higher than class 1 but lower than the class 3, and last class has the highest value per service. The lowest and highest values per service are based on the lowest and highest value of the set of providers described in this scenario. Class 1 has the minimum value and a class 7 has the maximum value. Therefore, no set of providers meet the budget with a value lower than class 1, and all providers met budget with values higher than class 7. The budget values indicate that only cloud services with values less than or equal to the attributed ones meet the cost requirement as described in (6). Figure 1(c) outcomes show that the average execution time increases alongside with the cost level, but after a certain amount the average execution time is stabilized. This happens because the higher the budget, the greater the search for a better solution.

4. **User requirements:** in these experiments we vary all three constraints: availability, response time and cost, in a manner that when the availability requirement increases, the cost and response time requirements decrease. Therefore, when the requirements class is increased on the x-axis of Fig. 1(d), we say that the constraints increases. Thus, we can observe that the average execution time in the graph decreases with the increase of the requirements. This outcome shows that the increase of the requirements reduces the number of providers that meet all requirements and that the average execution time converges to the same value.
5. **Set of Providers:** for this experiment we set up 5 set of providers, each with a multiple of 5 as the number of providers, in such manner that the first set has 10 providers and the last one has 30 providers, which have been configured according to (22). The requirements of availability, response time and cost were not modified during the experiment, maintaining the same value in all sets. Those requirements were configured using the intermediate values, i.e., they have neither the lowest nor the highest constraint. We can see in Fig. 1(e) that the increase in the number of providers influences the average selection time, since it increases the number of services that meet the applications requirements. Also, we can observe that the amount of microservices also influences the cloud providers selection average time.

As mentioned in Sect. 1, in the literature there are dealing with the cloud provider selection, but the focus of these works is different from ours. Some of them select one provider per application or service flow, and others are based on the cloud federation environment, which there is a collaboration agreement between providers. Besides, there is still one that deals with cloud selection to host microservice, but each service to microservice is hosted on a different cloud provider. The comparison of our proposal with these works is complex because each proposal has different purposes.

4 Related Work

Despite the growth of cloud computing, multiple clouds are still at their beginning [13], and very few research works related to cloud providers selections exist. In this Section, we compare our work with some of these previous works.

Wang et al. [5] proposed a service selection based on correlated QoS requirements, in which each service belongs to the same provider. Similarly to our work, [5] organizes the services within the provider by classes and considers that each required service belongs to a class and that several services candidates exist and each of them possesses the same functionality but different capabilities. Liu et al. [6] organizes services within the providers as our work does, but it proposed a service flow scheduling model with multiple service requirements in a cloud.

Zeng et al. [7] proposed a middleware which addresses the issue of selecting web services for their composition in a manner that maximizes user requirements. [7] leads the problems related to service selection the same way our work does.

Pietrabissa et al. [8] proposed a cloud resource allocation algorithm that manages the resource requests with the aim of maximizing the Cloud Manager Broker (CMB) revenue over time, focusing on the provider perspective and resource allocation in cloud federation, while our work focus on user perspective and multi-cloud environment.

Garg et al. [11] proposed a framework and mechanism that allow the evaluation of cloud offerings and rank them based on their ability to meet the user's requirements. Hence, the framework may permit a competition among cloud providers to satisfy their SLA and improve their QoS. This work differs from ours because we select the providers that best meet the user's requirements for each microservice of an application.

Sousa et al. [9] proposed an automated approach for the selection and configuration of cloud providers in multi-cloud microservices-based applications. [9] is similar to our work, but while [9] selects the services of a microservice in several providers, our work selects a provider to a microservice. Our work is based on Multi-Criteria Decision Making Methods (MCDM), while [9] is based on ontology reasoning and software product line techniques.

5 Conclusion

In this paper, we propose and formally describe a multiple cloud provider selection approach for PacificClouds, which must select the providers that best meet user requirements for each microservice of an application. We also use SAW to rank the candidate services for each microservice in each of the available providers according to its requirements. In addition, we mapped the provider selection process to the multi-choice knapsack problem. As a result, we have developed a tool that represents the proposed approach and we use dynamic programming to implement the multi-choice knapsack problem, then we evaluate the proposed approach using the developed tool. The outcomes show that the approach finds the best solution and that the approach proved to be viable even in the face of extreme restrictions. For future works, we intend to develop the provider selection process by using the knapsack problem via greedy algorithm, then we intend to compare it with this approach.

References

1. Mezgár, I., Rauschecker, U.: The challenge of networked enterprises for cloud computing interoperability. Comput. Ind. **65**(4), 657–674 (2014)
2. Silva, G.C., Rose, L.M., Calinescu, R.: A systematic review of cloud lock-in solutions. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, pp. 363–368 (2013)
3. Petcu, D.: Multi-cloud: expectations and current approaches. In: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, MultiCloud 2013, pp. 1–6 (2013)
4. Carvalho, J.O.D., Trinta, F., Vieira, D.: PacificClouds : a flexible MicroServices based architecture for interoperability in multi-cloud environments. In: CLOSER 2018, pp. 448–455 (2018)
5. Wang, Y., He, Q., Ye, D., Yang, Y.: Service selection based on correlated QoS requirements. In: 2017 IEEE International Conference on Services Computing (SCC), pp. 241–248 (2017)
6. Liu, H., Xu, D., Miao, H.K.: Ant colony optimization based service flow scheduling with various QoS requirements in cloud computing. In: Proceedings of the 1st ACIS International Symposium on Software and Network Engineering, SSNE 2011, pp. 53–58 (2011)
7. 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)
8. Pietrabissa, A., Priscoli, F.D., Di Giorgio, A., Giuseppi, A., Panfili, M., Suraci, V.: An approximate dynamic programming approach to resource management in multi-cloud scenarios. Int. J. Control **90**(3), 508–519 (2017)
9. Sousa, G., Rudametkin, W., Duchien, L.: Automated setup of multi-cloud environments for microservices-based applications. In: 9th IEEE International Conference on Cloud Computing (2016)
10. Al-Faifi, A.M., Song, B., Alamri, A., Alelaiwi, A., Xiang, Y.: A survey on multi-criteria decision making methods for evaluating cloud computing services. J. Internet Technol. **18**(3), 473–494 (2017)
11. Garg, S.K., Versteeg, S., Buyya, R.: A framework for ranking of cloud computing services. Future Gen. Comput. Syst. **29**(4), 1012–1023 (2013)
12. Hogendijk, J., Whiteside, A.E.S.D.: Sources and Studies in the History of Mathematics and Physical Sciences. Springer US (2011)
13. Petcu, D.: Consuming resources and services from multiple clouds. J. Grid Comput. **12**(2), 321–345 (2014)

Research Track: Cloud Data Storage

Teleporting Failed Writes with Cache Augmented Data Stores

Shahram Ghandeharizadeh(✉), Haoyu Huang, and Hieu Nguyen

USC Database Laboratory, Los Angeles, USA
{shahram,haoyuhua,hieun}@usc.edu

Abstract. Cache Augmented Data Stores enhance the performance of workloads that exhibit a high read to write ratio by extending a persistent data store (PStore) with a cache. When the PStore is unavailable, today's systems result in *failed* writes. With the cache available, we propose TARDIS, a family of techniques that teleport failed writes by buffering them in the cache and persisting them once the PStore becomes available. TARDIS preserves consistency of the application reads and writes by processing them in the context of buffered writes. TARDIS family of techniques is differentiated in how they apply buffered writes to PStore once it recovers. Each technique requires a different amount of mapping information for the writes performed while PStore was unavailable. The primary contribution of this study is an overview of TARDIS and its family of techniques.

1 Introduction

Person-to-person cloud service providers such as Facebook challenge today's software and hardware infrastructure [9,24]. Traditional web architectures struggle to process their large volume of requests issued by hundreds of millions of users. In addition to facilitating a near real-time communication, a social network infrastructure must provide an always-on experience in the presence of different forms of failure [9]. These requirements have motivated an architecture that augments a data store with a distributed in-memory cache manager such as memcached and Redis. We term this class of systems Cache Augmented Data Stores, **CADSs**.

Figure 1 shows a CADS consisting of Application Node (AppNode) servers that store and retrieve data from a persistent store (PStore) and use a cache for temporary staging of data [18,24,25]. The cache expedites processing of requests by either using faster storage medium, bringing data closer to the AppNode, or a combination of the two. An example PStore is a document store [10] that is either a solution such as MongoDB or a service such as Amazon DynamoDB [5,11] or MongoDB's Atlas [22]. An example cache is an in-memory key-value store such as memcached or Redis with Amazon ElastiCache [6] as an example service. Both the caching layer and PStore may employ data redundancy techniques to tolerate node failures.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 55–68, 2018.
https://doi.org/10.1007/978-3-319-94295-7_4

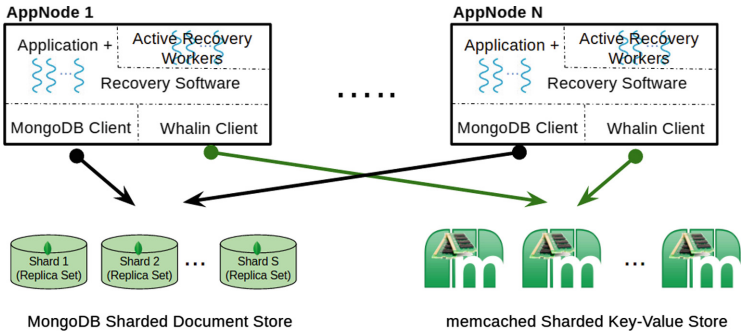


Fig. 1. CADS architecture.

The CADS architecture assumes a software developer provides application specific logic to identify cached keys and how their value is computed using PStore. Read actions look up key-value pairs. In case of a miss, they query PStore, compute the missing key-value pair, and insert it in the cache for future look up. Write actions maintain the key-value pairs consistent with data changes in PStore.

A read or a write request to PStore *fails* when it is not processed in a timely manner. This may result in denial of service for the end user. All requests issued to PStore fail when it is unavailable due to either hardware or software failures, natural disasters, power outages, human errors, and others. It is possible for a subset of PStore requests to fail when the PStore is either sharded or offered as a service. With a sharded PStore, the failed requests may reference shards that are either unavailable or slow due to load imbalance and background tasks such as backup. With PStore as a service, requests fail when the application exhausts its pre-allocated capacity. For example, with the Amazon DynamoDB and Google Cloud Datastore, the application must specify its read and write request rate (i.e., capacity) in advance with writes costing more [5,15]. If the application exhausts its pre-specified write capacity then its writes fail while its reads succeed.

An application may process its failed writes in a variety of ways. Simplest is to report the failures to the end users and system administrators. The obvious drawback of this approach is that the write fails and the end user is made aware of this to try again (or for the system administrator to perform some corrective action). Another variant is for the application to ignore the failed write. To illustrate, consider a failed write pertaining to Member A accepting Member B's friend invitation. The system may simply ignore this write operation and continue to show B's invitation to A to be accepted again. Assuming the failure of the PStore is short-lived then the user's re-try may succeed. This alternative loses those writes that are not recoverable. For example, when Member C invites Member B to be friends, dropping this write silently may not be displayed in an intuitive way for the end user to try again. A more complex approach is

for the application to buffer the failed writes. A system administrator would subsequently process and recover these writes.

This paper describes an automated solution that replaces the administrator with a smart algorithm that buffers failed writes and applies them to PStore at a later time once it is available. This approach constitutes the focus of TARDIS[1], a family of techniques for processing failed writes that are transparent to the user issuing actions.

Table 1. Number of failed writes with different BG workloads and PStore failure durations.

Failure duration	Read to write ratio	
	100:1	1000:1
1 min	5,957	561
5 min	34,019	3,070
10 min	71,359	6,383

TARDIS teleports failed writes by buffering them in the cache and performing them once the PStore is available. Buffered writes stored as key-value pairs in the cache are pinned to prevent the cache from evicting them. To tolerate cache server failures, TARDIS replicates buffered writes across multiple cache servers.

Table 1 shows the number of failed writes with different failure durations using a benchmark for interactive social networking actions named BG [8]. We consider two workloads with different read to write ratios. TARDIS teleports all failed writes and persists them once PStore becomes available.

TARDIS addresses the following challenges:

1. How to process PStore reads with pending buffered writes in the cache?
2. How to apply buffered writes to PStore while servicing end user requests in a timely manner?
3. How to process non-idempotent buffered writes with repeated failures during recovery phase?
4. What techniques to employ to enable TARDIS to scale? TARDIS must distribute load of failed writes evenly across the cache instances. Moreover, its imposed overhead must be minimal and independent of the number of cache servers.

TARDIS preserves consistency guarantees of its target application while teleporting failed writes. This is a significant improvement when compared with today's state of the art that loses failed writes always.

Advantages of TARDIS are two folds. First, it buffers failed writes in the cache when PStore is unavailable and applies them to PStore once it becomes

[1] Time and Relative Dimension in Space, TARDIS, is a fictional time machine and spacecraft that appears in the British science fiction television show Doctor Who.

available. Second, it enhances productivity of application developers and system administrators by providing a universal framework to process failed writes. This saves both time and money by minimizing complexity of the application software.

Assumptions of TARDIS include:

- AppNodes and the cache servers are in the same data center, communicating using a low latency network. This is a reasonable assumption because caches are deployed to enhance AppNode performance.
- The cache is available to an AppNode when PStore writes fail.
- A developer authors software to reconstruct a PStore document using one or more cached key-value pairs. This is the recovery software shown in Fig. 1 used by both the application software and Active Recovery (AR) workers.
- The PStore write operations are at the granularity of a single document and transition its state from one consistent state to another. In essence, the correctness of PStore writes are the responsibility of the application developer.
- There is no dependence between two or more buffered writes applied to *different*[2] documents. This is consistent with the design of a document store such as MongoDB to scale horizontally. Extensions to a relational data model that considers foreign key constraints is a future research direction.

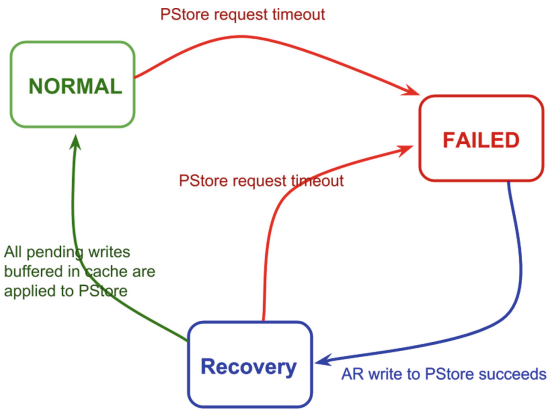


Fig. 2. AppNode state transition diagram.

The rest of this paper is organized as follows. Section 2 presents the design of TARDIS. Section 3 describes how undesirable race conditions may happen and our solution using contextual leases. We survey related work in Sect. 4. Brief future research directions are presented in Sect. 5.

[2] TARDIS preserves the order of two or more writes for the same document.

2 Overview

Each AppNode in TARDIS operates in 3 distinct modes: normal, failed, and recovery. Figure 2 shows the state transition diagram for these three modes. TARDIS operates in normal mode as long as PStore processes writes in a timely manner. A failed PStore write transitions AppNode to failed mode. In this mode, AppNode threads buffer their PStore writes in the cache. They maintain cached key-value pair impacted by the write as in normal mode of operation. Moreover, the AppNode starts one or more Active Recovery (AR) workers to detect when the PStore is available to process writes again. Once PStore processes a write by an AR worker, the AR worker transitions AppNode state to recovery mode.

In recovery mode, TARDIS applies buffered writes to PStore. TARDIS family of techniques is differentiated in how they perform this task. In its simplest form, termed TAR, only AR workers apply buffered writes. The next variant, termed DIS, extends TAR by requiring the write actions of the application to identify pending writes in the cache and apply them to PStore prior to performing their write. With DIS, a write action may merge the pending writes with its own into one PStore operation. With TARD, the developer provides a *mapping* between the read and buffered writes produced by write actions. This mapping enables the application to process reads that observe a cache miss by first applying the pending buffered writes to the PStore. Finally, TARDIS is a hybrid that includes features of both TARD and DIS.

The trade-off with the alternative techniques is as follows. With TAR, the application continues to produce buffered writes even though PStore is available. With both TAR and DIS, a cache miss continues to report a failure (even though PStore is available) until all pending writes are applied to PStore. DIS is different because it stops the application from producing buffered writes, enabling the recovery mode to end sooner. With TARD, a cache miss applies buffered writes to PStore while writes continue to be buffered. TARDIS is most efficient in processing application requests, ending the recovery process fastest. Due to novelty of TARDIS, applicability of TAR, TARD, DIS, and TARDIS in the context of different applications is a future research direction. It may be the case that DIS is applicable to most if not all applications.

When either AppNode or an AR worker incurs a failed write, it switches AppNode mode from recovery to failed. A PStore that processes writes intermittently may cause AppNode to toggle between failed and recovery modes repeatedly, see Fig. 2.

2.1 Failed Mode

In failed mode, a write for D_i generates a change δ_i for this PStore document and appends δ_i to the *buffered write* Δ_i of the document in the cache. In addition, this write appends P_i to the value of a key named Teleported Writes, TeleW. This key-value pair is also stored in the cache. It is used by AR workers to discover documents with pending buffered writes (Table 2).

Table 2. List of terms and their definition.

Term	Definition
PStore	A sharded data store that provides persistence
AR	Active Recovery worker migrates buffered writes to PStore eagerly
D_i	A PStore document identified by a primary key P_i
P_i	Primary key of document D_i. Also referred to as document id
$\{K_j\}$	A set of key-value pairs associated with a document D_i
Δ_i	A key whose value is a set of changes to document D_i
TeleW	A key whose value contains P_i of documents with teleported writes
ω	Number of TeleW keys

Δ_i may be redundant if the application is able to construct document D_i using its representation as a collection of cached key-value pairs. The value of keys $\{K_i\}$ in the cache may be sufficient for the AppNode to reconstruct the document D_i in recovery mode. However, generating a list of changes Δ_i for the document may expedite recovery time if it is faster to read and process than reading the cached value of $\{K_i\}$ to update PStore. An example is a member P_i with 1000 friends. If in failed mode, P_i makes an additional friend P_k, it makes sense for AppNode to both update the cached value and generate the change $\delta_i = \text{push}(P_k,\text{Friends})$. At recovery time, instead of reading an array of 1001 profile ids to apply the write to PStore document D_i, the system reads the change and applies it. Since the change is smaller and its read time is faster, this expedites recovery time.

In failed mode, AR workers try to apply buffered writes to PStore. An AR worker identifies these documents using the value of TeleW key. Each time AR's write to PStore fails, the AR may exponentially back-off before retrying the write with a different document. Once a fixed number of AR writes succeeds, an AR worker transitions the state of AppNode to recovery.

TARDIS prevents contention for TeleW by maintaining ω TeleW key-value pairs. It hash partitions documents across these using their primary key P_i. Moreover, it generates the key of each ω TeleW with the objective to distribute these keys across all cache servers. In failed mode, when the AppNode generates buffered writes for a document D_j, it appends the document to the value of the TeleW key computed using its P_j, see Fig. 3.

Algorithm 1 shows the pseudo-code for the AppNode in failed mode. This pseudo-code is invoked after AppNode executes application specific code to update the cache and generate changes δ_i (if any) for the target document D_i. It requires the AppNode to obtain a lease on its target document D_i. Next, it attempts to mark the document as having buffered writes by generating a key P_i+dirty with value dirty[3]. The memcached `Add` command inserts this key if

[3] Choice of P_i+dirty is arbitrary. The requirement is for the key to be unique. P_i+dirty is a marker and its value may be one byte.

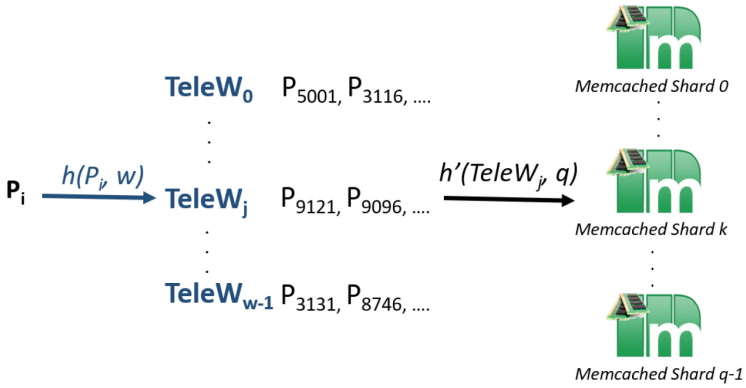


Fig. 3. Documents are partitioned across ω TeleW keys. TeleW keys are sharded across q memcached servers.

it does not exist. Otherwise, it returns NOT_STORED. Hence, the first time a document is inserted, it is appended to one of the ω TeleW keys. Repeated writes of D_i in failed mode do not insert a document D_i in TeleW again.

Algorithm 1. AppNode in Failed Mode (P_i)

1. **acquire lease** P_i
2. InsertAttempt = `Add`(P_i+dirty)
3. **release lease** P_i
4. **if** InsertAttempt is successful {
   TeleW$_i$ = hash(P_i, ω)
   acquire lease TeleW$_i$
   append(P_i, TeleW$_i$)
   release lease TeleW$_i$
   }

2.2 Recovery Mode

To perform an action that references a document D_i, AppNode checks to see if this document has buffered writes. It does so by obtaining a lease on P_i. Once the lease is granted, it looks up Δ_i. If it does not exist then it means an AR worker (or another AppNode thread) competed with it and propagated D_i's changes to PStore. In this case, it releases its lease and proceeds to service user's action. Otherwise, it applies the buffered writes to update D_i in PStore. If this is successful, AppNode deletes Δ_i to prevent an AR worker from applying buffered writes to D_i a second time.

TARDIS employs AR workers to eagerly apply buffered writes to PStore documents identified by TeleW. We minimize the recovery duration by maximizing

the probability of each AR worker to work on a disjoint set of documents. To achieve this, each AR worker randomly pick TeleW key as the starting point and visit other TeleW keys in a round-robin fashion. Also, it only recover α randomly selected documents in a TeleW. With AppNode, each time a user action references a document D_i with buffered writes, the AppNode applies its writes to the PStore prior to servicing the user action. In essence, during recovery, the AppNode stops producing buffered writes and propagates buffered writes to PStore.

Challenges of implementing TARDIS's recovery mode is two folds: (a) correctness: TARDIS must propagate buffered writes to PStore documents in a consistent manner, and (b) performance: TARDIS must process user's actions as fast as normal mode while completing recovery quickly.

We categorize buffered writes into idempotent and non-idempotent. Idempotent writes can be repeated multiple times and produce the same value. Non-idempotent writes lack this behavior. During recovery, multiple AR workers may perform idempotent writes multiple times without compromising correctness. However, this redundant work slows down the PStore for processing regular requests and makes the duration of recovery longer than necessary. Our implementation of recovery uses leases to apply buffered writes of a document to PStore exactly once even in the presence of AR worker or AppNode failures. The details are provided in the full technical report [26].

Algorithm 2 shows each iteration of AR worker in recovery mode. An AR worker picks a TeleW key randomly and looks up its value. This value is a list of documents written in failed mode. From this list, it selects α random documents $\{D\}$. For each document D_i, it looks up P_i+dirty to determine if its buffered writes still exist in the cache. If this key exists then it acquires a lease on D_i, and looks up P_i+dirty a second time. If this key still exists then it proceeds to apply the buffered writes to D_i in PStore. Should this PStore write succeed, the AR worker deletes P_i+dirty and buffered writes from the cache.

The AR worker maintains the primary key of those documents it successfully writes to PStore in the set `R`. It is possible for the AR worker's PStore write to D_i to fail. In this case, the document is not added `R`, leaving its changes in the cache to be applied once PStore is able to do so.

An iteration of the AR worker ends by removing the documents in `R` (if any) from its target TeleW key, Step 6 of Algorithm 2. Duplicate P_is may exist in TeleW$_T$. A race condition involving AppNode and AR worker generates these duplicates: A buffered write is generated between Steps 5 and 6 of Algorithm 2. Step 6 does a multi-get for TeleW$_T$ and the processed P_i+dirty values. For those P_is with a buffered write, it does not remove them from TeleW$_T$ value because they were inserted due to the race condition.

2.3 Cache Server Failures

A cache server failure makes buffered writes unavailable, potentially losing content of volatile memory all together. To maintain availability of buffered writes,

Algorithm 2. Each Iteration of AR Worker in Recovery Mode

```
1. Initialize R = {}
2. T = A random value between 0 and ω
3. V = get(TeleW_T)
4. {D} = Select α random documents from V

5. for each D_i in {D} do {
     V = get(P_i+dirty)
     if V exists {
          acquire lease P_i
          V = get(P_i+dirty)
          if V exists {
               success = Update P_i in PStore with buffered writes
               if success {
                    Delete(P_i+dirty)
                    Delete Δ_i (if any)
                    R = R ∪ P_i
               }
          } else R = R ∪ P_i
          release lease P_i
     } else R = R ∪ P_i
   }

6. if R is not empty {
     acquire lease TeleW_T
     Do a multiget on TeleW_T and all P_i+dirty in R
     V = value fetched for TeleW_T
     For those P_i+dirty with a value, remove them from R
     Remove documents in R from V
     put(TeleW_T,V)
     release lease TeleW_T
   }
```

TARDIS replicates buffered writes across two or more cache servers. Both Redis [1] and memcached [2] support this feature.

A cache server such as Redis may also persist buffered writes to its local storage, disk or flash [1]. This enables buffered writes to survive failures that destroy content of volatile memory. However, if a cache server fails when PStore recovers, its contained buffered writes become unavailable and TARDIS must suspend all reads and writes to preserve consistency. Thus, replication is still required to process reads and writes of an AppNode in recovery mode.

3 TARDIS Consistency

TARDIS consists of a large number of AppNodes. Each AppNode may operate in different modes as described in Sect. 2. While AppNode 1 operates in failed

or recovery mode, AppNode 2 may operate in normal mode. This may happen if AppNode 2 processes reads with 100% cache hit while AppNode 1 observes a failed write. This results in a variety of undesirable read-write and write-write race conditions when user requests for a document are directed to AppNode 1 and AppNode 2.

An example of a write-write race condition is that a user Bob who creates an Album and adds a photo to the Album. Assume Bob's "create Album" is directed to AppNode 1. If AppNode 1 is operating in failed mode then it buffers Bob's Album creation in the cache. Once PStore recovers and prior to propagating Bob's album creation to PStore, Bob's request to add a photo is issued to AppNode 2. If AppNode 2 is in normal mode then it issues this write to an album that does not exist in PStore.

An example of a read-write race condition is that Bob changes the permission on his profile page so that his manager Alice cannot see his status. Subsequently, he updates his profile to show he is looking for a job. Assume Bob's first action is processed by AppNode 1 in failed mode and Bob's profile is missing from the cache. This buffers the write as a change (Δ) in the cache. His second action, update to his profile, is directed to AppNode 2 (in normal mode) and is applied to PStore because it just recovered and became available. Now, before AppNode 1 propagates Bob's permission change to PStore, there is a window of time for Alice to see Bob's updated profile.

We present a solution, contextual leases, that employs stateful leases without requiring consensus among AppNodes. It implements the concept of sessions that supports atomicity across multiple keys along with commit and rollbacks. TARDIS with contextual leases maintains the three mode of operation described in Sect. 2. It incorporates Inhibit (I) and Quarantine (Q) leases of [13] and extends them with a marker.

Table 3. IQ lease compatibility.

		Granted	
		I lease	Q lease
Requesting	I lease	Back-off and retry	Back-off and retry
	Q lease	Grant Q and void I lease	Reject and abort requester

The framework of [13] requires: (1) a cache miss to obtain an I lease on its referenced key prior to querying the PStore to populate the cache, and (2) a cache update to obtain a Q lease on its referenced key prior to performing a Read-Modify-Write (R-M-W) or an incremental update such as `append`. Leases are released once a session either commits or aborts. TARDIS uses the write-through policy of IQ-framework. I leases and Q leases are incompatible with each other, see Table 3. A request for an I lease must back-off and retry if the key is already associated with another I or Q lease. A request for a Q lease may void an existing I lease or is rejected and aborted if the key is already associated with

another Q lease. IQ avoids thundering herds by requiring only one out of many requests that observes a cache miss to query PStore and populate the cache. It also eliminates all read-write and write-write race conditions.

In failed mode, a write action continues to obtain a Q lease on a key prior to performing its write. However, at commit time, a Q lease is converted to a P marker on the key. This marker identifies the key-value pair as having buffered writes. The P marker serves as the context for the I and Q leases granted on a key. It has no impact on their compatibility matrix. When the AppNode requests either an I or a Q lease on a key, if there is a P marker then the AppNode is informed of this marker should the lease be granted. In this case, the AppNode must recover the PStore document prior to processing the action.

In our example undesirable write-write race condition, Bob's Album creation using AppNode 1 generates a P marker for Bob's Album. Bob's addition of a photo to the album (using AppNode 2) must obtain a Q lease on the album that detects the marker and requires the application of buffered writes prior to processing this action. Similarly, with the undesirable read-write race condition, Bob's read (performed by AppNode 2) is a cache miss that must acquire an I lease. This lease request detects the P marker that causes the action to process the buffered writes. In both scenarios, when the write is applied to PStore and once the action commits, the P marker is removed as a part of releasing the Q leases.

4 Related Work

The CAP theorem states that a system designer must choose between strong consistency and availability in the presence of network partitions [21]. TARDIS improves availability while preserving consistency.

A weak form of data consistency known as eventual has multiple meanings in distributed systems [28]. In the context of a system with multiple replicas of a data item, this form of consistency implies writes to one replica will eventually apply to other replicas, and if all replicas receive the same set of writes, they will have the same values for all data. Historically, it renders data available for reads and writes in the presence of network partitions that separate different copies of a data item from one another and cause them to diverge [11]. TARDIS teleports failed writes due to either network partitions, PStore failures, or both while preserving consistency.

A write-back policy buffers writes in the cache and applies them to PStore asynchronously. It may not be possible to implement a write-back policy for all write actions of an application. This is because a write-back policy requires a mapping between reads and writes to be able to process a cache miss by applying the corresponding buffered writes to the PStore prior to querying PStore for the missing cache entry. In these scenarios, with TARDIS, one may use DIS (without TARD) that continues to report a failure for cache misses during recovery mode while applying buffered writes to PStore. A write-back policy does not provide

such flexibility. It also does not consider PStore availability. It improves performance of write-heavy workloads. TARDIS objective is to improve the availability of writes during PStore failure.

Everest [23] is a system designed to improve the performance of overloaded volumes during peak load. Each Everest client has a base volume and a store set. When the base volume is overloaded, the client off-loads writes to its idle stores. When the base volume load is below a threshold, the client uses background threads to reclaim writes. Everest clients do not share store set. With TARDIS, AppNodes share the cache and may have different view of PStore, requiring contextual leases. Moreover, its AppNode may fail during PSTore recovery, requiring conversion of non-idempotent writes into idempotent ones.

The race conditions encountered (see Sect. 3) are similar to those in geo-replicated data distributed across multiple data centers. Techniques such as causal consistency [20] and lazy replication [19] mark writes with their causal dependencies. They wait for those dependencies to be satisfied prior to applying them at a replica, preserving order across two causal writes. TARDIS is different because it uses Δ_i to maintain the order of writes for a document D_i that observes a cache miss and is referenced by one or more failed writes. With cache hits, the latest value of the keys reflects the order in which writes were performed. In recovery mode, TARDIS requires the AppNode to perform a read or a write action by either using the latest value of keys (cache hit) or Δ_i (cache miss) to restore the document in PStore prior to processing the action. Moreover, it employs AR workers to propagate writes to persistent store during recovery. It uses leases to coordinate AppNode and AR workers because its assumed data center setting provides a low latency network.

Numerous studies perform writes with a mobile device that caches data from a database (file) server. (See [27] as two examples.) Similar to TARDIS, these studies enable a write while the mobile device is disconnected from its shared persistent store. However, their architecture is different, making their design decisions inappropriate for our use and vice versa. These assume a mobile device implements an application with a local cache, i.e., AppNode and the cache are in one mobile device. In our environment, the cache is shared among multiple AppNodes and a write performed by one AppNode is visible to a different AppNode - this is not true with multiple mobile devices. Hence, we must use leases to detect and prevent undesirable race conditions between multiple AppNode threads issuing read and write actions to the cache, providing correctness.

Host-side Caches [4,16,17] (HsC) such as Flashcache [3] are application *transparent* caches that stage the frequently referenced disk pages onto NAND flash. These caches may be configured with either write-around, write-through, or write-back policy. They are an intermediary between a data store issuing read and write of blocks to devices managed by the operating system (OS). Application caches such as memcached are different than HsC because they require application specific software to maintain cached key-value pairs. TARDIS is somewhat similar to the write-back policy of HsC because it buffers writes in cache and propagates them to the PStore (HsC's disk) in the background. TARDIS is

different because it applies when PStore writes fail. Elements of TARDIS can be used to implement write-back policy with caches such as memcached, Redis, Google Guava [14], Apache Ignite [7], KOSAR [12], and others.

5 Conclusions and Future Research

TARDIS is a family of techniques designed for applications that must provide an always-on experience with low latency, e.g., social networking. In the presence of short-lived persistent store (PStore) failures, TARDIS teleports failed writes by buffering them in the cache and applying them once PStore is available.

An immediate research direction is to conduct a comprehensive evaluation of TARDIS and its variants (TAR, DIS, TARD) to quantify their tradeoff. This includes an evaluation of replication techniques that enhance availability of buffered writes per discussion of Sect. 2.3.

More longer term, we intend to investigate extensions of TARDIS to those logical data models that result in dependence between buffered writes. To elaborate, rows of tables of a relational data model have dependencies such as foreign key dependency. With these SQL systems, TARDIS must manage the dependence between the buffered writes to ensure they are teleported in the right order.

References

1. Redis. https://redis.io/
2. repcached: Data Replication with Memcached. http://repcached.lab.klab.org/
3. McDipper (2013). https://www.facebook.com/10151347090423920
4. Alabdulkarim, Y., Almaymoni, M., Cao, Z., Ghandeharizadeh, S., Nguyen, H., Song, L.: A comparison of flashcache with IQ-Twemcached. In: ICDE Workshops (2016)
5. Amazon DynamoDB (2016). https://aws.amazon.com/dynamodb/pricing/
6. Amazon ElastiCache (2016). https://aws.amazon.com/elasticache/
7. Apache: Ignite - In-Memory Data Fabric (2016). https://ignite.apache.org/
8. Barahmand, S., Ghandeharizadeh, S.: BG: a benchmark to evaluate interactive social networking actions. In: CIDR, January 2013
9. Bronson, N., Lento, T., Wiener, J.L.: Open data challenges at Facebook. In: ICDE (2015)
10. Cattell, R.: Scalable SQL and NoSQL data stores. SIGMOD Rec. **39**, 12–27 (2011)
11. Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: SOSP (2007)
12. Ghandeharizadeh, S., et al.: A Demonstration of KOSAR: an elastic, scalable, highly available SQL middleware. In: ACM Middleware (2014)
13. Ghandeharizadeh, S., Yap, J., Nguyen, H.: Strong consistency in cache augmented SQL systems. In: Middleware, December 2014
14. Google: Guava: Core Libraries for Java (2015). https://github.com/google/guava
15. Google App Engine (2016). https://cloud.google.com/appengine/quotas/

16. Graefe, G.: The five-minute rule twenty years later, and how flash memory changes the rules. In: DaMoN, p. 6 (2007)
17. Holland, D.A., Angelino, E., Wald, G., Seltzer, M.I.: Flash caching on the storage client. In: USENIXATC (2013)
18. Hu, X., Wang, X., Li, Y., Zhou, L., Luo, Y., Ding, C., Jiang, S., Wang, Z.: LAMA: optimized locality-aware memory allocation for key-value cache. In: 2015 USENIX Annual Technical Conference (USENIX ATC 15), July 2015
19. Ladin, R., Liskov, B., Shrira, L., Ghemawat, S.: Providing high availability using lazy replication. ACM Trans. Comput. Syst. **10**(4), 360–391 (1992)
20. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G.: Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In: SOSP (2011)
21. Lynch, N., Gilbert, S.: Brewer's conjecture and the feasibility of consistent, available partition-tolerant web services. ACM SIGACT News **33**, 51–59 (2002)
22. MongoDB Atlas (2016). https://www.mongodb.com/cloud
23. Narayanan, D., Donnelly, A., Thereska, E., Elnikety, S., Rowstron, A.: Everest: scaling down peak loads through I/O Off-loading. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI 2008, Berkeley, CA, USA, pp. 15–28. USENIX Association (2008)
24. Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H.C., McElroy, R., Paleczny, M., Peek, D., Saab, P., Stafford, D., Tung, T., Venkataramani, V.: Scaling memcache at Facebook. In: NSDI, Berkeley, CA, pp. 385–398. USENIX (2013)
25. Ports, D.R.K., Clements, A.T., Zhang, I., Madden, S., Liskov, B.: Transactional consistency and automatic management in an application data cache. In: OSDI. USENIX, October 2010
26. Shahram Ghandeharizadeh, H.H., Nguyen, H.: TARDIS: teleporting failed writes with cache augmented datastores. Technical report 2017–01, USC Database Laboratory (2017). http://dblab.usc.edu/Users/papers/TARDIS.pdf
27. Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.H.: Managing update conflicts in Bayou, a weakly connected replicated storage system. In: SOSP (1995)
28. Viotti, P., Vukolić, M.: Consistency in non-transactional distributed storage systems. ACM Comput. Surv. **49**(1) (2016)

2-Hop Eclipse: A Fast Algorithm for Bandwidth-Efficient Data Center Switching

Liang Liu(✉), Long Gong, Sen Yang, Jun (Jim) Xu, and Lance Fortnow

Georgia Institute of Technology, Atlanta, Georgia
{lliu315,gonglong,sen.yang}@gatech.edu, {jx,fortnow}@cc.gatech.edu

Abstract. A hybrid-switched data center network interconnects its racks of servers with a combination of a fast circuit switch that a schedule can reconfigure at significant cost and a much slower packet switch that a schedule can reconfigure at negligible cost. Given a traffic demand matrix between the racks, how can we best compute a good circuit switch configuration schedule that meets most of the traffic demand, leaving as little as possible for the packet switch to handle?

In this paper we propose 2-hop Eclipse, a new hybrid switch scheduling algorithm that strikes a much better tradeoff between the performance of the hybrid switch and the computational complexity of the algorithm, both in theory and in simulations, than the current state of the art solution Eclipse/Eclipse++.

1 Introduction

Fueled by the phenomenal growth of cloud computing services, data center networks (DCN) continue to grow relentlessly both in size, as measured by the number of racks of servers it has to interconnect, and in speed, as measured by the amount of traffic it has to transport per unit of time from/to each rack [1]. A traditional data center network architecture typically consists of a three-level multi-rooted tree of switches that start, at the lowest level, with the Top-of-Rack (ToR) switches, that each connects a rack of servers to the network [2]. However, such an architecture has become increasingly unable to scale with the explosive growth in both the size and the speed of the DCN, as we can no longer increase the transporting and switching capabilities of the underlying commodity packet switches without increasing their costs significantly.

A cost-effective solution approach to this scalability problem, called hybrid circuit and packet switching, has received considerable research attention in recent years [3–5]. In a hybrid-switched DCN, shown in Fig. 1, n racks of computers on the left hand side (LHS) are connected by both a circuit switch and a packet switch to n racks on the right hand side (RHS). Note that racks on the LHS are an identical copy of those on the RHS; however we restrict the role

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 69–83, 2018.
https://doi.org/10.1007/978-3-319-94295-7_5

of the former to only transmitting data and refer to them as *input ports*, and restrict the role of the latter to only receiving data and refer to them as *output ports*. The purpose of this duplication (of racks) and role restrictions is that the resulting hybrid data center topology can be modeled as a bipartite graph.

Each switch transmits data from input ports (racks on the LHS) to output ports (racks on the RHS) according to the configuration (modeled as a bipartite matching) of the switch at the moment. Usually, the circuit switch is an optical switch [3,6,7], and the packet switch is an electronic switch. Hence the circuit switch is typically an order of magnitude or more faster than the packet switch. For example, the circuit and packet switches might operate at the respective rates of 100 Gbps and 10 Gbps per port. The flip side of the coin however is that the circuit switch incurs a nontrivial reconfiguration delay δ when its configuration has to change. Depending on the underlying technology of the circuit switch, δ can range from tens of microseconds to tens of milliseconds [3,6–9].

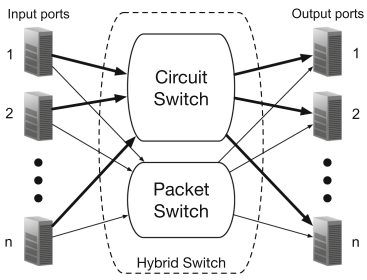


Fig. 1. Hybrid circuit and packet switch

In this paper, we study an important optimization problem stemming from hybrid circuit and packet switching: Given a traffic demand matrix D from input ports to output ports, how to schedule the circuit switch to best (*e.g.*, in the shortest amount of total transmission time) meet the demand? A schedule for the circuit switch consists of a sequence of configurations (matchings) and their time durations $(M_1, \alpha_1), (M_2, \alpha_2), \cdots, (M_K, \alpha_K)$. A workable schedule should let the circuit switch remove (*i.e.*, transmit) most of the traffic demand from D, so that every row or column sum of the remaining traffic matrix is small enough for the packet switch to handle. Since the problem of computing the optimal schedule for hybrid switching, in various forms, is NP-hard [10], almost all existing solutions are greedy heuristics.

1.1 State of the Art: Eclipse and Eclipse++

Since our solution builds upon the state of the art solution called Eclipse [11], we provide here a brief description of it, and its companion algorithm Eclipse++ [11]. Eclipse iteratively chooses a sequence of circuit switch configurations, one per iteration, according to the following greedy criteria: In each iteration, Eclipse tries to extract and subtract a matching (with its duration) from the $n \times n$ traffic demand matrix D that has the largest *cost-adjusted utility*, which we will specify precisely in Sect. 3. Eclipse, like most other hybrid switching algorithms, considers and allows only direct routing in the following sense: All circuit-switched data packets reach their respective final destinations in one-hop (*i.e.*, enters and exits the circuit switch only once).

However, restricting the solution strategy space to only direct routing algorithms may leave the circuit switch underutilized. For example, a connection

(edge) from input i_0 to output j_0 belongs to a matching M that lasts 50 µs, but at the start time of this matching, there is only 40 µs worth of traffic left for transmission from i_0 to j_0, leaving 10 µs of "slack" (*i.e., residue capacity*) along this connection. The existence of connections (edges) with such "slacks" makes it possible to perform indirect (*i.e.,* multi-hop) routing of remaining traffic via one or more relay nodes through a path consisting of such edges.

Besides Albedo [12], Eclipse++ [11] is the only other work that has explored indirect routing in hybrid switching. It was shown in [11] that optimal indirect routing using such "slacks", left over by a direct routing solution such as Eclipse, can be formulated as the maximum multi-commodity flow over a "slack graph", which is NP-complete [13–16]. Eclipse++ is a greedy heuristic that converts, with "precision loss" (otherwise P = NP), this multi-commodity flow computation to a large set of shortest-path computations. Hence the computational complexity of Eclipse++ is still extremely high: Both us and the authors of [11] found that Eclipse++ is roughly three orders of magnitude more computationally expensive than Eclipse [17] for a data center with $n = 100$ racks.

1.2 Our Solution

We develop a new problem formulation that allows the joint optimization of both direct and 2-hop indirect routing, at the same time, against the aforementioned cost-adjusted utility function. We obtain our solution, called 2-hop Eclipse, by applying the Eclipse algorithm as basis for a greedy heuristic to this new optimization problem. This 2-hop Eclipse algorithm, a slight yet subtle modification of Eclipse, has the same asymptotic computational complexity and comparable execution time as Eclipse, but has higher performance gains over Eclipse than Eclipse++, despite the fact that Eclipse++ is three orders of magnitude more computationally expensive.

We emphasize that 2-hop Eclipse uses a very different strategy than Eclipse++ and in particular has no resemblance to "Eclipse++ restricted to 2-hops", which according to our simulations performs slightly worse than, and has almost the same computational complexity as unrestricted Eclipse++.

The rest of the paper is organized as follows. In Sect. 2, we describe the system model and the design objective of this hybrid switch scheduling problem in details. In Sect. 3, we provides a more detailed description of Eclipse [11]. In Sect. 4, we present our solution, 2-hop Eclipse. In Sect. 5, we evaluate the performance of our solution against Eclipse and Eclipse++. Finally, we describe related work in Sect. 6 and conclude the paper in Sect. 7.

2 System Model and Problem Statement

In this section, we formulate the problem of hybrid circuit and packet switching precisely. We first specify the aforementioned traffic demand traffic D precisely. Borrowing the term virtual output queue (VOQ) from the crossbar switching literature [18], we refer to, the set of packets that arrive at input port i and

are destined for output j, as VOQ(i, j). The demand matrix entry $D(i, j)$ is the amount of VOQ(i, j) traffic, within a scheduling window, that needs to be scheduled for transmission by the hybrid switch. It was *effectively* assumed, in all prior works on hybrid switching except Albedo [12] (to be discussed in Sect. 6.1), that the demand matrix D is precisely known before the computation of the circuit switch schedule begins (say at time t). Consequently, all prior hybrid switching algorithms except Albedo [12] perform only batch scheduling of this D. In other words, given a demand matrix D, the schedules of the circuit and the packet switches are computed before the transmissions of the batch (*i.e.*, traffic in D) actually happen. Our 2-hop Eclipse algorithm also assumes that D is precisely known in advance and is designed for batch scheduling only. Since batch scheduling is offline in nature (*i.e.*, requires no irrevocable online decision-making), 2-hop Eclipse algorithm is allowed to "travel back in time" and modify the schedules of the packet and the circuit switches as needed.

In this work, we study this problem of hybrid switch scheduling under the following standard formulation that was introduced in [19]: to minimize the amount of time for the circuit and the packet switches working together to transmit a given traffic demand matrix D. We refer to this amount of time as *transmission time* throughout this paper. A schedule of the circuit switch consists of a sequence of circuit switch configurations and their durations: $(M_1, \alpha_1), (M_2, \alpha_2), \cdots, (M_K, \alpha_K)$. Each M_k is an $n \times n$ permutation (matching) matrix; $M_k(i, j) = 1$ if input i is connected to output j and $M_k(i, j) = 0$ otherwise. The total transmission time of the above schedule is $K\delta + \sum_{k=1}^{K} \alpha_k$, where δ is the reconfiguration delay, K is the total number of configurations in the schedule.

Since computing the optimal circuit switch schedule *alone* (*i.e.*, when there is no packet switch), in its full generality, is NP-hard [10], almost all existing solutions are greedy heuristics. Indeed, the typical workloads we see in data centers exhibit two characteristics that are favorable to such greedy heuristics: sparsity (the vast majority of the demand matrix elements have value 0 or close to 0) and skewness (few large elements in a row or column account for the majority of the row or column sum) [19].

3 Background on Eclipse

Since our 2-hop Eclipse algorithm builds upon Eclipse [11], we provide here a more detailed description of Eclipse. Eclipse iteratively chooses a sequence of configurations, one per iteration, according to the following greedy criteria: In each iteration, Eclipse tries to extract and subtract a matching from the demand matrix D that has the largest *cost-adjusted utility*, defined as follows. For a configuration (M, α) (using a permutation matrix M for a duration of α), its utility $U(M, \alpha)$, before adjusting for cost, is $U(M, \alpha) \triangleq \| \min(\alpha M, D_{\text{rem}}) \|_1$, where D_{rem} denotes what remains of the traffic demand (matrix) D after we subtract from it the amounts of traffic to be served by the circuit switch according to the previous matchings, *i.e.*, those computed in the previous iterations. Note

that $U(M, \alpha)$ is precisely the total amount of traffic the configuration (M, α) would remove from D. The cost of the configuration (M, α) is modeled as $\delta + \alpha$, which accounts for the reconfiguration delay δ. The cost-adjusted utility is simply their quotient $\frac{U(M,\alpha)}{\delta+\alpha}$. Although the problem of maximizing this cost-adjusted utility is very computationally expensive, it was shown in [11] that an computationally efficient heuristic algorithm solution exists that empirically produces the optimal value most of time. This solution, invoking the scaling algorithm for computing *maximum weighted matching* (MWM) [20] $O(\log n)$ times, has a (relatively) low computational complexity of $O(n^{5/2} \log n \log B)$, where B is the value of the largest element in D. Hence the computational complexity of Eclipse is $O(Kn^{5/2} \log n \log B)$, shown in Table 1, where K is the total number of matchings (iterations) used.

Table 1. Comparison of time complexities

Algorithm	Time complexity
Eclipse	$O(Kn^{5/2} \log n \log B)$
2-hop Eclipse	$O(Kn^{5/2} \log n \log B + \min(K, n)Kn^2)$
Eclipse++	$O(WKn^3(\log K + \log n)^2)$

4 Design of 2-Hop Eclipse

Unlike Eclipse, which considers only direct routing, 2-hop Eclipse considers both direct routing and 2-hop indirect routing in its optimization. More specifically, 2-hop Eclipse iteratively chooses a sequence of configurations that maximizes the cost-adjusted utility, just like Eclipse, but the cost-unadjusted utility $U(M, \alpha)$ here accounts for not only the traffic that can be transmitted under direct routing, but also that can be indirectly routed over all possible 2-hop paths.

We make a qualified analogy between this scheduling of the circuit switch and the scheduling of "flights". We view the connections (between the input ports and the output ports) in a matching M_k as "disjoint flights" (those that share neither a source nor a destination "airport") and the residue capacity on such a connection as "available seats". We view Eclipse, Eclipse++, and 2-hop Eclipse as different "flight booking" algorithms. Eclipse books "passengers" (traffic in the demand matrix) for "non-stop flights" only. Then Eclipse++ books the "remaining passengers" for "flights with stops" using only the "available seats" left over after Eclipse does its "bookings". Different than Eclipse++, 2-hop Eclipse "books passengers" for both "non-stop" and "one-stop flights" early on, although it does try to put "passengers" on "non-stop flights" as much as possible, since each "passenger" on a "one-stop flight" costs twice as many "total seats" as that on a "non-stop flight".

It will become clear shortly that the sole purpose of this qualified analogy is for us to distinguish two types of "passengers" in presenting the 2-hop Eclipse

Algorithm 1: 2-hop Eclipse

Input: Traffic demand D;
Output: Sequence of schedules $(M_k, \alpha_k)_{k=1,\ldots,K}$;

```
1  sch ← {};                                              ▷ schedule
2  D_rem ← D;                                             ▷ remaining demand
3  R ← 0;                                                 ▷ residue capacity
4  t_c ← 0;                                               ▷ transmission time of circuit switch
5  while ∃ any row or column sum of D_rem > r_p t_c do
6      Construct I_rem from (D_rem, R);                   ▷ 2-hop demand matrix
7      (M, α) ← argmax_{M∈M, α∈R_+} ‖min(αM, D_rem + I_rem)‖_1 / (δ + α);
8      sch ← sch ∪ {(M, α)};
9      t_c ← t_c + δ + α;
10     Update D_rem;
11     Update R;
12 end
```

algorithm: those "looking for a non-stop flight" whose counts are encoded as the remaining demand matrix D_{rem}, and those "looking for a connection flight" to complete their potential "one-stop itineraries", whose counts are encoded as a new $n \times n$ matrix I_{rem} that we will describe shortly. We emphasize that this analogy shall not be stretched any further, since it would otherwise lead to absurd inferences, such as that such a set of disjoint "flights" must span the same time duration and have the same number of "seats" on them.

4.1 The Pseudocode

The pseudocode of 2-hop Eclipse is shown in Algorithm 1. It is almost identical to that of Eclipse [11]. The only major difference is that in each iteration (of the "while" loop), 2-hop Eclipse searches for a matching (M, α) that maximizes $\frac{\|\min(\alpha M, D_{\text{rem}} + I_{\text{rem}})\|_1}{\delta + \alpha}$, whereas Eclipse searches for one that maximizes $\frac{\|\min(\alpha M, D_{\text{rem}})\|_1}{\delta + \alpha}$. In other words, in each iteration, 2-hop Eclipse first performs some preprocessing to obtain I_{rem} and then substitute the parameter D_{rem} by $D_{\text{rem}} + I_{\text{rem}}$ in making the "argmax" call (Line 7). The "while" loop of Algorithm 1 terminates when every row or column sum of D_{rem} is no more than $r_p t_c$, where r_p denotes the (per-port) transmission rate of the packet switch and t_c denotes the total transmission time used so far by the circuit switch, since the remaining traffic demand can be transmitted by the packet switch (in t_c time). Note there is no occurrence of r_c, the (per-port) transmission rate of the circuit switch, in Algorithm 1, because we normalize r_c to 1 throughout this paper.

4.2 The Matrix I_{rem}

Just like D_{rem}, the value of I_{rem} changes after each iteration. We now explain the value of I_{rem}, at the beginning of the k^{th} iteration ($k > 1$). To do so, we

need to first introduce another matrix R. As explained earlier, among the edges that belong to the matchings $(M_1, \alpha_1), (M_2, \alpha_2), \cdots, (M_{k-1}, \alpha_{k-1})$ computed in the previous $k-1$ iterations, some may have residue capacities. These residue capacities are captured in an $n \times n$ matrix R as follows: $R(l, i)$ is the total residue capacity of all edges from input l to output i that belong to one of these (previous) $k-1$ matchings. Under the qualified analogy above, $R(l, i)$ is the total number of "available seats on all previous flights from airport l to airport i". We refer to R as the *(cumulative) residue capacity matrix* in the sequel.

Now we are ready to define I_{rem}. Consider that, at the beginning of the k^{th} iteration, $D_{\text{rem}}(l, j)$ "local passengers" (*i.e.*, those who are originated at l) who need to fly to j remain to have their "flights" booked. Under Eclipse, they have to be booked on either a "non-stop flight" or a "bus" (*i.e.*, through the packet switch) to j. Under 2-hop Eclipse, however, there is a third option: a "one-stop flight" through an intermediate "airport". 2-hop Eclipse explores this option as follows. For each possible intermediate "airport" i such that $R(l, i) > 0$ (*i.e.*, there are "available seats" on one or more earlier "flights" from l to i), $I^{(l)}_{\text{rem}}(i, j)$ "passengers" will be on the "speculative standby list" at "airport" i, where

$$I^{(l)}_{\text{rem}}(i, j) \triangleq \min\big(D_{\text{rem}}(l, j), R(l, i)\big). \tag{1}$$

In other words, up to $I^{(l)}_{\text{rem}}(i, j)$ "passengers" could be booked on "earlier flights" from l to i that have $R(l, i)$ "available seats", and "speculatively stand by" for a "flight" from i to j that might materialize as a part of matching M_k.

The matrix element $I_{\text{rem}}(i, j)$ is the total number of "nonlocal passengers" who are originated at all "airports" other than i and j and are on the "speculative standby list" for a possible "flight" from i to j. In other words, we have

$$I_{\text{rem}}(i, j) \triangleq \sum_{l \in [n] \setminus \{i, j\}} I^{(l)}_{\text{rem}}(i, j). \tag{2}$$

Recall that $D_{\text{rem}}(i, j)$ is the number of "local passengers" (at i) that need to travel to j. Hence at the "airport" i, a total of $D_{\text{rem}}(i, j) + I_{\text{rem}}(i, j)$ "passengers", "local or nonlocal", could use a "flight" from i to j (if it materializes in M_k). We are now ready to precisely state the difference between Eclipse and 2-hop Eclipse: Whereas $\| \min(\alpha M, D_{\text{rem}}) \|_1$, the cost-unadjusted utility function used by Eclipse, accounts only for "local passengers", $\| \min(\alpha M, D_{\text{rem}} + I_{\text{rem}}) \|_1$, that used by 2-hop Eclipse, accounts for both "local" and "nonlocal passengers".

Note that the term $D_{\text{rem}}(l, j)$ appears in the definition of $I^{(l)}_{\text{rem}}(i, j)$ (Formula (1)), for all $i \in [n] \setminus \{l, j\}$. In other words, "passengers" originated at l who need to travel to j could be on the "speculative standby list" at multiple intermediate "airports". This is however not a problem (*i.e.*, will not result in "duplicate bookings") because at most one of these "flights" (to j) can materialize as a part of matching M_k.

4.3 Update D_{rem} and R

After the schedule (M_k, α_k) is determined by the "argmax call" (Line 7 in Algorithm 1) in the k^{th} iteration, the action should be taken on "booking" the right

set of "passengers" on the "flights" in M_k, and updating D_{rem} (Line 10) and R (Line 11) accordingly. Recall that we normalize r_c, the service rate of the circuit switch, to 1, so all these flights have $\alpha_k \times 1 = \alpha_k$ "available seats". We only describe how to do so for a single "flight" (say from i to j) in M_k; that for other "flights" in M_k is similar. Recall that $D_{\text{rem}}(i,j)$ "local passengers" and $I_{\text{rem}}(i,j)$ "nonlocal passengers" are eligible for a "seat" on this "flight". When there are not enough seats for all of them, 2-hop Eclipse prioritizes "local passengers" over "nonlocal passengers", because the former is more resource-efficient to serve than the latter, as explained earlier. There are three possible cases:

(I) $\alpha \leq D_{\text{rem}}(i,j)$. In this case, only a subset of "local passengers" (directly routed traffic), in the "amount" of α_k, are booked on this "flight", and $D_{\text{rem}}(i,j)$ is hence decreased by α. There is no "available seat" on this "flight" so the value of $R(i,j)$ is unchanged.

(II) $\alpha \geq D_{\text{rem}}(i,j)+I_{\text{rem}}(i,j)$. In this case, all "local" and "nonlocal passengers" are booked on this "flight". After all these "bookings", $D_{\text{rem}}(i,j)$ is set to 0 (all "local passengers" traveling to j gone), and for each $l \in [n] \setminus \{i,j\}$, $D_{\text{rem}}(l,j)$ and $R(l,i)$ each is decreased by $I^{(l)}_{\text{rem}}(i,j)$ to account for the resources consumed by the indirect routing of traffic demand (*i.e.*, "nonlocal passengers"), in the amount of $I^{(l)}_{\text{rem}}(i,j)$, from l to j via i. Finally, $R(i,j)$ is increased by $\alpha - \big(D_{\text{rem}}(i,j)+I_{\text{rem}}(i,j)\big)$, the number of "available seats" that remain on this flight after all these "bookings".

(III) $D_{\text{rem}}(i,j) < \alpha < D_{\text{rem}}(i,j) + I_{\text{rem}}(i,j)$. In this case, all "local passengers" are booked on this "flight", so $D_{\text{rem}}(i,j)$ is set to 0. However, different from the previous case, there are not enough "available seats" left on this "flight" to accommodate all $I_{\text{rem}}(i,j)$ "nonlocal passengers", so only a proper "subset" of them can be booked on this "flight". We allocate this proper "subset" proportionally to all origins $l \in [n] \setminus \{i,j\}$. More specifically, for each $l \in [n] \setminus \{i,j\}$, we book $\theta \cdot I^{(l)}_{\text{rem}}(i,j)$ "nonlocal passengers" originated at l on one or more "earlier flights" from l to i, and also on this "flight", where $\theta \triangleq \frac{\alpha - D_{\text{rem}(i,j)}}{I_{\text{rem}}(i,j)}$. Similar to that in the previous case, after these "bookings", $D_{\text{rem}}(l,j)$ and $R(l,i)$ each is decreased by $\theta \cdot I^{(l)}_{\text{rem}}(i,j)$. Finally, $R(i,j)$ is unchanged as this "flight" is full.

We restrict indirect routing to most 2 hops in 2-hop Eclipse because the aforementioned "duplicate bookings" could happen if indirect routing of 3 or more hops are allowed, making its computation not "embeddable" into the Eclipse algorithm and hence much more computationally expensive. This restriction is however by no means punitive: 2-hop indirect routing appears to have reaped most of the performance benefits from indirect routing, as shown in Sect. 5.5.

4.4 Complexities of 2-Hop Eclipse

Each iteration in 2-hop Eclipse has only a slightly higher computational complexity than that in Eclipse. This additional complexity comes from Lines 6, 10,

and 11 in Algorithm 1. We need only to analyze the complexity of Line 6 (for updating $I_{\text{rem}}^{(l)}$), since it dominates those of others. For each k, the complexity of Line 6 in the k^{th} iteration is $O(kn^2)$ because there were at most $(k-1)n$ "flights" in the past $k-1$ iterations, and for each such flight (say from l to i), we need to update at most $n-2$ variables, namely $I_{\text{rem}}^{(l)}(i,j)$ for all $j \in [n] \setminus \{l,i\}$. Hence the total additional complexity across all iterations is $O(\min(K,n)Kn^2)$, where K is the number of iterations actually executed by 2-hop Eclipse. Adding this to $O(n^{5/2}\log n \log B)$, the complexity of Eclipse, we arrive at the complexity of 2-hop Eclipse: $O(Kn^{5/2}\log n \log B + \min(K,n)Kn^2)$ (see Table 1). We found that the execution times of 2-hop Eclipse are only roughly 20% to 40% longer than that of Eclipse, for the instances (scheduling scenarios) used in our evaluations. Also shown in Table 1, the computational complexity of Eclipse++ is much higher than those of both Eclipse and 2-hop Eclipse. Here W denotes the maximum row/column sum of the demand matrix. Finally, it is not hard to check that the space (memory) complexity of 2-hop Eclipse is $O(\max(K,n)n)$, which is empirically only slightly larger than $O(n^2)$, that of Eclipse. This $O(Kn)$ additional space is needed to store the residue capacities (of no more than n links in each schedule) induced by each schedule (M_k, α_k).

5 Evaluation

In this section, we evaluate the performance of 2-hop Eclipse and compare it with those of Eclipse and Eclipse++, under various system parameter settings and traffic demands. We do not however have Eclipse++ in all performance figures because its computational complexity is so high that it usually takes a few hours to compute a schedule. However, those Eclipse++ simulation results we managed to obtain and present in Sect. 5.5 show conclusively that the small reductions in transmission time using Eclipse++ are not worth its extremely high computational complexity. We do not compare our solutions with Solstice [19] (to be described in Sect. 6.1) in these evaluations, since Solstice was shown in [11] to perform worse than Eclipse in all simulation scenarios. For all these comparisons, we use the same performance metric as that used in [19]: the total time needed for the hybrid switch to transmit the traffic demand D.

5.1 Traffic Demand Matrix D

For our simulations, we use the same traffic demand matrix D as used in other hybrid scheduling works [11,19]. In this matrix, each row (or column) contains n_L large equal-valued elements (large input-output flows) that as a whole account for c_L (percentage) of the total workload to the row (or column), n_S medium equal-valued elements (medium input-output flows) that as a whole account for the rest $c_S = 1 - c_L$ (percentage), and noises. Roughly speaking, we have

$$D = \left(\sum_{i=1}^{n_L} \frac{c_L}{n_L} P_i + \sum_{i=1}^{n_S} \frac{c_S}{n_S} P_i' + \mathcal{N}_1\right) \times 90\% + \mathcal{N}_2 \tag{3}$$

where P_i and P_i' are random $n \times n$ matching (permutation) matrices.

The parameters c_L and c_S control the aforementioned skewness (few large elements in a row or column account for the majority of the row or column sum) of the traffic demand. Like in [11,19], the default values of c_L and c_S are 0.7 (*i.e.*, 70%) and 0.3 (*i.e.*, 30%) respectively, and the default values of n_L and n_S are 4 and 12 respectively. In other words, in each row (or column) of the demand matrix, by default the 4 large flows account for 70% of the total traffic in the row (or column), and the 12 medium flows account for the rest 30%. We will also study how these hybrid switching algorithms perform when the traffic demand has other degrees of skewness by varying c_L and c_S.

As shown in Eq. (3), we also add two noise matrix terms $\mathcal{N}_1$ and $\mathcal{N}_2$ to D. Each nonzero element in $\mathcal{N}_1$ is a Gaussian random variable that is to be added to a traffic demand matrix element that was nonzero before the noises are added. This noise matrix $\mathcal{N}_1$ was also used in [11,19]. However, each nonzero (noise) element here in $\mathcal{N}_1$ has a larger standard deviation, which is equal to 1/5 of the value of the demand matrix element it is to be added to, than that in [11,19], which is equal to 0.3% of 1 (the normalized workload an input port receives during a scheduling window, *i.e.*, the sum of the corresponding row in D). We increase this additive noise here to highlight the performance robustness of our algorithm to such perturbations.

Different than in [11,19], we also add (truncated) positive Gaussian noises $\mathcal{N}_2$ to a portion of the zero entries in the demand matrix in accordance with the following observation. Previous measurement studies have shown that "mice flows" in the demand matrix are heavy-tailed [21] in the sense the total traffic volume of these "mice flows" is not insignificant. To incorporate this heavy-tail behavior in the traffic demand matrix, we add such a positive Gaussian noise – with standard deviation equal to 0.3% of 1 – to 50% of the zero entries of the demand matrix. This way the "mice flows" collectively carry approximately 10% of the total traffic volume. To bring the normalized workload back to 1, we scale the demand matrix by 90% before adding $\mathcal{N}_2$, as shown in Eq. (3).

5.2 System Parameters

In this section, we introduce the system parameters (of the hybrid switch) used in our simulations.

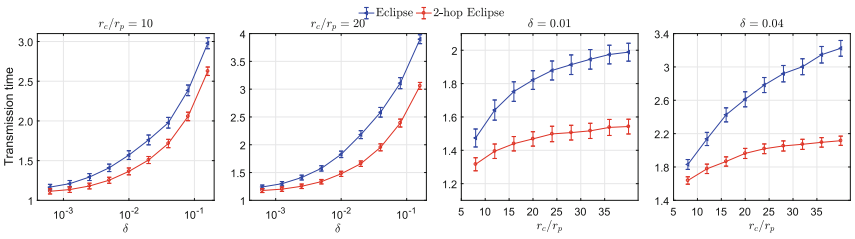


Fig. 2. Performance comparison under different system settings

Network Size: We consider the hybrid switch with $n = 100$ input/output ports. Other reasonably large (say $\geq$32) switch sizes produce similar results.

Circuit Switch Per-Port Rate r_c and Packet Switch Per-Port Rate r_p: As far as designing hybrid switching algorithms is concerned, only their ratio r_c/r_p matters. This ratio roughly corresponds to the percentage of traffic that needs to be transmitted by the circuit switch. The higher this ratio is, the higher percentage of traffic should be transmitted by the circuit switch. This ratio varies from 8 to 40 in our simulations. As explained earlier, we normalize r_c to 1 throughout this paper. Since both the traffic demand to each input port and the per-port rate of the circuit switch are all normalized to 1, the (idealistic) transmission time would be 1 when there was no packet switch, the scheduling was perfect (*i.e.*, no "slack" anywhere), and there was no reconfiguration penalty (*i.e.*, $\delta = 0$). Hence we should expect that all these algorithms result in transmission times larger than 1 under realistic "operating conditions" and parameter settings.

Reconfiguration Delay (of the Circuit Switch) δ: In general, the smaller this reconfiguration delay is, the less time the circuit switch has to spend on reconfigurations. Hence, given a traffic demand matrix, the transmission time should increase as δ increases.

5.3 Performances Under Different System Parameters

In this section, we evaluate the performances of Eclipse and 2-hop Eclipse for different value combinations of δ and r_c/r_p under the traffic demand matrix with the default parameter settings (4 large flows and 12 small flows accounting for roughly 70% and 30% of the total traffic demand into each input port). For each scenario, we perform 100 simulation runs, and report the average transmission time and the 95% confidence interval (the vertical error bar) in Fig. 2. As shown in Fig. 2, 2-hop Eclipse performs better than Eclipse, especially when reconfiguration delay δ and rate ratio r_c/r_p are large. For example, when $\delta = 0.01, r_c/r_p = 10$ (default setting), the average transmission time under 2-hop Eclipse is approximately 13% shorter than that under Eclipse, and when

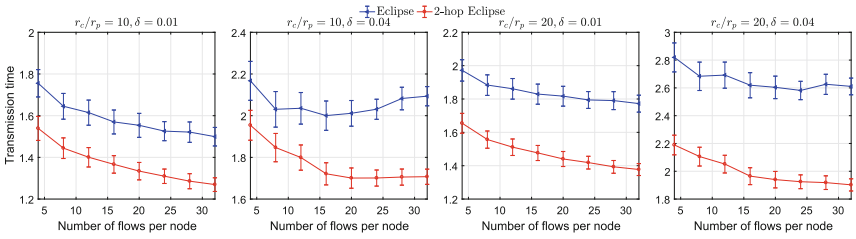


Fig. 3. Performance comparison while varying sparsity of demand matrix

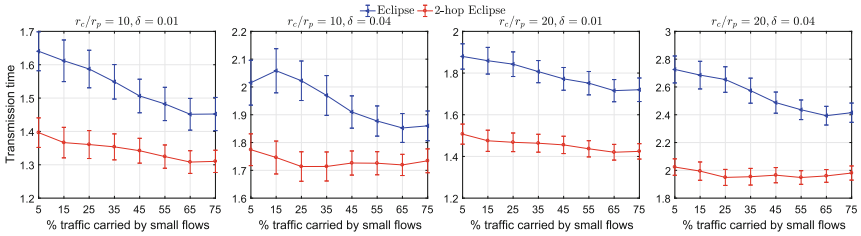


Fig. 4. Performance comparison while varying skewness of demand matrix

$\delta = 0.04, r_c/r_p = 20$, that under 2-hop Eclipse is 23% shorter. The performance of 2-hop Eclipse is also less variable than Eclipse: In all these scenarios, the confidence intervals (of the transmission time) under 2-hop Eclipse are slightly smaller than that under Eclipse.

5.4 Performances Under Different Traffic Demands

In this section, we evaluate the performance robustness of our algorithm 2-hop Eclipse under a large set of traffic demand matrices that vary by sparsity and skewness. We control the sparsity of the traffic demand matrix D by varying the total number of flows $(n_L + n_S)$ in each row from 4 to 32, while fixing the ratio of the number of large flow to that of small flows (n_L/n_S) at $1:3$. We control the skewness of D by varying c_S, the total percentage of traffic carried by small flows, from 5% (most skewed as large flows carry the rest 95%) to 75% (least skewed). In all these evaluations, we consider four different value combinations of system parameters δ and r_c/r_p: (1) $\delta = 0.01, r_c/r_p = 10$; (2) $\delta = 0.01, r_c/r_p = 20$; (3) $\delta = 0.04, r_c/r_p = 10$; and (4) $\delta = 0.04, r_c/r_p = 20$.

Figure 3 compares the transmission time of 2-hop Eclipse and Eclipse when the sparsity parameter $n_L + n_S$ varies from 4 to 32 and the value of the skewness parameter c_S is fixed at 0.3. Figure 4 compares the transmission time of 2-hop Eclipse and Eclipse when the the skewness parameter c_S varies from 5% to 75% and the sparsity parameter $n_L + n_S$ is fixed at 16 $(= 4 + 12)$. In each figure, the four subfigures correspond to the four value combinations of δ and r_c/r_p above.

Both Figs. 3 and 4 show that 2-hop Eclipse performs better than Eclipse under various traffic demand matrices, especially when the traffic demand matrix becomes dense (as the number of flows $n_L + n_S$ increases in Fig. 4). This shows that 2-hop indirect routing can reduce transmission time significantly under a dense traffic demand matrix. This is not surprising: Dense matrix means smaller matrix elements, and it is more likely for a small matrix element to be transmitted entirely by indirect routing (in which case there is no need to pay a large reconfiguration delay for the direct routing of it) than for a large one.

5.5 Compare 2-Hop Eclipse with Eclipse++

In this section, we compare the performances of 2-hop Eclipse and Eclipse++, both indirect routing algorithms, under the default parameter settings. Since Eclipse++ has a very high computational complexity, we perform only 50 simulation runs for each scenario. The results are shown in Fig. 5. They show that Eclipse++ slightly outperforms 2-hop Eclipse only when the reconfiguration delay is ridiculously large ($\delta = 0.64$ unit of time); note that, as explained earlier, the idealized transmission time is 1 (unit of time)! In all other cases, 2-hop Eclipse performs much better than Eclipse++, and Eclipse++ performs only slightly better than Eclipse.

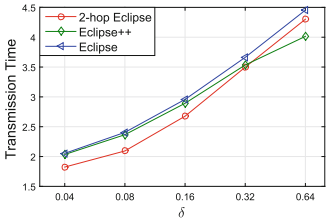


Fig. 5. Performance comparison of Eclipse, 2-hop Eclipse and Eclipse++

6 Related Work

6.1 Other Hybrid Switch Scheduling Algorithms

Liu et al. [19] first characterized the mathematical problem of the hybrid switch scheduling using direct routing only and proposed a greedy heuristic solution, called Solstice. In each iteration, Solstice effectively tries to find the Max-Min Weighted Matching (MMWM) in D, which is the full matching with the largest minimum element. The duration of this matching (configuration) is then set to this largest minimum element. Although its asymptotic computational complexity is a bit lower than Eclipse's, our experiments show that its actual execution time is similar to Eclipse's since Solstice has to compute a larger number of configurations K than Eclipse, which generally produces a tighter schedule.

This hybrid switching problem has also been considered in two other works [22,23]. Their problem formulations are a bit different than that in [11,19], and so are their solution approaches. In [22], matching senders with receivers is modeled as a stable marriage problem, in which a sender's preference score for a receiver equals to the age of the data the former has to transmit to the latter in a scheduling epoch, and is solved using a variant of the Gale-Shapely algorithm [24]. This solution is aimed at minimizing transmission latencies while avoiding starvations, and not at maximizing network throughput, or equivalently minimizing transmission time. The innovations of [23] are mostly in the aspect of systems building and are not on matching algorithm designs.

To the best of our knowledge, Albedo [12] is the only other indirect routing solution for hybrid switching, besides Eclipse++ [11]. Albedo however solves a very different hybrid switching problem: when the traffic demand matrix D is not precisely known in advance and a sizable portion of it has to be estimated. It works as follows. Based on an estimation of D, Albedo first computes a direct routing schedule using Eclipse or Solstice. Then any unexpected "extra workload" resulting from the inaccurate estimation is routed indirectly. However, the

computational complexity of Albedo, not mentioned in [12], appears at least as high as that of Eclipse++, because whereas Eclipse++ has to perform a shortest path computation for each VOQ (mentioned in Sect. 1.1), Albedo has to do so for each TCP/UDP flow belonging to the unexpected "extra workload".

6.2 Optical Switch Scheduling Algorithms

Scheduling of circuit switch alone (*i.e.*, no packet switch) has been studied for decades. Early works often assumed the reconfiguration delay to be either zero [9,25] or infinity [26–28]. Further studies, like DOUBLE [26], ADJUST [10] and other algorithms such as [27,29], take the actual reconfiguration delay into consideration. Recently, a solution called Adaptive MaxWeight (AMW) [30,31] was proposed for optical switches (with nonzero reconfiguration delays). The basic idea of AMW is that when the maximum weighted configuration (matching) has a much higher weight than the current configuration, the optical switch is reconfigured to the maximum weighted configuration; otherwise, the configuration of the optimal switch stays the same. However, this algorithm may lead to long queueing delays (for packets) since it usually reconfigures infrequently.

7 Conclusion

In this paper, we propose 2-hop Eclipse, a hybrid switching algorithm that can jointly optimize both direct and 2-hop indirect routing. It is a slight but non-trivial modification of, has nearly the same asymptotical complexity as, and significantly outperforms the state of the art algorithm Eclipse, which optimizes only direct routing.

Acknowledgments. This work is supported in part by US NSF through award CNS 1423182.

References

1. DeCusatis, C.: Optical interconnect networks for data communications. J. Lightw. Technol. **32**(4), 544–552 (2014)
2. Patel, P., et al.: Ananta: cloud scale load balancing. In: SIGCOMM Computer Communication Review, vol. 43, pp. 207–218. ACM (2013)
3. Farrington, N., et al.: Helios: a hybrid electrical/optical switch architecture for modular data centers. SIGCOMM Comput. Commun. Rev. **40**(4), 339–350 (2010)
4. Wang, H., et al.: Design and demonstration of an all-optical hybrid packet and circuit switched network platform for next generation data centers. In: OFC. Optical Society of America (2010)
5. Farrington, N., et al.: A 10 us hybrid optical-circuit/electrical-packet network for datacenters. In: OFC. Optical Society of America (2013)
6. Chen, K., et al.: OSA: an optical switching architecture for data center networks with unprecedented flexibility. IEEE/ACM Trans. Netw. **22**(2), 498–511 (2014)

7. Wang, G., et al.: c-through: part-time optics in data centers. In: SIGCOMM Computer Communication Review, vol. 40, pp. 327–338. ACM (2010)
8. Liu, H., et al.: Circuit switching under the radar with REACToR. NSDI **14**, 1–15 (2014)
9. Porter, G., et al.: Integrating microsecond circuit switching into the data center. SIGCOMM Comput. Commun. Rev. **43**(4), 447–458 (2013)
10. Li, X., et al.: On scheduling optical packet switches with reconfiguration delay. IEEE J. Sel. Areas Commun. **21**(7), 1156–1164 (2003)
11. Bojja Venkatakrishnan, S., et al.: Costly circuits, submodular schedules and approximate carathéodory theorems. In: SIGMETRICS, pp. 75–88. ACM (2016)
12. Li, C., et al.: Using indirect routing to recover from network traffic scheduling estimation error. In: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (2017)
13. Even, S., et al.: On the complexity of time table and multi-commodity flow problems. In: FOCS, pp. 184–193. IEEE (1975)
14. Ford Jr., L.R., et al.: A suggested computation for maximal multi-commodity network flows. Manage. Sci. **5**(1), 97–101 (1958)
15. Chiang, T.C.: Multi-commodity network flows. Oper. Res. **11**(3), 344–360 (1963)
16. Garg, N., et al.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM J. Comput. **37**(2), 630–652 (2007)
17. Bojja Venkatakrishnan, S.: In-Person Discussions at Sigmetrics Conference, June 2017
18. Mekkittikul, A., et al.: A starvation-free algorithm for achieving 100% throughput in an input-queued switch. In: Proceedings of ICCCN, vol. 96, pp. 226–231. Citeseer (1996)
19. Liu, H., et al.: Scheduling techniques for hybrid circuit/packet networks. In: ACM CoNEXT. CoNEXT 2015, pp. 41:1–41:13. ACM, New York. ACM (2015)
20. Duan, R., et al.: A scaling algorithm for maximum weight matching in bipartite graphs. In: SODA, pp. 1413–1424. SIAM (2012)
21. Benson, T., et al.: Network traffic characteristics of data centers in the wild. In: IMC, pp. 267–280. ACM (2010)
22. Ghobadi, M., et al.: ProjecTOR: agile reconfigurable data center interconnect. In: SIGCOMM, pp. 216–229 (2016)
23. Hamedazimi, N., et al.: FireFLY: a reconfigurable wireless data center fabric using free-space optics. In: SIGCOMM, pp. 319–330 (2014)
24. Gale, D., et al.: College admissions and the stability of marriage. Am. Math. Mon. **69**(1), 9–15 (1962)
25. Inukai, T.: An efficient SS/TDMA time slot assignment algorithm. IEEE Trans. Commun. **27**(10), 1449–1455 (1979)
26. Towles, B., et al.: Guaranteed scheduling for switches with configuration overhead. IEEE/ACM Trans. Netw. **11**(5), 835–847 (2003)
27. Wu, B., et al.: Nxg05-6: minimum delay scheduling in scalable hybrid electronic/optical packet switches. In: GLOBECOM, pp. 1–5. IEEE (2006)
28. Gopal, I.S., et al.: Minimizing the number of switchings in an SS/TDMA system. IEEE Trans. Commun. **33**(6), 497–501 (1985)
29. Fu, S., et al.: Cost and delay tradeoff in three-stage switch architecture for data center networks. In: HPSR, pp. 56–61. IEEE (2013)
30. Wang, C.-H., et al.: End-to-end scheduling for all-optical data centers. In: INFOCOM, pp. 406–414. IEEE (2015)
31. Wang, C.-H.: et al.: Heavy traffic queue length behavior in switches with reconfiguration delay. arXiv preprint arXiv:1701.05598 (2017)

A Prediction Approach to Define Checkpoint Intervals in Spot Instances

Jose Pergentino A. Neto[1(✉)], Donald M. Pianto[2], and Célia Ghedini Ralha[1]

[1] Department of Computer Science, University of Brasília, Brasília, DF, Brazil
paraujo@aluno.unb.br, ghedini@unb.br
[2] Statistics Department, University of Brasília, Brasília, DF, Brazil
dpianto@unb.br

Abstract. Cloud computing providers have started offering their idle resources in the form of virtual machines (VMs) without availability guarantees. Know as transient servers, these VMs can be revoked at any time without user intervention. Spot instances are transient servers offered by Amazon at lower prices than regular dedicated servers. A market model was used to create a bidding scenario for cloud users of servers without service reliability guarantees, where prices changed dynamically over time based on supply and demand. To prevent data loss, the use of fault tolerance techniques allows the exploration of transient resources. This paper proposes a strategy that addresses the problem of executing a distributed application, like bag-of-tasks, using spot instances. We implemented a heuristic model that uses checkpoint and restore techniques, supported by a statistical model that predicts time to revocation by analyzing price changes and defining the best checkpoint interval. Our experiments demonstrate that by using a bid strategy and the observed price variation history, our model is able to predict revocation time with high levels of accuracy. We evaluate our strategy through extensive simulations, which use the price change history, simulating bid strategies and comparing our model with real time to revocation events. Using instances with considerable price changes, our results achieve an 94% success with standard deviation of 1.36. Thus, the proposed model presents promising results under realistic working conditions.

Keywords: Cloud computing · Virtual Machine · Spot instance
Fault tolerance · Checkpoint · Machine learning · Statistical model

1 Introduction

Cloud Computing provides an environment with high scalability and flexibility and affordable pricing for users. It has emerged as an important community resource and is considered a new paradigm for the execution of applications with high levels of security and reliability [2,8]. According to [2], Cloud Computing is an environment where services and resources are available in a distributed way and on demand over the Internet. Furthermore, services and resources are

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 84–93, 2018.
https://doi.org/10.1007/978-3-319-94295-7_6

available in a virtual, abstract and managed platform with resources dynamically scalable in a transparent way to the user. A particular class of service is Infrastructure as a Service (IaaS), which offers a set of Virtual Machines (VMs) with different types and capacity, allowing users to choose instances according their needs.

Recently, cloud providers have been adopting a business model that offers the idle capacity of VMs resources in the form of transient servers [18]. These VMs are offered without a guarantee of their availability at considerably lower prices compared to normal servers. Amazon offers transient servers, namely Spot Instances (SIs), using a dynamic pricing model which uses a market model based on users' bids [14]. A user acquires a VM when the maximum value they are willing to pay, their bid, exceeds the current price of the instance. A bid fault (BF) occurs when the current price of the VM is above the user's bid price. Using these transient instances may weaken running applications because the environment is volatile, even if the user does not want to lose the instance. In [15], the authors state that, to effectively use transient cloud servers, it is necessary to choose appropriate fault-tolerance mechanisms and parameters.

In this paper, we present an heuristic model for efficient usage of transient instances, providing a novel strategy that uses machine learning to predict Time To Revocation (TTR) using historical data patterns and bid strategies. This approach allows the definition of fault tolerant mechanisms with appropriate parameters, reducing costs even more. These parameters are computed based on the features explained later in the paper (Sect. 4), addressing the impact of the checkpointing interval on the SIs to reduce monetary costs and the application's total execution time. The remainder of this paper is structured as follows. First, in Sect. 2, we briefly discuss related work in this domain. In Sect. 3 we describe the proposed heuristic for the definition of the parameters in a checkpoint and restore FT technique, based on a machine learning model that analyzes historical price changes of SIs. In Sect. 4 we present the results of exhaustive experiments that comply with our proposal. Finally, in Sect. 5 we summarize this study with conclusions and discuss elements for future research.

2 Related Work

Researchers have been doing significant work in the cloud computing area. The use of computing services as a utility is defined as "on demand delivery of infrastructure, applications, and business processes in a security-rich, shared, scalable, and computer based environment over the Internet for a fee" [13]. This model brings benefits to providers and consumers, especially because users can choose services and resources according to their application needs. It reduces both the monetary costs and idle resources, making it possible to provide them to other consumers. In recent years, a lot of research has been done regarding the use of FT techniques in cloud environments. Migration is considered as a FT mechanism in [6,16,17,19], and in [5] a framework is proposed to migrate servers

between VMs in an efficient way. Using a set of proposed virtualization techniques, [17] offers a platform that uses migration mechanisms to offer the cheapest environment, using unsecured spot instances instead of on-demand servers.

In a transient server perspective, much of the research focuses on exploring efficient ways to use SIs, either using techniques to define best user bids to avoid revocation [6,9,15,19] or providing FT mechanisms to guarantee application execution without data loss [17,21,22]. In the area of FT in SIs, recent works have focused on providing users a transparent environment with a high level of reliability, ensuring the execution of applications without data loss. Use of FT mechanisms are even more significant in transient instances than conventional servers where high availability is present. Even with additional overhead on running processes due to the necessity of save states, as shown in Fig. 1, checkpoint and restore is one of the most used FT mechanisms [3] and is explored in [4,10,22].

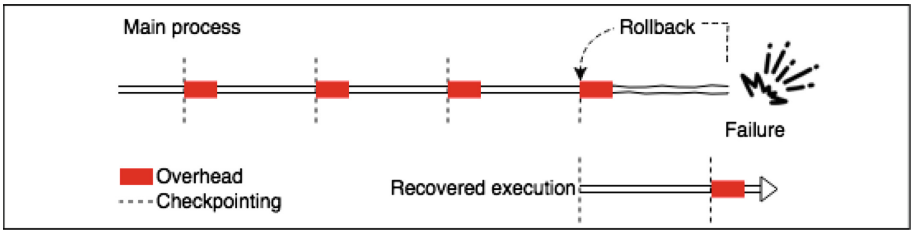


Fig. 1. Checkpoint intervals and restoration on failure of a single process execution.

From a checkpoint FT technique perspective, an important element is the checkpointing interval, because it affects total execution time. A process that executes in 30 h with 10 min of overhead checkpointing and an interval of 1 h will add 290 min of total execution time. A well defined interval time leads to lower costs and better use of execution time. Larger intervals imply faster execution.

In [19,20], the authors use a static interval of 60 min, arguing that the billing window used by the service is one hour, ignoring the fact that memory and data intensive applications need time for the save process. Alternatively, other authors present a strategy in which interval time is defined through monitoring price changes [21]. When a new price is registered, a new checkpoint is created. Using a set of defined times, [22] uses continuous checkpoints in static intervals of 10, 30 and 60 min. What is common among these related works is that the defined intervals are not dynamic. An important point is understanding the impact of the checkpointing interval on the SIs in order to reduce the monetary costs and the application's total execution time. This is the focus of this work.

3 Proposal

In this section, we present a high-level overview of our heuristic with a machine learning model that analyzes historical and current prices of SIs and their changes

to define appropriate checkpoint intervals. Compared to existing similar studies, our scheme uses observed price change patterns of SIs in terms of hour-in-day (HID) and day-of-week (DOW), as proposed in [7,8]. The performance and availability of VMs in a cloud environment varies according to instance types, region, zone and different times of day. A pattern can be observed in Fig. 2, that shows, in a range from April to December of 2017, how many price changes occurred on each day of the week (a) and during each hour of the day (b) in an *m1.large* SI, grouped by zones in the *US-WEST* region. The number of changes peaks during weekdays, as opposed to weekends, and after 5pm (17 h).

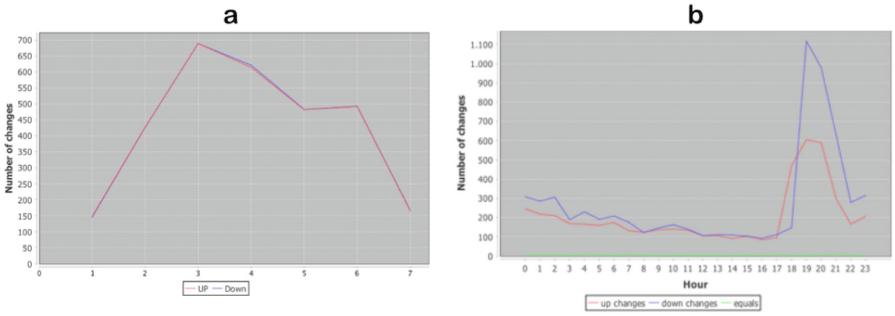


Fig. 2. Observed patterns in price changes in DOW and HID.

Compared to existing similar studies, our scheme uses a Case Based Reasoning (CBR) [1], that classifies price changes in order to comply with HID and DOW patterns. An intelligent system must be able to adapt to new situations, allowing for reasoning and understanding relationships between facts to recognize real situations and learn based on its experience. Applications that use CBR can learn through previous cases, adapting the knowledge database when evaluating new experiences.

Using a set of spot price history traces P_t, it is possible to perform simulations and create a set of real cases Δ. Algorithms that recover spot prices and simulate an auction are needed to estimate the time which a SI stays with a user until its revocation. Using a $\text{Est}_{bid}(P_t, n)$ function, that calculates the median price over the previous n days, simulations can produce scenarios where the user's bid U_{bid} assumes the returned value from Est_{bid}. With these assumptions, a set of cases can be defined as $\Delta = \{\delta_1, \delta_2 \ldots, \delta_n\}$, in which δ is a structure with the following attributes: $\delta.instance\text{-}type$, representing a type of VM instance; $\delta.day\text{-}of\text{-}week$, an integer number that determine a day of week, being 1 for Sunday and 7 for Saturday; $\delta.hour\text{-}in\text{-}day$ with a range of 0–23 representing a full hour; $\delta.price$ with instance price value; and $\delta.time\text{-}to\text{-}revocation$, that retains how much time (in minutes) a VM was alive until TTR.

In addition, our model uses a Survival Analysis (SA) statistical model [12]. SA is a class of statistical methods which are used when the data under analysis

represents the time to a specific event. In this paper, we use a nonparametric technique which incorporates censored data to avoid estimation bias. The event under study is a BF and the time to this event, TTR, is treated as a random variable.

We would like to estimate the largest survival time (T_S) for which we have a high confidence (98%) that our SI will not be revoked. This time is defined by $T_S = \arg\max_t \{t \in \mathbb{R} \mid P(TTR > t) \geq 0.98\}$ and can be calculated as the 98th percentile of TTR (TTR_{98}) from estimated Survival Curves (SC). The SCs, $\hat{S}$, are estimated using the Kaplan-Meier estimator [11], in which t_i represents the upper limit of a small time interval, D_i the number of deaths within that interval, and S_i the number of survivors at the beginning of the interval. If no deaths occur in a given interval, the SC does not decrease.

$$\hat{S}(t) = \prod_{i:t_i \leq t} (1 - \frac{D_i}{S_i}) \tag{1}$$

Figure 3 shows SCs for some SIs on Saturday at midnight. TTR_{98} for the *c3.large* instance is a little less than 100 h, while TTR_{98} for the *m3.large* is much smaller.

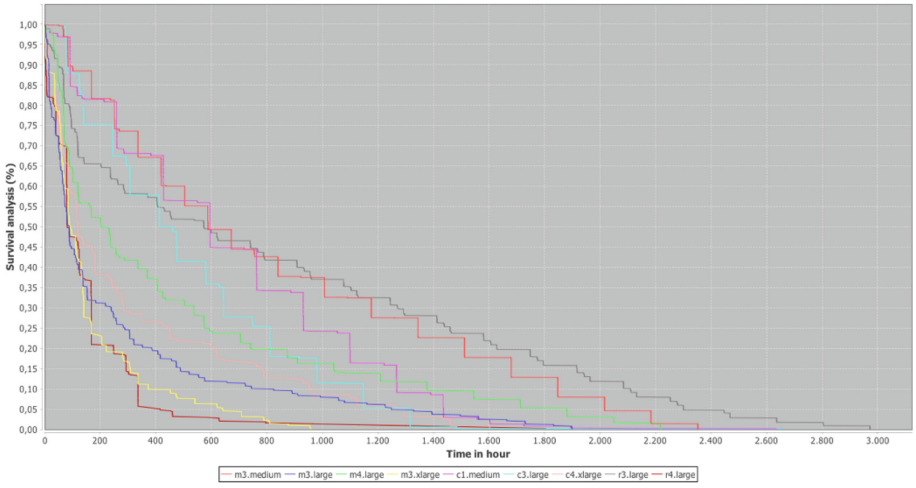


Fig. 3. Generated SC of selected SIs.

The main goal of our scheme is to find an optimal checkpointing interval that minimizes the total runtime and reduces costs. Using both CBR and SA, a SC is produced, allowing quantiles of the TTR to be used to find a probable T_S and use it as part of strategy, avoiding occurrences of BF.

4 Experimental Evaluation

We evaluate our proposed checkpointing scheme for 37 SIs in all zones of the *US-WEST* region. Using 6 months of real price changes, collected from April to December of 2017, our experiment simulates 389.576 scenarios with $U_{bid} = \text{Est}_{bid}(P_t, n) \mid n \in \mathbb{N} = \{1 \dots 7\}$, using all combinations of days and hours for the 13 weeks in October, November and December.

From the collected data, approximately 1 million cases were generated. Figure 4 shows a SC generated by the experiments. The value of TTR_{98} is indicated. We then repeated the same experiment, varying n in the Est_{bid} function between 1 and 7 days. Our results show that $n = 7$ is the best strategy, with all others performing worse.

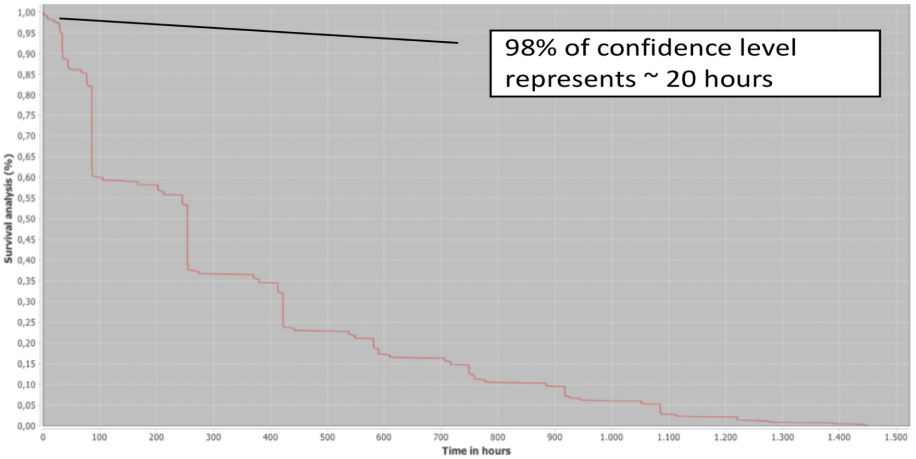


Fig. 4. SC of *c1.medium* on Sunday at 5am.

Each DOW and HID relationship has its own SC and a set of SC quantiles can be represented as a Survival Time Matrix (STM), as illustrated in Table 1, which shows TTR_{98} times, in minutes, obtained by 98th confidence level.

Table 1. Generated STM of *c1.medium* SI.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	95	129	76	220	142	100	86	130	75	706	636	569	201	168	119	330	270	208	150	90	65	151	117	94
2	83	134	78	221	159	102	84	107	68	134	123	173	129	93	68	118	95	167	109	75	69	210	121	101
3	103	129	73	226	143	100	63	154	80	445	360	309	132	189	223	156	250	192	133	74	66	345	142	101
4	108	136	80	138	111	90	199	172	79	124	360	300	215	149	102	61	261	199	141	81	68	361	194	154
5	173	138	87	119	73	102	69	128	80	717	634	574	461	438	372	318	249	191	138	85	63	227	160	128
6	85	70	69	198	134	81	67	128	78	591	253	471	169	109	123	231	189	130	89	93	64	84	80	70
7	73	120	60	207	114	99	77	134	74	680	575	346	120	111	220	238	210	164	116	89	67	130	77	120

To evaluate STM times, a set of experiments that compare STM values with real scenarios was created. Using the same experimental scenario, 80.808 simulations were executed. An example of the *c1.medium* instance is illustrated in Fig. 5. Very good results can be seen, with mean success rates around 80% with standard deviation of 1.36% points.

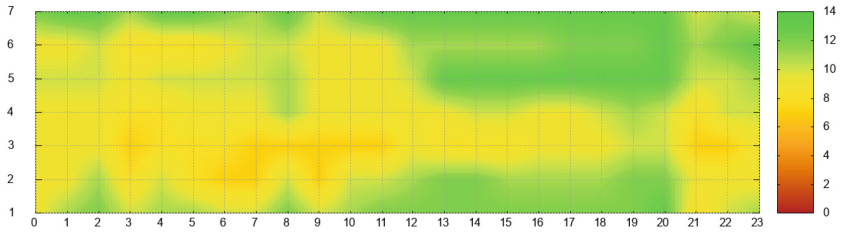


Fig. 5. Heat map of survival success compared of TTR times in *c1.medium* instance.

An unsecured gap can be observed on Monday and Tuesday between 1am and 10am, with 8 failures in 13 attempts, showing that another bid strategy is needed to increase the observed TTR and achieve better success rates. Considering all 37 instances' simulations, the success rate goes up to around 94%. This occurs because instances with expensive price values have stable variations, allowing long TTR times.

To achieve better results, a new strategy can be incorporated into the $\hat{S}$ function, where more recent results receive a greater weight. In this strategy, the time interval used in our experiment was reduced to 5 months, from April to August of 2017. With this change, a new STM was generated, creating a new matrix that represents the survival times over this period ($\mathrm{MT}_{S'}$).

Then, a new matrix that represents only September of 2017 was generated ($\mathrm{MT}_{S''}$) and a new STM was calculated as the median of $\mathrm{MT}_{S'}$ and $\mathrm{MT}_{S''}$. The results of this calculation are presented in Table 2.

Table 2. A new STM created after recency strategy in $\hat{S}$ function.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	263	203	143	83	1251	1191	1176	1090	1042	970	891	831	460	723	670	610	576	694	636	576	514	456	395	315
2	280	569	349	256	404	363	313	255	195	135	92	276	215	155	96	71	585	524	465	402	344	284	233	173
3	114	338	348	309	253	193	157	159	119	80	888	820	664	700	649	794	756	674	614	588	516	480	420	353
4	343	304	224	184	215	175	117	167	158	100	192	390	333	295	238	245	187	130	74	62	394	334	297	223
5	163	184	247	195	500	472	425	434	374	314	182	117	188	133	126	742	687	627	572	512	447	387	599	539
6	461	293	223	323	238	206	172	252	410	350	317	415	355	276	235	192	135	284	424	364	300	252	188	128
7	109	619	558	498	446	391	333	289	229	159	113	800	494	685	852	795	735	680	612	555	500	438	378	318

Better results were achieved with this change and can be observed in Fig. 6. Giving greater weight to more recent results allowed a gain of 8%, reaching an 88% success rate under the same conditions of our experiment. The new results for the DOW and HID relationship can be observed in Fig. 6.

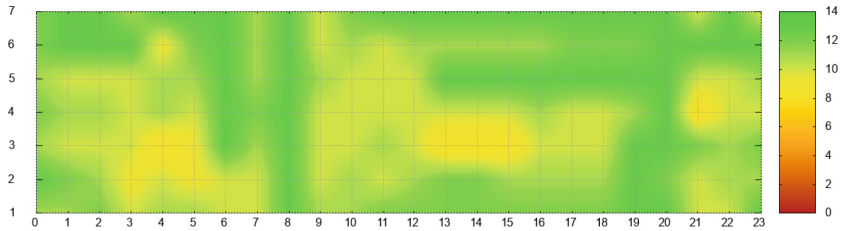


Fig. 6. Heat map of survival success after recency factor in $\hat{S}$.

Expanding our experiment to involve all 37 instances available in the US-WEST region, the success rate increases considerably, reaching 94% in our simulations, considering STM times with greater weight for more recent results. This occurs because when treating all instances, we include instances with advanced resources and with higher and more stable prices. The demand for these instances by cloud users is low and intermittent, allowing long TTR times.

Given the accuracy achieved in our experiments, we have shown that a survival curve can offer data to be used in Fault Tolerance (FT) parameters, e.g. the checkpointing interval. Given an application's required execution time (T_A) and T_S, estimated by STM, an interval can be given by Eq. 2.

$$interval = \begin{cases} \min(60, T_S) & T_A \geq T_S \\ 60 * \frac{T_S}{T_A} & T_A < T_S \end{cases} \tag{2}$$

Considering a data intensive application task that needs time $T_A = 500$ min, having checkpointing time $C_t = 10$ min, using intervals used in [19,22] (30 and 60), total execution varies between 580 and 620 min. A bigger interval means faster execution. With $T_S = 1000$ min, Eq. 2 results an interval of 120 min, leading to total execution time of 540 min. All collected data (30.3 GB) and source code are available at a public repository and can be used to reproduce our experiment and test other scenarios not contemplated in this paper.

5 Conclusions

In this paper, we present a heuristic model that uses price change traces of SIs as input in a machine learning and statistical model to predict TTR. To the best of our knowledge, there is not any available study that combines machine learning (CBR model) and SA to predict TTR in SIs. We demonstrate how our strategy can be used to define FT parameters, like the checkpointing interval or number of replications. Aspects related to the restoration of failed executions are beyond the scope of this paper. We consider that existing fault tolerance techniques can be used and their respective parameters can be defined using our approach. Furthermore, in order to achieve longer survival times, new bid strategies can be introduced, e.g. using bid values defined by a percent increase over actual instance prices.

The performed simulations have confirmed the efficiency and the accuracy of the proposed model when used in a real environment with real data. Better results are achieved when strategies which place more weight on recent data are explored. Future work can explore other strategies to exploit more recent data, such as comparing recent instance price changes with previously established change patterns to update FT parameters, like checkpoint intervals. Use of the proposed strategy to definite the checkpoint interval decreases the total time of execution, allowing more effective use of SIs and reducing monetary costs for cloud users.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
2. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: 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)
3. Elnozahy, E.N.(M.), Alvisi, L., Wang, Y.-M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. **34**(3), 375–408 (2002). https://doi.org/10.1145/568522.568525. ISSN 0360-0300
4. He, X., Shenoy, P., Sitaraman, R., Irwin, D.: Cutting the cost of hosting online services using cloud spot markets. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC 2015, pp. 207–218. ACN Press, New York (2015)
5. Huang, K., Gao, X., Zhang, F., Xiao, J.: COMS: customer oriented migration service. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 692–695. IEEE, June 2017
6. Jangjaimon, I., Tzeng, N.-F.: Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. IEEE Trans. Comput. **64**(2), 396–409 (2015)
7. Javadi, B., Thulasiram, R.K., Buyya, R.: Characterizing spot price dynamics in public cloud environments. Future Gener. Comput. Syst. **29**(4), 988–999 (2013)
8. Javadi, D., Thulasiramy, R.K., Buyya, R.: Statistical modeling of spot instance prices in public cloud environments. In: 2011 Fourth IEEE International Conference on Utility and Cloud Computing, pp. 219–228. IEEE, December 2011
9. Khandelwal, V., Chaturvedi, A., Gupta, C.P.: Amazon EC2 spot price prediction using regression random forests. IEEE Trans. Cloud Comput. **7161**(c), 1 (2017)
10. Lee, K., Son, M.: DeepSpotCloud: leveraging cross-region GPU spot instances for deep learning. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 98–105. IEEE, June 2017
11. Meeker, W.: Statistical Methods for Reliability Data. Wiley, New York (1998)
12. Miller Jr., R.G.: Survival Analysis, vol. 66. Wiley, Hoboken (2011)
13. Rappa, M.A.: The utility business model and the future of computing services. IBM Syst. J. **43**(1), 32–42 (2004)
14. Amazon Web Services: Amazon EC2 spot instances. https://aws.amazon.com/ec2/spot. Accessed Jan 2018
15. Sharma, P., Irwin, D., Shenoy, P.: Portfolio-driven resource management for transient cloud servers. Proc. ACM Meas. Anal. Comput. Syst. **1**(1), 5:1–5:23 (2017)

16. Sharma, P., Lee, S., Guo, T., Irwin, D., Shenoy, P.: SpotCheck: designing a derivative IaaS cloud on the spot market. In: Proceedings of the Tenth European Conference on Computer Systems - EuroSys 2015, pp. 1–15 (2015)
17. Sharma, P., Lee, S., Guo, T., Irwin, D., Shenoy, P.: Managing risk in a derivative IaaS cloud. IEEE Trans. Parallel Distrib. Syst. **9219**(c), 1 (2017)
18. Shastri, S., Rizk, A., Irwin, D.: Transient guarantees: maximizing the value of idle cloud capacity. In: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 992–1002. IEEE, November 2016
19. Voorsluys, W., Buyya, R.: Reliable provisioning of spot instances for compute-intensive applications. In: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, pp. 542–549. IEEE, March 2012
20. Voorsluys, W., Garg, S.K., Buyya, R.: Provisioning spot market cloud resources to create cost-effective virtual clusters. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) ICA3PP 2011 Part I. LNCS, vol. 7016, pp. 395–408. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24650-0_34
21. Yi, S., Andrzejak, A., Kondo, D.: Monetary cost-aware checkpointing and migration on amazon cloud spot instances. IEEE Trans. Serv. Comput. **5**(4), 512–524 (2012)
22. Zhou, J., Zhang, Y., Wong, W.-F.: Fault tolerant stencil computation on cloud-based GPU spot instances. IEEE Trans. Cloud Comput. **7161**(c), 1 (2017)

Research Track: Cloud Container

Cloud Service Brokerage and Service Arbitrage for Container-Based Cloud Services

Ruediger Schulze(✉)

IBM Germany Research and Development GmbH,
Schoenaicher Str. 220, 71034 Boeblingen, Germany
ruediger.schulze@de.ibm.com

Abstract. By exploiting the portability of application containers, platform- and software-as-a-service providers receive the flexibility to deploy and move their cloud services across infrastructure services delivered by different providers. The aim of this research is to apply the concepts of cloud service brokerage to container-based cloud services and to define a method for service arbitrage in an environment with multiple infrastructure-as-a-service (IaaS) providers. A new placement method based on constraint programming is introduced for the optimised deployment of containers across multiple IaaS providers. The benefits and limitations of the proposed method are discussed and the efficiency of the method is evaluated based on multiple deployment scenarios.

1 Introduction

Container-based applications are on the rise. Docker [1] as an open platform for application containers has become very popular since its first release in June 2014 and a complete ecosystem of supporting tools has grown up. With Docker, developers receive the option to package applications and their dependencies into self-contained images that can run as containers on any server. Multiple large Cloud Service Providers (CSP) have embraced Docker as container technology and announced alliances with the Docker company. Container-based applications are highly portable and independent of the hosting provider and hardware.

Cloud services such as Platform-as-a-services (PaaS) and Software-as-a-services (SaaS) have usually a topology that aligns with the components of a multi-tier architecture, including tiers for presentation, business logic and data access. The topology of the cloud service describes the structure of the IT service delivered by the CSP, the components of the service and the relationships between them. Cloud services designed for the use in enterprises have multiple Quality of Service (QoS) requirements such as High Availability (HA) and Disaster Recovery (DR) targets, performance and scalability requirements, and require compliance to security standards and data location regulations. The requirements for HA, DR and horizontal scaling are the drivers to design and deploy

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 97–111, 2018.
https://doi.org/10.1007/978-3-319-94295-7_7

the cloud services in multi-node topologies which may span multiple availability zones and geographical sites.

PaaS and SaaS providers in public or private cloud environments receive with Docker the option to deliver their cloud services using container-based applications or micro-services. The portability of the application containers enables them to easily move cloud services between IaaS providers and to locations where, for instance, the customer's data resides, the best SLA is achieved or where the hosting is most cost-effective. PaaS and SaaS providers will benefit from employing brokerage services that allow them to choose from a variety of IaaS providers and to optimise the deployment of their cloud services with respect to the QoS requirements, distribution and cost. By using a Cloud Service Broker (CSB), PaaS and SaaS providers will be enabled to structure their offering portfolio with additional flexibility and to quickly respond to new or changing customer requirements. New options arise from the easy deployment of the containers across the multiple IaaS providers. A PaaS or SaaS provider may deploy a container to a new geographic location by selecting a different IaaS provider when the primary one is not available there. A globally delivered cloud service may be provided by using resources from multiple IaaS providers. DR use cases may be realized by selecting a different provider for the backup site. Load-balancing and scalability built into the cloud service will allow to gradually migrating the service from one provider to another one with no downtime by simply moving the containers to the new provider.

Cloud service brokerage for application containers requires to define a new method for the optimised placement of the containers on IaaS resources of multiple providers. The method has to take the attributes of the containers and the IaaS resources into account, and honour the QoS requirements of enterprise-ready cloud services. The initial placement of the containers has to follow the specification of the topology of the cloud service. Aside of the attributes used for rating IaaS providers, the CSB has to consider container related attributes such as the built-in container support of the IaaS providers, the packaging of containers in virtual machines and the clustering of containers. The optimisation of the infrastructure resources of the container-based cloud services has to be without impact and visibility to the cloud service consumers. Access to user data and network connectivity to the cloud services have to be handled and delivered uninterrupted, and without the need to reconfigure clients.

A CSB for application containers may use a constraint programming-based engine to make the placement decision about the optimal IaaS provider. The engine will use as input the requirements of the container and information about each of the available IaaS providers. The objective of this research is to define a method for service arbitrage that allows for the optimised placement of containers in a cloud environment with multiple IaaS providers and to demonstrate the feasibility of the proposed method.

2 Background Research

2.1 Cloud Service Broker

The definition of a cloud broker as introduced by NIST is widely referenced by other authors, e.g., in [2–5]. In the NIST CCRA [6], a cloud broker is described as an "entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers." The cloud broker is an organisation that serves as a third-party entity as a centralised coordinator of cloud services for other organisations and enables users to interact through a single interface with multiple service providers [2,7]. Gartner [8] describes cloud service brokerage as "an IT role and business model in which a company or other entity adds value to one or more (public or private) cloud services on behalf of one or more consumers of that service via three primary roles including aggregation, integration and customisation brokerage."

NIST [6] and Gartner [9] define three categories of services that can be provided by a CSB. *Service intermediation* enables a CSB to enhance a service by improving some specific capability and providing value-added services to cloud consumers. Service intermediation is responsible for service access and identity management, performance reporting, enhanced security, service pricing and billing. A CSB uses *service aggregation* to combine and integrate multiple services into one or more new services while ensuring interoperability. The CSB is responsible for the integration and consolidation of data across multiple service providers, and ensures the secure movement of the data between the cloud consumer and the cloud providers. The key aspect of the service aggregation is to ease service selection and present services from separate providers as a unique set of services to the cloud service consumer. *Service arbitrage* is the process of determining the best CSP. The CSB has the flexibility to choose a service from multiple providers and can, for example, use a credit-scoring service to measure and select a provider with the best score. Service arbitrage adds flexibility and choice to service aggregation as the aggregated services are not fixed.

Six key attributes of CSB are derived in [7] mainly based on the categories defined by NIST [6] and Gartner [8,9], and used for evaluation of existing CSBs: *intermediation*, *aggregation*, *arbitrage*, *customisation*, *integration* and *standardisation*. According to [10], integration is focused on creating an unified, common system of services by integrating private and public clouds or bridging between CSPs. Customisation refers to the aggregation and integration with other value-added services, including the creation of new original services [10]. Both integration and customisation are closely interlinked with service aggregation and intermediation, and in-fact it is difficult to find distinct definitions of these attributes in the literature. Standardisation among the CSB mechanisms and across the cloud services of different providers enables interoperability, and supports the process of service selection by a CSB.

A comprehensive model of a CSB architecture is described in [2]. The CSB environment is built of the same components as the management platform for a single cloud but additional complexity is introduced into the system by the

requirement to support multiple CSPs. Bond [2] distinguishes between the functions of the Cloud Broker Portal, the Cloud Broker Operations and Management and the multiple CSPs, and aligns them in a layered model of a vendor-agnostic CSB architecture. Governance is introduced in addition as a set of functions which are orthogonal to the layers of the model and have to be realised for all functional components of the architecture.

A taxonomy of brokering mechanisms is given in [12]. *Externally managed* brokers are provided off-premise by a centralized entity, for instance, by a third-party cloud provider or SaaS offering. *Externally managed* brokers are transparent for applications using the services provided by the broker. *Directly managed* brokers are incorporated into an application, have to keep track of the application's performance and be built to meet the availability and dependability requirements of the applications. *Externally managed* brokers can be classified into SLA-based and trigger-action brokers. In case of *SLA-based broker*, a cloud user specifies the brokering requirements of a SLA in form of constraints and objectives, and the CSB finds the most suitable cloud service by taking into account the user requirements specified by the SLA. For trigger-action broker, a cloud user specifies a set of triggers and associates actions with them. A trigger becomes active when a predefined condition is meet, e.g., the threshold of a specific metric is exceeded, and the associated action is executed. Actions can be, for instance, scale-out and provisioning activities, and may include bursting into a public cloud when there is a spike in the demand for computing capacity.

In the context of the interoperability of clouds, the following challenges are described in [13] and applied here to CSBs. In an environment with multiple cloud service providers, each provider is expected to have its own SLA management mechanism. A CSB has to establish federation of the SLAs from each CSP in order to set up and enforce a global SLA. Methods and protocols for the negotiation of dynamic and flexible SLAs between the CSB and the multiple CSPs are required. Another important issue is the enforcement of the SLAs in environments with conflicting policies and goals, e.g., a CSB may offer a service with a SLA for HA, while none of the providers are willing to offer such a service. In addition to the SLA, there can be a Federation-Level Agreement that defines rules and conditions between the CSB and the CSPs, e.g., about pools of resources and the QoS such as the minimum expected availability. The CSB has to establish functions for matching the guaranteed QoS of cloud services offered by the CSPs with the QoS requirements of the end-user, and for monitoring that the promised QoS and SLA is provided to the cloud service consumer. The dependencies of a CSP to other providers have to be considered by the CSB as well. The QoS of a higher-layered service can be affected in cases when the CSP of the service itself uses external services. If one of the providers of the lower-layered services is not functioning properly, the performance of the higher-layered service may be affected and impact finally the SLA agreed by the CSB. The CSB has to guarantee the security, confidentiality and privacy of the data processed by the services provided. Within the country where the services are delivered, the CSB must comply with the legislation and laws concerning

the privacy and security of the data. Therefore, the CSB has to implement geolocation and legislation awareness policies and enforce compliance with those policies. As part of the SLA management, services of specific providers can be avoided or agreement can be made that placing data outside of a given country is prohibited.

The results of a survey of CSB projects are described in [14]. The authors consider four categories of CSB technologies: *CSBs for performance* to address issues of cloud performance comparison and prediction, *CSBs for application migration* which provide decision support when moving applications to the cloud, *theoretical models for CSBs* which describe purely theoretical and mathematical techniques and *data for CSBs* that summarises providers of data and metrics available for use by CSBs. A comprehensive list of commercial CSB projects is given in [10]. Recent research about CSBs has a significant focus on service arbitration across numerous CSPs, in particular on optimising the allocation of resources from different IaaS providers. The use of arbitration engines enables CSBs to automatically determine the best CSP and service for any given customer order. The attributes considered in the optimisation process vary depending on the proposed method. Typically attributes for rating IaaS providers are: the supported operating systems and configurations, geographical location, costs and rates, bandwidth and performance, SLA terms, legalisation and security, compliance and audit [15,16].

The placement of VMs in cloud and virtual environments is a critical operation as it has an direct impact on the performance, resource utilisation, power-consumption and cost. The subject of VM placement is widely discussed in the research literature. Detailed reviews of the current VM placement algorithms can be found in [17,18]. According to [18], the mono-objective, multi-objective solved as mono-objective and pure multi-objective approaches can be distinguished with respective optimisation objectives. *Mono-objective methods* are designed for the optimisation of a single objective or the individual optimisation of more objective functions, but one at a time. For *multi-objective solved as mono-objective approaches*, multiple objective functions are combined into a single objective function. The weighted sum method is used most often by this approach. This method defines one objective function as the linear combination of multiple objectives. A disadvantage of this approach is that it requires knowledge about the correct combination of the objective functions – which is not always available. Pure multi-objective approaches have a vector of multiple objective functions that are optimised. Only a small number of methods are described in the literature which use a pure multi-objective approach for VM placement.

A broad range of different VM placement schemes (18 different in total) are analysed in [17]. Constraint programming based VM placement is described as one of the considered placement schemes. The following classification of the placement schemes is proposed: *Resource-aware VM placement schemes* consider the infrastructure resource requirements of the VMs are considered in the placement decisions by these schemes. Efficient resource-aware placement tries to optimally place VMs on the hosts, so that the overall resource utilisation

is maximised. Most of the schemes consider CPU and memory resources, some network resources, and a minor number includes the device or disk I/O, or try to minimise the number of active hosts. Designed for the use by the CSPs, *power-aware VM placement schemes* try to make cloud data centres more efficient and to reduce the power consumption in order to enable green cloud computing. The objective of these schemes is to reduce the number of active host, networking and other data center components. The methods include VM consolidation and packaging of VMs on the same host or in the same rack, and powering off not needed VMs, hosts and network components. The attributes considered by the power-aware schemes include the CPU utilisation (e.g., based on the states: idle, average, active and over utilised), the server power usage and the host status (running, ready, sleep, off), costs for network routing and data center power usage, and the distance between VMs. *Network-aware VM placement schemes* try to reduce the network traffic or try to distribute network traffic evenly in order avoid congestion. The placement schemes allocate the VMs with more or extensive communication on the same host, to the same switch and rack, or within the same data center in order to reduce the network traffic within the data center and across data centres. Most common is the consideration of the traffic between VMs by the network-aware VM placement schemes, some evaluate the traffic between the hosts and selected schemes try to minimise the transfer time between the VMs and data, and the distance between the VMs. *Cost-aware VM placement schemes* try to reduce the costs for the CSPs while considering the QoS of the cloud services and honouring the SLAs. The schemes use different types of costs as attributes, such as the VM, physical machine, cooling and data center costs, and the distance between the VMs and the clients.

Conceptually a similar approach is taken in [18] with the classification of the objective functions. Based on the study of 56 different objective functions, the classification into five groups of objective functions is described: *Energy Consumption Minimisation*, *Network Traffic Minimisation*, *Economical Costs Optimisation*, *Performance Maximisation* and *Resource Utilisation Maximisation*. Most of the publications are focused on single-cloud environments, i.e. for use by CSPs. Seven of the methods are suitable for multi-cloud environments, i.e. use multiple cloud computing data centres from one or more CSP. Only two articles take a broker-oriented approach.

Different methods are employed by CSBs for rating IaaS providers, e.g., genetics algorithms [19,20] and rough sets [16,21]. Multiple projects propose CSBs which take advantage of the different price options such as for on-demand, reservation and spot instances [22], examples can be found in [23–26]. There are a couple of academic and open-source implementations of CSBs, e.g., STRATOS [27], QBROKAGE [19] and CompatibleOne [28].

2.2 Constraint Programming

Constraint programming is a form of declarative programming which uses variables and their domains, constraints and objective functions in order to solve

a given problem. The purpose of constraint programming is to solve constraint satisfaction problems as defined in [30,31]:

Definition 1 (Constraint Satisfaction Problem). *A Constraint Satisfaction Problem $\mathcal{P}$ is a triple $\mathcal{P} = (X, D, C)$ where X is an n-tuple of variables $X = (x_1, x_2, \ldots, x_n)$, D is a corresponding n-tuple of domains $D = (D_1, D_2, \ldots, D_n)$ such that $x_i \in D_i$, C is a t-tuple of constraints $C = (C_1, C_2, \ldots, C_t)$.*

The domain D_i of a variable x_i is a finite set of numbers, and can be continuous or of a discrete set of values. In order to describe a constraint satisfaction problem $\mathcal{P}$, a finite sequence of variables with their respective domains is used together with a finite set of constraints. A constraint over a sequence of variables is a subset of the Cartesian product of the variables' domains in the scope of the constraint.

Definition 2 (Constraints). *$C = (C_1, C_2, \ldots, C_t)$ is the set of constraints. A constraint C_j is a pair $\left(R_{S_j}, S_j\right)$ where R_{S_j} is a relation on the variables in $S_j = scope(C_j)$, i.e. the relation R_{S_j} is a subset of the Cartesian product $D_1 \times \ldots \times D_m$ of the domains $D_1, D_2, \ldots, D_m$ for the variables in S_j.*

A solution of the Constraint Satisfaction Problem $\mathcal{P}$ is defined as follows:

Definition 3 (Solution of $\mathcal{P}$). *A solution to the Constraint Satisfaction Problem $\mathcal{P}$ is a n-tuple $A = (a_1, a_2, \ldots, a_n)$ where $a_i \in D_i$ and each C_j is satisfied in that R_{S_j} holds the projection of A onto the scope of S_j.*

The definition of multiple global constraints such as the `alldifferent` constraint is described in the literature. The constraint `alldifferent` requires that the variables $x_{1,2}, \ldots, x_n$ take all different values. An overview of the most popular global constraints is given in [32].

Several publications focus on the use of CP-based cloud selection and VM placement methods. A method for cloud service match making based on QoS demand is introduced in [33]. CP is a convenient method for optimising the placement of VMs, as placement constraints can be directly expressed by variables representing the assignment of the VMs to the hosts and the allocation of resources for the VMs placed on each host. Resource-aware VM placement schemes are presented in [34–36]. A combined CP and heuristic algorithm is utilised in [35]. Special focus is put on fault tolerance and HA in [36]. In [37], the CP-based, open source VM scheduler BtrPlace [38] is used to exhibit SLA violations for discrete placement constraints, as these do not consider interims states of a reconfiguration process. As consequence, BtrPlace is extended with a preliminary version of continuous constraints and it is proved that these remove the temporary violation and improve the reliability. Power-aware methods are discussed in [39,40].

3 Method for Service Arbitrage

Aims of this research is to define a method for optimising the deployment of container-based applications across different CSPs. The objective is to find for

a given Docker container c the optimal host h. A host h is a virtual or physical machine that meets the requirements of the container c and is delivered by an IaaS provider. The optimisation problem to be solved can be described as transformation of a container domain $\mathbb{C}$ into a host domain $\mathbb{H}$:

$$f : \mathbb{C} \rightarrow \mathbb{H},\ c \mapsto h,\ \textit{where}\ c \in \mathbb{C}\ \textit{and}\ h \in \mathbb{H}. \tag{1}$$

The requirements of a container c are expressed as vector $r_c = (r_{c,1}, r_{c,i} \ldots, r_{c,n})$. Each attribute $r_{c,i}$ is an element or subset of a domain R_i with a finite number of elements. Likewise, the attributes of a host h are described by the a vector $a_h = (a_{h,1}, a_{h,j}, \ldots, a_{h,m})$ for which each attribute $a_{h,j}$ belongs to a domain A_j and $A = (A_1, A_j, \ldots, A_m)$. In order to solve the optimisation problem, the requirements $r_{c,i}$ of the containers and the attributes $a_{h,j}$ of the hosts have to be considered. As method for finding the optimal host h for a given container c, a CP model is used. As per Definition 1, a constraint satisfaction problem $\mathcal{P}$ is defined as the triple $\mathcal{P} = (X, D, C)$. The objective of the CP model is to provide solutions for the container placement problem $\mathcal{P}_{placement}$. The variables X and the corresponding domains D of the problem $\mathcal{P}_{placement}$ are defined by the attribute domains A of the hosts $\mathbb{H}$, i.e. $D = A$ and $X = (a_1, a_j, \ldots, a_m)$ where $a_j \in A_j$. Provided that the function *index* returns the index set I of any given set S, so that

$$\textit{index} : S \rightarrow I,\ s_i \mapsto i = \textit{index}(s), \tag{2}$$

then can be defined for the variables and domains of the host attributes the following:

$$\begin{aligned}
\textit{provider}: \quad & a_1 \in A_1 \text{ and } A_1 = \textit{index}(\mathbb{P}) \text{ where} && (3)\\
& \mathbb{P} = \{\text{AWS}, \text{DigitalOcean}, \text{Azure}, \text{SoftLayer}, \text{Packet}\} \\
\textit{host type}: \quad & a_2 \in A_2 \text{ and } A_2 = \textit{index}(\mathbb{T}) \text{ where} && (4)\\
& \mathbb{T} = \{\text{physical}, \text{virtual}\} \\
\textit{region}: \quad & a_3 \in A_3 \text{ and } A_3 = \textit{index}(\mathbb{R}) \text{ where} && (5)\\
& \mathbb{R} = \{\text{Australia}, \text{Brazil}, \text{Canada}, \text{France}, \text{Germany}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Great Britain}, \text{Hongkong}, \text{India}, \text{Ireland}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Italy}, \text{Japan}, \text{Mexico}, \text{Netherlands}, \} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Singapore}, \text{California}, \text{Iowa}, \text{New Jersey}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{New York}, \text{Oregon}, \text{Texas}, \text{Washington}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Virginia}\} && (6)
\end{aligned}$$

$$\textit{data centres}: \quad a_4 \in A_4 \text{ and } A_4 = \{1, .., 52\} \tag{7}$$
$$\textit{availability zone}: \quad a_5 \in A_5 \text{ and } A_5 = \{0, .., 5\} \tag{8}$$
$$\textit{cpu}: \quad a_6 \in A_6 \text{ and } A_6 = \{1, .., 40\} \tag{9}$$
$$\textit{memory (GB)}: \quad a_7 \in A_7 \text{ and } A_7 = \{1, .., 488\} \tag{10}$$
$$\textit{disk size (GB)}: \quad a_8 \in A_8 \text{ and } A_8 = \{1, .., 48000\} \tag{11}$$
$$\textit{disk type}: \quad a_9 \in A_9 \text{ and } A_9 = \textit{index}(\mathbb{D}) \text{ where} \tag{12}$$
$$\mathbb{D} = \{\text{HDD}, \text{SSD}\}$$
$$\textit{private}: \quad a_{10} \in A_{10} \text{ and } A_{10} = \{0, 1\} \tag{13}$$
$$\textit{optimised}: \quad a_{11} \in A_{11} \text{ and } A_{11} = \textit{index}(\mathbb{O}) \text{ where} \tag{14}$$
$$\mathbb{O} = \{\text{compute}, \text{memory}, \text{gpu}, \text{storage}\}$$
$$\mathbb{O} = \mathbb{O} \cup \{\text{network}, \text{none}\}$$
$$\textit{cost}: \quad a_{12} \in A_{12} \text{ and } A_{12} = \{1, .., 99999\} \tag{15}$$

In order to describe the constraints C, the requirements $r_c = (r_{c,1}, r_{c,i} \ldots, r_{c,n})$ of a container c have to be detailed first. The properties *ha_scale* and *dr_scale* are introduced to address the requirements for HA and DR. The requirement *ha_scale* describes how many containers have to be deployed across the data centres or available zones within one region and *dr_scale* is the number of containers that have to be deployed across multiple regions of the same or multiple providers in order to achieve protection against a disaster. The *op_factor* $r_{c,9}$ allows to request a larger host which can run $r_{c,9}$ instances of the container c. The *price limit* ($ 0.0001 per hour) specifies the maximum cost of host per hour which must not be exceeded. The attribute *private* $r_{c,11}$ allows to request to place the container c on a dedicated host.

$$\textit{host type}: \quad r_{c,1} \in R_1 \text{ and } R_1 = A_2 \tag{16}$$
$$\textit{region}: \quad r_{c,2} \in R_2 \text{ and } R_2 = A_3 \tag{17}$$
$$\textit{cpu}: \quad r_{c,3} \in R_3 \text{ and } R_3 = A_6 \tag{18}$$
$$\textit{memory (GB)}: \quad r_{c,4} \in R_4 \text{ and } R_4 = A_7 \tag{19}$$
$$\textit{disk size (GB)}: \quad r_{c,5} \in R_5 \text{ and } R_5 = A_8 \tag{20}$$
$$\textit{disk type}: \quad r_{c,6} \in R_6 \text{ and } R_6 = A_9 \tag{21}$$
$$\textit{ha\_scale}: \quad r_{c,7} \in R_7 \text{ and } R_7 = \{1, .., 5\} \tag{22}$$
$$\textit{dr\_scale}: \quad r_{c,8} \in R_8 \text{ and } R_8 = \{1, .., 5\} \tag{23}$$
$$\textit{op\_factor}: \quad r_{c,9} \in R_9 \text{ and } R_9 = \{1, .., 10\} \tag{24}$$
$$\textit{price limit}: \quad r_{c,10} \in R_{10} \text{ and } R_{10} = A_{12} \tag{25}$$
$$\textit{private}: \quad r_{c,11} \in R_{11} \text{ and } R_{11} = A_{10} \tag{26}$$
$$\textit{optimized}: \quad r_{c,12} \in R_{12} \text{ and } R_{12} = A_{11} \tag{27}$$
$$\textit{image}: \quad r_{c,13} \in R_{13} \text{ and } R_{13} = \mathbb{I} \text{ where} \tag{28}$$
$$\mathbb{I} \text{ is the set of Docker images}$$

In order to allow the placement of a container c either on an existing host or a new host, the domain $\mathbb{H}$ to be searched is defined as the union of the host templates T, i.e. the set of hosts which can be provisioned, and the already provisioned hosts H:

$$\mathbb{H} = T \cup H \tag{29}$$

wherein T is the superset of the templates t^p from all providers $\mathbb{P}$:

$$T = \bigcup_{p \in \mathbb{P}} T^p \text{ and } t^p \in T^p \tag{30}$$

The template t^p is described by the attribute vector a_t^p and domains A^p:

$$a_t^p = (a_{t,1}^p, a_{t,j}^p, \ldots, a_{t,m}^p) \text{ and } a_{t,j}^p \in A_j^p \tag{31}$$

$$A^p = A_1^p, A_j^p, \ldots, A_m^p \text{ and } p \in \mathbb{P} \tag{32}$$

The set of already provisioned hosts H is the union of the deployed hosts h^p at all providers $\mathbb{P}$:

$$H = \bigcup_{p \in \mathbb{P}} H^p \text{ and } h^p \in H^p \tag{33}$$

Provided that the host h^p is provisioned using the template h^p and that $\mathbb{C}_h$ denotes the set of containers deployed on the host h, the available resources on the host h^p can the be determined as follows:

$$cpu: \quad a_{h,6} = a_{t,6}^p - \sum_{c \in \mathbb{C}_h} r_{c,3} \tag{34}$$

$$memory(GB): \quad a_{h,7} = a_{t,7}^p - \sum_{c \in \mathbb{C}_h} r_{c,4} \tag{35}$$

$$disksize(GB): \quad a_{h,8} = a_{t,8}^p - \sum_{c \in \mathbb{C}_h} r_{c,5} \tag{36}$$

$$cost: \quad a_{h,12} = \frac{a_{t,12}^p}{|\mathbb{C}_h|} \tag{37}$$

The variables X represent the attributes of a single host. In order to respond to the requirements for HA and DR, multiple containers have to be placed on k anti-collocated hosts. $X_{e,f}$ denotes the variables and $a_j^{e,f}$ the attributes required to describe the k hosts $h_{e,f}$:

$$k = dr\_scale \cdot ha\_scale \tag{38}$$

$$X_{e,f} = (a_1^{e,f}, a_j^{e,f}, \ldots, a_m^{e,f}) \text{ where } a_j^{e,f} \in A_j \tag{39}$$

$$1 \leq e \leq dr\_scale \tag{40}$$

$$1 \leq f \leq ha\_scale \tag{41}$$

The constraints C can then be defined as follows:

$$hosts: \quad C_1 : \vee \Big(\wedge \big(a_j^{e,f} = a_{h,j}\big)\Big),\ \forall h \in \mathbb{H} \text{ and } 1 \leq j \leq 12 \tag{42}$$

$$host\ type: \quad C_2 : \begin{cases} a_2^{e,f} = r_{c,1}, & \text{if } r_{c,1} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{43}$$

$$region: \quad C_3 : \begin{cases} a_3^{e,f} = r_{c,2}, & \text{if } r_{c,2} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{44}$$

$$cpu: \quad C_4 : a_6^{e,f} \geq \big(r_{c,3} \cdot r_{c,9}\big) \tag{45}$$

$$memory: \quad C_5 : a_7^{e,f} \geq \big(r_{c,4} \cdot r_{c,9}\big) \tag{46}$$

$$disk\ size: \quad C_6 : a_8^{e,f} \geq \big(r_{c,5} \cdot r_{c,9}\big) \tag{47}$$

$$disk\ type: \quad C_7 : \begin{cases} a_9^{e,f} = r_{c,6}, & \text{if } r_{c,6} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{48}$$

$$price\ limit: \quad C_8 : \begin{cases} a_{12}^{e,f} \leq r_{c,10}, & \text{if } r_{c,10} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{49}$$

$$private: \quad C_9 : \begin{cases} a_{10}^{e,f} = r_{c,11}, & \text{if } r_{c,11} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{50}$$

$$optimised: \quad C_{10} : \begin{cases} a_{11}^{e,f} = r_{c,12}, & \text{if } r_{c,12} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{51}$$

$$HA: \quad C_{11} : \Big(\big(a_5^{e,f} \neq a_5^{e,g}\big) \wedge \big(a_4^{e,f} = a_4^{e,g}\big) \wedge \big(az_{enabled} = 1\big)\Big) \vee \Big(\big(a_4^{e,f} \neq a_4^{e,g}\big) \wedge \big(a_3^{e,f} = a_3^{e,g}\big)\Big) \tag{52}$$

where $1 \leq g \leq ha\_scale$ and $f \neq g$

$$DR: \quad C_{12} : \big(a_3^{d,f} \neq a_3^{e,g}\big) \tag{53}$$

where $1 \leq d \leq dr\_scale$ and $d \neq e$

The property $az_{enabled}$ indicates that the used technology generally supports the deployment of hosts into availability zones. The objective function f_{cost} for minimising the cost across the k hosts is defined as follows:

$$f_{cost} = minimise \sum a_{12}^{d,f}. \tag{54}$$

4 Results

In order to verify the effectiveness of the proposed method for service arbitrage, a series of deployment scenarios was executed with the objective to verify that the most cost-effective host for a given configuration is chosen for deployment. The verification was performed in an environment with five different IaaS providers.

In total, 3587 host templates with different server configurations in 22 regions and 52 data centres were given to the CP model as input. Actual prices from the CSPs were used for each of the configurations. The CP model was implemented using NumberJack [41] and the CP solver "Mistral" [42]. The execution of the test scenarios showed that the used CP solver is able to find optimal solutions for the CP model. The CP solver returns solutions in a reasonable time when only a single container has to be placed. The runtime of the CP solver increases significantly when multiple containers have to placed across different locations for HA and DR. In this case, the number of variables and constraints supplied to the CP model increase and the objective function becomes more complex. In order to validate if better performance results can be achieved with another CP solver supported by NumberJack, the "MiniSat" solver [43] was used. The test execution with "MiniSat" showed an increased CPU utilization on the hosting server and significant longer runtime. All further tests were executed using the "Mistral" solver afterwards. By applying a lower price limit to a deployment scenario with multiple containers, it was possible to obtain an optimal solution quicker. With the price limit applied, the initial number of host templates can be already reduced before the actual constraints are added to the CP model. The deployment scenarios with multiple containers shows as well that regions on different continents may be selected by the CP solver as optimal solution, e.g. Europe and North America. This solution may not be suitable for business use in all cases, e.g., when legal restrictions apply. Additional locality constraints or an objective function for minimising the distance between regions may be added to the CP model in future.

5 Summary and Conclusions

An important aspects of the proposed CP model is that the emphasis is not on rating the CSPs but the particular hosts for their capability to run a specific container, so that the CSP becomes only an attribute of the host. The proposed CP model was validated based on test data with host templates from five IaaS providers, 22 regions and 52 data centres. In this experimental research it is shown that the proposed CP model is capable to find the optimal placement for containers also in complex environments, and that HA and DR topologies of applications can be realised. It is further shown that the CP solver "Mistral" [42] used by NumberJack runs stable for large CP models. The duration of the process for finding the optimal solution increases significantly when multiple containers have to placed across different locations for HA and DR. Hence, the practical use of the proposed CP model in a production environment is not possible. Further research is required to reduce the complexity inherited from the input attributes before the actual CP model is constructed. Other CP solvers may be evaluated and the integration of rule-based algorithms such as Rete [44] into the CSB framework can be investigated. The objective of the integration with a rule-based approach will be to limit the number of CP variables and constraints to only the ones which are valid for a given request, so that the CP

solver can find a solution to the objective function within a few seconds. The advantage of the combination of the two approaches will be that the rule-based algorithm can provide a reduction of the solution space, while the CP solver is still used to find the best solution for the given objective function.

Aside of the runtime aspect, the CP model can be extended in future in various ways. The CP model uses a mono-objective function to minimise the cost. A multi-objective approach may take additional performance attributes into consideration and allow to maximise the service performance while providing the most cost-effective price. Such performance attributes could be gathered from monitoring data of the containers during runtime or be collected from IaaS benchmarking services like CloudHarmony [29]. The data model related the CP model is centred around host templates which represent virtual or physical servers. CSPs offer compute, storage and network resources today as independent services with various, flexible options for selection. The CP model may be extended to allow for better consumption and distinction of those services in the placement decisions. In addition, the CP model may be further extended to honour region specific prices, and optionally to take price differences for reservation, on-demand and spot instances into account. Further extension of the CP model can be done to support node clusters with multiple nodes and services with multiple containers.

With the proposed CP model, a first brokerage solution using service arbitrage for containers is provided. The underlying concepts were successful verified and allow for future research and development in this area.

References

1. What is Docker? https://www.docker.com/what-docker
2. Bond, J.: The Enterprise Cloud: Best Practices for Transforming Legacy IT. O'Reilly Media Inc., Newton (2015)
3. Amato, A., Di Martino, B., Venticinque, S.: Cloud brokering as a service. In: 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 9–16. IEEE (2013)
4. Park, J., An, Y., Yeom, K.: Virtual cloud bank: an architectural approach for intermediating cloud services. In: 2015 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 1–6. IEEE (2015)
5. Khanna, P., Jain, S., Babu, B.: BroCUR: distributed cloud broker in a cloud federation: brokerage peculiarities in a hybrid cloud. In: 2015 International Conference on Computing, Communication and Automation (ICCCA), pp. 729–734. IEEE (2015)
6. Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D.: NIST cloud computing reference architecture. NIST Spec. Publ. **500**(2011), 292 (2011)
7. Mostajeran, E., Ismail, B.I., Khalid, M.F., Ong, H.: A survey on sla-based brokering for inter-cloud computing. In: 2015 Second International Conference on Computing Technology and Information Management (ICCTIM), pp. 25–31. IEEE (2015)
8. Gartner: Cloud service brokerage. Gartner Research (2016). http://www.gartner.com/it-glossary/cloud-services-brokerage-csb

9. Gartner: Gartner says cloud consumers need brokerages to unlock the potential of cloud services. Gartner Research, July 2009. http://www.gartner.com/newsroom/id/1064712
10. Guzek, M., Gniewek, A., Bouvry, P., Musial, J., Blazewicz, J.: Cloud brokering: current practices and upcoming challenges. IEEE Cloud Comput. **2**, 40–47 (2015)
11. What is ITIL best practice? https://www.axelos.com/best-practice-solutions/itil/what-is-itil
12. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: taxonomy and survey. Softw. Pract. Exp. **44**(3), 369–390 (2014)
13. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. (CSUR) **47**(1), 7 (2014)
14. Barker, A., Varghese, B., Thai, L.: Cloud services brokerage: a survey and research roadmap. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), pp. 1029–1032. IEEE (2015)
15. Ngan, L.D., Tsai Flora, S., Keong, C.C., Kanagasabai, R.: Towards a common benchmark framework for cloud brokers. In: 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), pp. 750–754. IEEE (2012)
16. Tiwari, A., Nagaraju, A., Mahrishi, M.: An optimized scheduling algorithm for cloud broker using adaptive cost model. In: 2013 IEEE 3rd International Advance Computing Conference (IACC), pp. 28–33. IEEE (2013)
17. Masdari, M., Nabavi, S.S., Ahmadi, V.: An overview of virtual machine placement schemes in cloud computing. J. Netw. Comput. Appl. **66**, 106–127 (2016)
18. Lopez-Pires, F., Baran, B.: Virtual machine placement literature review. arXiv preprint arXiv:1506.01509 (2015)
19. Anastasi, G.F., Carlini, E., Coppola, M., Dazzi, P.: QBROKAGE: a genetic approach for QoS cloud brokering. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), pp. 304–311. IEEE (2014)
20. Kessaci, Y., Melab, N., Talbi, E.-G.: A Pareto-based genetic algorithm for optimized assignment of VM requests on a cloud brokering environment. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 2496–2503. IEEE (2013)
21. Srivastava, N.K., Singh, P., Singh, S.: Optimal adaptive CSP scheduling on basis of priority of specific service using cloud broker. In: 2014 9th International Conference on Industrial and Information Systems (ICIIS), pp. 1–6. IEEE (2014)
22. Amazon EC2 pricing. Amazon Web Services. https://aws.amazon.com/ec2/pricing/
23. Wang, W., Niu, D., Liang, B., Li, B.: Dynamic cloud instance acquisition via iaas cloud brokerage. IEEE Trans. Parallel Distrib. Syst. **26**(6), 1580–1593 (2015)
24. Liu, K., Peng, J., Liu, W., Yao, P., Huang, Z.: Dynamic resource reservation via broker federation in cloud service: a fine-grained heuristic-based approach. In: 2014 IEEE Global Communications Conference (GLOBECOM), pp. 2338–2343. IEEE (2014)
25. Nesmachnow, S., Iturriaga, S., Dorronsoro, B.: Efficient heuristics for profit optimization of virtual cloud brokers. IEEE Comput. Intell. Mag. **10**(1), 33–43 (2015)
26. Vieira, C., Bittencourt, L., Madeira, E.: A scheduling strategy based on redundancy of service requests on iaas providers. In: 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 497–504. IEEE (2015)
27. Pawluk, P., Simmons, B., Smit, M., Litoiu, M., Mankovski, S.: Introducing STRATOS: a cloud broker service. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 891–898. IEEE (2012)

28. Yangui, S., Marshall, I.-J., Laisne, J.-P., Tata, S.: CompatibleOne: the open source cloud broker. J. Grid Comput. **12**(1), 93–109 (2014)
29. Cloud Harmony - Transparency for the Cloud. Gartner, Inc. https://cloudharmony.com
30. Freuder, E.C., Mackworth, A.K.: Constraint satisfaction: an emerging paradigm. Handb. Constr. Program. **2**, 13–27 (2006)
31. Apt, K.: Principles of Constraint Programming. Cambridge University Press, Cambridge (2003)
32. Bockmayr, A., Hooker, J.N.: Constraint programming. Handb. Oper. Res. Manag. Sci. **12**, 559–600 (2005)
33. Zilci, B., Slawik, M., Kupper, A.: Cloud service matchmaking using constraint programming. In: 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 63–68. IEEE (2015)
34. Zhang, L., Zhuang, Y., Zhu, W.: Constraint programming based virtual cloud resources allocation model. Int. J. Hybrid Inf. Technol. **6**(6), 333–334 (2013)
35. Zhang, Y., Fu, X., Ramakrishnan, K.: Fine-grained multi-resource scheduling in cloud datacenters. In: 2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks (LANMAN), pp. 1–6. IEEE (2014)
36. Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E.K., Moatti, Y., Lorenz, D.H.: Guaranteeing high availability goals for virtual machine placement. In: 2011 31st International Conference on Distributed Computing Systems (ICDCS), pp. 700–709. IEEE (2011)
37. Dang, H.T., Hermenier, F.: Higher SLA satisfaction in datacenters with continuous VM placement constraints. In: Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, p. 1. ACM (2013)
38. BtrPlace - An Open-Source Flexible Virtual Machine Scheduler. http://www.btrplace.org
39. Van, H.N., Tran, F.D., Menaud, J.-M.: Performance and power management for cloud infrastructures. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 329–336. IEEE (2010)
40. Dupont, C., Giuliani, G., Hermenier, F., Schulze, T., Somov, A.: An energy aware framework for virtual machine placement in cloud federated data centres. In: 2012 Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), pp. 1–10. IEEE (2012)
41. NumberJack - A Python Contraint Programming platform. http://numberjack.ucc.ie
42. Mistral. http://homepages.laas.fr/ehebrard/mistral.html
43. The MiniSat Page. http://minisat.se
44. Forgy, C.L.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif. Intell. **19**(1), 17–37 (1982)

Fault Injection and Detection for Artificial Intelligence Applications in Container-Based Clouds

Kejiang Ye[1(✉)], Yangyang Liu[2], Guoyao Xu[1,3], and Cheng-Zhong Xu[1]

[1] Shenzhen Institutes of Advanced Technology,
Chinese Academy of Sciences, Shenzhen, China
{kj.ye,cz.xu}@siat.ac.cn
[2] School of Computer Science and Technology, Xidian University, Xi'an, China
liuyangyang@stu.xidian.edu.cn
[3] Department of Electrical and Computer Engineering,
Wayne State University, Detroit, USA
et8130@wayne.edu

Abstract. Container technique is increasingly used to build modern cloud computing systems to achieve higher efficiency and lower resource costs, as compared with traditional virtual machine technique. Artificial intelligence (AI) is a mainstream method to deal with big data, and is used in many areas to achieve better effectiveness. It is known that attacks happen every day in production cloud systems, however, the fault behaviors and interferences of up-to-date AI applications in container-based cloud systems is still not clear. This paper aims to study the reliability issue of container-based clouds. We first propose a fault injection framework for container-based cloud systems. We build a docker container environment installed with TensorFlow deep learning framework, and develop four typical attack programs, i.e., CPU attack, Memory attack, Disk attack and DDOS attack. Then, we inject the attack programs to the containers running AI applications (CNN, RNN, BRNN and DRNN), to observe fault behaviors and interferences phenomenon. After that, we design fault detection models based on quantile regression method to detect potential faults in containers. Experimental results show the proposed fault detection models can effectively detect the injected faults with more than 60% Precision, more than 90% Recall and nearly 100% Accuracy.

1 Introduction

Container technology is experiencing a rapid development with the support from industry like Google and Alibaba, and is being widely used in large scale production environments [1,2]. Container technology is also called operating-system-level virtualization, which allows multiple isolated user-space instances sharing the same operating system kernel and applies CGroups to take control of resources in the host. This provides functionality similar to a virtual machine

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 112–127, 2018.
https://doi.org/10.1007/978-3-319-94295-7_8

(VM) but with a lighter footprint. While traditional virtualization solutions (e.g., Hypervisor) need to interpose on various privileged operations (e.g., page-table lookups) and use roundabout techniques to infer resource usage (e.g., ballooning). The result is that hypervisors are heavyweight, with slow boot times as well as high run-time overheads.

Speedy launching time and tiny memory footprint are two outstanding features to make containers launch an application in less than a second and consume a very small amount of resources. Compared with virtual machines, adopting containers not only improves the performance of applications, but also allows hosts to sustain more applications simultaneously [3]. So far, there have been a number of container products released to the market, such as LXC (Linux Container) [4], Docker [5], rkt (Rocket) [6], and OpenVZ [7], etc. Docker [5] is a popular runtime system for managing Linux containers, providing both management tools and a simple file format. Docker technology introduces lightweight virtual machines on top of Linux containers called Docker containers. Major cloud platforms include Amazon Web Service [8] and Google Compute Engine [9] are also beginning to provide public container services for developers to deploy applications in the cloud. Undoubtedly, the emergence of container technology has virtually changed the trend of cloud computing market.

Deep learning is a new area of machine learning research. It is the application to learning tasks of artificial neural networks (ANNs) that contain more than one hidden layer. Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task specific algorithms. Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been widely applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation and bioinformatics, where they produced results comparable to or in some cases superior to human experts. TensorFlow [10] is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

There are already a lot of work focused on the performance issues of different applications running on containers [11–14]. However, the fault behavior and interference of container-based cloud is still not clear. In fact, reliability is very important and may cause serious consequences if faults are not timely handled, which not only hinders the runtime system, but also results in serious economic losses. For example, on February 28th 2017, Amazon Web Services, the popular hosting and storage platform used by a huge range of companies, experienced S3 service interruption for 4 hours in the Northern Virginia (US-EAST-1) Region, and then quickly spread other online service providers who rely on the S3 service [15]. This failure caused a large number of economic compensations for users. It is

because cloud computing service providers usually set a Service Level Agreement (SLA) with customers. For example, when customers require 99.99% availability, it means that 99.99% of the time must meet the requirement for 365 days per year. If the service breaks more than 0.01%, compensation is required.

Although reliability of cloud systems is important, it is not easy to solve this problem, mainly because: *(1) Large scale.* A typical data center involves more than 100,000 servers and 10,000 switches, more nodes usually mean higher probability of failure; *(2) Complex application structure.* Web search, e-commerce and other typical cloud programs have complex interactive behaviors. For example, an Amazon page request involves interactions with hundreds of components [16], error occurs in any one component will lead to anomalies of applications; *(3) Diverse causes.* Resource competition, resource bottlenecks, misconfiguration, defects of softwares, hardware failures and external attacks can cause anomalies or failures of cloud systems. Cloud systems are more prone to performance anomalies than traditional platforms [17].

In this paper, we focus on the reliability issue of container-based cloud. We first propose a fault injection framework for container-based cloud systems. We build a docker container environment installed with TensorFlow deep learning framework, and develop four typical attack programs, i.e., CPU attack, Memory attack, Disk attack and DDOS attack. Then, we observe fault behaviors and interferences phenomenon by injecting attack programs to the containers running artificial intelligence (AI) applications (CNN, RNN, BRNN and DRNN). We also design fault detection models based on quantile regression method to detect potential faults in containers.

Our main contributions are:

- We develop a *fault injection tool* as well as *four typical attack programs* for container-based cloud systems. The fault programs include CPU attack, memory attack, disk attack and network attack. The purpose of fault injection is to simulate abnormal behavior of containers.
- We investigate the *fault behaviors and interferences* between multiple containers. We focus on four mainstream AI applications: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Bidirectional Recurrent Neural Network (BRN) and Dynamic Recurrent Neural Network (DRNN).
- We design *fault detection models* for container-based cloud systems. The models are based on the quantile regression method. Part of the experimental data is used for training the models. Experimental results show that the proposed fault detection models can effectively detect the injected faults with more than 60% Precision, more than 90% Recall and nearly 100% Accuracy.

2 Fault Injection Framework

2.1 Fault Injection Framework

AI Applications in Target Systems. Our fault injection framework is mainly designed for container-based cloud systems (see Fig. 1), but it would be easily

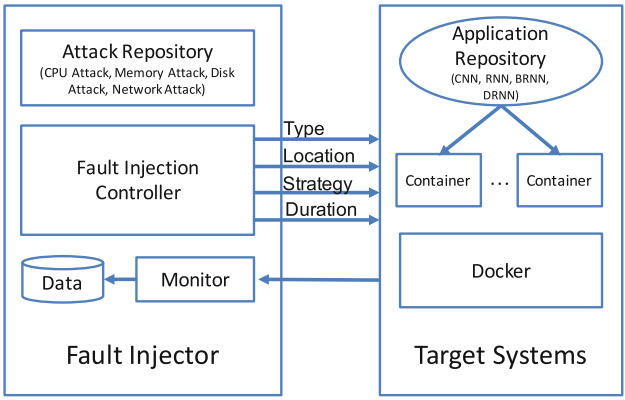


Fig. 1. Fault injection framework.

extended to other environments. We use docker to set up a target testing system and create several containers. The applications running in containers are the up-to-date AI applications based on tensorflow framework. We mainly focus on four types of typical AI applications:

- *Convolutional Neural Network (CNN):* is a class of deep, feed-forward artificial neural network that have been applied to analyzing visual imagery. Convolutional networks were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex. They have applications in image and video recognition, recommender systems and natural language processing.
- *Recurrent Neural Network (RNN):* is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.
- *Bidirectional Recurrent Neural Network (BRNN):* is introduced to increase the amount of input information available to the network. BRNNs do not require their input data to be fixed. Moreover, their future input information is reachable from the current state. The basic idea of BRNNs is to connect two hidden layers of opposite directions to the same output. By this structure, the output layer can get information from past and future states.
- *Dynamic Recurent Neural Network (DRNN):* contains architecturally both a huge number of feedforward and feedback synaptic connections between the neural units involved in the network and is mathematically a complex dynamic system described by a set of parameterized nonlinear differential or difference equations. The DRNNs have been proven to be capable of processing the time-varing spatio-temporal information represented by state space trajectories.

Attack Repository. In order to assess fault behaviors and interferences of container-based cloud systems from different dimensions, we develop four types of attack programs. Specifically, we use C language to develop the CPU, Memory, Disk and Network attack programs.

- *CPU attack program:* keeps CPU doing a lot of calculations and consuming a majority of CPU resources, which is used to simulate a competition scenario of CPU resources with other neighbor containers.
- *Memory attack program:* continuously allocates and consumes memory resources, which is used to simulate a competition scenario of memory resources with other neighbor containers.
- *DiskIO attack program:* uses IOzone benchmark to continuously read and write disk to consume almost all the disk bandwidth resources, which is used to simulate a competition scenario of disk bandwidth resources with other neighbor containers.
- *Network attack program:* uses third party DDOS clients to initiate a lot of invalid connections to the server, consuming all possible network bandwidth resources, which is used to simulate a competition scenario of network bandwidth resources with other neighbor containers.

Fault Injection Controller. Fault injection controller is responsible for triggering, injecting and activating attack programs. In most cases, the attack programs are not executing immediately, they would be triggered by user-defined conditions, such as the start time of execution. All the attack programs are selected from the *attack repository*, then been configured by users, and finally be injected to the target location, such as container, host, etc. The controlling parameters include *attack type, injection location, injection strategy, attack duration and other parameters.*

Monitor. Monitor is responsible for monitoring and collecting information of target systems. It is also responsible for checking if the attack program is activated, and how it affects the system. The detail monitored information includes *fault feedback data, container status information and performance data of applications running in containers, etc.*

2.2 DDOS Injection Method

Figure 2 shows the DDoS attack method in containers. We installed Apache Tomcat in the containers in host A, which provides web service through port 8080. Host B can access Tomcat service through port 8000 in host A, where port 8000 is mapped to the port 8080 in the container.

We use torshammer tool [18] to simulate the DDoS attack. Torshammer is a slow-rate DDoS attack tool that is efficient and very disruptive on most apache servers. Similar to regular volumetric DDoS attacks, slow-rate DDoS attacks exhaust web server resources by generating a large amount of connections as

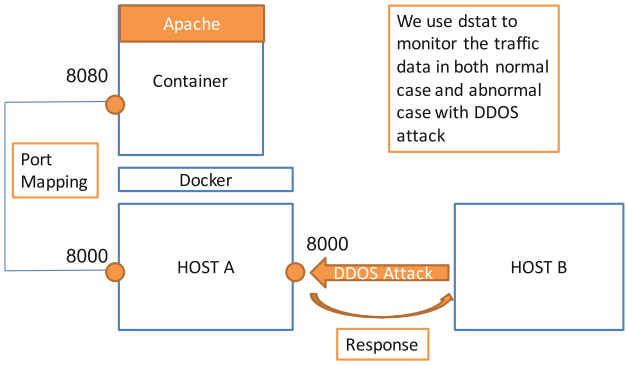


Fig. 2. The DDoS attack injection method.

long as possible. More technically, it uses the classic form of slow POST attack, generating HTTP POST requests and connections that will hold for around 1000–30000 s. Instead of leveraging huge attack bandwidth or large amounts of HTTP requests per second, slow-rate DDoS attacks simply exploit the maximum current connection time that apache servers can handle. We control the attack traffic through adjusting the attack threads. We use dstat tool [19] to collect the send and receive traffic.

3 Fault Behavior Analysis

In this section, we investigate fault behaviors and interferences by injecting different types of attacks in containers. We compared the performance of four up-to-date AI applications (CNN, RNN, BRNN and DRNN) before and after injecting those attacks.

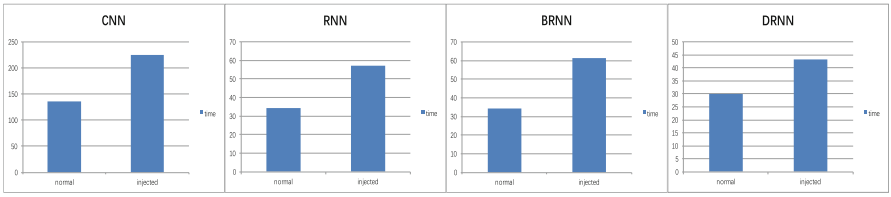


Fig. 3. Fault interference of four AI applications running in containers after injecting CPU attack.

3.1 Fault Interference Analysis

Figures 3, 4 and 5 show the fault interference of AI applications after injecting CPU, memory and disk attacks. It is very obvious that the performance

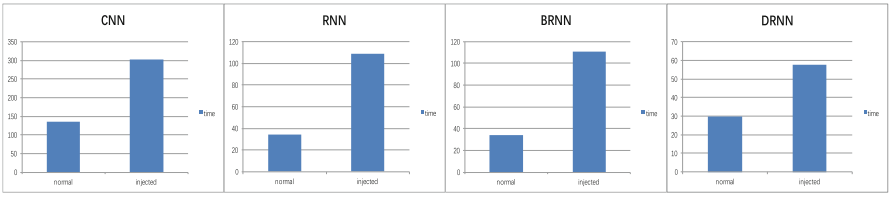


Fig. 4. Fault interference of four AI applications running in containers after injecting memory attack.

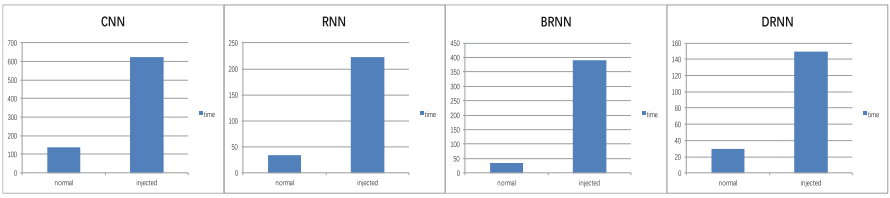


Fig. 5. Fault interference of four AI applications running in containers after injecting disk attack.

of AI applications is affected by varying degrees. For CPU attack, the running time of CNN, RNN, BRNN and DRNN is extended by 65.49%, 66.67%, 79.10%, and 44.89% respectively as compared with normal execution time. For memory attack, the running time is extended by 123.17%, 216.52%, 223.16%, and 93.31% respectively. For disk attack, the running time of is extended by 360.66%, 549.01%, 1045.43%, and 399.09% respectively.

From the experimental results, we observe: *(i) From the application perspective*, all the four AI applications are very sensitive to the attacks. The BRNN application is the most sensitive to the attacks while DRNN is least sensitive to the attacks. It is because BRNNs connects two hidden layers of opposite directions to the same output, which needs more resources to process the workloads. *(ii) From the attack perspective*, both memory and disk attack cause very serious degradation on application performance. Disk attacks causes most serious interference. It is because the applications are running on the distributed container cluster and need frequent data transfer, exchange, read and write. If disk bandwidth or memory space runs out, performance of applications would be certainly affected. *(iii) From the container perspective*, the fault isolation of containers is still not enough, which is although much better than traditional hypervisors like Xen or KVM. The CPU isolation is the best, memory isolation the second, and the disk isolation is the worst.

3.2 DDOS Attack Behavior Analysis

We further study DDoS attack behaviors in containers. DDoS attack is one of the most common network attacks in today's cloud systems. Figure 6 shows the

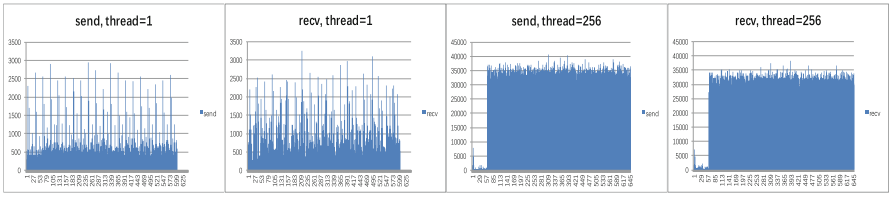


Fig. 6. Fault behavior of Tomcat web service after injecting DDoS attack in containers.

fault behaviors when triggering DDoS attack under different configurations of attack threads. We inject the DDoS attack at the 63th time points.

From the figure, we observe: (i) when the thread is set to 1, it didn't cause obvious changes on both send and receive traffic, indicating that the attack is similar or equivalent to an access to the server from a normal user. When the attack thread is set to 256, both the send and receive traffic suddenly increase to a very high degree, about more than 30 times higher than the original normal traffics. By doing so, the web server denies service for other normal users. (ii) According to the normal traffic analysis, we find even the normal traffic shows very fluctuating phenomenon, making it very challenging to accurately detect potential faults.

4 Fault Detection Models

In this section, we design several fault detection models based on the quantile regression method. Quantile regression is an optimization-based statistical method to model effects of factors on arbitrary quantiles, which has recently been successfully used for performance study in computer experiments [20]. We use this theory for anomaly detection. It is an extension of traditional regression method. The main difference is traditional regression builds a model on the *mean* of response variable, while quantile regression constructs one on any given *quantile* (having probabilistic sense), such as the median or the 99th percentile. Quantile regression has been successfully applied in various areas, such as ecology, healthcare and financial economics.

4.1 Metrics

The main objective of the fault detection method is to maximize the *Precision* and *Recall* of the proposed detection technique. The four common performance parameters for these objectives are *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)* and *False Negative (FN)* which are defined as follows:

True Positive (TP): This occurs when the detection model raises true alerts on a detected anomaly traffic.

True Negative (TN): This occurs when there is no anomaly activity taking place in the system, and the detection model is thus not raising any alarm.

False Positive (FP): This occurs when the detection model erroneously raises a false alarm over a legitimate activity in the system.

False Negative (FN): This occurs when the detection model fails to detect an anomaly activity taking place in the system.

To better understand the relationship between these four metrics, we give an intuitive judgment matrix in Table 1. We further define the metrics of Precision, Recall and Accuracy to quantify the detection efficiency.

Table 1. Confusion matrix for TP, TN, FP and FN.

	Real fault	Real not anomaly
Detected fault	True Positive	False Positive
Detected not anomaly	False Negative	True Negative

Precision is used to indicate the proportion of correct predictions, which represents the correctness of the prediction:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

Recall represents the percentage of actual failures that the model predicts in all occurrences of the environment, which represents the accuracy of the prediction:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

Accuracy indicates that the predicted model correctly determines the percentage of events in the total event:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

4.2 Detection Models for CPU, Memory and Disk Attacks

Figure 7 shows the fault detection results for the four AI applications (CNN, RNN, BRNN and DRNN) running on TensorFlow framework in containers. We set tau value of 0.1 as lower boundary for the fitting model, and tau value of 0.9 as the upper boundary of the fitting model. All the points in the interval are considered normal traffic, the remaining points are considered as abnormal traffic. In the figure, sequence 5, 10, and 15 are the points where the faults (CPU, memory and disk fault) are injected. From our quantile regression based detection model, we find these points are not within interval of upper and lower boundaries of the fitting model, so this method can detect the injected faults. Tables 2, 3, 4 and 5 show the four basis detection metrics TP, FP, FN, and TN,

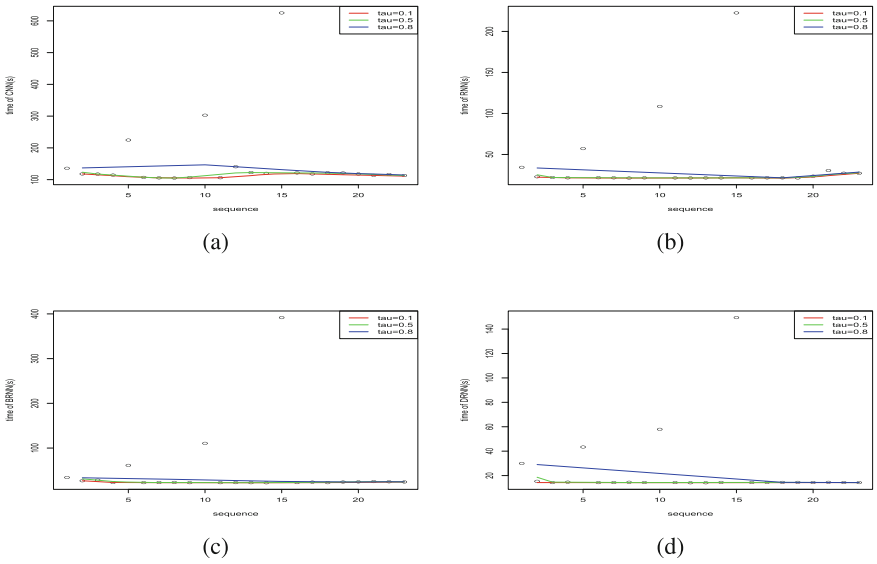


Fig. 7. Fault detection result for four AI applications: CNN, RNN, BRNN and DRNN.

Table 2. TP, TN, FP and FN for CNN.

	Real fault	Real not anomaly
Detected fault	3 (TP)	2 (FP)
Detected not anomaly	0 (FN)	18 (TN)

Table 3. TP, TN, FP and FN for RNN.

	Real fault	Real not anomaly
Detected fault	3 (TP)	2 (FP)
Detected not anomaly	0 (FN)	18 (TN)

Table 4. TP, TN, FP and FN for BRNN.

	Real fault	Real not anomaly
Detected fault	3 (TP)	1 (FP)
Detected not anomaly	0 (FN)	19 (TN)

Table 5. TP, TN, FP and FN for DRNN.

	Real fault	Real not anomaly
Detected fault	3 (TP)	1 (FP)
Detected not anomaly	0 (FN)	19 (TN)

Table 6. The Precision, Recall and Accuracy of the detection models for different AI applications.

	Precision	Recall	Accuracy
CNN application	60%	100%	91.30%
RNN application	60%	100%	91.30%
BRNN application	75%	100%	95.65%
DRNN application	75%	100%	95.65%

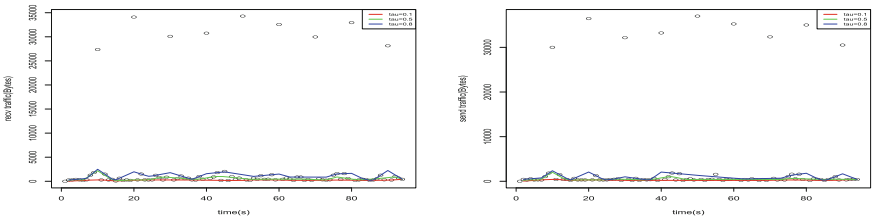
based on which, we calculate the Precision, Recall and Accuracy of the detection models in Table 6.

From the above experimental results, it can be seen that the fault detection method based on the quantile regression can effectively detect the potential faults. For CNN and RNN fault detection model, the Precision is 60%, Recall is 100%, and Accuracy is 91.30%. For BRNN and DRNN fault detection model, the Precision is 75%, Recall is 100%, and Accuracy is 95.65%. We find the Precision is relatively low, it is because the number of fault points in the experiment is very small (3 points) which affects the detection Precision. But if we look at the Accuracy, it is nearly 100% because the number of sampling data is relatively large (23 points). We plan to increase the testing scale in the next step of our work.

4.3 Detection Models for DDoS Attack

DDOS fault injection detection model needs two sets of data, namely, the receive traffic and send traffic of the Apache web server. In the experiment, we uses dstat tool to collect traffic information. Traffic data is a typical time series type of data, making it ideal for processing using time series models. In the experiment, the quantile regression method is used to process the traffic data in the time series. We totally calculate three groups of fitting values, named fit1, fit2 and fit3, the tau value in each set is set as 0.1, 0.5 and 0.8 respectively. The tau value of 0.1 indicates the lower boundary of the receive or send traffic, tau value of 0.5 indicates the average value of the receive or send traffic, and the tau value of 0.8 indicates the upper boundary of the receive or send traffic. Based on the trained model, we can detect potential DDoS attacks.

Figure 8(a) and (b) shows detection result for both receive and send traffic of Apache Tomcat server under DDoS attacks. In the figure, we inject DDoS attack at point 10, 20, 30, 40, 50, 60, 70, 80, and 90. All the traffic point inside this range is determined as the normal traffic, while the traffic points outside the range means potential DDOS attack. From the figure, we find the traffic points with attacks are obviously outside the confidential interval of the fitting model, which means the fitting model fits the normal traffic data well and can be used to accurately detect potential DDOS attacks.



(a) (b)

Fig. 8. Fault detection result for DDoS attacks.

Table 7. TP, TN, FP and FN for the recv traffic.

	Real fault	Real not anomaly
Detected fault	9 (TP)	0 (FP)
Detected not anomaly	0 (FN)	85 (TN)

Table 8. TP, TN, FP and FN for the send traffic.

	Real fault	Real not anomaly
Detected fault	9 (TP)	1 (FP)
Detected not anomaly	0 (FN)	84 (TN)

Table 9. The Precision, Recall and Accuracy of the detection model for DDoS recv traffic and send traffic.

	Precision	Recall	Accuracy
Receive traffic	100%	100%	100%
Send application	90%	100%	98.94%

Table 9 shows precision, recall and accuracy results of the detection model for DDoS recv traffic and send traffic, which is based on the results of TP, TN, FP and FN as shown in Tables 7 and 8. For the receive traffic detection model, the Precision is 100%, Recall is 100% and Accuracy is also 100%. For the send traffic detection model, the Precision is 90%, Recall is 100% and Accuracy is 98.94%, indicating our test models perform very well.

5 Related Work

Recently, there is an increasing interest in the performance and reliability issues of containers.

Performance. Felter et al. [3] explored the performance of traditional virtual machine (VM) deployments, and compared them with the use of Linux

containers. They used KVM as a representative hypervisor and Docker as a container manager. The results showed that containers result in equal or better performance than VMs in most cases. Both VMs and containers require tuning to support I/O intensive applications. Ruan et al. [21] conducted a series of experiments to measure performance differences between application containers and system containers and find system containers are more suitable to sustain I/O-bound workload than application containers. Ye and Ji [13] proposed a performance model to predict the application performance running in containers. Higgins et al. [12] evaluated the suitability of Docker containers as a runtime for high performance parallel execution. Their findings suggest that containers can be used to tailor the run time environment for an MPI application without compromising performance. Yu et al. [14] designed a flexible container-based tuning system (FlexTuner) allowing users to create a farm of lightweight virtual machines (containers) on host machines. They mainly focus on the impact of network topology and routing algorithm on the communication performance of big data applications. Harter et al. [11] proposed Slacker, a new Docker storage driver optimized for fast container startup. Docker workers quickly provision container storage using backend clones and minimize startup latency by lazily fetching container data.

Reliability. Duffield et al. [22] proposed a rule-based anomaly detection on the IP network which correlates the packet and flow level information. Cherkasova et al. [23] presented an integrated framework of using regression based transaction models and application performance signatures to detect anomalous application behavior. Sharma et al. [24] used the Auto-Regressive models and a time-invariant relationships based approach to detect the fault. Pannu et al. [25] presented an adaptive anomaly detection framework that can self adapt by learning from observed anomalies at runtime. Tan et al. presented two anomaly prediction systems PREPARE [26] and ALERT [27] that integrate online anomaly prediction, learning-based cause inference, and predictive prevention actuation to minimize the performance anomaly penalty without human intervention. They also investigated the anomalous behavior of three datasets [28]. Bronevetsky et al. [29] designed a novel technique that combine classification algorithms with information on the abnormality of application behavior to improve detection. Gu et al. [30] developed an attack detection system LEAPS based on supervised statistical learning to classify benign and malicious system events. Besides, Fu and Xu [31] implemented a failure prediction framework PREdictor that explores correlations among failures and forecasts the time-between-failure of future instances. Nguyen et al. [32] presented a black-box online fault localization system called FChain that can pinpoint faulty components immediately after a performance anomaly is detected. Most recently, Arnautov et al. [33] described SCONE, a secure container mechanism for Docker that uses the SGX trusted execution support of Intel CPUs to protect container processes from outside attacks.

In comparison, we study fault behaviors and interferences of artificial intelligence applications on Docker containers. We also propose fault detection models based on quantile regression method to accurately detection the faults in containers.

6 Conclusion

In this paper, we studied the reliability issue of container-based cloud. We first proposed a fault injection framework for container-based cloud systems. We built a docker container environment installed with TensorFlow deep learning framework, and developed four typical attack programs, i.e., CPU attack, Memory attack, Disk attack and DDOS attack. Then, we injected the attack programs to the containers running typical artificial intelligence applications (CNN, RNN, BRNN and DRNN), to observe the fault behavior and interference phenomenon. After that, we designed different fault detection models based on quantile regression method to detect potential faults in containers. Experimental results show that the proposed fault detection models can effectively detect the injected faults with more than 60% Precision, more than 90% Recall and nearly 100% Accuracy. Note that, in this paper we verified that quantile regression based fault detection model can efficiently detect potential faults, but the experimental scale is still very small. In the future, we plan to expand the experimental scale and continue to test our approach in more complex environments.

Acknowledgments. This work is supported by the National Basic Research Program of China (Grant No. 2015CB352400), National Natural Science Foundation of China (Grant No. 61702492, U1401258), and Shenzhen Basic Research Program (Grant No. JCYJ20170818153016513, JCYJ20170307164747920).

References

1. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at Google with Borg. In: Proceedings of the Tenth European Conference on Computer Systems, p. 18. ACM (2015)
2. Lu, C., Ye, K., Xu, G., Xu, C.-Z., Bai, T.: Imbalance in the cloud: an analysis on Alibaba cluster trace. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 2884–2892. IEEE (2017)
3. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171–172. IEEE (2015)
4. Linux Containers (2017). https://linuxcontainers.org/
5. Docker (2017). https://www.docker.com/
6. Rkt (2017). https://github.com/rkt/rkt
7. OpenVZ (2017). https://openvz.org/
8. Amazon EC2 Container Service (2017). https://aws.amazon.com/docker/
9. Containers on Google Compute Engine (2017). https://cloud.google.com/compute/docs/containers/
10. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: TensorFlow: a system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283 (2016)
11. Harter, T., Salmon, B., Liu, R., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Slacker: fast distribution with lazy Docker containers. In: FAST, pp. 181–195 (2016)

12. Higgins, J., Holmes, V., Venters, C.: Orchestrating Docker containers in the HPC environment. In: Kunkel, J.M., Ludwig, T. (eds.) ISC High Performance 2015. LNCS, vol. 9137, pp. 506–513. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20119-1_36
13. Ye, K., Ji, Y.: Performance tuning and modeling for big data applications in Docker containers. In: 2017 12th IEEE International Conference on Networking, Architecture, and Storage (NAS 2017). IEEE (2017)
14. Yu, Y., Zou, H., Tang, W., Liu, L., Teng, F.: Flex tuner: a flexible container-based tuning system for cloud applications. In: 2015 IEEE International Conference on Cloud Engineering (IC2E), pp. 145–154. IEEE (2015)
15. Summary of the Amazon s3 Service Disruption in the Northern Virginia (us-east-1) Region (2017). https://aws.amazon.com/message/41926/
16. Veeraraghavan, K., et al.: Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In: OSDI (2016)
17. Gunawi, H.S., et al.: Why does the cloud stop computing? Lessons from hundreds of service outages. In: SoCC (2016)
18. Tor's Hammer: A Slow Post DOS Testing Tool (2017). https://sourceforge.net/projects/torshammer/
19. Dstat: Versatile Resource Statistics Tool (2017). https://github.com/dagwieers/dstat
20. De Oliveira, A.B., Fischmeister, S., Diwan, A., Hauswirth, M., Sweeney, P.F.: Why you should care about quantile regression. In: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 207–218. ACM (2013)
21. Ruan, B., Huang, H., Wu, S., Jin, H.: A performance study of containers in cloud environment. In: Wang, G., Han, Y., Martínez Pérez, G. (eds.) APSCC 2016. LNCS, vol. 10065, pp. 343–356. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49178-3_27
22. Duffield, N., Haffner, P., Krishnamurthy, B., Ringberg, H.: Rule-based anomaly detection on IP flows. In: IEEE International Conference on Computer Communications (INFOCOM), pp. 424–432. IEEE (2009)
23. Cherkasova, L., Ozonat, K., Mi, N., Symons, J., Smirni, E.: Anomaly? Application change? Or workload change? Towards automated detection of application performance anomaly and change. In: 2008 IEEE International Conference on Dependable Systems and Networks (DSN), pp. 452–461. IEEE (2008)
24. Sharma, A.B., Chen, H., Ding, M., Yoshihira, K., Jiang, G.: Fault detection and localization in distributed systems using invariant relationships. In: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–8. IEEE (2013)
25. Pannu, H.S., Liu, J., Fu, S.: Aad: adaptive anomaly detection system for cloud computing infrastructures. In: 2012 IEEE 31st Symposium on Reliable Distributed Systems (SRDS), pp. 396–397. IEEE (2012)
26. Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., Rajan, D.: Prepare: predictive performance anomaly prevention for virtualized cloud systems. In: 2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS), pp. 285–294. IEEE (2012)
27. Tan, Y., Gu, X., Wang, H.: Adaptive system anomaly prediction for large-scale hosting infrastructures. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), pp. 173–182. ACM (2010)

28. Tan, Y., Gu, X.: On predictability of system anomalies in real world. In: 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 133–140. IEEE (2010)
29. Bronevetsky, G., Laguna, I., De Supinski, B.R., Bagchi, S.: Automatic fault characterization via abnormality-enhanced classification. In: 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–12. IEEE (2012)
30. Gu, Z., Pei, K., Wang, Q., Si, L., Zhang, X., Xu, D.: Leaps: detecting camouflaged attacks with statistical learning guided by program analysis. pp. 57–68 (2015)
31. Fu, S., Xu, C.-Z.: Exploring event correlation for failure prediction in coalitions of clusters. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC), pp. 1–12. IEEE (2007)
32. Nguyen, H., Shen, Z., Tan, Y., Gu, X.: Fchain: toward black-box online fault localization for cloud systems. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS), pp. 21–30. IEEE (2013)
33. Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., OKeeffe, D., Stillwell, M.L., et al.: Scone: Secure Linux containers with Intel SGX. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2016)

Container-VM-PM Architecture: A Novel Architecture for Docker Container Placement

Rong Zhang[1,2], A-min Zhong[1,2], Bo Dong[1], Feng Tian[1,2(✉)], and Rui Li[1]

[1] Shaanxi Province Key Laboratory of Satellite and Terrestrial Network Tech. R&D, Xi'an Jiaotong University, Xi'an 710049, China
fengtian@mail.xjtu.edu.cn

[2] School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, People's Republic of China

Abstract. Docker is a mature containerization technique used to perform operating system level virtualization. One open issue in the cloud environment is how to properly choose a virtual machine (VM) to initialize its instance, i.e., container, which is similar to the conventional problem of VM placement towards physical machines (PMs). Current studies mainly focus on container placement and VM placement independently, but rarely take into consideration of the two placements' systematic collaboration. However, we view it as a main reason for scattered distribution of containers in a data center, which finally results in worse physical resource utilization. In this paper, we propose a definition named "**Container-VM-PM**" architecture and propose a novel container placement strategy by simultaneously taking into account the three involved entities. Furthermore, we model a fitness function for the selection of VM and PM. Simulation experiments show that our method is superior to the existing strategy with regarding to the physical resource utilization.

Keywords: Docker container · Virtual machine · Resource fragment
Three-tier architecture

1 Introduction

Currently, studies in cloud computing mainly focus on the placement of container to Virtual Machine (VM) and the placement of VM to Physical Machine (PM) [1–3], which we denote as "Container-VM" architecture and "**VM-PM**" architecture respectively, shown in Fig. 1(a) and (b). They only consider the resource relation between two objects, so we call these architectures as **two-tier** architecture. Technically, when a new Docker container is created, the Swarm master will choose a node to host it, which has the highest ranking under a given strategy [4, 5]. Provided that the node is a VM, the decision making mechanism only concerns the resource relationship between the Docker container and VM. Moreover, when a new VM needs to be created, the OpenStack nova filter scheduler selects correspond PMs according to a given weighting functions [6, 7]. The new VM placement decision making thus only concerns the resource relationship between the new VM and PMs, which ignores the

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 128–140, 2018.
https://doi.org/10.1007/978-3-319-94295-7_9

resource request from the real workload, i.e. Docker container. In short, Swarm master and OpenStack nova just like two separate divisions and there is no uniform operation, currently. In other words, both **Container-VM** architecture and **VM-PM** architecture are autonomous working; they do not systematically considering the resource relationships between container, VM and PM as a whole. For convenience, we name this combination of the two autonomous working **two-tier** architectures as **double two-tier "Container-VM add VM-PM" (CV_VP)** architecture, shown in Fig. 1(c).

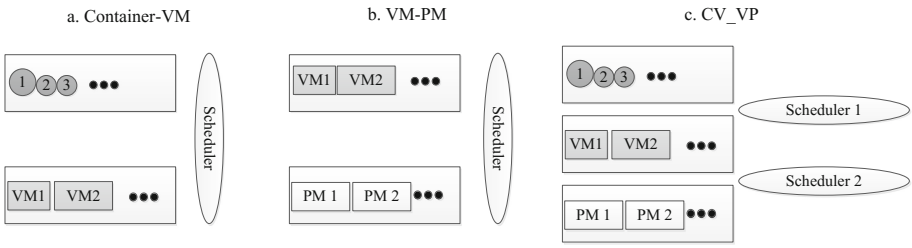


Fig. 1. Three different types of architecture.

In fact, current studies in this field rarely consider the potential influence caused by physical servers. When placing new containers, it needs to choose suitable VMs to host them. If the selection algorithm only considers the "**Container-VM**" architecture, it is easy to cause containers in a scattered distribution situation. As showed in Fig. 2, the small circles with different sizes in a queue represent the New coming Docker containers, the small circle with white color and a dotted line represents the New Docker container which is under the placement decision making process, the dotted lines with arrows indicates that they are feasible to place. The new Docker container placed in VM1, VM2 or VM4 can make full use of their resource; the placement makes the same impacts to the three VMs, but different impacts in physical resource utilization to the Swarm system. If the new Docker container is no placed in VM1, then VM1 is no-load and can be cancel or remove from PM1, therefore PM1 is no-load and can be shut down or turn into sleep model. In fact, the selection algorithm which only considers the "**Container-VM**" architecture cannot ensure no to choose VM1 as the target VM. As a result, this may lead to use more active PMs for the placement and worse resource utilization.

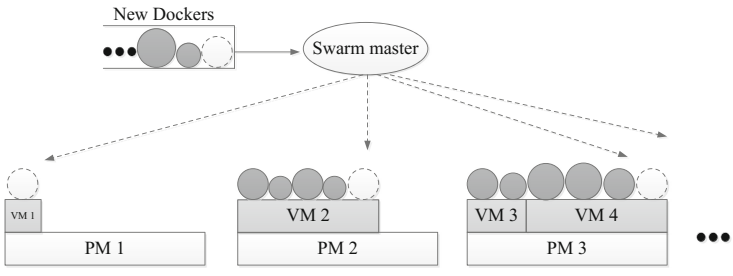


Fig. 2. An example of Docker container placement.

To solve this problem, the resource relationships between container, VM and PM should be systematically considered as a whole, we denote their collaborative operating architecture as a **three-tier** "**Container-VM-PM**" (**CVP**) architecture, shown in Fig. 3.

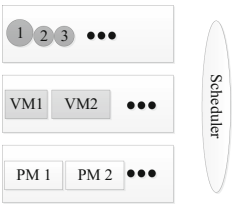


Fig. 3. The CVP architecture.

However, current studies rarely consider the three as a whole, there is very little literature about this concept. Tchana et al. [8] proposed the implementation of the software consolidation based on the assumption that Docker is used for software isolation and the same VM be shared between several soft applications. The authors want to extend analysis of best coordinate software consolidation on VMs with VM consolidation on physical machines in their future work. Their idea involves **CVP** architecture which coincides with ours, although it is their future work. Still, it inspires us. In this paper, we try to solve the new Docker container placement problem under the **CVP** architecture, taking the minimum active PMs as the first goal and maximum resource utilization as the second goal.

Due to the collaborative operations, the placement algorithm towards new Docker containers is more complicated under the **CVP** architecture, which raises three challenges:

- Which PM should be chosen?
- Which VM should be chosen?
- How to coordinate the placement of Docker containers to VMs, considering the placement of VMs to PMs simultaneously.

We propose a container placement strategy under **CVP** architecture and make a fitness model to select the optimal VMs and PMs. We also leverage the Best-fit algorithm to search the optimal mapping. Finally, we derive the mapping relationship between container, VM and PM. To the best of our knowledge, this work is the first that systematically considers the resource relationships between container, VM and PM and coordinates the placement of Docker containers to VMs with the placement of VMs to PMs simultaneously.

Our main contributions are as follows:

(1) We focus on the Docker container placement problem in order to improve the physical resource utilization.
(2) We systematically consider the resource relationships between container, VM and PM as a whole and extend the study from **two-tier** architecture to a **three-tier** "**Container-VM-PM**" architecture.

(3) We propose a Docker container placement strategy under the **three-tier** "**Container-VM-PM**" architecture and establish a VM and PM selection model.

The rest of this paper is organized as follows. Related works and some important definitions are reviewed in Sect. 2. In Sect. 3, we describe the key problem and our proposed method. Furthermore, we propose an evaluation formula for the placement performance comparison. Experiments and results are showed in Sect. 4. Conclude and future work in Sect. 5.

2 Background

In this section, we introduce the concept of Docker container and Virtual Machine. More concretely, we first compare the pros and cons of the two techniques and review their related research in the literature. Then, we introduce the definitions of resource utilization and resource fragmentation. Last, we introduce the Best-fit algorithm that will adopt.

2.1 Docker Container and Virtual Machine

Docker [9], an open sourced project, provides a lightweight virtualization mechanism at system level and automates a faster deployment of Linux applications by extending a common container format called Linux Containers (LXC) [10]. It can be deployed on VMs or "bare metal" physical servers. In most cases, a container can be created and destroyed in almost real-time and introduces negligible performance overhead towards CPU and memory [11]. It thus is suitable to quickly build and restructure application services. Compared with VMs, containers have advantages of fast deployment and migration, but they also suffer from weak isolation and security. Therefore, many cloud service providers take a compromise solution by combining the two techniques, i.e., they place containers in VMs. As cloud business services thrive in recent years, the requirement for efficiently placing containers while maximizing resource utilization has attracted a wide range of attention [12].

VMs are widely used in cloud computing. Various kinds of service models are offered by Cloud computing service, there are three kinds of service models that have been popular accepted: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [13]. Many enterprises use their cloud platforms to make VMs available to the end user and even run services inside VMs. One of extensively used open source cloud platform is OpenStack. Nova is an important compute component of OpenStack. OpenStack Nova is responsible for provisioning and management of VMs [14]. The detail information about its scheduling strategy is referring to [15, 16].

Many Platform providers built their PaaS and SaaS on IaaS, and run all the workload inside VMs. Swarm plays an important role in PaaS and SaaS, which is used to manage the Docker cluster. Scheduling strategy on Swarm can be introduced from [17, 18] in detail.

2.2 Resource Utilization

Both the Container placement problem under **Container-VM** architecture and the VM placement problem under **VM-PM** architecture can be described as a bin packing problem. For this reason, we can transform the **Container-VM** architecture as Fig. 4(a), and the **VM-PM** architecture could be showed as Fig. 4(b).

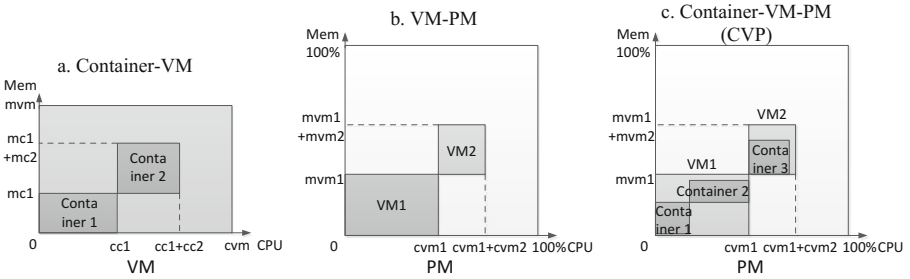


Fig. 4. Three different types of architecture in two-dimension.

As shown in the Fig. 4(a), the real workloads are Container 1 and Container 2. Although the VM requests for *cvm* CPU and *mvm* Memory resources, but the effective resources utilization are *(cc1 + cc2)* CPU and *(mc1 + mc2)* Memory. As shown in the Fig. 4(b), it only concerned about the resources of VM and ignores the container layer. The physical resource utilization is calculated by summing resources occupied by the VMs hosted by the PM.

The **CVP** architecture can be transformed as Fig. 4(c). It systematically considers the resource relationships between container, VM and PM as a whole. Because of the real workloads are containers, the effective resources utilization are calculated by the resources of containers. As a result, the physical resource utilization is calculated by summing resources occupied by the containers run in the PM. As the resource information of the three layers is clear, it can help to place new Docker containers more effectively.

2.3 Resource Fragmentation

Different containers need to occupy different dimensions of resources. A node can host a container when it meets the container's resources requirement in every dimension. After new container placement, the unused resources are called Resource Fragmentation [17, 19].

If the resource fragmentations are too small, it may unable to accept more new objects' placement. So, the size of resource fragmentations is the bigger the better in the case of using same amount of active PMs.

2.4 Best-Fit Algorithm

Best-fit algorithm is a fast allocation algorithm and it is commonly used in solving the bin packing problem. For an upon arrival item, Best-fit algorithm chooses a fullest bin, in which it fits from a list of current bins to host the item; in case this process fails, a new bin for the item will be opened [20]. As a result, when Best-fit algorithm is used in the placement of Docker container to VM, we treat Docker containers as the items and treat VMs as bins. Similarly, when Best-fit algorithm is used in the placement of VM to PM, we treat VMs as the items and treat PMs as bins. Furthermore, resource fragmentation is used to evaluate the bin in order to judge if it is the fullest one or not.

3 Methods and Problem Formulation

The key in the new Docker container placement problem is how to choose the suitable VM and PM. In this section, we will give a new Docker container placement strategy referring to the placement strategies in [8, 9]. As it's different in the view point of resource relations between **CV_VP** and **CVP** architecture, the VM and PM selection strategies are different too. We thus model the VM and PM selection fitness function for **CV_VP** and **CVP** architecture respectively. Furthermore, we propose an evaluation formula for the placement performance comparison.

To simplicity and clarity, we make the following assumptions:

(1) Each Docker container should be hosted by one VM, and several Docker containers can be placed in a same VM.
(2) All PMs have the same configuration.
(3) Only focus on two-dimension resources, they are CPU and Memory.

Considering taking the minimum active PMs as the first goal and maximum resource utilization as the second goal, we propose a placement strategy for the new Docker container and the VM and PM selection strategy for the two architectures, respectively. The notations used in this paper are described in Table 1.

3.1 The Placement Strategy Under CV_VP Architecture

When dealing with a new Docker container, the Docker container should be placed in the current existing VMs preferentially. There are two situations that need to be considered.

(1) If there does not exists any VM that can host the new Docker container, the smallest VM which can host the new Docker container should be selected from the VM provision set as the new VM. This new VM is the target VM. Certainly we need to choose a suitable PM to host the new VM. There are two possible scenarios.
One scenario: Suppose that there are m active PMs which can host the new VM, where $m > 0$. Because of the **CV_VP** architecture, the PM selection is only depending on the new VM's resource requests. For each PM in the m active PMs Set, we take the sum of its CPU and memory resources utilization after the new

Table 1. Notations.

Symbol	Description
cpu_j^{Vused}	The sum of the VMs' CPU resource in the j^{th} active PM
mem_j^{Vused}	The sum of the VMs' Memory resource in the j^{th} active PM
cpu_j^{pm}	The total CPU resource of the j^{th} active PM
mem_j^{pm}	The total Memory resource of the j^{th} active PM
cpu_i^{Dused}	The sum of the Docker containers' CPU resource in the i^{th} VM
mem_i^{Dused}	The sum of the Docker containers' Memory resource in the i^{th} VM
cpu_i^{vm}	The total CPU resource of the i^{th} VM
mem_i^{vm}	The total Memory resource of the i^{th} VM
cpu_{new}^{Docker}	The CPU resource that the new Docker container request for
mem_{new}^{Docker}	The Memory resource that the new Docker container request for
cpu_j^{Dused}	The sum of the Docker containers' CPU resource in the j^{th} active PM
mem_j^{Dused}	The sum of the Docker containers' Memory resource in the j^{th} active PM
N_{PM}^{active}	The number of active PM
cpu_j^{remain}	The remain CPU resources of the j^{th} active PM
mem_j^{remain}	The remain Memory resources of the j^{th} active PM

VM placement as its fitness. So, the fitness function $PAfit(j)$ for the j active PM can be defined as:

$$PAfit(j) = \frac{cpu_j^{Vused} + cpu_{new}^{vm}}{cpu_j^{pm}} * \alpha + \frac{mem_j^{Vused} + mem_{new}^{vm}}{mem_j^{pm}} * \beta, \tag{1}$$

where, $j = 1, \cdots, m$, α and β are two constants, represent the two parts' weights of the formula. At last, the PM which has the max fitness will be chosen to host the new VM. This PM will be the target PM.

The other scenario: Suppose that there does not exist any active PM that can host the new VM, then an idle PM should be start. The new active PM will be the target PM.

(2) If there exists n VMs that can host the new Docker, where $n > 0$. Because of the **CV_VP** architecture, the VM selection is only depending on the new Docker's resource requests. For each VM in the n exist VMs Set, we take the sum of its CPU and memory resources utilization after the new Docker container placement as its fitness. So, the fitness function $VAfit(i)$ for the VM i can be defined as:

$$VAfit(i) = \frac{cpu_i^{Dused} + cpu_{new}^{Docker}}{cpu_i^{vm}} * \alpha + \frac{mem_i^{Dused} + mem_{new}^{Docker}}{mem_i^{vm}} * \beta, \tag{2}$$

where, $i = 1, \cdots, n$. At last, the VM which has the max value of *VAfit* will be chosen to host the new Docker container. This VM will be the target VM, and the PM which hosts this VM will be the target PM.

3.2 The Placement Strategy Under CVP Architecture

There are two situations that need to be considered too, but some parts of the calculation are different from **CV_VP** architecture.

(1) If there does not exists any VM that can host the new Docker, the smallest VM which can host the new Docker should be selected from the VM provision set as the new VM. This new VM is the target VM. Certainly we need to choose a suitable PM to host the new VM. There are two possible scenarios.
One scenario: Suppose there are m active PMs which can host the new VM, where $m > 0$. We define the fitness function $PBfit(j)$ for the j active PM as:

$$PBfit(j) = \frac{cpu_j^{Dused} + cpu_{new}^{Docker}}{cpu_j^{pm}} * \alpha + \frac{mem_j^{Dused} + mem_{new}^{Docker}}{mem_j^{pm}} * \beta, \tag{3}$$

where, $j = 1, \cdots, m$. At last, the PM which has the max value of PBfit will be chosen to host the new Docker container. This PM will be the target PM.
The other scenario: Suppose there does not exist any active PM that can host the new VM, then an idle PM should be start. The new active PM will be the target PM.

(2) If there exists n VMs that can host the new Docker, where $n > 0$. The VM selection is not only depending on the new Docker's resource requests but also depending on the real resource utilizations because of the **CVP** architecture. The placement of a new container should better improve the resource utilization of the target PM and the target VM. So, the fitness function $VBfit(i)$ for the VM i can be defined as:

$$VBfit(i) = \frac{cpu_i^{used} + cpu_{new}^{Docker}}{cpu_i^{vm}} * \alpha + \frac{mem_i^{used} + mem_{new}^{Docker}}{mem_i^{vm}} * \beta + PBfit(k) * \delta_{ik}, \tag{4}$$

$$i = 1, \cdots, n, \delta_{ik} = \begin{cases} 1, & \textit{if PM k hosts VM i} \\ 0, & \textit{the other} \end{cases},$$

where, the first part and the second part of the fitness function are used to represent the new Docker container placement impact on VM resources utilization, the third part represent the impact on PM resources utilization. At last, the VM which has the max value of *VBfit* will be chosen to host the new Docker container. This VM will be the target VM, and the PM which hosts this VM will be the target PM.

3.3 Evaluation Model

Because of the requested resource by a VM is not fully utilized in most situations, it is more accurate to calculate the real resource utilization by the real workloads' resource

requests. The real workloads are Docker containers, so the evaluation formula ***Escore*** can be given as follows:

$$\boldsymbol{Escore} = \alpha_E * N_{PM}^{\text{active}} + \beta_E * \sum\nolimits_j \left(cpu_j^{remain} * mem_j^{remain} \right), \tag{5}$$

$$cpu_j^{remain} = cpu_j^{PM} - cpu_j^{Dused}, \tag{6}$$

$$mem_j^{remain} = mem_j^{PM} - mem_j^{Dused}, \tag{7}$$

where, α_E and β_E are two constants, represent the two parts' weights of ***Escore***. $\alpha_E > 0 > \beta_E$ and $\alpha_E > > |\beta_E|$. The first part of ***Escore*** is to make sure that taking the least active PMs as the first goal, and the second part is used to distinguish the size of resource fragmentations when using the same amount of active PMs. The evaluation score is the small the better.

4 Experiments and Results

In this section, a method is introduced to generate Docker container Synthetic Instances and 16 types of VM instance from Amazon EC2 are constituted as our VM provision set. The experiment simulates to deal with 9 batches of new Docker containers' placement consecutively. Finally we evaluate the physical resource utilization between **CV_VP** and **CVP** architecture after the new Docker container placement.

4.1 Docker Container Synthetic Dataset

A method is introduced to generate Docker container Synthetic Instances from [21, 22] into this paper. The method is showed in Algorithm 1.

Algorithm 1 Generation of Docker container Synthetic Instances

Input: n (the number of Docker container)
Output: $< u_i^{cpu}, u_i^{mem} >$ Set
for i=1 to n **do**
  $u_i^{cpu} = 2 * rand(\overline{u^{cpu}})$;
  $u_i^{mem} = rand(\overline{u^{mem}})$;
  $r = rand(1)$;
  if $((r < p) \wedge (u_i^{cpu} \geq \overline{u^{cpu}}) \vee (r \geq p) \wedge (u_i^{cpu} < \overline{u^{cpu}}))$ **then**
    $u_i^{mem} = u_i^{mem} + \overline{u^{mem}}$;
  end if
end for
return $< u_i^{cpu}, u_i^{mem} >$ Set

where, $\boldsymbol{rand}(\boldsymbol{a})$ will return a random number in the range $[0, \boldsymbol{a})$; $\overline{\boldsymbol{u^{cpu}}}$ and $\overline{\boldsymbol{u^{mem}}}$ are represent the reference CPU and Memory utilization respectively. The probability P can be used to control the correlation of CPU and Memory utilizations.

4.2 VM Dataset

As outlined in Table 2, 16 types of VM instance from Amazon EC2 are imported as our VM provision set.

Table 2. VM instance types from Amazon EC2.

VM instance type	vCPU	Memory (GiB)
t2.micro	1	1
t2.small	1	2
t2.medium	2	4
t2.large	2	8
m3.medium	1	3.75
m3.large	2	7.5
c4.large	4	7.5
r4.xlarge	4	30.5
r4.2xlarge	8	61
r4.4xlarge	16	122
r4.8xlarge	32	244
m4.4xlarge	16	64
m4.10xlarge	40	160
g2.8xlarge	32	60
g3.4xlarge	16	122
d2.8xlarge	36	244

4.3 Parameter Setting

In this paper, our parameters are outlined in Table 3.

Table 3. The related parameter values.

Parameters
$\alpha = \beta = \alpha_E = 100, \beta_E = -13, \overline{u^{cpu}} = 0.25, \overline{u^{mem}} = 0.25, p = 0.02$

4.4 Experimental Comparison

After dealing with 9 batches of new Docker container placement consecutively, we evaluate the physical resource utilization between **CV_VP** and **CVP** architecture. The results are showed as Table 4 and Fig. 5.

As shown in Table 4, after the placement of first batch of containers, the total number of Docker container is 787. It uses 292 active PMs and its evaluation score is 28727.866 under **CV_VP** architecture, however it uses only 244 active PMs that is

Table 4. Comparison of the placement under CV_VP and CVP architecture.

No.	Docker containers	CV_VP		CVP		The ratio of CVP to CV_VP	
		Active Pms	Escore	Active Pms	Escore	Active Pms	Escore
1	787	292	28727.866	**244**	**24283.900**	0.8356	0.8453
2	1019	377	37094.847	**311**	**30978.971**	0.8249	0.8351
3	2297	827	81556.531	**707**	**70406.169**	0.8549	0.8633
4	5014	1819	179197.533	**1493**	**148878.131**	0.8208	0.8308
5	10129	3613	356197.028	**3011**	**300239.999**	0.8334	0.8429
6	12569	4485	356197.028	**3704**	**369394.082**	0.8259	0.8358
7	124984	44516	4386860.736	**36563**	**3647070.301**	0.8213	0.8314
8	149601	53116	5235826.403	**43893**	**4377877.318**	0.8264	0.8361
9	225627	80223	7906899.000	**66103**	**6593462.821**	0.8240	0.8399

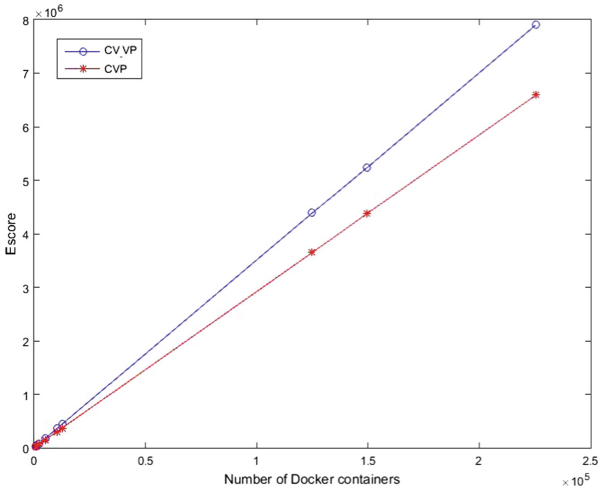


Fig. 5. Escore comparison.

0.8536 times of **CV_VP** and its evaluation score is 24283.9 under **CVP** architecture that is 0.8453 times of **CV_VP**. After the placement of nine batches of Docker containers, the total number of Docker container is 225627. It uses 80223 active PMs and its evaluation score is 7906899 under **CV_VP** architecture, however it uses only 66103 active PMs that is 0.8240 times of **CV_VP** and its evaluation score is 6593462.821 under **CVP** architecture that is 0.8399 times of **CV_VP**. The experimental results show that the placement under **CVP** architecture uses less active PMs and get a better evaluation score than the placement under **CV_VP** architecture. New Docker container placement is more effective in physical resource utilization under **CVP** architecture.

5 Conclusion and Future Work

In this paper, we have extended the study from **two-tier** architecture to **three-tier** "**Container-VM-PM**" (**CVP**) architecture. We focus on the new Docker container placement problem under the **CVP** architecture, and compare it with the placement under **CV_VP** architecture. The results show that the former is more effective than the latter in physical resource utilization.

The resource utilization can be observed in a global view under **CVP** architecture, so it can be more conducive to utilize the resource effectively. Based on this advantage, we want to extend our study on Container consolidation and load balance under the **CVP** architecture in our future work. We also want to test more algorithms in order to fully show their performances under the **CVP** architecture.

Acknowledgement. This work is supported by National Key Research and Development Program of China under Grant No. 2016YFB1000303, Innovative Research Group of the National Natural Science Foundation of China (61721002), Innovation Research Team of Ministry of Education (IRT_17R86), the National Natural Science Foundation of China (61502379, 61532015, 61672410, 61472315, 61428206, 61532015 and 61532004), the Project of China Knowledge Centre for Engineering Science and Technology.

References

1. Affetti, L., Bresciani, G., Guinea, S.: aDock: a cloud infrastructure experimentation environment based on open stack and Docker. In: IEEE International Conference on Cloud Computing (2015)
2. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for Docker using Ant Colony Optimization. In: International Conference on Knowledge and Smart Technology, pp. 254–259. IEEE (2017)
3. Rathor, V.S., Pateriya, R.K., Gupta, R.K.: Survey on load balancing through virtual machine scheduling in cloud computing environment. Int. J. Cloud Comput. Serv. Sci. (IJ-CLOSER) **3**(1), 37 (2014)
4. Docker Swarm Strategies. https://docs.docker.com/swarm/scheduler/strategy/
5. Docker/Awarm. https://github.com/docker/swarm/tree/master/scheduler/strategy
6. Filter Scheduler. https://docs.openstack.org/nova/latest/user/filter-scheduler.html
7. OpenStack/Nova. https://github.com/openstack/nova/tree/master/nova/scheduler
8. Tchana, A., Palma, N.D., Safieddine, I., Hagimont, D., Diot, B., Vuillerme, N.: Software consolidation as an efficient energy and cost saving solution for a SaaS/PaaS cloud model. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015. LNCS, vol. 9233, pp. 305–316. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_24
9. Naik, N.: Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker swarm. In: Computational Intelligence (2017)
10. Bernstein, D.: Containers and cloud: from LXC to Docker to kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2015)
11. Felter, W., et al.: An updated performance comparison of virtual machines and Linux containers. In: IEEE International Symposium on PERFORMANCE Analysis of Systems and Software (2015)

12. Mao, Y., et al.: DRAPS: dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster. In: IEEE – International PERFORMANCE Computing and Communications Conference (2017)
13. Bhardwaj, T., Kumar, M., Sharma, S.C.: Megh: a private cloud provisioning various IaaS and SaaS. In: Pant, M., Ray, K., Sharma, T.K., Rawat, S., Bandyopadhyay, A. (eds.) Soft Computing: Theories and Applications. AISC, vol. 584, pp. 485–494. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-5699-4_45
14. Datt, A., Goel, A., Gupta, S.C.: Analysis of infrastructure monitoring requirements for OpenStack Nova. Procedia Comput. Sci. **54**, 127–136 (2015)
15. Hu, B., Yu, H.: Research of scheduling strategy on OpenStack. In: International Conference on Cloud Computing and Big Data (2014)
16. Sahasrabudhe, S., Sonawani, S.S.: Improved filter-weight algorithm for utilization-aware resource scheduling in OpenStack. In: International Conference on Information Processing (2016)
17. Tseng, H.W., Wu, R.Y., Chang, T.S.: An effective VM migration scheme for reducing resource fragments in cloud data centers (2014)
18. Lu, S., Ni, M., Zhang, H.: The optimization of scheduling strategy based on the Docker swarm cluster. Information Technology, pp. 147–155 (2016)
19. Huang, W., Li, X., Qian, Z.: An energy efficient virtual machine placement algorithm with balanced resource utilization. In: Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (2013)
20. Kenyon, C.: Best-fit bin-packing with random order. In: ACM-SIAM Symposium on Discrete Algorithms (1996)
21. Gao, Y., et al.: A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. **79**(8), 1230–1242 (2013)
22. Tian, F., et al.: Deadlock-free migration for virtual machine consolidation using Chicken Swarm Optimization algorithm. J. Intell. Fuzzy Syst. **32**, 1389–1400 (2017)

Research Track: Cloud Resource Management

Renewable Energy Curtailment via Incentivized Inter-datacenter Workload Migration

Ahmed Abada(✉) and Marc St-Hilaire

Department of Systems and Computer Engineering,
Carleton University, Ottawa, ON, Canada
{ahmed.abada,marc_st_hilaire}@carleton.ca

Abstract. Continuous Grid balancing is essential for ensuring the reliable operation of modern smart grids. Current smart grid systems lack practical large-scale energy storage capabilities and therefore their supply and demand levels must always be kept equal in order to avoid system instability and failure. Grid balancing has become more relevant in recent years following the increasing desire to integrate more Renewable Energy Sources (RESs) into the generation mix of modern grids. RESs produce intermittent energy supply that can't always be predicted accurately [1] and necessitates that effective balancing mechanisms are put in place to compensate for their supply variability [2,3]. In this work, we propose a new energy curtailment scheme for balancing excess RESs energy using data centers as managed loads. Our scheme uses incentivized inter-datacenter workload migration to increase the computational energy consumption at a destination datacenter by the amount necessary to balance the grid. Incentivised workload migration is achieved by offering discounted energy prices (in the form of Energy Credits) to large-scale cloud clients in order to influence their workload placement algorithms to favor datacenters where the energy credits can be used. Implementations of our system using the CPLEX ILP solver as well as the Best Fit Decreasing (BFD) heuristic [4] for workload placement on data centers showed that using energy credits is an effective mechanism to speed-up/control the energy consumption rates at datacenters especially at low system loads and that they result in increased profits for the cloud clients due to the higher profit margins associated with using the proposed credits.

1 Introduction

The lack of large-scale energy storage solutions in modern smart-grids necessitates that their energy generation and consumption levels are always kept equal in order to avoid system instability and service interruptions. Since making online fine adjustments to the energy levels of the generators supplying the grid can be costly (because of the different start-up/shut-down delay constraints of each generator and the monetary costs associated with making such adjustments),

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 143–157, 2018.
https://doi.org/10.1007/978-3-319-94295-7_10

many Demand-Response (DR) proposals were introduced by the utility companies to exploit the demand-side flexibility of energy consumers for grid balancing. DR programs generally aim to control the consumers demand for electricity by using monetary incentives such as variable pricing or time-of-use pricing [5–7]. Commercial class energy consumers have been the main focus of many DR proposals because of their higher consumption rates and the higher demand elasticity of some commercial consumers compared to their residential counterparts. Datacenter plants in particular have received a significant share of DR proposals because of their consistent rapid growth in recent years (as a result of the increasing popularity of cloud computing applications) and their ability to control their energy consumption levels by using different energy management techniques or adjusting the scheduling of their computational workloads [8–11]

Data centers are known to consume a lot of energy to operate since they typically comprise several thousands of computing and networking equipments in a physically monitored environment [12]. The high energy consumption of datacenters has attracted many optimization research efforts [13]. Traditionally the main focus has been on minimizing their energy consumption cost either by sever consolidation [14–16], using DVFS [17] or by proposing different workload scheduling/migration techniques that take advantage of variable real-time energy prices to process more workloads when the prices are low [18,19]. More recently, following the growing demand to integrate more green RESs into the generation mix of modern smart grids, more attention was given to datacenters as controllable loads that could play a major role in enabling the integration of more RESs by compensating for their intermittent supply and help maintain the grid balance. Such DR related research is mainly concerned with the effective integration of data centers as a "resource" or a "managed load" into the various smart grid based DR programs [5,8,20].

Because of their highly intermittent supply, it can be a challenge for the grid to maintain its balance at times of excess RESs energy generation. Hence, the ability of the grid to achieve timely and effective renewable energy curtailment have been a distinctive hurdle that needed to be addressed more effectively in order to increase the amount of renewable energy deployment into the generation mix of modern grids [21]. Events of unexpected excess RESs energy generation can lead to system instability and utility companies opt at times to apply negative energy prices in order to re-balance the grid in a timely manner. However, grid failures can still occur due to such events [22,23].

The curtailment of renewable energy have been studied in the past to ensure balanced grid operation [3,24]. However, unlike existing curtailment strategies that blocks excess RESs energy from entering into the system [25], in this work, we try to leverage the demand side flexibility of datacenters in order to effectively balance the excess RESs energy. Hence, our proposal has the advantage of putting the excess RESs energy to use rather than simply discarding it. We present a new energy curtailment approach that uses data centers as managed loads to allow the grid system more "downward flexibility" (i.e. the ability to cope with extra supply or low demand). Our approach is based on offering dis-

counted energy cost (a monetary incentive in the form of Energy Credits) to large-scale cloud users to incentivize them to migrate their computational workloads to datacenters connected to the part of the grid having excess RESs energy supply. The amount of Energy credits offered is made equal to the amount of energy that a grid would need to consume in a timely manner in order to stay balanced. We formulate our problem as a linear integer programming problem with the objective of minimizing the cost of assigning computational workloads on available datacenters. We present our simulation results from implementing our model using the CPLEX ILP solver as well as the Best Fit Decrteasing (BFD) workload assignment heuristic [4] that can be more practical in large scale operations.

The remainder of this paper is organized as follows: in the next section we briefly discuss previous work related to earlier data center based demand-response efforts and discuss the importance of these efforts as an enabler for integrating more renewable energy sources into the generation mix of modern smart grids. Section 3 introduces our cloud-broker based grid balancing proposal and presents its system model and optimization formulation. We present the performance analysis conducted on our system in Sect. 4 and finally conclude the paper in Sect. 5.

2 Related Work

Since data centers represent a significant load for the smart grids that they draw power from, many research efforts have proposed closer cooperation between the two in order to ensure a smooth operation on both sides [11,26]. An example of such tight coupling between the power grid and the data center is in [27] which tries to save on the datacenter energy cost by deploying backup batteries locally at the datacenter and charging them at times of low electricity prices then using them to power the datacenter when the prices are higher. The significance of data centers as energy loads and their ability to increase/decrease their energy consumption by adjusting their computational workload scheduling represents a valuable opportunity for the smart grid operators to essentially integrate them as managed loads for grid balancing purposes using DR [6,10].

A successful integration between the datacenters and smart grids would allow the datacenters to be considered as a "resource" by the smart grid operator which in turn can then use it for grid balancing purposes. However, a main obstacle towards this integration is that modern data centers are increasingly becoming colocation based [28], where multiple organizations host their computing/networking equipment at a shared physical location in order to save on management and maintenance costs. In such environments, the colocation operator has no direct control on the workload scheduling or the energy-saving/server-consolodation techniques applied by the tenants. This makes the involvement of such data centers in modern DR programs more challenging compared to the traditional case of owner-operated data centers as described in [29]. Collocation operators suffer from the "split-incentive" phenomenon that hinders their participation in DR programs. On one hand, they desire to get the monetary gains

associated with participating in DR programs, yet, they have no direct way to control the energy consumption of their tenants.

Effective grid balancing is also necessary for enabling large-scale integration of RESs into the smart-grid. The integration of such sources into the generation mix of modern smart grids have made significant progress in recent years. However, because of the indeterministic nature of the power generated by such sources (wind/solar) and the unavailability of large-scale energy storage facilities in modern grid systems, their rate of adoption into the generation-mix of modern grids has been mainly limited by the grid's ability to stay in balance given the nature of their intermittent supply [9]. Recent research [1] points to the fact that excess energy from RESs could often need to be discarded in order to keep the grid in balance. In some cases negative pricing is applied just to achieve a timely balancing of the grid and avoid service interruption. Negative/free energy pricing is applied at times of excess RESs supply when it is cheaper to offer the excess energy for free or at a negative price than to shut down generation facilities and risk future service interruptions. The rapid increase of datacenter deployments in recent years is positioned to play an important role in enabling the integration of more RESs into the smart-grids by acting as a managed load that can effectively absorb the introduced supply variability of RESs sources.

A main challenge to overcome in this regard is how to effectively influence the workload scheduling on the datacenter servers in order to comply with the DR requirements. This is not easy to accomplish in current colocation based datacenters since they lack coordination between the colocation operator and the tenants in charge of workloads scheduling on the servers. On one hand, the colocation operator is interested in complying with DR programs so it can qualify for financial incentives, but on the other hand, the colocation tenants are only interested in achieving maximum performance for their workloads since they have no direct relationship with the grid operator and would not receive any financial gain for adjusting their workload if they comply with the requirements of the DR programs. We suggest in this work that establishing a direct relationship between large scale colocation tenants (such as cloud brokers) and the utility company/grid can lead to a more effective DR implementation in the datacenter domain. Previous work in this area has investigated several mechanisms for extending demand response programs to colocation based datacenters [29–31]. However with the continuing demand for increased renewable energy integration in modern smart grids, a tighter coupling between the grid and datacenters (including their tenants) is of great value in order to take full advantage of the datacenters as managed loads.

3 Cloud-Broker Based Grid Balancing

As opposed to existing demand-response approaches that only target data center operators (that don't necessarily have direct control on workload scheduling in the case of colocations), we propose a new mechanism that would allow extending such programs to the enterprise-type tenants (such as Cloud Brokers

and SaaS/PaaS operators) of colocation data centers. Such tenants typically host their hardware at multiple colocations in different geographical locations to achieve high redundancy and shorter response times. Their workloads account for a large share of the total workloads handled by modern colocation data centers. Hence, they can be considered as a resource or a managed load that smart grids can use to scale up/down the energy consumption in order to meet DR goals.

Our approach is based on introducing a "Market" entity as a communication medium between the grid/utility company and the cloud-broker/colocation-tenant as shown in Fig. 1. When a grid/utility company needs to balance a certain amount of excess RESs energy, it generates a number of energy credits equal to the amount of excess RESs energy that needs to be balanced/consumed. Such energy credits are then offered (at a discounted cost) to the cloud brokers/collocation tenants to incentivize them to migrate their workloads from other datacenter locations to a certain datacenter whose energy consumption needs to be increased to balance the excess RESs energy. The proposed energy credits are made available to the brokers via the central market entity ("EC Market") that handles all the energy credits assignments/transactions between the different utility companies and cloud brokers/colocation tenants. Our system model shown in Fig. 1 comprises four main entities, cloud brokers/colocation tenants, colocation data centers, utility companies (grid) and the EC market. We consider a single cloud broker in our model for simplicity, however, the system can be expanded to handle multiple brokers via game theoretic approaches. Each data center is assumed to adopt a different pricing model based on its popularity and the offered performance guarantees. Cloud brokers continuously receive computing workloads (i.e. Requests) of different weights from their clients and they need to optimally assign/schedule the received workloads on the available data centers to maximizes their return.

We assume that each incoming workload (WL) received by a broker has three main attributes, a Computing weight (WL^{cpu}) that describes how much computing resources it requires, a Memory weight (WL^{memory}) that describes how much storage capacity it requires and an Energy Consumption weight (WL^{kWh}) that describes how much energy it is rated to consume per unit time. We further assume that the cloud broker implements a fixed pricing model whereby its clients are charged solely based on the computing and storage weights of their submitted workloads at the rates C_B^{cpu} and C_B^{memory} respectively. Accordingly, the broker's profit P that the broker charges for handling a request j with a computing weight of WL_j^{cpu} and a memory storage weight of WLj^{memory} can be computed as P_j in Eq. (1) per each allocation interval for the time of its execution.

$$P_j = WL_j^{cpu} * C_B^{cpu} + WL_j^{mem} * C_B^{mem} \tag{1}$$

Datacenters on the other hand can charge the requests/workloads submitted to them via cloud brokers according to the individual pricing model of each datacenter. Additionally, data centers also charge the cloud brokers for the energy

consumption of their submitted workloads. Accordingly, if a certain workload consumes WL_j^{kWh} units of energy to run on a certain data center, the data center would charge the broker that submitted the workload a total of C_j^k as per Eq. (2):

$$C_j^k = WL_j^{cpu} * C_{DC}^{cpu} + WL_j^{mem} * C_{DC_k}^{mem} + WL_j^{kWh} * C_{DC_k}^{kWh} \quad (2)$$

Our proposal introduces the concept of "energy credits" which represent a certain amount of energy refund that a cloud broker can use to help offset the otherwise regular energy cost (the last term ($WL^{KWh} * C_{DC}^{kWh}$) in Eq. (2)). As a result, using such credits results in reducing the total amount payable by the brokers to the datacenter where the credits are available. Energy credits are issued by the grid operators when they needs to consume a certain amount of excess RESs energy in order to maintain balance. Energy credits can be claimed against the energy consumption charges on the datacenter connected to the smart grid that issued the credits.

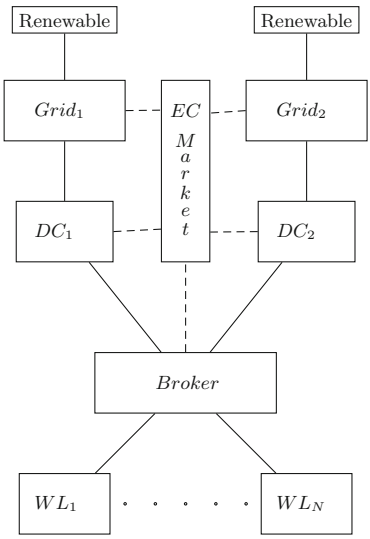


Fig. 1. System model

This direct interaction between the smart grid operators and large scale cloud brokers allows the grid operators to directly influence the scheduling activities of the cloud brokers (such as workload migration from one datacenter to another or workload rescheduling on the same datacenter) on the datacenters that they connect to (by offering reduced energy costs at certain datacenters) and in turn achieve the desired DR objectives. We next introduce the optimization variables and equations of our system model before describing its operation in more detail.

C_B^{cpu} the cost that broker B charges for the processing capacity allocated per time slot (in \$/FLOPS)
C_B^{mem} the cost that broker B charges for disk-space usage per time slot (in \$/MB)
A_{jk}^{next} a binary variable that is set to 1 if job j is assigned to data center k in the next time slot
$A_{jk}^{current}$ a binary variable that is set to 1 if job j is assigned to data center k in the current time slot (initial value = 0)
UEC_{jk} a binary variable that indicates if job j can claim energy credits available at data center k
M the number of available data centers
$C_{DC_k}^{cpu}$ the processing cost at datacenter k per time slot allocated (in \$/FLOPS)
$C_{DC_k}^{mem}$ the disk-space cost at datacenter k per time slot allocated (in \$/MB)
$C_{DC_k}^{kWh}$ the price of energy at datacenter k (in \$/kWh)
EC_k the number of energy credits available at data center k (in kWh)
$MigCost_{kl}$ a normalized value in the range of [0,1] that represents the cost of migrating data from datacenter k to datacenter l (distance-based)
$Cap_{DC_k}^{cpu}$ the maximum processing capacity of datacenter k (in FLOPS)
$Cap_{DC_k}^{mem}$ the maximum disk-storage capacity at datacenter k (in MB)
N the number of client workloads that needs to be assigned to data centers
WL_j^{cpu} the processing capacity needed to process the workload of job j (in FLOPS)
WL_j^{mem} the amount of disk storage needed to host the workload of job j (in MB)
WL_j^{kWh} the energy consumption of job j per each time slot allocated to it (in kWh)
L_{jk} a normalized value (between 0.8–1) that represents the latency between request j and datacenter k. (distance-based)

$$max \sum_{k=1}^{M}\left(\sum_{j=1}^{N}\left(\left[WL_j^{mem} * C_B^{mem} + WL_j^{cpu} * C_B^{cpu}\right] * L_{jk} * A_{jk}^{next}\right)\right) \tag{3}$$

$$-\sum_{k=1}^{M}\left(\sum_{j=1}^{N}\left(\left[WL_j^{mem} * C_{DC_k}^{mem} + WL_j^{cpu} * C_{DC_k}^{cpu} + WL_j^{KWh} * C_{DC_k}^{kWH}\right] * A_{jk}^{next}\right)\right) \tag{4}$$

$$+\sum_{k=1}^{M}\left(\sum_{j=1}^{N}\left(UEC_{jk} * WL_j^{KWh} * C_{DC_k}^{kWH}\right)\right) \tag{5}$$

$$-\sum_{k=1}^{M}\left(\sum_{j=1}^{N}\left(UEC_{jk} * WL_j^{KWh} * 0.1\ C_{DC_k}^{kWH}\right)\right) \tag{6}$$

$$-\sum_{j=1}^{N}\left(\sum_{k=1}^{M} A_{jk}^{current}\left(\sum_{l=1}^{M} A_{jl}^{next} * MigCost[k][l] * WL_j^{mem}\right)\right) \tag{7}$$

$$s.t. \sum_{j=1}^{N} \left(WL_j^{KWh} * UEC_{jk}\right) < EC_k, \quad \forall k \in M \tag{8}$$

$$\sum_{k=1}^{M} A_{jk}^{next} \leq 1, \quad \forall j \in N \tag{9}$$

$$UEC_{jk} <= A_{jk}^{next}, \quad \forall j \in N,\ k \in M \tag{10}$$

$$\sum_{j=1}^{N} WL_j^{cpu} * A_{jk}^{next} \leq Cap_k^{cpu}, \quad \forall k \in M \tag{11}$$

$$\sum_{j=1}^{N} WL_j^{mem} * A_{jk}^{next} \leq Cap_k^{mem}, \quad \forall k \in M \tag{12}$$

We assume that the cloud broker continuously receives end-user requests (workloads) of different processing (WL^{cpu}), storage (WL^{mem}) and energy consumption (WL^{kWh}) weights and that it needs to find the most cost efficient allocation for the received workloads on the available data centers in order to maximize its revenue. We also assume that each workload and data center in the system is associated with a location coordinate (x, y) that is used to determine the distance-based latency L_{jk} between a workload j and a data center k as well as the migration cost factor $MigCost[k][l]$ between data centers k and l. We consider that time is divided into discrete time intervals of equal duration and that the cloud broker needs to decide a cost efficient allocation for its workloads at the beginning of each interval. We further assume that the pricing coefficients of the cloud broker (C_B^{cpu} and C_B^{mem}) remain fixed over all allocation intervals while the pricing coefficients of the different data centers (C_{DC}^{cpu}, C_{DC}^{cpu} and C_{DC}^{cpu}) do change (within a certain range) from one interval to another in order to simulate a dynamic datacenter pricing and allow the broker to adjust its workload allocation on each interval according to the new prices in order to maximize its revenue. We also assume that the cost of purchasing energy credits can be any fraction of the regular price as dictated by the EC market and consider several example values for this fraction (0.1, 0.4 and 0.7) in our results section to show the effect of this parameter on our model. This allows the broker to have enough incentive to reschedule/migrate its workloads in order to take advantage of the low cost energy and balance the grid in the process.

Our optimization formulation introduced above seeks to maximize the financial gain of the cloud broker subject to system constraints. This optimization is evaluated by the cloud broker at the beginning of each allocation interval in order to decide which data center each of its workloads should be assigned to and whether the allocated workload can benefit from using energy credits or not.

The decision variables of our optimization model are the matrices A_{jk}^{next} and UEC_{jk} introduced earlier. A_{jk}^{next} represents the allocation/assignment result of the broker's workloads (incoming requests) on the different available data centers, whereas UEC_{jk} on the other hand represents whether each workload is counted towards the consumption of energy credits at the datacenters that the

workloads were assigned to (when such credits are available). Our optimization formulation seeks to choose the appropriate values for A_{jk}^{next} and UEC_{jk} such that the objective function is maximized. Assuming that the cloud broker needs to assign a total of N workload requests, each to one of the M available data centers, the above decision variables A_{jk}^{next} and UEC_{jk} can then be represented as two 2-dimensional boolean matrices of size $[N][M]$ where each element A_{jk} (s.t. $j \in N$ and $k \in M$) is assigned the value of "1" if workload j was assigned to be hosted on data center k in the next allocation interval and "0" otherwise. Similarly, each element UEC_{jk} is assigned the value of "1" if workload j is counted towards the consumption of the energy credits available at data center k in the next allocation interval and "0" otherwise.

Using A_{jk}^{next} and UEC_{jk} as the decision variables in the above optimization formulation, the objective function tries to maximize the financial gain of the cloud broker given the different cost coefficients (C_{DC}^{cpu}, C_{DC}^{mem}, C_{DC}^{kWH}) of the M available data centers and the weights (WL^{cpu}, WL^{mem}, WL^{kWh}) of the N workloads that needs to be assigned. The net financial gain is estimated as the sum of total generated revenues (represented with a "+" sign in the objective function) minus the sum of total costs (represented with a "−" sign in the objective function). The first term (3) in the objective function represents the money that the cloud broker generates from its clients in exchange for hosting their workloads on the different data centers. If a workload WL_j was successfully assigned to be hosted on a data center k (i.e. $A_{jk}^{next} = 1$), that workload is said to be generating revenue in the amount shown by the first term of the objective function. The whole first term is multiplied by a scaling-down factor L_{jk} that ranges between [0.8, 1] and represents the distance-based latency between a workload j and data center k where as the distance between a workload and a data center increases, L_{jk} will decrease to approach (0.8). The purpose of scaling down the first term by L_{jk} is to enforce the cloud broker to favor assigning workloads to data centers that are geographically closer to them in order to minimize the latency. The second term (4) of the objective function represents the money that the cloud broker would need to pay to the data center that the workload was assigned to. It is simply the weights of the workload multiplied by the corresponding cost at the data center and finally multiplied by the allocation decision variable A_{jk}^{next}. The third term (5) of the objective function represents the regular value of the used energy credits if they were to be bought at the regular price. The fourth term (6) is the actual (reduced) cost price paid for acquiring the energy credits. Accordingly, the difference between the third and fourth terms (actual and reduced costs) represents a monetary gain that the cloud broker achieves by using the reduced cost energy credits offered by the smart grid. The last term (7) of the objective function represents the cost of migrating a workload of size WL^{mem} from data center k to data center l. For this term, we utilize the $[M] \times [M]$ size $MigCost$ matrix that contains distance-based migration cost factors between the different data centers. Migration cost factors range from [0, 1] and approach the value of 0 as the distance between data centers gets closer. In order to detect a migration event we use $A_{jk}^{current}$ to

hold the allocation results from the previous iteration and we detect a migration if for a workload WL_j we have $A_{jk}^{current} = 1$ and $A_{jl}^{next} = 1$ and $k \neq l$.

The first optimization constraint (8) states that the total weight of workloads that can claim usage of energy credits on a data center may not exceed the total amount of energy credits available on that data center. The second constraint (9) limits the number of data center assignments that a workload can get to a maximum of "1", since we assume that workloads can only be assigned to a maximum of one datacenter at a time. The third constraint (10) ensures that a workload has to be assigned to a data center in order for it to be counted towards the energy credits consumption at that data center (i.e. energy credits can not be claimed at a certain data center if the workload is not assigned to be hosted there). The fourth and fifth constraints (11), (12) are data center capacity constraints to ensure that each data center does not get assigned more workloads than what its capacity can accommodate.

4 Performance Analysis

Since our request/workload assignment problem is essentially a multidimensional binpacking problem where each request is defined by three dimensions (cpu, memory and kWh), the computational complexity of our problem is known to be NP-Hard [32]. Therefore, we have implemented our system model using two approaches, first, as a linear optimization problem into the CPLEX ILP solver and second, using the more scalable approach of the Best Fit Decreasing (BFD) assignment heuristic, so we can evaluate our model under the two implementations. In this section we present our simulation results that measures two different aspects related to our model, namely the effect of the amount of energy credits introduced in the system on the time needed to balance a certain amount of excess energy and the effect of the amount of introduced energy credits on the revenue generated by a cloud broker. We used a system model consisting of three data centers and a single cloud broker as discussed before to maintain system simplicity. Time is modeled in discrete intervals of equal size and the CPLEX/BFD implementations are ran by the broker at the beginning of each interval to determine the allocation/mapping of the requests/workloads on the different available datacenters. Our shown simulation results are the averages of 100 randomized runs where the cost coefficients (C_{DC}^{mem}, C_{DC}^{cpu}, C_{DC}^{kWH}) and location coordinates of all data centers are randomized (within the specified ranges as shown in Table 1) at the beginning of each allocation interval in order to mimic a dynamic cost system that would require the broker to optimize/adjust its workload allocations at the beginning of each new interval and induce workload migrations between data centers. We have set the range of the datacenter energy consumption cost parameter C_{DC}^{kWH} slightly higher than the other datacenter costs in order to make datacenters with available energy credits more attractive as hosts. On the other hand, the cloud broker adopts fixed cost coefficients (C_B^{mem}, C_B^{cpu}) over all allocation intervals in order to shield its clients from the uncertainty of the volatile live pricing model applied by datacenters.

The fixed costs of the broker are set higher than the randomized datacenter costs to allow it to still make profit under any cost applied by the datacenters. We also show the effect of variable system loads on the measured parameters by using different workload arrival rates (10, 20, 30, 40, 50 new request per allocation interval) for each simulation. The list of the different parameters used in our performance testing are listed in Table 1.

Table 1. Simulation parameters

Parameter	Min	Max
C_{DC}^{mem}, C_{DC}^{cpu}	5	15
C_{DC}^{kWH}	25	30
C_{B}^{mem}, C_{B}^{cpu}	100	100
WL^{mem}, WL^{cpu}, WL^{kwh}	5	15
$WL^{duration}$	3	5
Available energy credits (EC)	0	3000
Workload arrival rate	10	50
Datacenter CPU capacity	5000	5000
Datacenter memory capacity	5000	5000

Figure 2 shows the effect of using different cost fractions (0.1, 0.4 and 0.7) for energy credits prices on the time needed to balance the grid given that a certain amount of energy needs to be consumed. Here we assume that 3000 units of energy needs to be consumed by a datacenter in order to balance the grid and we observe how long it takes (in number of allocation intervals) the datacenter to consume this amount when different cost fractions of the original price are offered as energy credits. We notice that reducing the price of offered energy credits always speeds up the time needed for grid balancing especially under light system loads as the workloads quickly become concentrated at the data center that has the energy credits. We also notice that both CPLEX and BFD perform almost exactly the same in all simulated cases. This is due to the big size difference between the request/workload size requirements and the capacity of datacenters (as shown in Table 1), which is a valid assumption in most realistic situations. Because request sizes are considerably smaller than datacenter capacities and the fact that the BFD algorithm always orders the items in a decreasing fashion before assignment, BFD can achieve as optimal binpacking results as CPLEX except for the last "critical element" that gets rejected for size capacity violations. Therefore, the size of the last element that gets rejected from a bin represents an upper bound on the efficiency of the bin assignment. This upper bound is minimized under the assumption of small request sizes. Figure 3 shows the effect of using different cost fractions (0.1, 0.4 and 0.7) for energy credits prices on the broker's generated revenues. We can see that the generated revenues increase as the cost fraction of energy credits

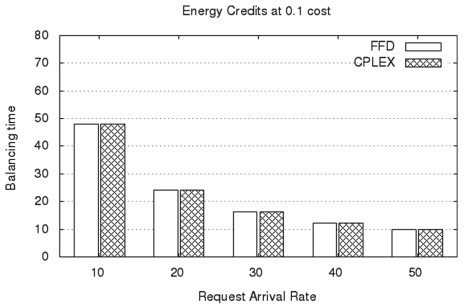


Fig. 2. Effect of varying ECs cost on balancing time

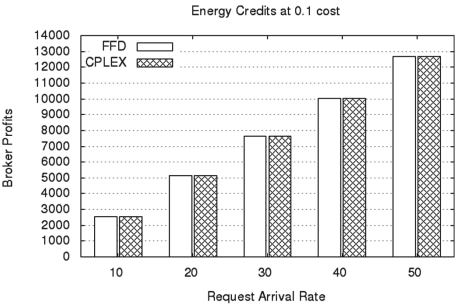


Fig. 3. Effect of varying ECs cost on Broker's profit

decrease. This is expected as energy credits represent discounted energy costs and the lower the cost paid towards acquiring the energy credits the more profits are generated.

5 Conclusion and Future Work

Effective grid balancing is essential for ensuring the reliable operation of modern smart grids. In this work we presented a new linear optimization approach that uses energy credits as an incentive to control the energy consumption levels of large scale datacenter clients (such as cloud brokers/colocation tenant) for grid balancing purposes. Simulations performed on our optimization model using

two different implementation approaches showed that using energy credits as an incentive can speed up the grid balancing process and increases the brokers profit margins. In future work we plan to expand the proposed system to include multiple cloud brokers to compete for the available energy credits by way of game theory bidding mechanism in order to maximize the smart grid's return on the offered energy credits.

References

1. Wang, J., Botterud, A., Bessa, R., Keko, H., Carvalho, L., Issicaba, D., Sumaili, J., Miranda, V.: Wind power forecasting uncertainty and unit commitment. Appl. Energy **88**(11), 4014–4023 (2011)
2. Nicolosi, M.: Wind power integration and power system flexibility-an empirical analysis of extreme events in Germany under the new negative price regime. Energy Policy **38**(11), 7257–7268 (2010). Energy Efficiency Policies and Strategies with regular papers
3. Olson, A., Jones, R.A., Hart, E., Hargreaves, J.: Renewable curtailment as a power system flexibility resource. Electr. J. **27**(9), 49–61 (2014)
4. Khuri, S., Schütz, M., Heitkötter, J.: Evolutionary heuristics for the bin packing problem. In: Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.) Artificial Neural Nets and Genetic Algorithms, pp. 285–288. Springer, Vienna (1995). https://doi.org/10.1007/978-3-7091-7535-4_75
5. Albadi, M.H., El-Saadany, E.F.: Demand response in electricity markets: an overview. In: 2007 IEEE Power Engineering Society General Meeting, pp. 1–5, June 2007
6. Basmadjian, R., Lovasz, G., Beck, M., Meer, H.D., Hesselbach-Serra, X., Botero, J.F., Klingert, S., Ortega, M.P., Lopez, J.C., Stam, A., Krevelen, R.V., Girolamo, M.D.: A generic architecture for demand response: the ALL4Green approach. In: 2013 International Conference on Cloud and Green Computing, pp. 464–471, September 2013
7. Spees, K., Lave, L.B.: Demand response and electricity market efficiency. Electr. J. **20**(3), 69–85 (2007)
8. Chen, H., Caramanis, M.C., Coskun, A.K.: The data center as a grid load stabilizer. In: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 105–112, January 2014
9. Chiş, A., Rajasekharan, J., Lundén, J., Koivunen, V.: Demand response for renewable energy integration and load balancing in smart grid communities. In: 2016 24th European Signal Processing Conference (EUSIPCO), pp. 1423–1427, August 2016
10. Kim, H., Kim, Y.J., Yang, K., Thottan, M.: Cloud-based demand response for smart grid: architecture and distributed algorithms. In: 2011 IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 398–403, October 2011
11. Wierman, A., Liu, Z., Liu, I., Mohsenian-Rad, H.: Opportunities and challenges for data center demand response. In: International Green Computing Conference, pp. 1–10, November 2014
12. Schomaker, G., Janacek, S., Schlitt, D.: The energy demand of data centers. In: Hilty, L.M., Aebischer, B. (eds.) ICT Innovations for Sustainability. AISC, vol. 310, pp. 113–124. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-09228-7_6

13. Rong, H., Zhang, H., Xiao, S., Li, C., Hu, C.: Optimizing energy consumption for data centers. Renew. Sustain. Energy Rev. **58**, 674–691 (2016)
14. Ferreto, T.C., Netto, M.A., Calheiros, R.N., Rose, C.A.D.: Server consolidation with migration control for virtualized data centers. Future Gener. Comput. Syst. **27**(8), 1027–1034 (2011)
15. Varasteh, A., Goudarzi, M.: Server consolidation techniques in virtualized data centers: a survey. IEEE Syst. J. **11**(2), 772–783 (2017)
16. Van, H.N., Tran, F.D., Menaud, J.M.: SLA-aware virtual resource management for cloud infrastructures. In: 2009 Ninth IEEE International Conference on Computer and Information Technology, vol. 1, pp. 357–362, October 2009
17. Wu, C.M., Chang, R.S., Chan, H.Y.: A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters. Future Gener. Comput. Syst. **37**, 141–147 (2014)
18. Buchbinder, N., Jain, N., Menache, I.: Online job-migration for reducing the electricity bill in the cloud. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011. LNCS, vol. 6640, pp. 172–185. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20757-0_14
19. Rao, L., Liu, X., Xie, L., Liu, W.: Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9, March 2010
20. Chiu, D., Stewart, C., McManus, B.: Electric grid balancing through lowcost workload migration. SIGMETRICS Perform. Eval. Rev. **40**(3), 48–52 (2012)
21. Bollen, M.: The Smart Grid: Adapting the Power System to New Challenges. Morgan and Claypool, San Rafael (2011)
22. Vos, K.D.: Negative wholesale electricity prices in the German, French and Belgian day-ahead, intra-day and real-time markets. Electr. J. **28**(4), 36–50 (2015)
23. Brandstätt, C., Brunekreeft, G., Jahnke, K.: How to deal with negative power price spikes?—flexible voluntary curtailment agreements for large-scale integration of wind. Energy Policy **39**(6), 3732–3740 (2011)
24. Kane, L., Ault, G.: A review and analysis of renewable energy curtailment schemes and principles of access: transitioning towards business as usual. Energy Policy **72**, 67–77 (2014)
25. Subramanian, A., Bitar, E., Khargonekar, P., Poolla, K.: Market induced curtailment of wind power. In: 2012 IEEE Power and Energy Society General Meeting, pp. 1–8, July 2012
26. Kavanagh, R., Armstrong, D., Djemame, K., Sommacampagna, D., Blasi, L.: Towards an energy-aware cloud architecture for smart grids. In: Altmann, J., Silaghi, G.C., Rana, O.F. (eds.) GECON 2015. LNCS, vol. 9512, pp. 190–204. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43177-2_13
27. Zhang, S., Ran, Y., Wu, X., Yang, J.: Electricity cost optimization for data centers in smart grid environment. In: 11th IEEE International Conference on Control Automation (ICCA), pp. 290–295, June 2014
28. https://www.rackspace.com/managed-hosting/colocation
29. Ren, S., Islam, M.A.: Colocation demand response: why do i turn off my servers? In: 11th International Conference on Autonomic Computing (ICAC 2014), pp. 201–208. USENIX Association, Philadelphia (2014)
30. Zhan, Y., Ghamkhari, M., Xu, D., Mohsenian-Rad, H.: Propagating electricity bill onto cloud tenants: using a novel pricing mechanism. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6, December 2015

31. Zhan, Y., Ghamkhari, M., Xu, D., Ren, S., Mohsenian-Rad, H.: Extending demand response to tenants in cloud data centers via non-intrusive workload flexibility pricing. IEEE Trans. Smart Grid **99**, 1–1 (2016)
32. Christensen, H.I., Khan, A., Pokutta, S., Tetali, P.: Approximation and online algorithms for multidimensional bin packing: a survey. Comput. Sci. Rev. **24**, 63–79 (2017)

Pricing Cloud Resource Based on Reinforcement Learning in the Competing Environment

Bing Shi[1,2(✉)], Hangxing Zhu[1], Han Yuan[1], Rongjian Shi[1], and Jinwen Wang[1]

[1] School of Computer Science and Technology, Wuhan University of Technology, Wuhan, People's Republic of China
bingshi@whut.edu.cn

[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, People's Republic of China

Abstract. Multiple cloud providers compete against each other in order to attract cloud users and make profits in the cloud market. In doing so, each provider needs to charge fees to users in a proper way. In this paper, we will analyze how a cloud provider sets price effectively when competing against other cloud providers. Specifically, we model this problem as a Markov game, and then use minimax-Q and Q learning algorithms to design the pricing policies respectively. Based on this, we run extensive experiments to analyze the effectiveness of minimax-Q and Q learning based pricing policies. We find that although minimax-Q is more suitable in analyzing the competing game with multiple self-interested cloud providers, Q learning based pricing policy performs better in terms of making profits. We also find that minimax-Q learning based pricing policy performs better in terms of keeping cloud users. Our experimental results can provide useful insights on designing practical pricing policies in different situations.

Keywords: Competing cloud providers · Pricing policy
Reinforcement learning · Minimax-Q learning

1 Introduction

During the past few years, the development of cloud computing has achieved significant success in the industry since it can provide economical, scalable, and elastic access to computing resources, thus liberating people from installing, configuring, securing, and updating a variety of hardware and software [1–3]. More and more firms and personal users have been using cloud computing services over Internet, which contribute to the development of the cloud computing market. The global cloud computing market is expected to grow at a 30% compound annual growth rate (CAGR) reaching $270 billion in 2020. To compete for hundreds of billions of dollars, many firms as service providers have been

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 158–171, 2018.
https://doi.org/10.1007/978-3-319-94295-7_11

participating in the cloud market [4]. Now, there exist many dominating cloud platforms offering cloud services, such as Microsoft's Azure, IBM's SoftLayer and Amazon's AWS. In the cloud market with multiple cloud providers, cloud users have various choices, and they usually participate in the provider which can satisfy their demands and charge the lowest price to them. Actually, when multiple providers offer similar quality of service [5–7], the price will significantly affect users' choices and thus providers' profits. Therefore, cloud providers need to set effective prices to compete against each other. Furthermore, the competition among providers usually lasts for a long time, i.e. the providers compete against each other repeatedly, and thus they need to maximize the long-term profits. In this paper, we will analyze how a cloud provider designs an appropriate pricing policy to maximize the long-term profit and also remain attractive to cloud users.

There exist some works on designing pricing policies for cloud providers. In [8], a non-cooperative competing model based on game theory has been proposed which computes the equilibrium price for one-shot game and does not consider the long-term profits. In [9,10], the authors assume that there is only one provider, while in today's cloud market multiple providers exist and compete against each other. Then the authors in [11,12] analyze the user behavior with respect to the providers' prices, but ignore the competition among providers. In [13], the authors analyze the pricing policy in the competing environment by assuming that there is only one proactive provider, and other providers just follow the proactive one's pricing policy. Some other works, such as [14,15], consider the competition among providers but does not capture the market dynamics, and their algorithms can only be applied to a very small market with few users.

To the best of our knowledge, few works have considered the situation of multiple providers competing against each other repeatedly. In this paper, we will analyze how the competing cloud provider sets price effectively to maximize the long-term profits in the context with two competing providers.[1] In more detail, we first describe basic settings of cloud users and providers. Specifically, we consider the uncertainty of users choosing cloud providers in the setting, which is consistent with the realistic user behavior. Furthermore, how users choosing cloud providers is affected by the prices, and how cloud providers setting prices is affected by users' choices, and therefore it is a sequential-decision problem. Moreover, this problem involves two self-interested cloud providers, and thus it is a Markov game [16]. In this paper, we model the competition between cloud providers as a Markov game, and then use two typical reinforcement learning algorithms, minimax-Q learning [17] and Q learning [18], to solve this game and design the pricing policy. We then run extensive experiments to evaluate the policies in different situations. We find that although minimax-Q learning, which was specifically designed for Markov games, is more suitable to be applied in this issue, Q learning based pricing policy performs better in terms of making profits. We also find that minimax-Q based pricing policy is better for remaining attractive to cloud users.

[1] Our model can be easily extended to the case with more than two cloud providers.

The structure of the paper is as follows. In Sect. 2, we describe basic settings of cloud users and providers. In Sect. 3, we describe how to use Q learning and minimax-Q learning algorithms to design the pricing policy. We run extensive experiments to evaluate the pricing policies in different situations in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Basic Settings

In this section, we describe the basic settings of cloud providers and users. We assume that there are N users and two cloud providers, A and B. Cloud providers compete against each other repeatedly, i.e. the competition consists of multiple stages. At the beginning of each stage, each provider publishes its price, and then each user chooses to be served by which provider based on its choice model. According to users' choices, the two providers compute the obtained profits at the current stage, and the competition enters into the next stage.

2.1 Cloud Providers

Cloud providers can make profits by charging fees to users, while they also need to pay for the cost of offering services (e.g. power, hardware, infrastructure maintenance cost and so on). At stage t, provider i should pay for the cost of offering per unit service [19], which is denoted as $c_{i,t}$. We assume that each user only requests one-unit service. Therefore, the amount of requested service at stage t is equal to the number of users. At the beginning of the competition, the initial marginal cost of provider i is $c_{i,0}$. At stage t, the amount of users choosing provider i is $N_{i,t}$, and then at this stage, the marginal cost is:

$$c_{i,t} = c_{i,0}(N_{i,t})^{-\beta} e^{-\theta t} \tag{1}$$

This equation indicates that as more users requiring the service and as time goes, the marginal cost decreases [20]. Specifically, when the provider receives more demands of services, its marginal cost would be decreased because of economics of scale, where β is the parameter for the economics of scale, and $\beta > 0$. Furthermore, the reduction of hardware cost and the development of technology contribute to the temporal decaying factor of the marginal cost, where θ is the parameter of temporal decaying factor, and $\theta > 0$.

We assume that the price is denoted as p, and all allowable prices constitute a finite set P. The price is actually the *action* used in Sect. 3. After providers publishing the prices, users make the choices of providers. We can calculate the immediate reward of each provider, which is the immediate profit made at the current stage t:

$$r_{i,t} = N_{i,t}(p_{i,t} - c_{i,t}) \tag{2}$$

where $p_{i,t}$ is the price set by provider i at stage t.

2.2 Cloud Users

Each user has a marginal value on per-unit requested service, which is denoted as δ. At stage t, after all providers publish the prices, user j can calculate its expected revenue when entering provider i, which is:

$$R_{j,i}^{t} = \delta_j - p_{i,t} \tag{3}$$

Intuitively, based on Eq. 3, cloud users can determine in which provider they can obtain the maximal revenue at the current stage, and then choose that provider. However, in the real world, users keep requiring cloud services, and they usually take into account the prices at previous stages. Specifically, in this paper, we assume that the users will consider the prices at the current stage t and the last stage $t-1$ when choosing the cloud providers. We do not need to consider the prices at all previous stages since in this paper, the providers' prices and the users' choices are affected by each other, and thus the price of the last stage actually implies the dynamic interaction of all previous stages. Therefore, the expected utility that user j can make when entering provider i is:

$$v_{j,i}^{t} = \xi R_{j,i}^{t} + (1-\xi) R_{j,i}^{t-1} \tag{4}$$

where ξ is the weight of the price considered by the user at this stage. Furthermore, in reality, when agents make decisions, their choices are affected by some unobservable factors [21], such as customers' loyalty on some product brand, which is denoted as $\eta_{j,i}$. This part introduces the uncertainty of users' choice. Now the utility that cloud user j makes in provider i at stage t is defined as follows:

$$u_{j,i}^{t} = v_{j,i}^{t} + \eta_{j,i} \tag{5}$$

We assume that the random variable $\eta_{j,i}$ is an independently, identically distributed extreme value, i.e. it follows Gumbel and type I extreme value distribution [21], and the density of $\eta_{j,i}$ is

$$f(\eta_{j,i}) = e^{-\eta_{j,i}} e^{-e^{-\eta_{j,i}}} \tag{6}$$

and the cumulative distribution is

$$F(\eta_{j,i}) = e^{-e^{-\eta_{j,i}}} \tag{7}$$

The probability of user j choosing provider i at stage t, which is denoted as $P_{j,i}^{t}$

$$\begin{aligned} P_{j,i}^{t} &= Prob(u_{j,i}^{t} > u_{j,i'}^{t}, \forall i' \neq i) \\ &= Prob(v_{j,i}^{t} + \eta_{j,i} > v_{j,i'}^{t} + \eta_{j,i'}, \forall i' \neq i) \\ &= Prob(\eta_{j,i'} < \eta_{j,i} + v_{j,i}^{t} - v_{j,i'}^{t}, \forall i' \neq i) \end{aligned} \tag{8}$$

According to (7), $P_{j,i}^{t}$ is

$$P_{j,i}^{t} = e^{-e^{(\eta_{j,i} + v_{j,i}^{t} - v_{j,i'}^{t})}} \tag{9}$$

Since $\eta_{j,i}$ is independent, the cumulative distribution over all $i \neq i'$ is the product of the individual cumulative distributions

$$P_{j,i}^t \mid \eta_{j,i} = \prod e^{-e^{-(\eta_{j,i}+v_{j,i}^t - v_{j,i'}^t)}} \tag{10}$$

And $\eta_{j,i}$ is unknown to the providers, so the choice probability is the integral of $P_{j,i}^t \mid \eta_{j,i}$ over all values of $\eta_{j,i}$ weighted by its density

$$P_{j,i}^t = \int (\prod_{i' \neq i} e^{-e^{-(\eta_{j,i}+p_{i,t}-p_{i',t})}}) e^{-\eta_{j,i}} e^{-e^{-\eta_{j,i}}} d\eta_{j,i} \tag{11}$$

The closed-form expression is

$$P_{j,i}^t = \frac{e^{v_{j,i}^t}}{\sum_{i'} e^{v_{j,i'}^t}} \tag{12}$$

which is the probability of user j choosing to be served by provider i at stage t.

3 Reinforcement Learning Algorithms

After describing the basic settings, we now introduce how to design a pricing policy for the cloud provider. How to set an effective price is a decision-making problem, and reinforcement learning algorithms have been widely used to solve similar issues. Specifically, we adopt Q learning algorithm [18] to determine how the provider sets the price. Note that Q learning algorithm is usually used to solve the sequential decision problem involving only one agent, and therefore when using Q learning algorithm, we let the opponent's action be part of the environment. Moreover, since our problem actually involves two providers competing against each other repeatedly, it can be modeled as a Markov game [16]. In such a game, we use minimax-Q learning algorithm [17] to solve this issue[2]. In the following, we introduce how to design the pricing policy based on Q learning and minimax-Q learning algorithms respectively.

At stage t, provider A sets price according to its own and the opponent B's price at the last stage $t-1$, which is denoted as state $s_{t-1} = (p_{A,t-1}, p_{B,t-1})$. Note that the state does not involve the amount of users participating in each provider since the price has implied users' choices and therefore we only use the prices to represent the state. The state space is denoted as $S = P \times P$. For simplicity, in the following, we use $a \in P$ and $b \in P$ to represent the actions of providers A and B respectively. The pricing policies of provider A and B are denote as Π_A

[2] minimax-Q learning was designed to solve Markov game when its stage game is a zero-sum game. In this paper, although the sum of both providers' payoffs is not zero, the gain of one provider (users choosing this provider) is indeed the loss of the other provider (users not choosing that provider). Therefore, it is actually a zero-sum game, and we use minimax-Q learning algorithm to design the pricing policy in this competing environment.

Algorithm 1. Q learning

Input: pricing space P; B's pricing policy Π_B
Output: A's pricing policy Π_A

for $\forall s \in S, V(s) = 0, and\ \forall a \in P, Q(s,a) = 0$
for $\forall s \in S, \forall a \in P, \Pi_A(s,a) = 1/|P|$
repeat
  at the current state s, given ϵ, generate a random number $rand$ $(0 < rand < 1)$; when $rand \leq \epsilon$, A chooses a price $a \in P$ randomly; when $rand > \epsilon$, A chooses a price $a \in P$ according to the pricing policy Π_A
  B chooses the price b (according to the pricing policy Π_B), the next state is $s' = (a, b)$
  $Q(s,a) = (1-\alpha) * Q(s,a) + \alpha * (r_{i,t} + \gamma * V(s'))$
  $\Pi_A(s,\cdot) = argmax_{\Pi'_A(s,\cdot)}(\sum_{a'}(\Pi_A(s,a') * Q(s,a')))$
  $V(s) = \sum_a(\Pi_A(s,a) * Q(s,a))$
until $(\Pi_A(s,\cdot)$ is converged)

Algorithm 2. minimax-Q learning

Input: pricing space P; B's pricing policy Π_B
Output: A's pricing policy Π_A

for $\forall s \in S, V(s) = 0, and\ \forall a \in P, Q(s,a,b) = 0$
for $\forall s \in S, \forall a \in P, \Pi_A(s,a) \leftarrow 1/|P|$
repeat
  at the current state s, given ϵ, generate a random number $rand$ $(0 < rand < 1)$; when $rand \leq \epsilon$, A chooses a price $a \in P$ randomly; when $rand > \epsilon$, A chooses a price $a \in P$ according to the pricing policy Π_A
  B chooses the price b (according to the pricing policy Π_B), the next state is $s' = (a, b)$
  $Q(s,a,b) = (1-\alpha) * Q(s,a,b) + \alpha * (r_{i,t} + \gamma * V(s'))$
  $\Pi_A(s,\cdot) = argmax_{\Pi'_A(s,\cdot)}(min_{b'}\sum_{a'}(\Pi_A(s,a') * Q(s,a',b')))$
  $V(s) = min_{b'}(\sum_{a'}(\Pi_A(s',a') * Q(s',a',b')))$
until $(\Pi_A(s,\cdot)$ is converged)

and Π_B respectively. Based on these notations, Q learning algorithm is shown in Algorithm 1, and minimax-Q learning algorithm is shown in Algorithm 2. In this setting, it is guaranteed that both algorithms will converge [17,18]. The final output Π_A is the designed pricing policy.

4 Experimental Analysis

In this section, we run numerical simulations to analyze the reinforcement learning based pricing policies in different situations. We first describe the parameter setup in the experiments in the following.

Table 1. Experimental parameters

Parameter setup	Description
$c_{.,0} = 5$	cloud provider's marginal cost at the initial stage
$P = \{10, 20, \ldots, 100\}$	the set of allowable prices
$N = 100$	the amount of users in the market
$\delta \in [50, 150]$	the cloud user's marginal value δ follows a uniform distribution supported on [50,150]
$\beta = 0.01$	parameter β in Eq. 1
$\theta = 0.001$	parameter θ in Eq. 1
$\xi = 0.8$	parameter ξ in Eq. 4
$\epsilon = 0.2$	exploration rate in Q learning and minimax-Q learning algorithms
$\gamma = 0.8$	discount factor in Q learning and minimax-Q learning algorithms

4.1 Experimental Parameters

First, we assume that each cloud provider has the same initial marginal cost, i.e. $c_{.,0} = 5$, and the marginal cost is decreased as the demand increases. In addition, we assume that the set of allowable prices P chosen by cloud providers is $\{10, 20, \ldots, 100\}$. Furthermore, we assume that there are $N = 100$ cloud users in total. The marginal values of users δ are independent random variable, and for illustrative purpose, we assume that they are drawn from a uniform distribution with support $[50, 150]$. Other parameters used in the following simulations follow the typical setting in the related literature, and are shown in Table 1.

4.2 Pricing Policy

We first describe the pricing policies trained and output by minimax-Q and Q learning algorithms respectively. We consider the case that the cloud provider takes Q learning and minimax-Q learning algorithms against the opponent choosing actions randomly, and the case that both cloud providers are trained in Q learning and minimax-Q learning algorithms. We name the trained pricing policies as QR, QQ, MR and MM respectively, and for example QQ means that both cloud providers adopt Q learning algorithm and are trained against each other. We show these four pricing policies in Fig. 1. From these figures, we can find the probability of the provider choosing each price (action) at each state. Note that in this paper, the state is a tuple including two providers' prices at the last stage, and in order to show the state in one-dimension *state*-axis, we map state (10, 10) to 1, map state (10, 20) to 2, ..., map (20, 10) to 11, ..., and map (100, 100) to 100. Furthermore, we find that no provider intends to

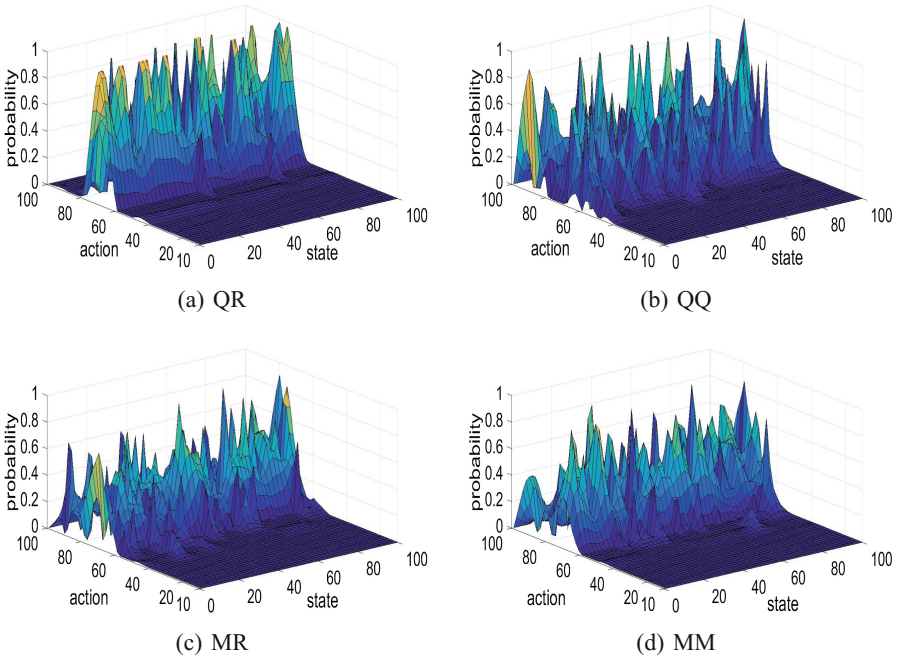
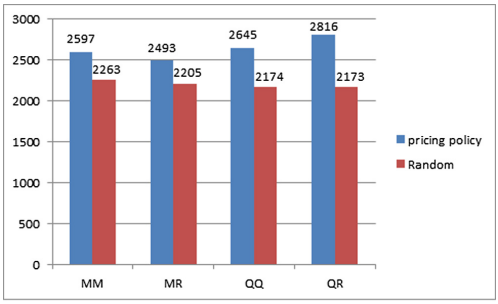


Fig. 1. Pricing policies trained in Q and minimax-Q learning algorithms

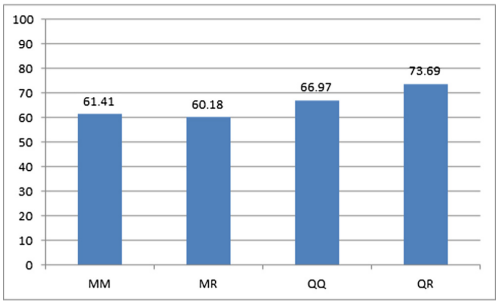
set a minimal price 10 to attract users, since on one hand, a low price is not beneficial for the long-term profit, and on the other hand, cloud users' choices of providers are also affected by some unobservable factors, and thus a provider with a minimal price cannot attract all users, but lose some profits. Furthermore, we find that these pricing policies will not set the highest price since such a high price will drive all users to leave. Moreover, we find that the surface of QQ and QR is sharper than MR and MM's. Specifically, at some state, QR and QQ will choose a deterministic action, but MR and MM have mixed actions. This is because in contrast to Q learning trying to maximizing the profit regardless of the opponent's action, minimax-Q learning needs to randomize the action in order to maximize the profit in the worst case.

4.3 Evaluation

In this section, we run simulations to evaluate the above pricing policies in different situations. Specifically, we use the average profit and the winning percentage as the evaluation metrics.



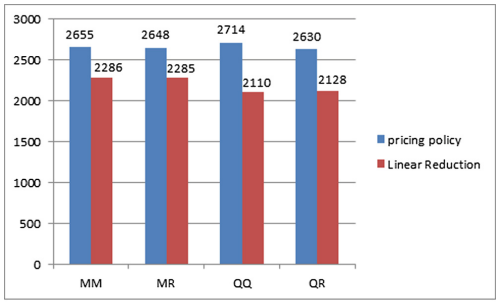
(a) Average profit



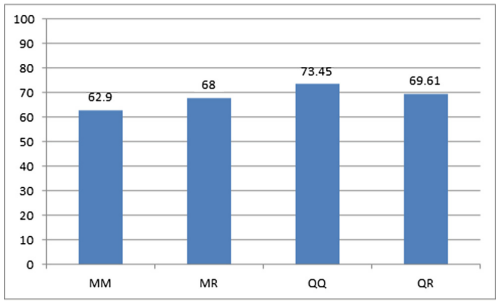
(b) Winning percentage

Fig. 2. MM, MR, QQ, QR vs. *Random*

vs. Random Pricing Policy. We first evaluate these four pricing policies against the opponent adopting a random pricing policy, which chooses each price with equal probability. The reason for doing this is that when participating in the cloud market, some fresh cloud providers often explore the competing environment randomly in order to collect more market information. The competing results are shown in Fig. 2, where Fig. 2(a) is the average profit over 10000 stages, and Fig. 2(b) is the winning percentage of these four policies competing against the random pricing policy. We find that all these four pricing policy can beat the opponent. Not surprisingly, we find that QR is the best one among these four policies when competing against the random pricing policy since QR is trained specifically against the random policy. In contrast, we find that MR is the worst one. This is because when the opponent chooses the price randomly, i.e. not trying to make the other side be the worst, minimax-Q cannot perform well. In fact, we find that QQ and QR perform better than MR and MM. This may indicate that the agent should adopt Q learning when its opponent cannot take action in an intelligent way.



(a) Average profit

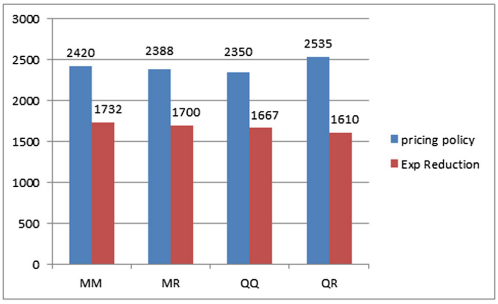


(b) Winning percentage

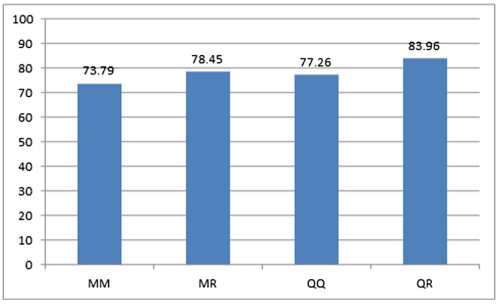
Fig. 3. MM, MR, QQ, QR vs. *Linear Reduction*

vs. Price Reduction Policy. In the real world, cloud providers usually attract cloud users by decreasing the price continuously. For example, when a fresh cloud provider enters the market, it may keep decreasing the price to attract users. We also evaluate these four pricing policies against the cloud provider which keeps reducing the price. Specifically, we consider two typical price reduction policies, Linear Reduction and Exp Reduction. In Linear Reduction policy, the price decreases linearly with respect to time, where at stage t the price is $p_t = p_0 - 0.01t$ (p_0 is the initial price, and we set it as the maximal price, i.e. 100), while in Exp Reduction, the price decreases exponentially with time, where $p_t = p_0 * e^{-0.0003t}$ ($p_0 = 100$ which is the same as before). The results of QQ, QR, MM, MR competing against Linear Reduction policy and Exp Reduction policy are shown in Figs. 3 and 4. We still find that our reinforcement learning-based pricing policies can beat these price reduction policies. Again, we find that MM and MR cannot outperform the reduction policies significantly than that QQ and QR do.

***Q*-*X* vs. *X*.** In the above, it seems that when competing against the opponent using simple pricing policies (i.e. random or price reduction), Q learning based pricing policy is better. However, after investigating the fundamentals of minimax-Q and Q learning algorithms, we can see that minimax-Q is more suitable in this Markov game with two competing providers. Since this is not proved in the above experiments, in the following we further investigate this issue by



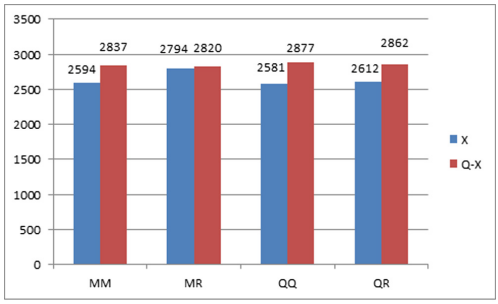
(a) Average profit



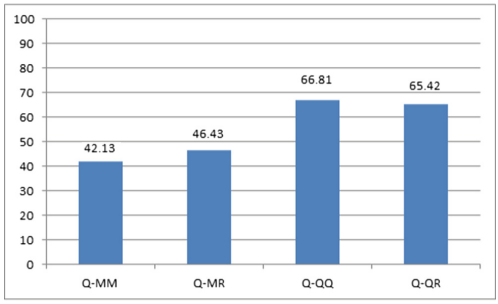
(b) Winning percentage

Fig. 4. MM, MR, QQ, QR vs. *Exp Reduction*

using Q learning algorithm to train the pricing policy against the above four policies, i.e. QQ, QR, MR, MM. We then obtained four new pricing policies, Q-QR, Q-QQ, Q-MR, and Q-MM. Q-QR is a pricing policy based on Q learning competing against QR pricing policy in the above section. We run the following simulations including Q-MM vs MM, Q-MR vs MR, Q-QQ vs QQ and Q-QR vs QR. The results are shown in Fig. 5, where QQ, QR, MR, MM are denoted as X. From Fig. 5(b), in terms of winning percentage, we find that Q-QQ outperforms QQ and Q-QR outperforms QR, i.e. the winning percentage is more than 50%. However, even though Q-MM is trained against MM, it is outperformed by MM, i.e. the winning percentage is less than 50%. The similar result happens for Q-MR. However, from Fig. 5(a), we find that even though Q-MM is outperformed by MM in terms of winning percentage, it obtains more profits than MM. It is similar for Q-MR. This is because minimax-Q based pricing policies try to do the best in the worst case, and therefore its winning percentage can be kept at a good level. However, Q learning based policies try to maximize the profits at all times, and therefore perform better in terms of making profits.



(a) Average profit



(b) Winning percentage

Fig. 5. Q-X vs. X

5 Conclusions

How to set prices effectively is an important issue for the cloud provider, especially in the environment with multiple cloud providers competing against each other. In this paper, we use reinforcement learning algorithms to address this issue. Specifically, we model the issue as a Markov game, and use minimax-Q and Q learning algorithms to design the pricing policies respectively. We then run extensive experiments to analyze the pricing policies. We find that although minimax-Q is more suitable in analyzing the competing game with multiple self-interested agents, Q learning based pricing policy performs better in terms of making profits. We also find that minimax-Q based pricing policy is better for remaining attractive to cloud users. The experimental results can provide useful insights on designing practical pricing policies in different situations.

Acknowledgment. This paper was funded by the National Natural Science Foundation of China (No. 61402344), the Excellent Dissertation Cultivation Funds of Wuhan University of Technology (No. 2017-YS-065) and the Fundamental Research Funds for the Central Universities (WUT:2017II14XZ).

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing: clearing the clouds away from the true potential and obstacles posed by this computing capability. Commun. ACM **53**(4), 50–58 (2010)
2. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandicl, I.: 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)
3. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. In: Grid Computing Environments Workshop, pp. 1–10 (2008)
4. Buyya, R., Yeo, C., Venugopal, S.: Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities. In: The 10th IEEE International Conference on High Performance Computing and Communications, pp. 5–13 (2008)
5. Laatikainen, G., Ojala, A., Mazhelis, O.: Cloud services pricing models. In: Herzwurm, G., Margaria, T. (eds.) ICSOB 2013. LNBIP, vol. 150, pp. 117–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39336-5_12
6. Sharma, B., Thulasiram, R.K., Thulasiraman, P., Garg, S.K., Buyya, R.: Pricing cloud compute commodities: a novel financial economic model. In: The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 451–457 (2012)
7. Wang, H., Jing, Q., Chen, R., He, B., Qian, Z., Zhou, L.: Distributed systems meet economics: pricing in the cloud. In: The 2nd USENIX Conference on Hot Topics in Cloud Computing, pp. 1–6 (2010)
8. Feng, Y., Li, B., Li, B.: Price competition in an oligopoly market with multiple iaas cloud providers. IEEE Trans. Comput. **63**(1), 59–73 (2014)
9. Kantere, V., Dash, D., Francois, G., Kyriakopoulou, S., Ailamaki, A.: Optimal service pricing for a cloud cache. IEEE Trans. Knowl. Data Eng. **23**(9), 1345–1358 (2011)
10. Xu, H., Li, B.: Maximizing revenue with dynamic cloud pricing: the infinite horizon case. In: IEEE International Conference on Communications, pp. 2929–2933 (2012)
11. Vengerov, D.: A gradient-based reinforcement learning approach to dynamic pricing in partially-observable environments. Future Gener. Comput. Syst. **24**(7), 687–693 (2008)
12. Xu, H., Li, B.: Dynamic cloud pricing for revenue maximization. IEEE Trans. Cloud Comput. **1**(2), 158–171 (2013)
13. Xu, B., Qin, T., Qiu, G., Liu, T.Y.: Optimal pricing for the competitive and evolutionary cloud market. In: The 24th International Joint Conference on Artificial Intelligence, pp. 139–145 (2015)
14. Truong-Huu, T., Tham, C.K.: A game-theoretic model for dynamic pricing and competition among cloud providers. In: The 6th International Conference on Utility and Cloud Computing, pp. 235–238 (2013)
15. Truong-Huu, T., Tham, C.K.: A novel model for competition and cooperation among cloud providers. IEEE Trans. Cloud Comput. **2**(3), 251–265 (2014)
16. Wal, J.: Stochastic dynamic programming. Methematisch Centrum (1980)
17. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: 11th International Conference on Machine Learning, pp. 157–163 (1994)
18. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)

19. Adams, I.F., Long, D.D.E., Miller, E.L., Pasupathy, S., Storer, M.W.: Maximizing efficiency by trading storage for computation. In: The 1st USENIX Conference on Hot Topics in Cloud Computing. vol. 7 (2009)
20. Jung, H., Klein, C.M.: Optimal inventory policies under decreasing cost functions via geometric programming. Eur. J. Oper. Res. **132**(3), 628–642 (2001)
21. Train, K.E.: Discrete Choice Methods with Simulation. Cambridge University Press, Cambridge (2003)

An Effective Offloading Trade-Off to Economize Energy and Performance in Edge Computing

Yuting Cao, Haopeng Chen(✉), and Zihao Zhao

Shanghai Jiao Tong University, Dongchuan Rd., Shanghai, China
{seniyuting,chen-hp,1059198449}@sjtu.edu.cn

Abstract. Mobile Edge Computing is a new technology which aims to reduce latency, to ensure highly efficient network operation and to offer an improved user experience. Considering offloading will introduce additional wireless transmission overhead, the key technical challenge of mobile edge computing is tradeoff between computation cost and wireless transmission cost, reducing energy consumption of mobile edge devices and response time of computation task at the same time. A Mobile Edge Computing System composed of Mobile Edge Device and Edge Cloud, connecting with Wireless Stations, comes out. To protect user privacy, Data Preprocessing is proposed which includes irrelevant property clean and data segmentation. Aimed at reducing total energy consumption and response time, an energy consumption priority offloading (ECPO) algorithm and a response time priority offloading (RTPO) algorithm are put forward, based on Energy Consumption Model and Response Time Model. Combining both ECPO and RTPO, a dynamic computing offloading algorithm is raised which is more universal. Finally, simulations in four scenarios, including network normal scenario, network congested scenario, device low battery scenario and task time limited scenario, demonstrate that our algorithms can effectively reduce energy consumption of mobile edge device and response time of computation task.

Keywords: Mobile edge computing · ECPO · RTPO · Offloading

1 Introduction

With the rapid development of Internet of Things [1], mobile edge devices generate a lot of data, performing as not only data consumers but also data producers [2]. Under this circumstance, traditional cloud computing is no longer a wise choice since it has some obvious drawbacks. Firstly, it is not acceptable for real time computation task on mobile edge devices because bandwidth of network would be the bottleneck. Secondly, user privacy is another problem to be solved for cloud computing.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 172–186, 2018.
https://doi.org/10.1007/978-3-319-94295-7_12

Mobile Edge Computing is a new technology which provides an IT service environment and cloud computing capabilities at the edge of mobile network, within the Radio Access Network and in close proximity to mobile subscribers. The aim is to reduce latency, to ensure highly efficient network operation and service delivery, and to offer an improved user experience [3]. In mobile edge computing scenario, network latency can be reduced by enabling computation and storage capacity at the edge network. And mobile edge devices can perform computation offloading for computing intensive applications to leverage the context-aware mobile edge computing service by using real time radio access network information [4]. Since offloading introduces additional communication overhead, a key technical challenge is how to balance between computation cost and communication cost to support applications with enhanced user experience, such as lower response time and energy consumption [5].

In this paper, we design a Mobile Edge Computing System composed of Mobile Edge Device and Edge Cloud, connecting with Wireless Station. To protect user privacy, Data Preprocessing is proposed including irrelevant property clean and data segmentation. Assuming that horizontal segmentation of input data will not affect computing result and execution time of computing is proportional to data size, input data can be divided into multiple data blocks and data block can be divided into multiple data slices with fixed data size. Only one data slice can be executed each time. In our research, we focus on reducing energy consumption of mobile edge device and response time of computation task, proposing an Energy Consumption Model and a Response Time Model.

In terms of mobile edge computing offloading problem, considering the change of network environment and the power of mobile edge device to make a dynamic decision will work better than making an unchanging decision at the beginning. Therefore, an energy consumption priority offloading (ECPO) algorithm and a response time priority offloading (RTPO) algorithm are raised in order to reduce energy consumption and response time. Furthermore, combining both ECPO algorithm and RTPO algorithm, we propose a new dynamic computing offloading algorithm as the greatest contribution of our paper. Finally, simulations in four scenarios: network normal scenario, network congested scenario, device low battery scenario and task time limited scenario, demonstrate that our proposed algorithms can effectively reduce energy consumption and response time.

The structure of this paper is as follows. Section 2 presents the related work. Section 3 describes system design and system model. In Sect. 4, we propose ECPO alogorithm, RTPO algorithm and a dynamic computing offloading algorithm combining previous two. Section 5 shows the simulation results and analysis. Section 6 is about conclusion and future work.

2 Related Work

Recently, computing offloading from mobile devices into cloud [6], as a key technical challenge of Mobile Edge Computing [7], has been extensively studied in many area including power management [8], cloud computing task migration [9,10] and virtual machine migration [11].

In order for better understanding of offloading, Orsini et al. provide a design guideline for the selection of suitable concepts for different classes of common cloud-augmented mobile applications and present open issues that developers and researchers should be aware of when designing their mobile cloud computing approach [12].

There are some existing offloading algorithms, like energy-optimal partial computation offloading (EPCO) algorithm [5], Lyapunov optimization-based dynamic computation offloading (LODCO) algorithm [13], distributed computation offloading algorithm [14] and the actor-model programming paradigm [15]. Sardellitti et al. consider an MIMO multicell system where multiple mobile users ask for computation offloading to a common cloud server and propose an iterative algorithm, based on a novel successive convex approximation technique, converging to a local optimal solution of original nonconvex problem [16].

In order to save energy, Zhao et al. design a threshold-based policy to improve the QoS of Mobile Cloud Computing by cooperation of local cloud and Internet cloud resources, which takes advantages of low latency of local cloud and abundant computational resources of Internet cloud simultaneously [17]. Ge et al. propose a game-theoretic approach to optimize the overall energy in a mobile cloud computing system and formulate the energy minimization problem as a congestion game, where each mobile device is a player to select one server to offload computation to minimize the overall energy consumption [18]. Wang and Giannakis investigate resource allocation policies for time-division multiple access over fading channels in the power-limited regime [19].

What's more, Chen et al. study the multi-user computation offloading problem for mobile-edge cloud computing in a multi-channel wireless interference environment and design a distributed computation offloading algorithm that can achieve a Nash equilibrium, deriving the upper bound of the convergence time and quantifying its efficiency ratio over the centralized optimal solution in terms of two important performance metrics [14]. You et al. study resource allocation for a multiuser MECO system based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA) to solve the problem and characterize its policy structure, proposing a low-complexity sub-optimal algorithm by transforming the OFDMA problem to its TDMA counterpart [20].

3 System Design and System Model

In this section, we introduce system architecture design and data preprocessing method. The energy consumption model and response time model are also presented. Partial parameters used in the following are listed in Table 1.

3.1 System Architecture Design

The system is composed of two requisite parts: Mobile Edge Device and Edge Cloud, while data transmission of them is Wireless Station. The system architecture is shown in Fig. 1.

Table 1. Partial parameters

	Description
C_e	Number of CPU cycle required for computing 1-bit data at edge cloud
C_m	Number of CPU cycle required for computing 1-bit data at mobile edge device
D_l	Size of data slice in local computing
D_t	Size of data slice for offloading
D	Size of the whole data slice
E_c	Energy consumption per CPU cycle for a mobile edge device
E_l	Energy consumption for local computing of one data slice
E_t	Energy consumption for wireless transmission of one data slice
E	Energy consumption to compute the whole data slice
E_{total}	Total energy consumption of one computing task
f_e	CPU frequency for edge cloud
f_m	CPU frequency for mobile edge device
g	Channel gain
N	Variance of complex white Gaussian channel noise
P_t	Transmission power
R_t	Channel transmission rate
T_e	Response time for computing one data slice in edge cloud
T_l	Response time for local computing of one data slice
T_t	Transmission time of one data slice
T	Response time to compute the whole data slice
T_{total}	Total response time of one computation task
W	Channel bandwidth

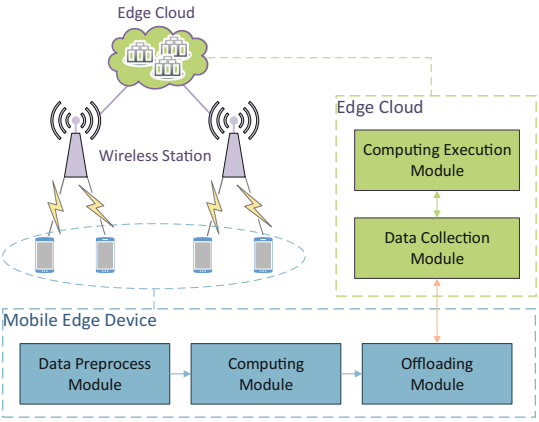


Fig. 1. System architecture

Mobile Edge Device. Mobile Edge Device is both data consumer and data producer in edge computing, including three modules: Data Preprocess Module, Computing Module and Offloading Module. Data Preprocess Module is responsible for irrelevant property clean and data segmentation. Computing Module is used for local computing. Offloading Module is used to develop calculation migration strategy.

Edge Cloud. There are two modules in Edge Cloud: Computing Execution Module and Data Collection Module. Computing Execution Module is a module that performs computational tasks, while Data Collection Module is applied for collecting data from Mobile Edge Device.

3.2 Data Preprocessing

There are two kinds of data preprocessing: Irrelevant Property Clean and Data Segmentation. On the one hand, the original data may contain some properties which are related to user private and computing irrelevant. On the other hand, in the case of horizontal segmentation of input data not affecting computing result, we propose to segment from multiple granularities on the basic definition of Data Unit, smallest data processing unit for performing computational tasks. Data Unit can be divided into multiple Data Blocks while Data Block can be divided into multiple Data Slices as shown in Fig. 2. The purpose of multi-granularity data segmentation is to dynamically adjust the computing migration mechanism for different granularity data which will be analyzed in Sect. 4.

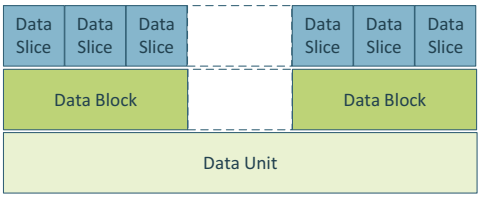


Fig. 2. Data segmentation

3.3 Energy Consumption Model

One of the main purposes of mobile edge computing is to reduce energy consumption of mobile edge devices as much as possible, including energy consumption for local computing and for wireless transmission.

Energy Consumption for Local Computing. For a mobile edge device, let E_c denote the energy consumption per CPU cycle for local computing, C_m denote the number of CPU cycles required for computing 1-bit of input data and D_l denote the size of data slice. Then the total energy consumption for local computing of a data slice is E_l, given by Formula 1.

$$E_l = C_m \cdot D_l \cdot E_c \tag{1}$$

Energy Consumption for Wireless Transmission. Let W denote the channel bandwidth, g denote the channel gain, P_t denote the transmission power and N denote the variance of complex white Gaussian channel noise. Then the channel transmission rate denoted by R_t, is Formula 2, according to Shannon formula.

$$R_t = W \cdot \log_2(1 + \frac{g \cdot P_t}{W \cdot N}) \tag{2}$$

Provided that W, P_t and N are constant, R_t will have a positive correlation with changeable g. However, if time slot is short enough, P_t and R_t can be considered constant, resulting to the total energy consumption for wireless transmission of a data slice as E_t, given by Formula 3, where T_t and D_t are transmission time and size of offloading data slice respectively.

$$E_t = P_t \cdot T_t = \frac{P_t \cdot D_t}{R_t} \tag{3}$$

By the way, energy consumption for wireless transmission of calculation result is negligible in this case due to size of result is much smaller than that of original data normally.

Total Energy Consumption. Considering the size of data slice is small enough, the local computing time and transmission time of data slice is short. Under this circumstances, analysis of energy consumption is acceptable.

Let D denote size of the whole data slice, D_l denote the size of data slice in local computing with $0 \leq D_l \leq D$, then the size of offloading data slice is $D_t = D - D_l$. Total energy consumption denoted as E is given by Formula 4.

$$E = E_l + E_t = \frac{P_t}{R_t} \cdot D + (C_m \cdot E_c - \frac{P_t}{R_t}) \cdot D_l \tag{4}$$

For the case of $C_m \cdot E_c - \frac{P_t}{R_t} > 0$, E will be minimum when $D_l = 0$. Oppositely, supposing $C_m \cdot E_c - \frac{P_t}{R_t} \leq 0$, E will be minimum when $D_l = D$. It shows that for a data slice small enough, to make energy consumption minimum, the whole data slice is either executed in local computing or offloaded to edge cloud.

The total energy consumption of one computation task can be expressed as Formula 5.

$$E_{total} = \sum E_{slice}, \ \textit{for each data slice} \tag{5}$$

3.4 Response Time Model

Another purpose of mobile edge computing is to reduce computing response time, which is mainly affected by three factors: local computing, wireless transmission and cloud computing.

Response Time for Local Computing. For a mobile edge device, let f_m denote CPU frequency for local computing which means the number of CPU cycles per second, C_m denote the number of CPU cycles required for computing 1-bit data and D_l denote the size of data slice in local computing. Then the total response time for local computing of a data slice is T_l, given by the following Formula 6.

$$T_l = \frac{C_m \cdot D_l}{f_m} \tag{6}$$

Response Time for Wireless Transmission. If the time slot is short enough, the channel transmission rate R_t can be considered to be constant. In this case, the total response time for wireless transmission of a data slice of input data is T_t, given by Formula 7.

$$T_t = \frac{D_t}{R_t} = \frac{D - D_l}{R_t} \tag{7}$$

where D_t is the size of offloading data slice. Response time for wireless transmission of computing results is ignored due to the relative smaller sizes.

Response Time for Cloud Computing. Obviously in edge cloud, the CPU frequency denoted as f_e and the number of CPU cycles required for computing 1-bit data denoted as C_e can be considered constant within a time slot which is short enough. Then the total response time for cloud computing of a data slice is T_e, given by Formula 8.

$$T_e = \frac{C_e \cdot D_t}{f_e} = \frac{C_e \cdot (D - D_l)}{f_e} \tag{8}$$

Total Response Time. For a data slice, local execution and offloading to edge cloud are simultaneous, which leads to the total response time be the maximum of them, given by Formula 9.

$$T = max(T_l, T_t + T_e) \tag{9}$$

Total response time of a computation task can be expressed as Formula 10.

$$T_{total} = \sum T_{slice}, \ for\ each\ data\ slice \tag{10}$$

4 Computing Offloading Mechanism

Reducing energy consumption and response time is two main purpose for computing offloading mechanism in edge computing. In this section, we proposed an energy consumption priority offloading algorithm named ECPO and a response time priority offloading algorithm named RTPO. Ultimately, we put forword a dynamic computing offloading algorithm based on ECPO and RTPO.

4.1 Energy Consumption Priority Offloading Algorithm

To reduce the total energy consumption of mobile edge device, an energy consumption priority offloading (ECPO) algorithm is proposed in Algorithm 1.

Algorithm 1: ECPO Algorithm

```
Input: data: input data or data block
Divide data into data slices
for each DataSlice do
    if DataSlice is non-offloadable then
        Execute DataSlice in local computing
    else
        Get connection information of wireless station
        Calculate C_m, E_c, P_t and R_t in Formula 4
        Set Param = C_m · E_c − P_t/R_t
        if Param > 0 then
            Offloading DataSlice to Edge Cloud
        else
            Execute DataSlice in local computing
```

As is shown in Fig. 3, in ECPO algorithm, data will be divided into multiple data slices with fixed data size under the assumption that horizontal segmentation will not affect computing result. For each data slice, it will be either executed in local computing or offloaded to edge cloud. Only one data slice can be executed each time. What's more, total energy consumption of one data slice E in ECPO algorithm is shown in Formula 4. As is proved by Energy Consumption Model, E will be minimum when using ECPO algorithm.

4.2 Response Time Priority Offloading Algorithm

Considering that ECPO algorithm has space for improving response time of computation task, an response time priority offloading (RTPO) algorithm is proposed in Algorithm 2.

As is shown in Fig. 4, in RTPO algorithm, the offloading weight of a data slice is calculated according to the power percentage of mobile edge device and

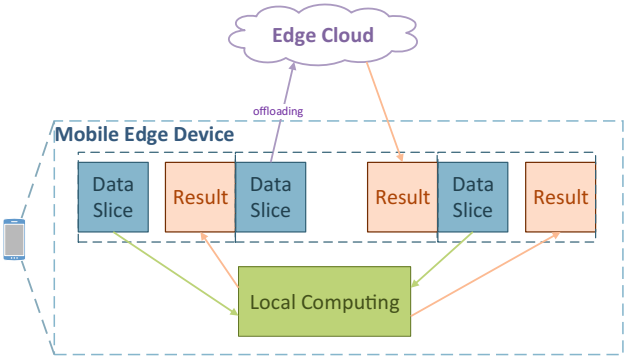


Fig. 3. Energy consumption priority offloading

Algorithm 2: RTPO Algorithm

Input: *data*: input data or data block
Divide *data* into data slices
for *each DataSlice* **do**
  if *DataSlice is non-offloadable* **then**
    Execute *DataSlice* in local computing
  else
    Get connection information of wireless station
    Calculate current channel transmission rate R_t
    $D_l = (f_m \cdot f_e + C_e \cdot f_m \cdot R_t) \cdot D/(C_m \cdot f_e \cdot R_t + f_m \cdot f_e + C_e \cdot f_m \cdot R_t)$
    Execute local computing and offload to Edge Cloud at the same time

network status. In other words, RTPO algorithm allows one data slice to execute both in local and edge cloud to earn less response time.

For each data slice, the response time for executing local computing to process a data unit is $\frac{C_m}{f_m}$, while time for offloading to edge cloud is $\frac{1}{R_t} + \frac{C_e}{f_e}$. So for a data slice, in order to equalize this two latencies, we get the weight for local computing D_l in Formula 11. As is analyzed previously in Formula 4, total energy consumption of one data slice E in RTPO algorithm will not be the worst case as D_l always satisfy the condition $0 < D_l < D$.

$$D_l = \frac{(f_m \cdot f_e + C_e \cdot f_m \cdot R_t) \cdot D}{C_m \cdot f_e \cdot R_t + f_m \cdot f_e + C_e \cdot f_m \cdot R_t} \tag{11}$$

4.3 Dynamic Computing Offloading Algorithm

In order to deal with more complex scenarios, we propose a dynamic computing offloading algorithm in Algorithm 3, combining ECPO and RTPO algorithm.

The dynamic computing offloading algorithm takes both power of mobile edge device and user requirements into consideration. On the one hand, if mobile edge

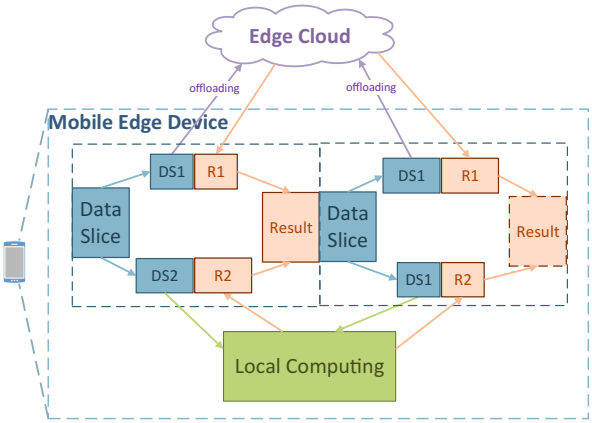


Fig. 4. Response time priority offloading

Algorithm 3: Dynamic Computing Offloading Algorithm

Input: w_e: weight of energy consumption, w_t: weight of response time

Get current power of Mobile Edge Device: *power*
Get maximum power of Mobile Edge Device: *power_max*
Get expected minimum energy consumption: E_{min}
if $power < E_{min}$ **then**
    return
else
    Divide input data into data blocks
    for *each DataBlock* **do**
        Divide *DataBlock* into data slices
        Get current power of Mobile Edge Device: *power*
        if $power > power\_max \cdot threshold\_power$ **then**
            Calculate $E_{ECPO}, E_{RTPO}, T_{ECPO}, T_{RTPO}$
            Set $Overhead_{ECPO} = E_{ECPO} \cdot w_e + T_{ECPO} \cdot w_t$
            Set $Overhead_{RTPO} = E_{RTPO} \cdot w_e + T_{RTPO} \cdot w_t$
            if $Overhead_{RTPO} < Overhead_{ECPO}$ **then**
                Use RTPO algorithm
            else
                Use ECPO algorithm
        else
            Use ECPO algorithm

device does not have enough power to support, task will be abandoned. Besides, let *threshold_power* denote the threshold of percentage of battery charge and compare power before processing each data block. On the other hand, let w_e and w_t denote the weight of reducing energy consumption and shortening response time respectively according to user requirements to calculate overhead as

Formula 12. Finally, we select algorithm with lower overhead to deal with current data block.

$$Overhead = E \cdot w_e + T \cdot w_t \tag{12}$$

The energy consumption model and time model of one computaion task are the sum of cost in each data block, which can be expressed as Formula 13.

$$\begin{aligned} E_{total} &= \sum E_{block},\ for\ each\ data\ block \\ T_{total} &= \sum T_{block},\ for\ each\ data\ block \end{aligned} \tag{13}$$

5 Evaluation

As 5G technology has not fully applied currently, it is scarcely possible to carry out experiments in real environment. In this section, we analyze our simulation result in four scenarios to evaluate performance of the proposed dynamic computing offloading mechanism, including network normal scenario, network congested scenario, device low battery scenario and task time limited scenario.

5.1 Experimental Parameter

Settings of our simulation are as follows. The size of data block and data slice are 100 MB and 1 MB separately. Energy consumption per CPU cycle for mobile edge device $E_c = 1.0 \times 10^{-10}$ J/cycle. The number of CPU cycles required for computing 1-bit data at mobile edge device and edge cloud is $C_m = C_e = 1000$ cycles/bits. CPU frequency for a mobile edge device is $f_m = 2.0 \times 10^9$ cycles/s while it for edge cloud is $f_e = 1.0 \times 10^{10}$ cycles/s. Transmission power is $P_t \in [0, 0.2]$ J/s. Channel transmission rate is $R_t \in [0, 2000]$ KB/s. Energy of mobile edge device at full charge is 20000 J and *threshold_power* is set to 30%.

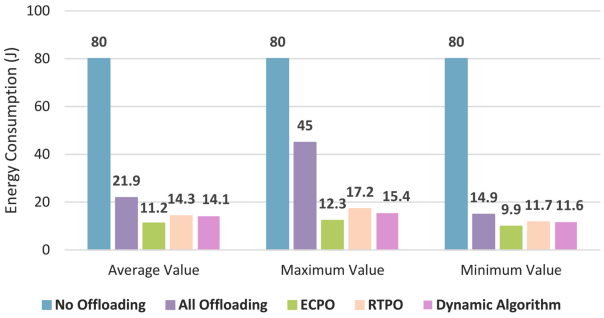
Moreover, as Formula 14 shows, $WirelessStation$ consists of n channels and m devices are maintained for each channel, of which m has negative correlation of the wireless transmission rate.

$$\begin{aligned} &WirelessStation(Channel_1, Channel_2, ..., Channel_{n-1}, Channel_n) \\ &Channel(Device_1, Device_2, ..., Device_{m-1}, Device_m) \end{aligned} \tag{14}$$

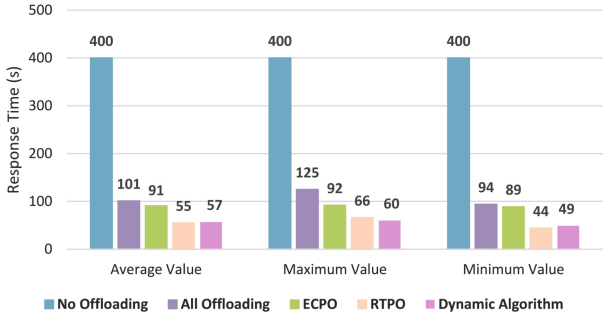
Assuming that $Number_{Channel}$ and $Number_{Device}$ represent the number of channels and mobile edge devices severally, we define the ratio of $Number_{Channel}$ and $Number_{Device}$ equalizing 20 as congested network, while equalizing 2 as normal network in our experiment.

5.2 Simulation Results Analysis

Network Normal Scenario. Processing computation task with size of 100 MB in normal network scenario with $Number_{Device}/Number_{Channel} = 2$, total energy consumption and response time of edge device are shown in Fig. 5. Apparently, ECPO, RTPO and Dynamic Algorithm have a great advantage over executing locally and all offloading to edge cloud. However, in terms of energy consumption, compared with ECPO and RTPO, advantage of Dynamic Algorithm is not obvious, as even slightly worse than ECPO. While in terms of response time, combining the superiority of RTPO, it performs better for Dynamic Algorithm than ECPO while a little worse than RTPO. Therefore, Dynamic Algorithm plays a role as a compromise between ECPO and RTPO.



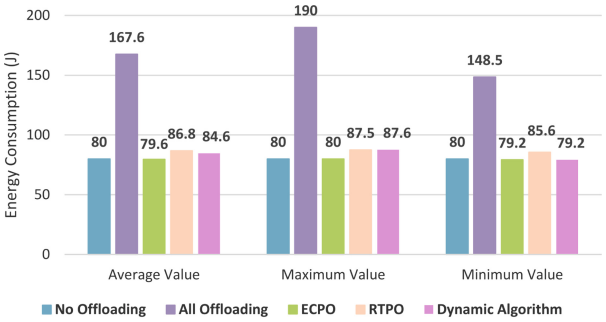
(a) Energy Consumption Analysis



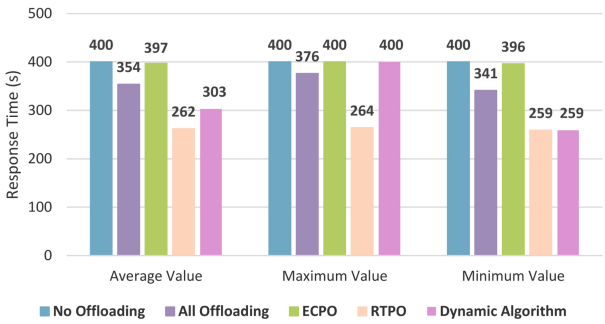
(b) Response Time Analysis

Fig. 5. Analysis in normal network scenario

Network Congested Scenario. Dealing with task with size of 100 MB in congested network scenario with $Number_{Device}/Number_{Channel} = 20$, results are shown in Fig. 6. In the respect of energy consumption, cost of Dynamic Algorithm is slightly higher than local execution due to high transmission cost in congested network, while is half as low as it of all offloading. From another perspective, response time of Dynamic Algorithm has distinct improvement over no offloading and all offloading, but still in the midst of ECPO and RTPO.



(a) Energy Consumption Analysis



(b) Response Time Analysis

Fig. 6. Analysis in Congested Network Scenario

Device Low Battery Scenario. The initial power of mobile edge device is set lower than 30% randomly in low battery scenario and the original data size of each task is 5 GB. Number of initial tasks is 100. Results of computing offloading is shown in Table 2. The failure of computing tasks is due to exhaustion of mobile edge devices. Success rates of ECPO and RTPO are 91% and 75% each. Dynamic Algorithm will estimate energy consumption before executing tasks and abandon calculation task when do not have enough power, resulting to 100% success rate.

Table 2. Result of computing offloading in device low battery scenario

	Initialization	Abandon	Failure	Success	SuccessRate
ECPO	100	0	9	91	91%
RTPO	100	0	25	75	75%
Dynamic	100	10	0	90	100%

Task Time Limited Scenario The initial power of mobile edge device is set randomly and the original data size of each task is 5 GB. Number of initial tasks is 100. In addition, each computing task is limited to 1 h. Results of computing offloading is shown in Table 3. For ECPO, there are 2 tasks failed due to depletion of mobile edge devices, while the remaining 98% violate the time limit due to timeout. For RTPO, although its success rate is 95%, 5 tasks failed due to exhaustion of mobile edge device. For Dynamic Algorithm, success rate is 83% without failure which is optimal entirely.

Table 3. Result of computing offloading in task time limited scenario

	Initialization	Abandon	Failure	Timeout	Punctuality	SuccessRate
ECPO	100	0	2	98	0	0%
RTPO	100	0	5	0	95	95%
Dynamic	100	4	0	16	80	83%

5.3 Brief Summary

The simulation results conducted in 4 scenarios show that our computing offloading machanism can comprehensively consider energy consumption, response time of computing tasks and load of edge cloud, by dynamically adjusting and calculating offloading strategy. The machanism is far better than local execution and all offloading to edge cloud.

6 Conclusion

We study computing offloading mechanism for a mobile edge computing system composed of mobile edge device and edge cloud, connecting with wireless station, to reduce energy consumption of mobile edge devices and response time of computation tasks. We propose ECPO and RTPO algorithm according to Energy Consumption Model and Response Time Model. Ultimately, we put forward a dynamic offloading algorithm combining the previous two, which is proved to be an effective way to achieve the original goals entirely. In the future, after full application of 5G technology, we will make further effort to build a real mobile edge computing scenarios to verify the effectiveness of our mechanism.

References

1. Xia, F., Yang, L.T., Wang, L., Vinel, A.: Internet of Things. Int. J. Commun. Syst. **25**(9), 1101 (2012)
2. Shi, W., Dustdar, S.: The promise of edge computing. Computer **49**(5), 78–81 (2016)

3. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing a key technology towards 5G. ETSI White Pap. **11**(11), 1–16 (2015)
4. Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1–8 (2016)
5. Wang, Y., Sheng, M., Wang, X., Wang, L., Li, J.: Mobile-edge computing: partial computation offloading using dynamic voltage scaling. IEEE Trans. Commun. **64**(10), 4268–4282 (2016)
6. Kumar, K., Lu, Y.H.: Cloud computing for mobile users: can offloading computation save energy? Computer **43**(4), 51–56 (2010)
7. Dinh, H.T., Lee, C., Niyato, D., Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wirel. Commun. Mob. Comput. **13**(18), 1587–1611 (2013)
8. Van, H.N., Tran, F.D., Menaud, J.M.: Performance and power management for cloud infrastructures. In: 2010 IEEE 3rd International Conference Cloud Computing (CLOUD), pp. 329–336 (2010)
9. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer Systems, pp. 301–314 (2011)
10. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: INFOCOM, 2012 Proceedings IEEE, pp. 945–953 (2012)
11. Xiao, Z., Song, W., Chen, Q.: Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1107–1117 (2013)
12. Orsini, G., Bade, D., Lamersdorf, W.: Context-aware computation offloading for mobile cloud computing: requirements analysis, survey and design guideline. Procedia Comput. Sci. **56**, 10–17 (2015)
13. Mao, Y., Zhang, J., Letaief, K.B.: Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J. Sel. Areas Commun. **34**(12), 3590–3605 (2016)
14. 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)
15. Haubenwaller, A.M., Vandikas, K.: Computations on the edge in the internet of Things. Procedia Comput. Sci. **52**, 29–34 (2015)
16. Sardellitti, S., Scutari, G., Barbarossa, S.: Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Sig. Inf. Process. Netw. **1**(2), 89–103 (2015)
17. Zhao, T., Zhou, S., Guo, X., Zhao, Y., Niu, Z.: A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. In: 2015 IEEE Globecom Workshops (GC Wkshps), pp. 1–6 (2015)
18. Ge, Y., Zhang, Y., Qiu, Q., Lu, Y.H.: A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 279–284 (2012)
19. Wang, X., Giannakis, G.B.: Power-efficient resource allocation for time-division multiple access over fading channels. IEEE Trans. Inf. Theory **54**(3), 1225–1240 (2008)
20. You, C., Huang, K., Chae, H., Kim, B.H.: Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wirel. Commun. **16**(3), 1397–1411 (2017)

Research Track: Cloud Management

Implementation and Comparative Evaluation of an Outsourcing Approach to Real-Time Network Services in Commodity Hosted Environments

Oscar Garcia[1(✉)], Yasushi Shinjo[1(✉)], and Calton Pu[2(✉)]

[1] Department of Computer Science, University of Tsukuba, Tsukuba, Ibaraki, Japan
oscar@softlab.cs.tsukuba.ac.jp, yas@cs.tsukuba.ac.jp
[2] College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
calton.pu@cc.gatech.edu

Abstract. Commodity operating systems (OS) often sacrifice real-time (RT) performance (e.g., consistent low latency) in favor of optimized average latency and throughput. This can cause latency variance problems when an OS hosts virtual machines that run network services. This paper proposes a software-based RT method in Linux KVM-based hosted environments. First, this method solves the priority inversion problem in interrupt handling of vanilla Linux using the RT Preempt patch. Second, this method solves another priority inversion problem in the softirq mechanism of Linux by explicitly separating the RT softirq handling from the non-RT softirq handling. Finally, this method mitigates the cache pollution problem by co-located non-RT services and avoids the second priority inversion in a guest OS by socket outsourcing. Compared to the RT Preempt Patch Only method, the proposed method has the 76% lower standard deviation, 15% higher throughput, and 33% lower CPU overhead. Compared to the dedicated processor method, the proposed method has the 63% lower standard deviation, higher total throughput by a factor of 2, and avoids under-utilization of the dedicated processor.

1 Introduction

Large-scale applications with real-time (RT) or stringent quality of service (QoS) requirements, ranging from large distributed systems such as electronic trading, NextGen air traffic control [1], and web-facing e-commerce n-tier applications to small sensors and edge servers in Internet of Things (IoT), smart applications need fast and consistent network services [2–6]. Owing to the variety of environments in which these applications operate, there is a growing need for commodity operating system (OS) with systematic improvements that can satisfy real-time and stringent QoS performance requirements in both speed and consistency. An indication of this trend is the number of real-time operating systems [7] that build on commodity operating systems (Sect. 5).

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 189–205, 2018.
https://doi.org/10.1007/978-3-319-94295-7_13

Because of the need and effort to maximize average performance in commodity operating systems, they tend to have larger latency variabilities and wider spread in the latency of kernel services, particularly for I/O devices. These latency variabilities are not bugs in the traditional sense, since the system functionality is correctly implemented. These latency variance problems can also cause problems in hosted virtual machines (VM) that run network services.

In this study, we have found three major sources of variance in "vanilla" Linux: two priority inversion problems (Sects. 2.1 and 2.3) and cache pollution by co-located non-RT services (Sect. 2.5). We describe a new approach to real-time network services in Linux KVM-based commodity hosted environments, and evaluate our approach by comparing two current production RT methods. We call our approach the "outsourcing plus separation of RT Softirq" method or the outsourcing method for short. First, we solve the first priority inversion problem in interrupt handling of vanilla Linux using the RT Preempt patch [8]. However, using this patch only has the second priority inversion problem in the softirq mechanism of Linux in the host OS (Sect. 2.3). We can avoid this problem by dedicating a processor exclusively for RT threads (Sect. 2.4). However, this method has disadvantages of low CPU utilization and low total throughput. Therefore, we solve the second priority inversion problem in the host OS by explicitly separating the RT softirq handling from the non-RT softirq handling (Sect. 3.1). Finally, we mitigate the cache pollution problem and avoid the second priority inversion in a guest OS by outsourcing [9] (Sect. 3.2).

Compared to the RT Preempt Patch Only method, the outsourcing method has the 76% lower standard deviation, 33% lower CPU overhead, and 15% higher throughput. Compared to the dedicated processor method, the outsourcing method has the 63% lower standard deviation and higher total throughput by a factor of 2, and avoids under-utilization of the dedicated processor.

2 The Latency Variance Problems in Vanilla Linux and Two Production RT Methods

In this section, we illustrate the latency variance problems in vanilla Linux and two representative production RT methods with a common mix of an RT service and co-located non-RT network services called NetRT and NetStream (Fig. 1). A NetRT server is an RT network service that receives requests from clients sporadically and replies with response messages to the clients. A NetStream server is a non-RT network service that receives messages continuously from clients in a best effort manner but it does not send response messages. While a NetRT server requires short and predictable response times, a NetStream server desires high throughput.

The NetRT server uses an RT network and the NetStream servers use non-RT networks. In the following, we refer to the Network Interface Cards (NICs) connected to these networks as RT NIC and non-RT NICs, respectively. We assume that the network delay and bandwidth of the RT network are guaranteed by using the methods described by [10–13]. The non-RT networks are best-effort networks.

In this figure, we run a network service in a VM. We allocate a single vCPU to each VM of NetStream, and the vCPU corresponds to a host thread with a normal priority. We allocate two vCPUs to the VM of NetRT. The first vCPU (non-RT vCPU) executes system tasks (e.g. housekeeping tasks) and it corresponds to a host thread with a normal priority. The second vCPU (RT vCPU) executes the NetRT server and it corresponds to a host thread with a high priority. Each VM has a vNIC thread that executes a backend driver of the VM network.

2.1 Interrupt Handling of Network Devices in Vanilla Linux

Linux implements the split interrupt handling model to handle interrupts. Figure 1a shows the interrupt handling in vanilla Linux. Each device driver of a NIC has two interrupt handlers: the *hard IRQ handler* and *softirq handler*. The hard IRQ processes the essential interrupt tasks while interrupts are disabled and the softirq handler processes the remaining interrupt tasks including heavy TCP and bridge processing while interrupts are enabled.

Each CPU has an instance of the softirq mechanism and this instance is shared by multiple device drivers. To ensure cache affinity, the softirq handler of a device driver is executed by the same CPU that receives the IRQ from the device and that executes the hard IRQ handler. The hard IRQ handler of a NIC put the RT softirq handler into the *poll_list*, which is the list of pending softirq handlers in a per-CPU variable.

The interrupt handling in vanilla Linux has a priority inversion problem. Interrupt handlers are executed prior to user processes. For instance, in Fig. 1a, the RT vCPU thread, which is a high-priority user process, can be delayed by the softirq handler of a non-RT NIC.

In this configuration, message copying is performed two times, i.e., once between the host kernel and a guest kernel by a vNIC thread, and another from the guest kernel to a guest user process by a guest kernel. This message copying causes a cache pollution problem. We will discuss about it in Sect. 2.5.

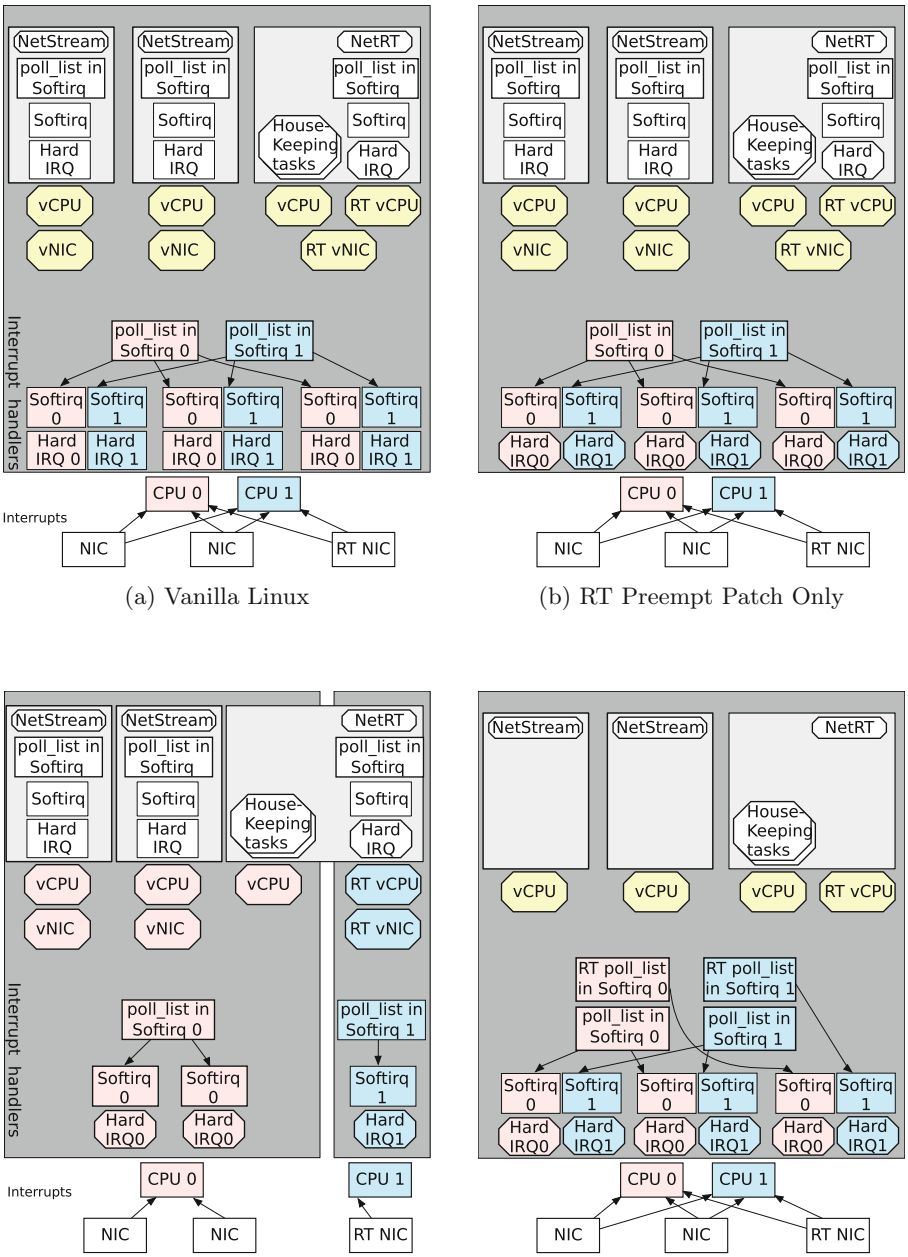
2.2 The RT Preempt Patch Only Method

We can solve the priority inversion problem in Sect. 2.1 by using the RT Preempt patch [8]. We call this method the *RT Preempt Patch Only method*. This patch makes the kernel more preemptible by the following features:

- Hard IRQ handlers are executed as threads (IRQ threads).
- Spin locks are translated into mutexes with a priority inheritance function.

Figure 1b shows that the threads of VMs and interrupt handlers in the RT Preempt Patch Only method. In this figure, each NIC has two IRQ threads for two CPUs. Each IRQ thread is bounded to a CPU.

This method solves the priority inversion problem in Sect. 2.1. Each IRQ thread calls hard and softirq handlers with its own priority. In Fig. 1b, the IRQ



(a) Vanilla Linux

(b) RT Preempt Patch Only

(c) Dedicated processor

(d) Outsourcing

Fig. 1. Configurations of vanilla Linux and the RT methods.

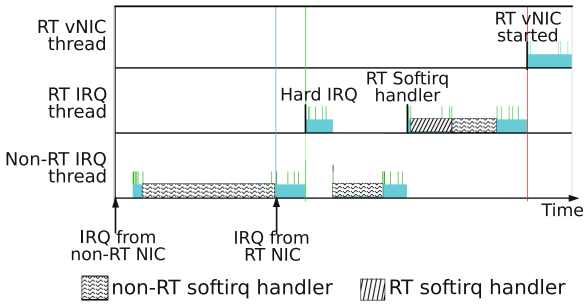


Fig. 2. Priority inversion in interrupt handling of the host OS.

threads of the non-RT NICs do not preempt the threads of the RT VM. This method can yield high achievable CPU utilization because all CPUs execute any vCPU and vNIC threads.

2.3 The Priority Inversion in the Softirq Mechanism of Linux

The RT Preempt patch solves the priority inversion in Sect. 2.1. However, there exists another type of priority inversion in the softirq mechanism of Linux.

Figure 2 shows a trace of the kernel activities using the RT Preempt Patch Only method, where we obtained this plot using KernelShark [14][1]. In this figure, while the CPU was executing the non-RT IRQ thread that called the non-RT softirq handler, the CPU received an interrupt from an RT NIC. The CPU activated the RT IRQ thread, and called the RT hard IRQ handler. The RT hard IRQ handler put the RT softirq handler into the poll_list in a per-CPU variable.

After finishing the RT hard IRQ handler, the RT IRQ thread entered the softirq mechanism. This thread tried to lock the per-CPU variable but it was locked by the non-RT IRQ thread. Therefore, the CPU suspended the RT IRQ thread and executed the non-RT IRQ thread. This thread continued the non-RT softirq handler. At this time, this thread ran with a high priority according to the priority inheritance function. Therefore, the RT IRQ thread had to wait until the non-RT IRQ thread finished. This is a priority inversion.

Next, in Fig. 2, the non-RT softirq handler exceeded the execution quota. Therefore, the non-RT softirq handler was put at the end of the poll_list. Next, the RT IRQ thread obtained the lock and called the RT softirq handler. This handler processed messages from the RT NIC, placed messages into a queue, and activated the RT vNIC thread. Next, the RT IRQ thread also called the non-RT softirq handler with high priority, which created a priority inversion.

[1] The results of KernelShark may include large probe effects.

2.4 Dedicated Processor Method

We can avoid the priority inversion in Sect. 2.3 by dedicating processors to RT threads. This is a popular production method to run RT services in commodity hosted environments [15–18]. We call this method *the dedicated processor method*.

Figure 1c shows that this method allocates a group of RT threads to a dedicated CPU. We call such a CPU an RT-CPU. In this method, an RT NIC injects interrupts to an RT-CPU and a non-RT NIC injects interrupts to a non-RT CPU. Interrupt handling of non-RT NICs do not disturb the execution of the RT threads.

While this method can achieve consistently low latency, it has a drawback. Because RT CPUs do not help to execute non-RT threads, this method yields less achievable CPU utilization.

2.5 Cache Pollution Problem

The CPUs (cores) of a system are explicitly-shared resources and we can control them through priorities of threads and dedication. On the other hand, the Last Level Cache (LLC) is an implicitly-shared resource. When co-located non-RT services pollute the LLC, this can interfere the execution of RT services. For example, in Fig. 1b, when the NetStream servers receive messages, vNIC threads and guest operating systems copy these messages and this copying pollutes the LLC. This changes the response times of the NetRT server.

Note that we cannot avoid this problem using the dedicated processor method. This is because in many CPU architectures, the LLC is shared among CPU cores.

3 Outsourcing Method

Sections 2.2 and 2.4 described two production RT methods, the RT Preempt Patch only method and the dedicated processor method. This section shows our proposed method, *the outsourcing method*.

Figure 1d shows the configuration of the method. This method is an extension of the RT Preempt Patch Only method. It uses the RT Preempt patch and it assigns high priorities to RT threads. This method avoids the priority inversion in Sect. 2.3 by adding a new poll_list for RT services as shown in Fig. 1d. In addition, this method mitigates the cache pollution problem in Sect. 2.5 and avoids the priority inversion in a guest OS by RT socket outsourcing.

3.1 Adding an RT Poll_list for RT Services

In the outsourcing method, we divide the poll_list of the softirq mechanism into two lists.

- The poll_list for non-RT NIC handlers (the non-RT poll_list).
- The poll_list for RT NIC handlers (the RT poll_list).

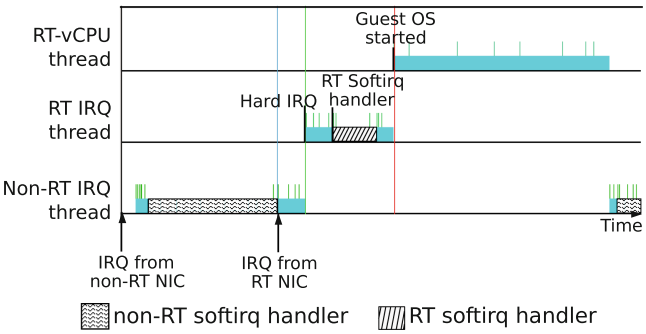


Fig. 3. Separating RT interrupt handling from non-RT interrupt handling.

Figure 3 shows interrupt handling in the outsourcing method. While the CPU was executing the non-RT softirq handler, the CPU received an interrupt from an RT NIC as in Fig. 2. The CPU activated the RT IRQ thread, and called the RT hard IRQ handler. The RT hard IRQ handler placed the RT softirq handler into the RT poll_list.

After finishing the RT hard IRQ handler, the RT IRQ thread entered the softirq mechanism. This thread obtained the lock of the per-CPU variable and called the RT softirq handler. In contrast to the RT Preempt Patch Only method, it called the RT softirq handler and it did not call the non-RT softirq handler because the RT poll_list only contained the RT softirq handler. After the RT IRQ thread finished processing the RT message, it made the CPU available to the RT vCPU thread. In contrast to Fig. 2, there is no priority inversion in Fig. 3.

We implemented the RT poll_list in Linux, which required the changing of 150 lines of code. First, we added the code for the RT poll_list by duplicating that for the base poll_list. Second, we changed the function `napi_schedule_irqoff()`. This function uses the RT poll_list instead of the non-RT poll_list if the IRQ thread is labeled as RT. This allows the reuse of the existing device drivers without modification. We did not change the device driver of the Intel X520 NIC. We added the sysctl parameter `net.core.rtnet_prio` to label the IRQ threads as RT. For example, by calling `sysctl -w net.core.rtnet_prio=47`, an IRQ thread running with a priority equal to or higher than 47 uses the RT poll_list in `napi_schedule_irqoff()`.

3.2 RT Socket Outsourcing

To overcome the latency variance caused by cache pollution, we extend socket outsourcing [9]. Socket outsourcing allows a guest kernel to delegate high-level network operations to the host kernel. When a guest process invokes a socket operation, its processing is delegated to the host. The incoming network messages arriving at a guest process are handled by the host network stack.

Socket outsourcing is implemented using VMRPCs [9]. VMRPCs are remote procedure calls for hosted VMs that allow a guest client to invoke a procedure

on a host server. The parameters for this procedure are passed through the shared memory. VMRPCs are synchronous in a similar manner to system calls, so a naive implementation may block a client until the host procedure returns a response message. Conventional socket outsourcing uses virtual interrupts to solve this problem. Therefore, it has the priority inversion problem in the softirq mechanism of a guest OS.

We solve this problem by eliminating interrupt handling from a guest OS. We call this *RT socket outsourcing*. In RT socket outsourcing, the vCPU running an RT server waits for incoming messages in the idle process. The idle process in the guest OS executes the halt instruction and this makes the vCPU thread sleep in the host OS. When new messages arrive in the host OS, the host vCPU thread and the guest idle process are resumed. The idle process checks the event queue and the structure with the states of the sockets in the shared memory, wakes up the receiving processes and goes back to the scheduler. The scheduler executes these processes immediately without interrupt handling. Further, receiving new messages does not interfere with a running RT service. When the RT service is running and a new message arrives, the guest kernel does not handle the new message immediately. The guest kernel handles it when the RT service issues a receive system call or the guest kernel becomes idle.

Message copying is performed two times in current production methods, including the RT Preempt patch method and dedicated processor method, i.e., once between the host kernel and a guest kernel, and another from the guest kernel to a guest user process. By contrast, in socket outsourcing, message copying is performed once from the host kernel to a guest user process. This omission of copying makes the footprint smaller and reduces the cache pollution by non-RT services. This contributes lower latency variance of RT services.

We have implemented RT socket outsourcing as kernel modules. The guest module replaces the socket layer functions with those that perform VMRPCs to the host procedures. We modified the idle process in the guest, which examines the event queue and the socket status structure. A module in the host contains procedures for handling the requests from guest clients.

4 Experimental Evaluation

4.1 Experimental Setup

Figure 4 shows the experimental environment. We used netperf [19] as the NetRT server. Using a remote client, we measured round trip times. We modified the client of netperf, which sent request messages at random intervals ranging from 1 to 10 ms using UDP and received the response messages. We ran iperf [20] in server mode as a NetStream server. The iperf client sent continuous messages using TCP at the maximum speed.

It should be noted that changing the inter-arrival time had a similar effect on the CPU cache as changing the streaming workload. When we made the inter-arrival time shorter, the cache could retain more contents of the NetRT server, which corresponded to making the load lighter. When we made the inter-arrival

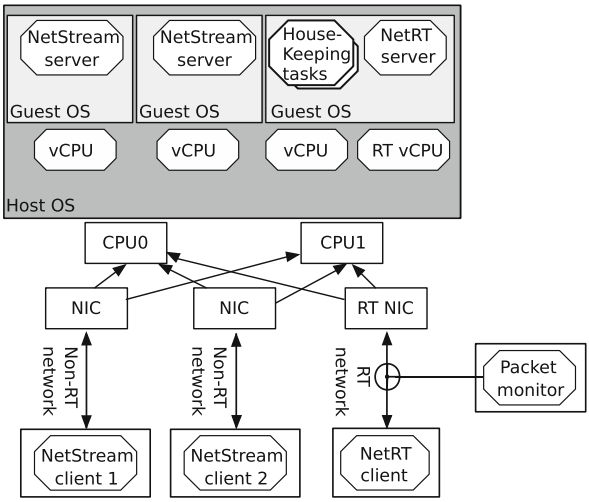


Fig. 4. The experimental environment.

Table 1. Configurations of the machines and their active cores in the experiments.

Machine	CPU/Cache (MB)	Cores	OS
VM host	Intel Core i7-6700K/8	2	Linux 4.1
NetRT client	Intel Core i7-6700K/8	4	Linux 4.1
NetStream client 1	Intel Core i7-3820/10	4	Linux 4.1
NetStream client 2	Intel Core i7-3820/10	4	Linux 4.1
Monitor	Intel Core i7-3820/10	4	Linux 3.16

time longer, the cache could retain fewer contents of the NetRT server, which corresponded to making the load heavier.

We connected the host of the VMs with a single RT network and two non-RT networks, as shown in Fig. 4. These networks comprised 10GBASE-LR optical fiber Ethernet systems. The NICs were Intel X520 Ethernet converged network adapters. We used two non-RT network links to make the CPUs busy on the VM host. When we used a single link, this link became the bottleneck and the CPUs had idle times. We set the maximum transfer unit (MTU) for these networks to 1500 bytes.

We measured the response times of the NetRT server using a hardware monitor, i.e., an Endace DAG10X2-S card [21]. We inserted optical taps into the RT network and directed packets to the Endace DAG card. This card captured and timestamped both the request and response packets at a resolution of 4 ns.

Table 1 shows the specifications of the physical machines used in the experiments. To avoid fluctuations in the frequency of the CPUs, we disabled the

Table 2. NetRT latency comparison between the RT methods (microseconds).

Method	Mean	99th percentile	Standard deviation
No method (vanilla Linux)	123.0	184.9	22.0
RT Preempt Patch Only	91.8	120.5	11.8
Dedicated processor	58.1	80.8	7.7
Outsourcing	32.6	49.5	2.8
Outsourcing (only RT poll_list)	93.2	131.1	11.4
Outsourcing (only RT socket outsourcing)	41.1	64.0	12.2

following processor features: Hyper-threading, TurboBoost and C-States[2]. In addition, we set the `CONFIG_NO_HZ_FULL` option in both the host and guest kernels to reduce the number of clock ticks in the CPU that executed the NetRT server. For the dedicated processor method, we assigned CPU 0 as the non-RT CPU and CPU 1 as the RT CPU. All guest OSes were Linux 4.1.

4.2 Experimental Results

Figure 5 shows the response times of the NetRT server, and Table 2 shows their statistical values (the mean, 99th percentile, and standard deviation (SD)). Figure 6 shows the throughputs and Fig. 7 shows the achievable CPU utilization.

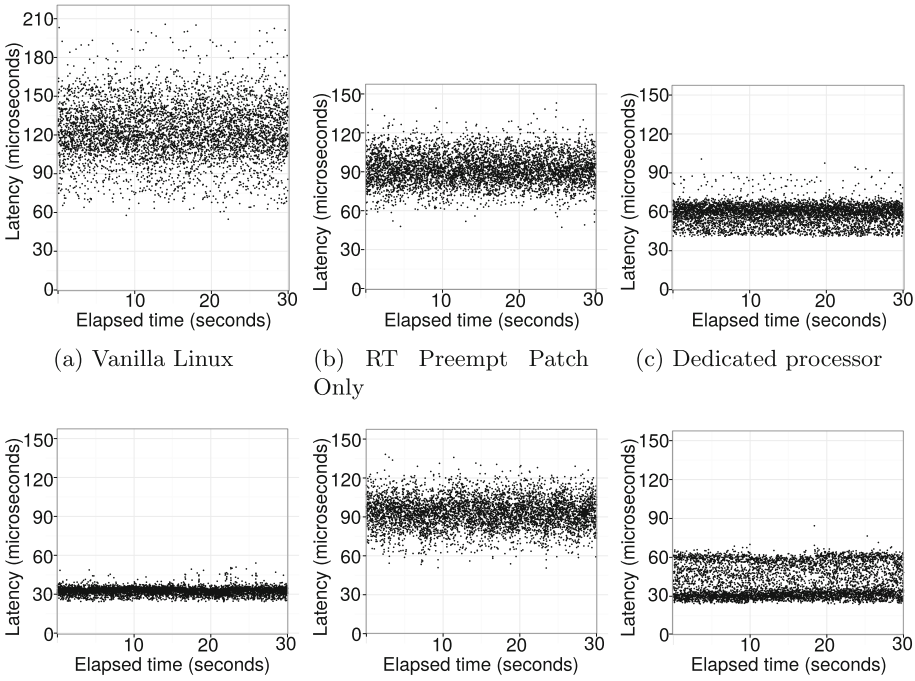
Figure 5a shows that in vanilla Linux, the NetRT server had high latency variance. On the other hand, the execution of NetStream servers presented a high performance as shown in Fig. 6, and NetStream servers got a throughput of 18.8 Gbps over an aggregated link capacity of 20 Gbps. The activities of the NetRT server and the NetStream servers did not consume all the CPU resources, as shown in Fig. 7.

Figure 5b shows that the RT Preempt Patch Only method improved the system realtimeness compared with vanilla Linux. Despite this improvement, the NetStream servers interfered with the response times of the NetRT server. The CPUs reached their maximum capacities in this method, which limited the volume of data received by the NetStream servers. Consequently, the NetStream servers had a throughput of 16.0 Gbps.

Figure 5c shows the response times obtained using the dedicated processor method. This method reduced the latency variability. However, the dedicated processor method could only use 50% of the CPU resources to execute the NetStream servers, which limited the total throughput to 8.6 Gbps.

Figure 5d shows the response times using the outsourcing method that comprises the two techniques: adding an RT poll_list (Sect. 3.1) and RT socket outsourcing (Sect. 3.2). This method had the lowest latency variability among the

[2] C-states are CPU modes for saving power. C-state transitions degrade the performance of RT services. We disabled C-states in the BIOS and in the Linux kernel using the parameters `intel_idle.max_cstate=0` and `idle=poll`.



(a) Vanilla Linux (b) RT Preempt Patch Only (c) Dedicated processor

(d) Outsourcing (using both techniques) (e) Outsourcing (only adding an RT poll_list). (f) Outsourcing (only RT socket outsourcing).

Fig. 5. Distribution of the NetRT server response times.

three methods. In addition, the outsourcing method maintained a high throughput of 18.8 Gbps with the lowest CPU consumption.

In summary, compared to the RT Preempt Patch Only method, the outsourcing method has the 76% lower standard deviation, 15% higher throughput, and 33% lower CPU overhead. Compared to the dedicated processor method, the outsourcing method has the 63% lower standard deviation and higher total throughput by a factor of 2, and avoids under-utilization of the dedicated processor.

We performed the experiment by enabling one of two techniques: adding an RT poll_list and RT socket outsourcing. Figure 5e and f show the results. When we enabled only one of the two techniques, we obtained large variability in the response times.

4.3 Application Benchmarks

In Sect. 4.2, we compared the three RT methods using netperf as a NetRT server. In this section, we compare these methods using two time-sensitive applications as NetRT servers. We used the same experimental environment and configurations described in Sect. 4.1.

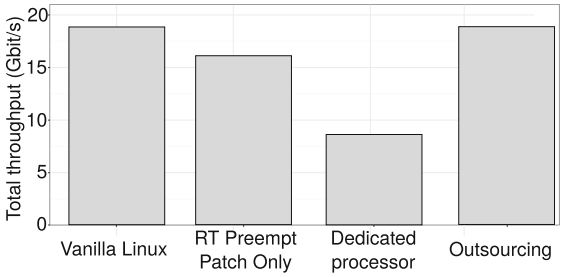


Fig. 6. Total throughput.

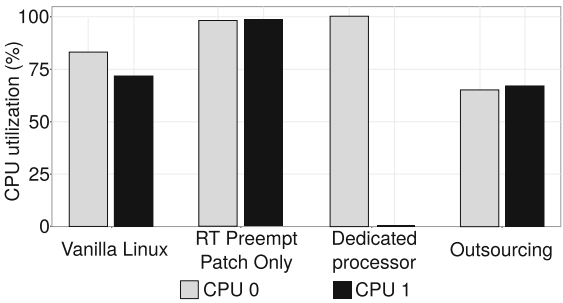


Fig. 7. Achievable CPU utilization.

A Voice over IP Server (VoIP). We measured the forward delays of a VoIP server that used the Session Initiation Protocol (SIP). We measured the impact of the NetStream servers on the activity of the NetRT server with various requesting rates. In this experiment, we ran Kamailio [22], a VoIP server, as the NetRT server. In a remote machine, we ran a SIPp [23] instance as a user agent client (UAC) and another instance as a user agent server (UAS). The VoIP server relayed messages between the UAC and the UAS.

We measured the forward delays between the message the VoIP server received and the message the VoIP server sent in SIP calls. In a single SIP call, the server forwarded six messages. The UAC first sent an INVITE message to the server, the server replied with a TRYING message and forwarded the INVITE message to the UAS. Next, the server forwarded a RINGING and OK message from the UAS to the UAC. Next, the server forwarded an ACK and BYE message from the UAC to the UAS. Finally, the server forwarded an OK message from the UAS to the UAC. We measured these forward delays using the hardware monitor, the Endace DAG card. We modified SIPp to make calls at random rates ranging from 17 to 167 calls per second. This means that the server forwarded 100 to 1000 messages in a second.

Figure 8 shows the 50^{th}, 99^{th} and 99.9^{th} percentiles of the NetRT response times. The response times with the outsourcing method had lower tail latencies than those with the RT Preempt Patch Only method and the dedicated processor

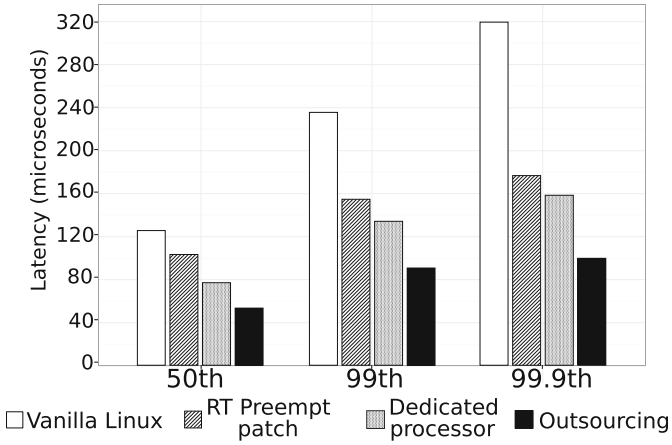


Fig. 8. The forward delays of a voice over IP server (Kamailio).

method. For example, in the 99th percentiles, the outsourcing method had 41% lower latency compared with the RT preempt Patch Only method and 32% lower latency compared with the dedicated processor method.

Memcached. We performed another application experiment using Memcached [24] as a NetRT server. Memcached is a distributed caching server that stores key-value pairs. The NetRT server received requests from a remote client called memaslap [25]. We measured the response times using random request intervals ranging from 100 to 1000 requests per second with the hardware monitor, the Endace DAG card. Memaslap sent GET/SET requests at a ratio of 9:1. The size of a key was 64 bytes and the size of the value was 1024 bytes. We ran the same NetStream servers employed in the previous experiments.

Figure 9 presents the 50th, 99th, and 99.9th percentiles of the response times of the GET requests. Similar to the previous experiment, the outsourcing method obtained better results than the RT Preempt Patch Only and the dedicated processor methods. In the 99th percentiles, the outsourcing method had 59% lower latency compared with the RT preempt Patch Only method and 54% lower latency compared with the dedicated processor method.

5 Related Work

The RT Preempt Patch Only and dedicated processor methods [8,15,16,18] are among the favorite choices in production-use operating systems that support RT services. Classic proposals for adding real-time support to commodity operating systems include RTLinux [26], Time-Sensitive Linux (TSL) [27], and Xenomai [28]. Although these proposed systems have been shown to be effective in reducing latency variance, the results are affected by the continuous evolution of the

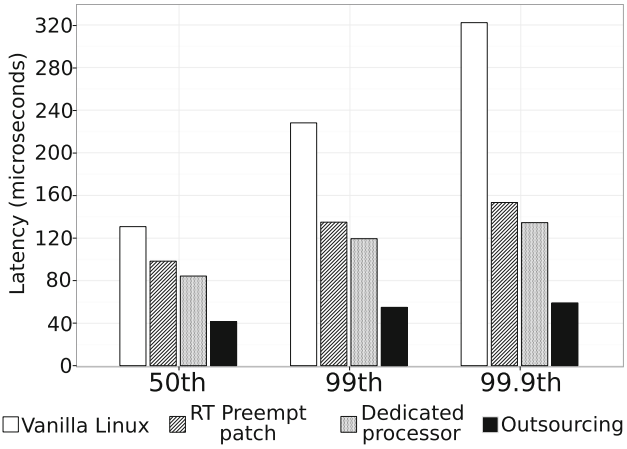


Fig. 9. The response times of Memcached.

underlying OS code. Examples of new sources of variance described in Sects. 2.3 and 2.5 include priority inversion and cache pollution.

Other proposals to address the priority inversion problem in network processing include lazy receiver processing (LRP) [29], which postpones the interrupt handling process until execution of the receiver task, and prioritized interrupt handling [30] for asynchronous transfer mode networks. More generally, some user space I/O frameworks [31–33] employ a polling mode to avoid latencies caused by interrupt handling and allow applications to send and receive packets directly from the DMA buffers of a NIC. Other alternatives to improve network throughput include polling threads [31], Data Plane Development Kit (DPDK)/vhostuser [32] and Netmap [33]. These design and implementation alternatives have varied trade-offs in throughput, latency, variance in latency, and achievable CPU utilization, as well as implementation difficulty in commodity systems such as Linux and KVM.

The study of software-based methods in this paper complements the advanced hardware assist techniques to improve I/O performance in VM environments [10–12,34–36]. Concrete examples include: Exit-Less Interrupts (ELI) [34], Efficient and Scalable Paravirtual I/O System (ELVIS) [35], and Direct Interrupt Delivery (DID) [36], which employ advanced hardware features, such as Single Root I/O Virtualization (SR-IOV) and Advanced Programmable Interrupt Controller virtualization (APICv) by Intel, Advanced Virtual Interrupt Controller (AVIC) by AMD, and Virtual Generic Interrupt Controller (VGIC) by ARM. These new hardware features are able to bypass some of the software layers in a consolidated environment that includes a guest OS, the host OS, and the VMM. The combination of software techniques such as outsourcing with advanced hardware assist is another interesting area of future research.

Previous outsourcing and similar techniques focus on improving throughput [9,37–39]. This is the first paper that uses outsourcing for reducing latency and variance in latency.

6 Conclusion

In this study, we have proposed the outsourcing method of real-time network services in KVM-based commodity hosted environments, and evaluated our method by comparing with two production methods, the RT Preempt Patch Only method and the dedicated processor method.

First, we found that the RT Preempt Patch Only method is able to reduce and remove the sources of latency variance in interrupt handling, primarily due to the first priority inversion problem between RT user processes and non-RT interrupt handling. However, the second priority inversion problem in the softirq mechanism of Linux remains. The dedicated processor method dedicates an exclusive processor for RT threads, including softirq, thus removing all thread-related priority inversion problems. Its drawback is that the low utilization of the dedicated processor can significantly reduce the total achievable throughput. The main contribution of the paper is the outsourcing method, which is implemented with two modest modifications to Linux (in addition to the RT Preempt patch). The first modification explicitly separates RT softirq handling from non-RT softirq handling, removing the second priority inversion problem. The second modification outsources the processing of network services from the guest OS to the host OS, thereby mitigating the cache pollution problem and avoiding the second priority inversion in a guest OS.

Compared to the RT Preempt Patch Only method, the outsourcing method has the 76% lower standard deviation, 15% higher throughput, and 33% lower CPU overhead. Compared to the dedicated processor method, the outsourcing method has the 63% lower standard deviation, higher total throughput by a factor of 2, and avoids under-utilization of the dedicated processor.

Acknowledgments. This work was partially supported by JSPS KAKENHI Grant Number 25540022 and 16K12410.

References

1. Prevot, T.: NextGen technologies for mid-term and far-term air traffic control operations. In: IEEE/AIAA 28th Digital Avionics Systems Conference, pp. 2.A.4-1–2.A.4-16 (2009)
2. Chen, I.R., Guo, J., Tsai, J.J.P.: Trust as a service for SOA-based IoT systems. Serv. Trans. Internet Things (STIOT) **1**(1), 43–52 (2017)
3. Page, A., Hijazi, S., Askan, D., Kantarci, B., Soyata, T.: Support systems for health monitoring using Internet-of-Things driven data acquisition. Int. J. Serv. Comput. (IJSC) **4**(4), 18–34 (2016)

4. Chang, S.F., Chang, J.C., Lin, K.H., Yu, B., Lee, Y.C., Tsai, S.B., Zhou, J., Wu, C., Yan, Z.C.: Measuring the service quality of E-Commerce and competitive strategies. Int. J. Web Serv. Res. (IJWSR) **11**(3), 96–115 (2014)
5. LiDuan, Y., Chen, J.: Event-driven SOA for IoT services. Int. J. Serv. Comput. (IJSC) **2**(2), 30–43 (2014)
6. Sun, Y., Qiao, X., Tan, W., Cheng, B., Shi, R., Chen, J.: A low-delay, light-weight publish/subscribe architecture for delay-sensitive IoT services. Int. J. Web Serv. Res. (IJWSR) **10**(3), 60–81 (2013)
7. dmoz.org: List of real-time operating systems. https://dmoztools.net/Computers/Software/Operating_Systems/Realtime/. Accessed 1 Nov 2017
8. Rostedt, S., Hart, D.V.: Internals of the RT patch. In: Proceedings of the Linux Symposium, pp. 161–172 (2007)
9. Eiraku, H., Shinjo, Y., Pu, C., Koh, Y., Kato, K.: Fast networking with socket-outsourcing in hosted virtual machine environments. In: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), pp. 310–317 (2009)
10. Jang, K., Sherry, J., Ballani, H., Moncaster, T.: Silo: predictable message latency in the cloud. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM 2015), pp. 435–448 (2015)
11. Chowdhury, M., Liu, Z., Ghodsi, A., Stoica, I.: HUG: multi-resource fairness for correlated and elastic demands. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp. 407–424 (2016)
12. Xiong, P., Hacigumus, H., Naughton, J.F.: A software-defined networking based approach for performance management of analytical queries on distributed data stores. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 955–966 (2014)
13. Werner, C., Buschmann, C., Jäcker, T., Fischer, S.: Bandwidth and latency considerations for efficient SOAP messaging. Int. J. Web Serv. Res. **3**(1), 49–67 (2006)
14. KernekShark: KernelShark - a front end reader of trace-cmd (2017). http://rostedt.homelinux.com/kernelshark/. Accessed 27 Oct 2017
15. Red Hat Inc.: Enterprise Linux for Real Time. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_for_Real_Time/7/html/Installation_Guide. Accessed 13 May 2017
16. Christofferson, M.: 4 ways to improve performance in embedded Linux systems. Korea Linux Forum (2013)
17. Crespo, A., Ripoll, I., Masmano, M.: Partitioned embedded architecture based on hypervisor: the XtratuM approach. In: Proceedings of the IEEE European Dependable Computing Conference (EDCC), pp. 67–72 (2010)
18. van Riel, R.: Real-time KVM from the ground up. KVM Forum, August 2015
19. Jones, R.: Netperf (1996). https://hewlettpackard.github.io/netperf/. Accessed 27 May 2017
20. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: iPerf: the TCP/UDP bandwidth measurement tool (2005). http://iperf.sourceforge.net 20 Apr 2017
21. Endace Technology Limited: DAG10X2-S datasheet (2015). https://www.endace.com/dag-10x2-s-datasheet.pdf. Accessed 12 June 2017
22. The Kamailio SIP Server Project: Kamailio SIP server. https://www.kamailio.org/w/. Accessed 12 May 2017
23. Gayraud, R., Jacques, O.: SIPp benchmark tool. http://sipp.sourceforge.net/. Accessed 23 Apr 2017
24. Memcached: Memcached - a distributed memory object caching system (2015). https://memcached.org/. Accessed 3 July 2017

25. Aker, B.: Libmemcached-Memaslap. http://libmemcached.org/libMemcached.html. Accessed 1 Nov 2017
26. Barabanov, M., Yodaiken, V.: Introducing real-time Linux. Linux J. **34** 9 p. (1997)
27. Goel, A., Abeni, L., Krasic, C., Snow, J., Walpole, J.: Supporting time-sensitive applications on a commodity OS. In: the USENIX 5th Symposium on Operating Systems Design and implementation, pp. 165–180 (2002)
28. Gerum, P.: Xenomai - Implementing a RTOS emulation framework on GNU/Linux. http://www.xenomai.org/documentation/xenomai-2.5/pdf/xenomai.pdf. Accessed 4 Nov 2017
29. Druschel, P., Banga, G.: Lazy receiver processing (LRP): a network subsystem architecture for server systems. In: the 2nd USENIX Symposium on Operating Systems Design and Implementation, pp. 261–275 (1996)
30. Kuhns, F., Schmidt, D.C., Levine, D.L.: The design and performance of a real-time I/O subsystem. In: Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium, pp. 154–163 (1999)
31. Liu, J., Abali, B.: Virtualization polling engine (VPE): using dedicated CPU cores to accelerate I/O virtualization. In: The ACM 23rd International Conference on Supercomputing, pp. 225–234 (2009)
32. Intel Corporation: DPDK: Data Plane Development Kit. http://dpdk.org/. Accessed 28 Sept 2017
33. Rizzo, L.: Netmap: a novel framework for fast packet I/O. In: Proceedings of the 2012 USENIX Conference on Annual Technical Conference, p. 9 (2012)
34. Gordon, A., Amit, N., Har'El, N., Ben-Yehuda, M., Landau, A., Schuster, A., Tsafrir, D.: ELI: bare-metal Performance for I/O Virtualization. In: Proceedings of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 411–422 (2012)
35. Gordon, A., Har'El, N., Landau, A., Ben-Yehuda, M., Traeger, A.: Towards exitless and efficient paravirtual I/O. In: Proceedings of the 5th ACM Annual International Systems and Storage Conference (SYSTOR), pp. 1–6 (2012)
36. Tu, C.C., Ferdman, M., Lee, C., Chiueh, T.: A comprehensive implementation and evaluation of direct interrupt delivery. In: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), pp. 1–15 (2015)
37. Gamage, S., Kompella, R.R., Xu, D., Kangarlou, A.: Protocol responsibility offloading to improve TCP throughput in virtualized environments. ACM Trans. Comput. Syst. (TOCS) **31**(3), 1–34 (2013)
38. Nordal, A., Kvalnes, Å., Johansen, D.: Paravirtualizing TCP. In: Proceedings of the 6th ACM International Workshop on Virtualization Technologies in Distributed Computing Date (VTDC), pp. 3–10 (2012)
39. Lin, Q., Qi, Z., Wu, J., Dong, Y., Guan, H.: Optimizing virtual machines using hybrid virtualization. Elsevier J. Syst. Softw. **85**(11), 2593–2603 (2012)

Identity and Access Management for Cloud Services Used by the Payment Card Industry

Ruediger Schulze(✉)

IBM Germany Research and Development GmbH, Schoenaicher Street 220, 71034 Boeblingen, Germany
ruediger.schulze@de.ibm.com

Abstract. The Payment Card Industry Data Security Standard (PCI DSS) mandates that any entity of the cardholder data environment (CDE) involved in the credit card payment process has to be compliant to the requirements of the standard. Hence, cloud services which are used in the CDE have to adhere to the PCI DSS requirements too. Identity and access management (IAM) are essential functions for controlling the access to the resources of cloud services. The aim of this research is to investigate the aspects of IAM required by the PCI DSS and to describe current concepts of IAM for cloud services and how they relate to the requirements of the PCI DSS.

1 Introduction

Credit card frauds are committed using different methods. According to [1], 60% of the value of fraudulent transactions within the Single Euro Payments Area (SEPA) were caused by card-not-present (CNP) payments in 2012. CNP frauds are committed without the actual physical use of the credit card, e.g. in online, phone or mail-order transactions. Online merchants are at an increased risks due to attacks which target to exploit vulnerabilities and obtain credit card records for use in CNP frauds and identity theft. The PCI DSS has been established to ensure the safety of the merchant's payment systems and has to be implemented by all merchants that provide payment with credit cards. The PCI DSS describes a set of 12 technical and operational requirements in five categories to protect credit card data [2]. The importance of the requirements claimed by PCI DSS can be illustrated at the example of the TJX data breach [3]. The TJX Companies, Inc., a department store chain in the United States, kept the magnetic strip data from customers credit cards in unencrypted from and was attacked by thieves first time in July 2005. In total, data of 94,000,000 cards was stolen. The PCI DSS defines some very prescriptive requirements to avoid such scenarios, e.g. sensitive authentication data must not be stored on storage at all and the primary account number has to rendered in unreadable format when stored. The requirements of the PCI DSS apply to all entities involved into the payment process, including service providers. Cloud services involved with credit

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 206–218, 2018.
https://doi.org/10.1007/978-3-319-94295-7_14

card payment have to be PCI DSS compliant. Cloud Service Providers (CSP) which are engaged by merchants as third-part service providers (TPSP) have to provide evidence of their PCI DSS compliance status [4]. IAM are essential functions when interacting with cloud services. IAM has to ensure that access to cloud services and related data is only granted to authorized users and that compliance with internal policies and external regulations is maintained. The aim of this research is to examine the IAM related requirements of the PCI DSS and to investigate current methods of IAM in the context of cloud services and PCI DSS. In the following, the requirements of the PCI DSS and the essential functions of IAM for cloud services are described, the PCI DSS requirements are assessed for their relation to the IAM functions and different methods of IAM for cloud services are analysed.

2 Payment Card Industry Data Security Standard

The PCI DSS has been issued to ensure the security of credit card and account data [2]. The latest version 3.2 of the standard has been released in April 2016 [2]. The technical and operational requirements of the PCI DSS apply to all entities involved in the process of the credit card payment. The scope of the PCI DSS is extensive. Any component included in the CDE is to be considered, including people, processes and technologies. The entities to be compliant with PCI DSS include merchants, processors, acquirers, issuers and services providers but also any entities that store, process or transmit cardholder data and sensitive authentication data. PCI DSS defines 12 requirements and corresponding test procedures which are combined into a security assessment tool for use during PCI DSS compliance assessments. The following requirements are to be assessed using the test procedures in the validation process [2]:

2.1 Build and Maintain a Secure Network and Systems

1. *Install and maintain a firewall configuration to protect cardholder data*
   Firewalls control the traffic between an entity's internal network and the untrusted, external networks, and the traffic in and out of sensitive areas within the internal trusted network. Firewalls and router configurations have to be established and implemented following a formal process for approving and testing all network connections and using documentation of business justification and network diagrams.
2. *Do not use vendor-supplied defaults for system passwords and other security parameters*
   Intruders (external and internal) use default passwords or exploits based on default system settings to compromise systems. The information about default passwords and settings are widely known and can easily be obtained from public documentation. Vendor-supplied defaults must always be changed and unnecessary default accounts be removed before a system is connected to the network.

2.2 Protect Cardholder Data

3. *Protect stored cardholder data*
   Cardholder data refers to any information present in any form on a payment card. The cardholder data must be protected by the entities accepting payment cards and unauthorized use has to be prevented. Methods like encryption, cutting, masking and hashing have to be used in order to transfer and process the data in an unreadable form. Only limited cardholder data should be stored on storage. Sensitive authentication data must not be stored after authentication.
4. *Encrypt transmission of cardholder data across open, public networks*
   Intruders exploit the misconfiguration of wireless network devices and weaknesses in legacy encryption and authentication protocols to gain access to CDE. Strong encryption and security protocols must be used to protect the cardholder data during transmission over open and public networks.

2.3 Maintain a Vulnerability Management Program

5. *Protect all system against malware and regularly update anti-virus software or programs*
   Malicious software (malware) such as viruses, worms and trojans can enter the network during approved business activities, e.g. via employee e-mail and the use of the Internet, and target to exploit system vulnerabilities. Anti-virus software has to be installed on all systems potentially affected by malicious software. The anti-virus software has to be capable to detect, remove and protect against all types of malicious software. It has to be ensured that the anti-virus mechanisms can not be disabled and are kept current, i.e. virus signature files are regularly updated and scans are executed.
6. *Develop and maintain secure systems and applications*
   Intrudes use security vulnerabilities to gain privileged access to the systems. Many of these vulnerabilities can be fixed by security patches provided by the vendors. Therefore, a process has to be established to regularly identify security vulnerabilities, rank them by risk (e.g. “high”, “medium”, “low”) and install applicable vendor-supplied security patches. Critical patches must be installed within a month after release.

2.4 Implement Strong Access Control Measures

7. *Restrict access to cardholder data by business need to know*
   The access to cardholder data must be limited to authorized personnel. Systems and processes have to be in place to control and restrict the access to the cardholder data. Access must only be granted on the base of business needs and in accordance with the job responsibility.
8. *Identify and authenticate access to system components*
   An unique identification (ID) has to be assigned to each person with access to

the CDE. Personalized IDs ensure that each individual can be held accountable for their actions. Using IDs, activities performed on critical data and systems can be restricted to authorized users only and be traced for auditing. An authentication system must be in place which implements the policies for adding, deleting and modifying user IDs and credentials. The authentication system has to effectively enforce protection, e.g. by limiting the number of access attempts and requesting regular password changes.

9. *Restrict physical access to cardholder data by business need to know*
   Physical access to systems that host cardholder data must be restricted to avoid unauthorised access to devices or data, and prevent removal or deletion of devices or hard-copies. Appropriate facility entry controls must be in place to limit and monitor the physical access to the systems of the CDE.

2.5 Regularly Monitor and Test Networks

10. *Track and monitor all access to network resources and cardholder data*
    All access to the cardholder data must be tracked and monitored. Logging mechanisms and tracking of user activities enable effective forensics and vulnerability management. The existence of logs in all environments allows for thorough tracking, warning and analysis in the event of a fault. Audit trails have to be implemented that link all access to the system to an individual user and allow to reconstruct events such the user access to the cardholder data and invalid access attempts. Regular reviews of logs and security events have to be established to identify anomalies and suspicious activities.
11. *Regularly test security system and processes*
    Vulnerabilities are continually discovered by both intruders and researchers. New vulnerabilities are often introduced by new software. System components, processes and custom software have to be tested regularly to ensure that security is maintained over time. Tests have to include the verification of the security controls for wireless access points, vulnerability scans of the internal and external network, penetration tests, intrusion detection and change detection for unauthorized modification.

2.6 Maintain an Information Security Policy

12. *Maintain a policy that addresses information security for all personnel*
    All personnel have to understand the sensitivity of the data and their responsibility to protect it. Strong security policies have to be enforced for the entire entity of the CDE and inform all personnel what is expected of them.

In [5], the fact that the PCI DSS 3.0 implies many changes compared to the version 2.0 is investigated. Even organisations which are already PCI DSS 2.0 compliant may encounter challenges when moving to the new version. The changes between PCI DSS 2.0 and 3.0 are identified and PCI DSS 3.0 is implemented at the example of the largest company for online payment services in Indonesia to measure an organisation's compliance level. As a result of this

research 182 new controls are identified which simplify the adoption of PCI DSS 3.0 by an organisation that is already compliant to the PCI DSS 2.0. The company was found to be 77.43% compliant to the PCI DSS 3.0 requirements. One of the recommendations in [5] is that the company should implement strict access controls to limit the access to card holder data as per PCI DSS requirement 7. The PCI Compliance Report from 2015 [6] shows that victims of data breaches struggle with establishing restricted access and access authentication more than other organisations. The report positively points out that the compliance with requirement 7 increased over the last year to 89% of all investigated companies in 2014. The compliance with requirement 8 for access authentication was at 69% in 2014. Some companies still have challenges with establishing appropriate handling of credentials (8.2) and preventing shared use of credentials (8.5).

3 Identity and Access Management for Cloud Services

With the adoption of cloud services, functions of the IAM have to be extended into the CSP. The following IAM functions are essential for the successful and effective management of identities when using cloud services [7]:

1. *Identity Provisioning and User Management*
   The existing processes for user management within an enterprise have to be extended to cloud services. Identity provisioning has to handle the secure and timely management of on-boarding and off-boarding the users of cloud services. The users of the cloud services represent either individuals or the organisation and are maintained by the CSP in order to support authentication, authorisation, federation, billing and auditing processes. The provisioning of the users for a specific cloud service has to be done with appropriate privileges and based on their roles as cloud service consumer, e.g. business manager, administrator, service integrator and end user. CSP have to provide APIs which support the provisioning of users and consumer organisations may use automated identity management solutions exploiting those APIs.
2. *Authentication and Credential Management*
   Authentication validates the credentials provided by a user. Organisations that become cloud service consumers have to invest into properly handling credential management, strong authentication and delegated authentication, and how trust is managed across multiple cloud services. Credential management involves the processes for managing passwords, digital certificates and dynamic credentials. One time passwords and strong authentication such as multi-factor authentication should be used to reduce for instance the impact of compromised password. Credentials may be stolen by various types of intrusion attempts such as phishing, key-logging or man-in-the-middle attacks. The impact of stolen credentials is fatal, data can be monitored or manipulated, and compromised valid user accounts can be used for denial-of-service attacks or obtaining root level access to VMs and hosts [8].

3. *Federation*
   Federation enables cloud service consumers to authenticate as users of cloud services using their chosen identity provider. The identity provider is an authoritative source which provides authentication of the users. The federation process requires that identity attributes are securely exchanged between a service provider and the identity provider. The service provider can be either an internally deployed application or a cloud service. Multiple federation standards exist today. The Security Assertion Markup Language (SAML) is a widely accepted federation standard and supports single sign-on (SSO). As organisations start to consume multiple cloud services, SSO becomes important in order to manage authentication across these different services consistently using a central identity provider.
4. *Authorisation and User Profile Management*
   Authorisation grants access to requested resources based on user profile attributes and access policies. The user profile stores a set of attributes which the CSP uses to customize the service and to restrict or enable access to subsets of functions of the service. For users which act on behalf of an organisation, some of the user's profile attributes, e.g. the user role, may be assigned by the organisation. The organisation is in this case the authoritative source for those attributes and owns the access control policy to be applied to the users. Organisations have to manage the profile attributes for their users of cloud service and have to confirm with the CSP based on their requirements that adequate access control mechanisms of the cloud resources are supported.
5. *Compliance*
   Cloud service consumers have to understand how CSPs enable compliance with internal and external regulatory requirements. When consuming cloud services, some of the information required to satisfy audit and compliance reporting requirements has to be delivered by the CSP.

The classification of different Identity Management Systems (IDMS) for cloud services are discussed in [9–11]:

1. *Isolated IDMS*
   A single server is used as service provider and identity provider. The system does not rely on a trusted third party for credential issuance and verification. User accounts are replicated from the on-premise user directory into the cloud environment. Using this approach introduces security exposures when sensitive data like passwords or keys are exposed into the cloud. Changes to roles and user ID removal become effective delayed as the replication occurs asynchronous.
2. *Centralised IDMS*
   The service provider(s) and the identity provider run on different servers. One dedicated server is used as identity provider for issuing and managing user identities, while any other servers are responsible for providing cloud services.

3. *Federated IDMS*
   Federation allows to manage user identities within an organisation using an on-premise user directory. Users of federated IDMS can use the same credentials for authenticating to different cloud services. User identities are validated using tokens issued by the identity provider and presented to the cloud service interface. A cloud-based form of federated IDMS are Identity Management-as-a-Service (IDMaaS) systems. In this case, a separate cloud service manages the identity, provides identity federation with other cloud services and allows an identity to be linked with multiple cloud services.
4. *Anonymous IDMS*
   The user's identity management information is not disclosed to others during authentication and the user is kept anonymous.

4 PCI DSS Requirements for Identity and Access Management

The PCI DSS requirements impose a set of requirements related to IAM, in particular the requirements for strong access control are related to the IAM functions [2]. In [12], an approach for migrating PCI DSS compliance to an Infrastructure-as-a-Service (IaaS) provider is described. The article emphasises the shared responsibility of organisations and CSPs for ensuring PCI DSS compliance for the IAM related requirements. Typically, IaaS providers enable PCI DSS compliance at the physical infrastructure level, and support multi-factor authentication and role-based policies, while the users of the organisations have to perform the IaaS account management.

4.1 Identity Provisioning and User Management

PCI DSS requirement 8 formulates the detailed requirements for managing identity within the CDE[1] [2]:

1. All users of the CDE must receive a unique ID before allowing them to access components of the CDE or cardholder data (8.1.1).
2. Strong processes for managing the life cycle of the user IDs must be in place. User accounts must be valid and be associated with a person currently present in the organisation. Changes such as adding new user IDs, modifying or deleting existing ones must be performed under the control of these processes (8.1.2).
3. User IDs of terminated users must be revoked immediately (8.1.3).
4. User accounts which are not used within the last 90 days have to be removed or disabled (8.1.4).
5. User IDs of external vendors should only be enabled during the time period when needed and be disabled when not in use. Timely unlimited access to the CDE by vendors increases the risk of unauthorised access. The use of the user IDs has to be monitored (8.1.5).

[1] The numbers in parenthesis refer to the specific PCI DSS requirements.

6. Security policies and operations procedures for identification must be documented, in use and known to all affected parties (8.8).

4.2 Authentication and Credential Management

PCI DSS requirement 8 demands the implementation of the following authentication and credential management related features by the IAM system of the cloud services [2]:

1. Authentication to the CDE must be done by using the user ID and at least one of methods (8.2):
   (a) Something you know (password, passphrase)
   (b) Something you have (smart card, token device)
   (c) Something you are (biometric)
2. Two-factor authentication must be used when accessing the CDE from remote networks. Two of the three authentication methods above have to be used for authentication (8.3).
3. Strong processes to control the modification of credentials have to be established (8.1.2).
4. The number of attempts to use invalid credentials for an user IDs has to be limited. After not more than six attempts the user ID has to be locked out (8.1.6).
5. Once an user ID is locked out due to too many attempts with invalid credentials, the lockout duration has to be at least 30 min or until an administrator enables the user ID again (8.1.7).
6. Re-authentication is required after a session is idle for more than 15 min (8.1.8).
7. All credentials must be rendered unreadable using strong cryptography during transmission and when stored on storage (8.2.1).
8. Before the modification of any authentication credentials, the identity of the users has to be verified (8.2.2).
9. Passwords and passphrases must have a minimum length of at least seven characters and contain both numeric and alphabetic characters (8.2.3)
10. Passwords and passphrases must be changed at least every 90 days (8.2.4).
11. When a user changes the credentials of its user ID, it must to be ensured that the new password or passphrases is not the same for any of the last four passwords or passphrases used before (8.2.5).
12. Passwords and passphrases for first-time use or upon reset have to be unique for each user and must be changed immediately after the first login (8.2.6).
13. Group, shared or generic IDs and passwords must not be used for authentication (8.5).
14. Authentication policies and procedures must be documented and communicated (8.4).
15. Security policies and operation procedures for authentication must be documented, in use and known to all affected parties (8.8).

4.3 Federation

The PCI DSS does not give any explicit requirements related to federation. The identity provider for authenticating users when working with federation belongs to the CDE. In case of cloud services which are used as identity provider, the provider of the services is to be treated as TPSP from the perspective of the CDE and the requirements of the PCI DSS apply to it.

4.4 Authorisation

The authorisation functions of the cloud IAM have to support the following requirements of the PCI DSS [2]:

1. Access to the components of the CDE and cardholder data must to be limited to users whose job requires such access (7.1).
2. Based on the job responsibilities and functions, access needs and required privileges have to be defined for each user role (7.1.1).
3. The access of privileged user IDs must be restricted to the least privileges necessary to perform a job responsibility (7.1.2).
4. Access must only be assigned based on the user's individual job classification and function (7.1.3).
5. Privileges must be assigned to users according to their job classification and function, and only after documented approval (7.2.2, 8.1.1).
6. The access control system must be set to "deny all" for all components of the CDE and only grant access to users with the need to know as per their assigned role and privileges (7.2, 7.2.1, 7.2.3).
7. Direct access to the databases containing cardholder data is restricted to only database administrators. All other access to the database is through programmatic methods using dedicated application IDs. Application IDs are restricted for only the use by applications (8.7).
8. Security policies and operation procedures for restricting access to cardholder data must be documented, in use and known to all affected parties (7.3).

4.5 Compliance

In order for cloud services to be conform with the PCI DSS, the following mechanisms for tracking user activities using audit logs have to be established [2]:

1. All user access to cardholder data must be logged (10.2.1).
2. All actions taken by an user with root or administrative privileges must be logged (10.2.2).
3. All access to audit logs must be logged (10.2.3).
4. All invalid access attempts must be logged (10.2.4).
5. All activities of user management and elevation of privileges must be logged (10.2.5).
6. The initialization, stopping or pausing of the audit logs must be logged (10.2.6).

7. Creation and deletion of system-level objects must be logged (10.2.7).
8. User identification, type of event, date and time, success or failure identification, origination of the event and identification or name of the affected data, system component or resources must be logged at least for each of the above events (10.3).
9. Time-synchronization of all clocks must be implemented to ensure events captured in different logs can be correlated (10.4).
10. Audit logs must be secured so that they can not be altered (10.5).
11. Audit logs must be reviewed regularly, and in particular all security events and logs of critical components daily (10.6.).
12. Audit logs must be retained at least one year, with a minimum of three month to be available immediately (10.7).
13. Security policies and operation procedures for monitoring all access to network resources and cardholder data must be documented, in use and known to all affected parties (10.8).

5 IAM Methods for Cloud Services

Methods for cloud IAM are investigated by multiple authors. In [13], a model for identity and trust management in cloud environments is proposed. In this model, both CSP and users have to register to a trusted authority and obtain a certificate access token. By validating the user's certificate, the CSP can be sure that the user's request comes from an authentic source. A high performance IAM system on the base of OpenStack Keystone is proposed in [14]. An intrusion tolerant identity provider is described in [15] and an approach for federation and SSO on the base of claim-based identity management is discussed in [16].

Strong user authentication is a requirement of the PCI DSS. The methods published in [17,18] are described below in detail. In [17], a method is proposed which combines different security features such as identity management, mutual authentication and session key agreement between users. Based on two-factor authentication and two different communication channels, the following authentication flow is described:

1. A smart card is issued to the user. The data on the smart card is encrypted using secret numbers for both the user and server.
2. The user logs into a terminal using a smart card, user ID and password. The authenticity of the user is verified based on the smart card, user ID and password.
3. The user sends from the terminal a login request to the cloud server.
4. The cloud server generates a one-time key and calculates some hash string.
5. The cloud server sends back the hash string to the terminal.
6. The cloud server sends the one-time key to the mobile phone of the user via SMS.
7. The user enters the one-time key into the terminal and the key is validated based on the hash string and data on the smart card.

8. An authentication request is sent to the cloud server using the data of the smart card, the user ID and the one-time key.
9. The cloud server checks if the maximum legal time of an authentication session is not yet passed.
10. The cloud server calculates the validation data based the authentication request and verifies the user's identity.
11. The cloud server authenticates the user by calculating a session key which is sent as a hash string back to the user's terminal.

Methods with communication via two channels and authentication using additional one-time keys are already used by some financial institutes today. The proposed method enhances these existing mechanisms by combining them with additional flows for user and server authentication and secure encoding of messages. The method fulfils the PCI DSS requirement 8.3 to use two-factor authentication when interacting with cloud services within the CDE from remote networks. In order to use the method in a PCI DSS compliant environment, the other requirements for authentication and managing credentials must be satisfied as well. The methods assumes that the user's password is only stored on smart card. In this case, the terminal software would have to ensure that the requirements for password such as minimum length and difference from previous passwords are ensured.

Another method focusing on strong user authentication is proposed in [18]. The suggested model uses virtual device authorisation (VDA) and adds an authentication agent to the user's web browser. VDA is a Software-as-a-Service (SaaS) application that acts as an gateway to other cloud services. The VDA is used to identify the ownership of private devices and to establish the linkage between the clients and the cloud services. A cloud service user may register multiple devices in the VDA for either temporary or permanent authentication to communicate with a cloud service. During the registration of a device, approval is obtained that the device is allowed to access the cloud service and the unique access code of the authentication agent is generated by the VDA using the device's MAC address, the access type (permanent or temporary) and an user-provided passcode. The authentication agent is sent to the client and installed into the user's web browser. When the user makes a request to a cloud service, the authentication agents confirms the identity of the user based on the passcode. Once the VDA receives confirmation about the identity of the user, the VDA validates the authorisation of the device. With the proposed method, a CSP can restrict the access to cloud services for only approved devices and users. The method leaves it open to establish further mechanisms for user authentication for each of the cloud services. The method implements a form of two-factor authentication (1. something you have - an uniquely identified and approved device and 2. something you know - the user's passcode) and addresses therefore requirement 8.3 of the PCI DSS. As with the other authentication method described previously, it is necessary to also establish mechanisms to satisfy the other PCI DSS requirements for authentication and credential management. For instance, a re-registration of each

device will be required at least every 90 days, as the passcode is used for generating the unique code of the authentication agent.

6 Conclusions

In this article, the PCI DSS was analysed for the IAM specific requirements and how they are related to the use of cloud services. Two existing methods for authenticating cloud services were analysed. Both methods fulfill the PCI DSS requirement 8.3 for two-factor authentication. With respect to the IAM related requirements of the PCI DSS, it was found that current research primarily focuses on the requirements for strong authentication. Further research needs to be done for establishing effective methods that address the other PCI DSS requirements for IAM in cloud environments.

References

1. Third report on card fraud, European Central Bank, February 2014. https://www.ecb.europa.eu/pub/pdf/other/cardfraudreport201402en.pdf
2. Payment Card Industry (PCI) Data Security Standard Version 3.2, April 2016. https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf
3. Shaw, A.: Data breach: from notification to prevention using PCI DSS. Colum. JL Soc. Probs. **43**, 517 (2009)
4. PCI Data Security Standard (PCI DSS) 3.0 Information Supplement: Third-Party Security Assurance, August 2014. https://www.pcisecuritystandards.org/documents/PCI_DSS_V3.0_Third_Party_Security_Assurance.pdf
5. Shihab, M., Misdianti, F.: Moving towards PCI DSS 3.0 compliance: a case study of credit card data security audit in an online payment company. In: 2014 International Conference on Advanced Computer Science and Information Systems (ICACSIS), pp. 151–156, October 2014
6. PCI Compliance Report, Verizon 2015. http://www.verizonenterprise.com/pcireport/2015/
7. Kumaraswamy, S., Lakshminarayanan, S., Stein, M.R.J., Wilson, Y.: Domain 12: guidance for identity & access management v2.1. Cloud Secur. Alliance (CSA), **10** (2010)
8. Fernandes, D.A., Soares, L.F., Gomes, J.V., Freire, M.M., Inácio, P.R.: Security issues in cloud environments: a survey. Int. J. Inf. Secur. **13**(2), 113–170 (2014)
9. Habiba, U., Abassi, A., Masood, R., Shibli, M.: Assessment criteria for cloud identity management systems. In: 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 188–195, December 2013
10. Ma, X.: Managing identities in cloud computing environments. In: 2015 2nd International Conference on Information Science and Control Engineering (ICISCE), pp. 290–292, April 2015
11. Understanding and Selecting Identity and Access Management for Cloud Services, Securosis, June 2013. https://securosis.com/assets/library/reports/Understanding_IAM_For_Cloud_Full.pdf
12. Ensuring PCI DSS Compliance in the Cloud, Cognizant (2014). http://www.cognizant.com/InsightsWhitepapers/Ensuring-PCI-DSS-Compliance-in-the-Cloud-codex879.pdf

13. Nida, Teli, B.: An efficient and secure means for identity and trust management in cloud. In: 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), pp. 677–682, March 2015
14. Faraji, M., Kang, J.-M., Bannazadeh, H., Leon-Garcia, A.: Identity access management for multi-tier cloud infrastructures. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–9, May 2014
15. Barreto, L., Siqueira, F., Fraga, J., Feitosa, E.: An intrusion tolerant identity management infrastructure for cloud computing services, In: 2013 IEEE 20th International Conference on Web Services (ICWS), pp. 155–162, June 2013
16. Singh, A., Chatterjee, K.: Identity management in cloud computing through claim-based solution. In: 2015 Fifth International Conference on Advanced Computing Communication Technologies (ACCT), pp. 524–529, February 2015
17. Choudhury, A., Kumar, P., Sain, M., Lim, H., Jae-Lee, H.: A strong user authentication framework for cloud computing. In: 2011 IEEE Asia-Pacific Services Computing Conference (APSCC), pp. 110–115, December 2011
18. Fatemi Moghaddam, F., Khanezaei, N., Manavi, S., Eslami, M., Samar, A.: UAA: user authentication agent for managing user identities in cloud computing environments. In: 2014 IEEE 5th Control and System Graduate Research Colloquium (ICSGRC), pp. 208–212, August 2014

Network Anomaly Detection and Identification Based on Deep Learning Methods

Mingyi Zhu, Kejiang Ye(✉), and Cheng-Zhong Xu

Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
{my.zhu,kj.ye,cz.xu}@siat.ac.cn

Abstract. Network anomaly detection is the process of determining when network behavior has deviated from the normal behavior. The detection of abnormal events in large dynamic network has become increasingly important as networks grow in size and complexity. However, fast and accurate network anomaly detection is very challenging. Deep learning is a potential method for network anomaly detection due to its good feature modeling capability. This paper presents a new anomaly detection method based on deep learning models, specifically the feed-forward neural network (FNN) model and convolutional neural network (CNN) model. The performance of the models is evaluated by several experiments with a popular NSL-KDD dataset. From the experimental results, we find the FNN and CNN models not only have a strong modeling ability for network anomaly detection, but also have high accuracy. Compared with several traditional machine learning methods, such as J48, Naive Bayes, NB Tree, Random Forest, Random Tree and SVM, the proposed models obtain a higher accuracy and detection rate with lower false positive rate. The deep learning models can effectively improve both the detection accuracy and the ability to identify anomaly types.

1 Introduction

In today's large-scale computer networks, an immense amount of flow data are observed each day, among which there are some records that do not conform to the normal network behavior. Some of them are malicious and can cause serious damage to the network. Therefore, it is important to sift through traffic data and detect anomalous events as they occur to ensure timely corrective actions. Network anomalies stand for a large fraction of the Internet traffic and compromise the performance of the network resources [1]. Possible reasons for traffic anomalies are changes in the network topology (e.g., newly connected hosts, routing changes) or network usage (e.g., changed customer behavior, new applications). Anomalies may also be caused by failures of network devices as well as by malicious worm or attack traffic [2]. The early detection of such events is of particular interest as they may impair the safe and reliable operation of the network.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 219–234, 2018.
https://doi.org/10.1007/978-3-319-94295-7_15

Network anomaly detection aims to detect patterns in a given network traffic data that do not conform to an established normal behavior [3] and has become an important area for both commercial interests as well as academic research. Applications of anomaly detection typically stem from the perspectives of network monitoring and network security. In network monitoring, a service provider is often interested in capturing such network characteristics as heavy flows that use a link with a given capacity, flow size distributions, and the number of distinct flows. In network security, the interest lies in characterizing known or unknown anomalous patterns of an attack or a virus. The anomalies may waste network resources, cause performance degradation of network devices and end hosts, and lead to security issues concerning all Internet users. Although network anomaly detection has been widely studied, it remains a challenging task due to the following factors:

- In large and complicated networks, the normal behavior can be multi-modal, and the boundary between normal and anomalous events is often not precise.
- Usually the network attacks adapt themselves continuously to cheat the firewalls and security filters, making the anomaly detection problem more difficult.
- Network traffic data often contains noise which tends to be similar to the true anomalies, and it is difficult to remove them.

In recent years, deep learning has grown very fast and achieved good results in many scenarios. There is a trend to use deep learning technologies for anomaly detection [4]. These feature learning approaches and models have been successful to a certain extent and match or exceed state of the art techniques.

This paper presents an anomaly detection method using deep learning models, specifically the feedforward neural network (FNN) model and convolutional neural network (CNN) model. The performance of the model is evaluated by several experiments with a popular NSL-KDD dataset [5]. From the experimental results, we find the FNN and CNN models not only have a strong modeling ability for network anomaly detection, but also have high accuracy. Compared with several traditional machine learning methods, such as J48, Naive Bayes, NB Tree, Random Forest, Random Tree and SVM, the proposed models obtain a higher accuracy and detection rate with lower false positive rate. The deep learning models can effectively improve both the detection accuracy and the ability to identify anomaly types.

2 The Models

2.1 Feedforward Neural Networks

Feedforward neural networks (FNN) are called networks because they are typically represented by composing together many different functions. The model is associated with a graph describing how the functions are composed together. For example, we might have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a

chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. These chain structures are the most commonly used structures of neural networks. In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The overall length of the chain gives the depth of the model. It is from this terminology that the name deep learning arises. The final layer of a feedforward network is called the output layer.

During neural network training, we drive $f(x)$ to match $f^*(x)$ to get our model. The training data provides us with noisy, approximate examples of $f^*(x)$ evaluated at different training points. Each example x is accompanied by a label $y \approx f^*(x)$. The training examples specify directly what the output layer like at each point x; it must produce a value that is close to y. The behavior of other layers is not determined by the training data. The learning algorithm must decide how to use these layers to produce the desired output, but the training data does not show what each individual layer does. Instead, the learning algorithm must know how to use these layers to achieve best implementation. Because the training data does not show the output for each of these layers, so that they are called hidden layers.

2.2 Convolutional Neural Network

A CNN (Convolutional Neural Network) consists of an input and an output layer, as well as multiple hidden layers, see Fig. 1 as an example. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution.

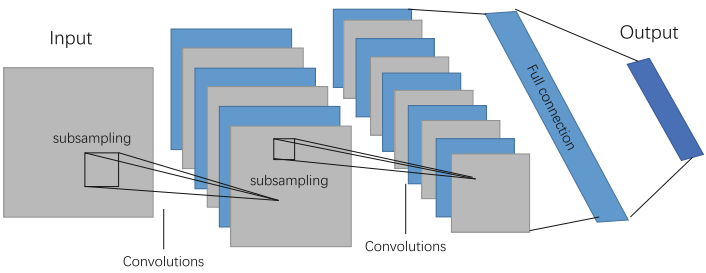


Fig. 1. Convolutional Neural Network

Convolutional. Convolutional layers apply a convolution operation to the input data, passing the result to the next layer. The convolution emulates the response of an individual neuron to stimulation. Each convolutional processes data only when it just in its responsed field. Although fully connected feedforward neural networks can be used to learn features to classify data, a very high number of neurons would be necessary, when we design a deep architecture. The

convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

Pooling. Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. Usually, we use the average output of these neurons as a result.

Fully Connected. Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Weights. CNNs share weights in convolutional layers, which means that the same filter is used for each receptive field in one layer repeatedly, this can reduces memory use and improves performance.

3 The Proposed Methods

3.1 The Dataset

We used NSL-KDD dataset [5] in our work. NSL-KDD is a dataset suggested to solve some of the inherent problems of the KDD'99 dataset. The KDD Cup includes normal and different kinds of attack traffic, such as DoS, Probing, user-to-root (U2R), and root-to-local (R2L). The network traffic for training was collected for seven weeks. The KDD Cup dataset has been widely used as a benchmark dataset for many years in the evaluation of NIDS (Network Intrusion Detection Systems) before. However, one of the major drawback with the dataset is that it contains an enormous amount of redundant records both in the training and testing data. It was observed that almost 78% and 75% records are redundant in the training and test dataset, respectively. This redundancy makes the learning algorithms more sensitive towards the frequent attack records and leads to poor classification results for the infrequent, but harmful records.

NSL-KDD was proposed to overcome the limitation of KDD Cup dataset. The dataset is derived from the KDD Cup dataset. It improved the previous dataset in four ways. First, It does not include redundant records in the training set, so the classifiers will not be biased towards more frequent records. Second, There is no duplicate records in the proposed testing sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on the frequent records. Further, The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD dataset. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have

an accurate evaluation of different learning techniques. Moreover, the number of records in the training and testing sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research works will be consistent and comparable.

This dataset contains a training set, KDDTrain+ and two testing sets called KDDTest+ and KDDTest21. KDDTrain+ includes the full NSL-KDD training set with attack-type labels in CSV format and KDDTest+ includes the full NSL-KDD testing set with attack-type labels in CSV format. In addition, the dataset providers analyzed the difficulty level of the records in KDD dataset. Surprisingly, about 98% of the records in the training set and 86% of the records in the testing set were correctly classified with 21 learners. So, they deleted the examples which are detected by all 21 learners from KDDTest+ and form another testing set - KDDTest21.

Each record in the NSL-KDD dataset consists of 41 features mentioned in Table 1 and is labeled with either normal or a particular kind of attack. These features include basic features derived directly from a TCP/IP connection, traffic features accumulated in a window interval, either time, e.g. two seconds, or several connections, and content features extracted from the application layer data of connections, that shows in Table 1. Among 41 features, three are nominal, four are binary, and remaining 34 are continuous. The training data contains 23 traffic classes that include 22 classes of attack and one normal class. The testing data contains of 38 traffic classes that include 21 attack classes from the training data, 16 novel attacks, and one normal class.

The NSL-KDD also has five categories: Normal, DoS, Probe, user-to-root (U2R), and root-to-local (R2L). Table 2 shows the sample number for the five categories. There are totally 125973 samples in training set with 67343 Normal samples, 45927 DoS samples, 995 R2L samples, 52 U2R samples and 11656 Probe samples. As well as in testing set, there are 22544 samples in all, including 9711 Normal samples, 7458 DoS samples, 2754 R2L samples, 200 U2R samples and 2421 Probe samples.

3.2 Data Preprocessing

Numericalization. There are 38 numeric features and 3 non-numeric features in the NSL-KDD dataset. Because the input value of our models should be a numeric matrix, we must convert some non-numeric features, such as 'protocol_type', 'service' and 'flag' features, into numeric form. For instance, the feature 'protocol_type' has three kinds of attributes, which are 'tcp', 'udp', and 'icmp'. And then we encode these values into binary vector (1,0,0), (0,1,0) and (0,0,1). Similarly, the feature service has 70 kinds of attributes, and the feature 'flag' has 11 kinds of attributes. Using the method we mentioned before, 41-dimensional features map into 122-dimensional features after transformation. Because for the convenience of our CNN model, we append 22 '0' each line total 144 features to make a $12 * 12$ matrix.

Table 1. Features in NSL-KDD Dataset

No.	Feature	Types	No.	Feature	Types
1	Duration	Continuous	22	Is_guest_login	Symbolic
2	Protocol_type	Symbolic	23	Count	Continuous
3	Service	Symbolic	24	Srv_count	Continuous
4	Flag	Symbolic	25	Serror_rate	Continuous
5	Src_bytes	Continuous	26	Srv_serror_rate	Continuous
6	Dst_bytes	Continuous	27	Rerror_rate	Continuous
7	Land	Symbolic	28	Srv_rerror_rate	Continuous
8	Wrong_fragment	Continuous	29	Same_srv_rate	Continuous
9	Urgent	Continuous	30	Diff_srv_rate	Continuous
10	Hot	Continuous	31	Srv_diff_host_rate	Continuous
11	Num_failed_logins	Continuous	32	Dst_host_count	Continuous
12	Logged_in	Symbolic	33	Dst_host_srv_count	Continuous
13	Num_compromised	Continuous	34	Dst_host_same_srv_count	Continuous
14	Root_shell	Continuous	35	Dst_host_diff_srv_rate	Continuous
15	Su_attempted	Continuous	36	Dst_host_same_srv_port_ra	Continuous
16	Num_root	Continuous	37	Dst_host_srv_diff_host_rat	Continuous
17	Num_file_creations	Continuous	38	Dst_host_serror_rate	Continuous
18	Num_shells	Continuous	39	Dst_host_srv_serror_rate	Continuous
19	Num_access_files	Continuous	40	Dst_host_rerror_rate	Continuous
20	Num_outbound_cmds	Continuous	41	Dst_host_srv_rerror_rate	Continuous
21	Is_host_login	Symbolic			

Table 2. Anomaly statistics in the training and testing sets

Intrusion type	Train samples	Testing samples
Total	125973	22544
Normal	67343	9711
DoS	45927	7458
R2L	995	2754
U2R	52	200
Probe	11656	2421

Normalization. First, according to some features, such as 'duration [0, 58329]' and 'src_bytes [0, $1.3 * 10^9$]', where the difference between the maximum and minimum values has a very large scope, we apply the logarithmic scaling method to reduce scaling scope. Second, the value of every feature is mapped to the [0,1] range linearly according to formula below, where Max denotes the maximum value and Min denotes minimum value for each feature.

$$x_i = \frac{x_i - Min}{Max - Min}$$

3.3 The Proposed Models

We designed a couple of Feedforward neural network models to test their performance including models with one hidden layer and two hidden layers. In one hidden layer case, we also tried different models with 100, 80 and 60 nodes in hidden layer. Meanwhile, the models with two hidden layers are designed as 100/80, 80/60 and 60/40 nodes in each hidden layer. Furthermore, we changed their learning rate to observe their performance. These models are shown in Sect. 4 in detail.

We also designed a CNN model to detect network abnormal traffic. It is obvious that the training of the CNN model consists of two parts: Forward Propagation and Back Propagation. Forward Propagation is responsible for calculating the output values. Back Propagation is responsible for passing the residuals that were accumulated to update the weights, which is not fundamentally different from the normal neural network training.

Here, we first reshape the handled 144 features to a $12 * 12$ matrix, as the input to our CNN model. The convolution layers are treated as a feature extractor to get feature maps, and then the average of each feature map is computed by Global Average Pooling. Finally, the resulting vector is fed directly into Softmax layer for classification. The generalization ability of the network is improved by this method. Global Average Pooling also be found that it can be compute to get a generic localization deep representation in. A single fully-connected layer is added between Global Average pooling layer and Softmax layer, and Class Activation Maps (CAM) is computed with the weights of this fully-connected layer like Fig. 2.

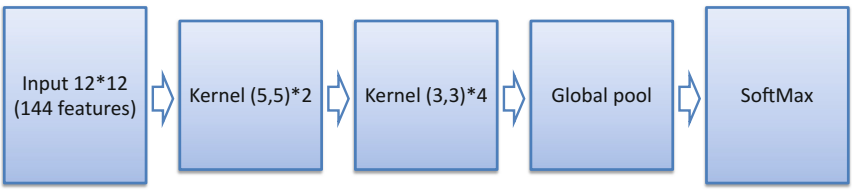


Fig. 2. Our CNN model

4 Evaluation

4.1 The Metrics

In our model, the most important performance indicator of network anomaly detection is used to measure the performance of our deep learning models. In addition, we use the detection rate and false positive rate to measure the performance. The True Positive (TP) means the number of anomaly traffic that are identified as anomaly. The False Positive (FP) denotes the number of normal records that are identified as anomaly. The True Negative (TN) is equivalent to

those correctly admitted, and it means the number of normal traffic that are identified as normal. The False Negative (FN) denotes the number of anomaly traffic that are identified as normal. Table 3 shows the definition of confusion matrix. We have the following notation:

Table 3. Confusion matrix

Predicted class		
Actual class	Anomaly	Normal
Anomaly	TP	FN
Normal	FP	TN

Accuracy: the percentage of the number of records classified correctly versus total the records shown below.

$$AC = \frac{TP + TN}{TP + TN + FP + FN}$$

True Positive Rate (TPR): as the equivalent of the Detection Rate (DR), it shows the percentage of the number of records identified correctly over the total number of anomaly records, as shown below.

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR): the percentage of the number of records rejected incorrectly is divided by the total number of normal records, as shown below.

$$FPR = \frac{FP}{FP + TN}$$

Hence, the motivation for the network anomaly detection models is to obtain a higher accuracy and detection rate with a lower false positive rate.

4.2 Experimental Results

In this paper, we used one of the most current and popular deep learning frameworks - Tensorflow [6]. Tensorflow is an excellent machine learning framework belonging to Google. Its powerful capabilities enable researchers to create the machine learning models they designed more simply. The experiment is performed on a server, with a configuration of an Intel Xeon E5-2630 v3 (15M Cache, 2.40 GHz) and 32GB memory. Some experiments have been designed to study the performance of our two different models for five-category classification, such as Normal, DoS, R2L, U2R and Probe. In order to demonstrate the efficiency of the proposed models, we also compare the results with several classical machine learning algorithms, such as J48, Naive Bayes, NB Tree, Random Forest, Random Tree and SVM.

FNN Model. Figure 3 shows the training process on KDDTrain+ and testing process on KDDTest+ and KDDtest21. From the figure, we can see the accuracy rapidly rise before about 3000 epochs and gradually tend to flatten out after about 15000 epochs. Furthermore, the accuracy on the training dataset is the highest which is close to 1, it is because the model is trained from the same dataset directly and captures all the anomaly characteristics. While the detection accuracy on dataset KDDTest+ and KDDTest21 are about 80%, and 50% respectively. The accuracy on KDDTest21 is lower than KDDTest+. It is because KDDTest21 removes all the anomalies that can be easily detected, resulting in a low detection accuracy.

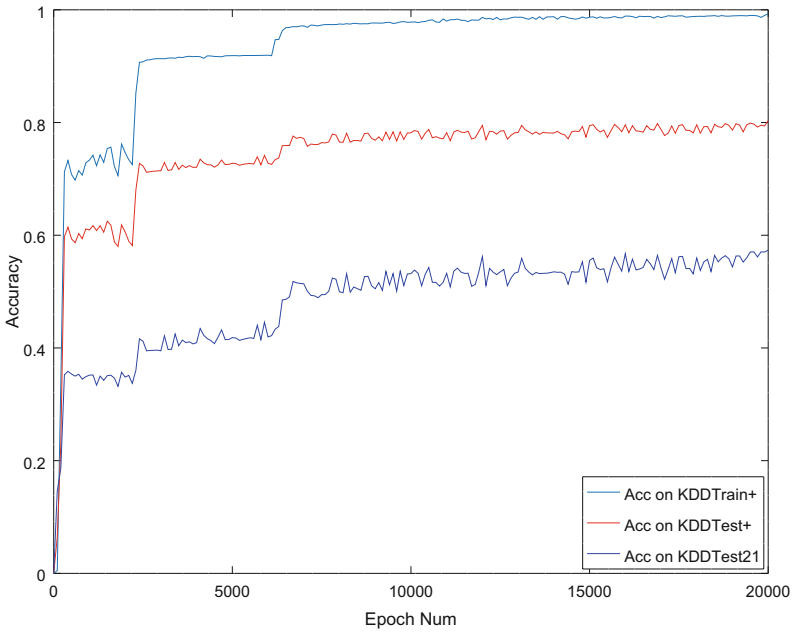


Fig. 3. The Accuracy of FNN

Table 4 shows the accuracy with different FNN models. In the five-category classification experiments, different kinds of Feedforward Neural Network was tested. The parameters include number of hidden layers, number of nodes in each layer and the learning rate of algorithm. As we can see, the best result 80.34% shows when there are 100 nodes in hidden layer, meanwhile the learning rate is 0.5. When the number of nodes below 100, as nodes increase, there is a raising tendency for precision. However, when we add the node number to 120, the precious comes to a slightly drop. Results are getting complicated when it comes to the Feedforward Neural Network with two hidden layers. We can't describe a obvious rule under such circumstances, just think a little lower performance than the Feedforward Neural Network with just one hidden layer. However, there is

Table 4. The accuracy with different FNN models

Structure	Learning rate	Accuracy
Hidden layer nodes = 120	0.5	78.9%
Hidden layer nodes = 100	0.5	80.34%
Hidden layer nodes = 80	0.5	80.01%
Hidden layer nodes = 60	0.5	76.75%
Hidden layer nodes = 100	0.8	80.18%
Hidden layer nodes = 100	0.1	77.57%
Hidden layer nodes = 100 * 80	0.5	79.2%
Hidden layer nodes = 80 * 60	0.5	78.8%
Hidden layer nodes = 60 * 40	0.5	79.01%

also a nice result (80.3%) when first hidden layer has 80 nodes and the second hidden layer has 60 nodes with 0.8 learning rate.

CNN Model. Table 5 shows the detection result for CNN model. For CNN, we can not design the enough layers since the hardware limitation. As a result, we just tried a simple model with a drop optimization and it reaches an result that accuracy comes to 77.8%. We also find the accuracy of CNN model is insensitive with the size of kernel, number of full connect layer nodes and learning rate. Compared with FNN model, the accuracy of CNN is relatively lower. The reason is that the connection among the features in the dataset may not be so strong. Another possible reason may be the layer is not deep enough.

4.3 Comparison with Other Methods

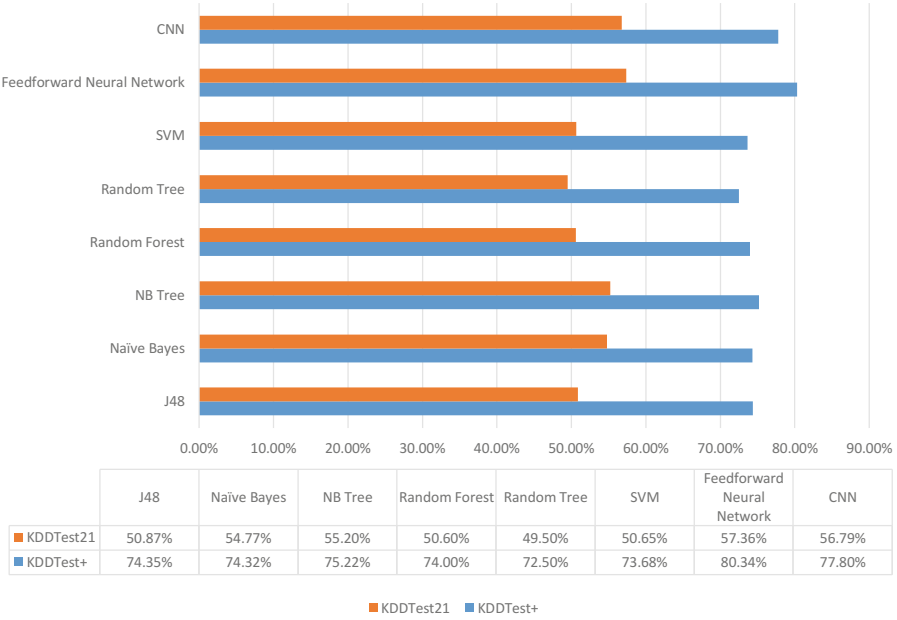
To make our results more intuitive, we used another famous classic machine learning framework - scikit-learn to implement J48, Naive Bayes, NB Tree Random Forest, Random Tree and SVM on the same benchmark dataset. The results of the experiment are described in the Fig. 4. Compared with these classic machine learning methods, the performance of deep learning algorithms shows a remarkable improvement. The classical machine learning algorithms generally achieve about 75% accuracy, and the best NB tree reached 75.22% accuracy. However, the deep learning models can achieve at least 77.8% accuracy, and the best one comes to 80.34% accuracy.

Table 6 shows the result for the Feedforward Neural Network on testing set KDDTest+ in the five-category classification experiments. The model has achieved considerable results in the recognition of DOS and probe, but not good in the rest two categories. When look at the dataset in Table 2, we find the anomaly U2R has few samples in both training set and testing set, so the accuracy is greatly affected. While the anomaly R2L have enough samples in testing set whereas have few samples in training set, the data in the testing set is about

Table 5. The accuracy with different CNN models

Structure	Learning rate	Accuracy
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 120	0.5	74.38%
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 100	0.5	77.80%
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 80	0.5	74.40%
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 60	0.5	77.64%
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 100	0.1	70.83%
Convolutional neural network Kernel1(5,5)*2 Kernel2(3,3)*4 Full connect layer nodes = 100	0.8	74.49%
Convolutional neural network Kernel1(4,4)*2 Kernel2(2,2)*4 Full connect layer nodes = 100	0.5	74.84%
Convolutional neural network Kernel1(7,7)*2 Kernel2(5,5)*4 Full connect layer nodes = 100	0.5	74.63%

3 times of the training set. A small amount of training data can not produce a model with sufficient generalization capability, resulting in low detection rate. Furthermore, the FPR of R2L and U2R are very low at the same time, which shows these models have excellent ability to identify the subclasses trained by training set and the subclasses which could not be identified may be these classes



	J48	Naïve Bayes	NB Tree	Random Forest	Random Tree	SVM	Feedforward Neural Network	CNN
KDDTest21	50.87%	54.77%	55.20%	50.60%	49.50%	50.65%	57.36%	56.79%
KDDTest+	74.35%	74.32%	75.22%	74.00%	72.50%	73.68%	80.34%	77.80%

Fig. 4. Comparartion in different machine learning algorithm

Table 6. The detection results for different anomaly types

Anomaly type	FPR(%)	DR(%)
DoS	2.04	83.44
R2L	0.79	24.52
U2R	0.07	10.30
Probe	2.17	82.34

containing no or just a few samples in the training set. As a result, the deep learning models is practical for network anomaly detection.

5 Discussion

Based on the same benchmark, using KDDTrain+ as the training set and KDDTest+ and KDDTest-21 as the testing set, the experimental results show that the network anomaly detection models both have higher accuracy than the other machine learning methods. In addition, there's still a lot of space for the improvement of network anomaly detection using deep learning.

First of all, we should refine this model's pertinence to the specific aspect. As Robin Sommer mention [7], we should keep the scope narrow. Due to the diversity of network traffic, we should not have the idea that design a model

which can detect all types of anomalies. Oppositely, a model which can detect the specific anomalies under specific environment should be established. These models may not require strong generalization and migration capabilities, but we can draw on the thought of boosting and bagging, which means overlay and integrate these models to become a model detecting different anomalies in one environment, this can cause the weakness of generalization ability to a certain extent. For example, we can design different models to detect DoS anomaly and U2R anomaly, then we use these models sequentially to get the result of each anomaly. At last we combine the two results comprehensively to achieve the final result. We believe this could reach a better result than a general model.

Meanwhile, we can not test a deep enough CNN model because of the limitation of hardware. The huge amount of practice in different areas confirm the big effect on CNN. It avoids explicit feature extraction and implicitly learns from training data, this guarantee it can learn the inherent relation in features. When solute anomaly detection problem (whose essential issue is the classification problem), we are confident of the better performance CNN will show when we use the new device.

6 Related Work

Network anomaly detection is an important and dynamic research area. Many network anomaly detection methods and systems have been proposed in the literature.

Duffield et al. [8] proposed a rule-based anomaly detection on the IP network which correlates the packet and flow level information. Cherkasova et al. [9] presented an integrated framework of using regression based transaction models and application performance signatures to detect anomalous application behavior. Sharma et al. [10] used the Auto-Regressive models and a time-invariant relationships based approach to detect the fault. Pannu et al. [11] presented an adaptive anomaly detection framework that can self adapt by learning from observed anomalies at runtime. Tan et al. presented two anomaly prediction systems PREPARE [12] and ALERT [13] that integrate online anomaly prediction, learning-based cause inference, and predictive prevention actuation to minimize the performance anomaly penalty without human intervention. They also investigated the anomalous behavior of three datasets [14]. Bronevetsky et al. [15] designed a novel technique that combine classification algorithms with information on the abnormality of application behavior to improve detection. Gu et al. [16] developed an attack detection system LEAPS based on supervised statistical learning to classify benign and malicious system events. Tati et al. [17] proposed an algorithm to efficiently diagnose large-scale clustered failures in computer networks which is based on greedy approach. Besides, several works focus on the anomaly/fault pattern analysis. Birke et al. [18] conducted a failure pattern analysis on 10K virtual and physical machines hosted on five commercial datacenters over an observation period of one year. Their objective is to establish a sound understanding of the differences and similarities between failures of

physical and virtual machines. Rosa et al. [19] studied three types of unsuccessful executions in traces of a Google datacenter, namely fail, kill, and eviction. Their objective is to identify their resource waste, impacts on application performance, and root causes.

Most recently, with fast development of deep learning, there is a trend to use deep learning on network anomaly detection. Maimó et al. [20] used deep learning method for anomaly detection in 5G networks. Tang et al. [21] applied a deep learning approach for flow-based anomaly detection in an SDN environment. They just use six basic features (that can be easily obtained in an SDN environment) taken from the forty-one features of NSL-KDD Dataset. Yin et al. [22] proposed a deep learning approach for intrusion detection using recurrent neural networks (RNN). Javaid et al. [23] used Self-taught Learning (STL), a deep learning based technique, to develop a Network Intrusion Detection System. Roy et al. [24] used Deep Neural Network as a classifier for the different types of intrusion attacks and did a comparative study with Support Vector Machine (SVM). Li et al. [25] proposed a image conversion method of NSL-KDD data and evaluated the performance of the image conversion method by binary class classification experiments.

In comparison, this paper applies feedforward neural network model and convolutional neural network model for network anomaly detection and studies the impact of different model parameters. We also perform a detailed comparison of detection accuracy with several traditional machine learning methods and demonstrate the deep learning based detection models can achieve a better accuracy.

7 Conclusion

This paper presents a new anomaly detection method based on deep learning models, specifically the feedforward neural network (FNN) model and convolutional neural network (CNN) model. The performance of the models is evaluated by several experiments with a popular NSL-KDD dataset. From the experimental results, we find the FNN and CNN models not only have a strong modeling ability for network anomaly detection, but also have relatively high accuracy. Compared with several traditional machine learning methods, such as J48, Naive Bayes, NB Tree, Random Forest, Random Tree and SVM, the proposed models obtain a higher accuracy and detection rate with lower false positive rate. The deep learning models can effectively improve both the detection accuracy and the ability to identify the anomaly types. With the continuous advancement of deep learning technology and the development of hardware accelerators, we believe better performance can be achieved in network anomaly detection by using deep learning methods in the future.

Acknowledgments. This work is supported by the National Basic Research Program of China (Grant No. 2015CB352400), National Natural Science Foundation of China (Grant No. 61702492, U1401258), and Shenzhen Basic Research Program (Grant No. JCYJ20170818153016513, JCYJ20170307164747920).

References

1. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 267–280. ACM (2010)
2. Gill, P., Jain, N., Nagappan, N.: Understanding network failures in data centers: measurement, analysis, and implications. In: ACM SIGCOMM Computer Communication Review, vol. 41, no. 4, pp. 350–361. ACM (2011)
3. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: methods, systems and tools. IEEE Commun. Surv. Tutor. **16**(1), 303–336 (2014)
4. Kwon, D., Kim, H., Kim, J., Suh, S.C., Kim, I., Kim, K.J.: A survey of deep learning-based network anomaly detection. Clust. Comput. 1–13 (2017)
5. Nsl-kdd dataset (2018). http://www.unb.ca/cic/datasets/nsl.html
6. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283 (2016)
7. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy (SP), pp. 305–316. IEEE (2010)
8. Duffield, N., Haffner, P., Krishnamurthy, B., Ringberg, H.: Rule-based anomaly detection on Ip flows. In: INFOCOM (2009)
9. Cherkasova, L., Ozonat, K., Mi, N., Symons, J., Smirni, E.: Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In: DSN (2008)
10. Sharma, A.B., Chen, H., Ding, M., Yoshihira, K., Jiang, G.: Fault detection and localization in distributed systems using invariant relationships. In: DSN (2013)
11. Pannu, H.S., Liu, J., Fu, S.: AAD: adaptive anomaly detection system for cloud computing infrastructures. In: SRDS (2012)
12. Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., Rajan, D.: Prepare: predictive performance anomaly prevention for virtualized cloud systems. In: ICDCS (2012)
13. Tan, Y., Gu, X., Wang, H.: Adaptive system anomaly prediction for large-scale hosting infrastructures. In: PODC (2010)
14. Tan, Y., Gu, X.: On predictability of system anomalies in real world. In: MASCOTS (2010)
15. Bronevetsky, G., Laguna, I., De Supinski, B.R., Bagchi, S.: Automatic fault characterization via abnormality-enhanced classification. In: DSN (2012)
16. Gu, Z., Pei, K., Wang, Q., Si, L., Zhang, X., Xu, D.: Leaps: detecting camouflaged attacks with statistical learning guided by program analysis. In: DSN (2015)
17. Tati, S., Ko, B.J., Cao, G., Swami, A., Porta, T.L.: Adaptive algorithms for diagnosing large-scale failures in computer networks. In: DSN (2012)
18. Birke, R., Giurgiu, I., Chen, L.Y., Wiesmann, D., Engbersen, T.: Failure analysis of virtual and physical machines: patterns, causes and characteristics. In: DSN (2014)
19. Rosa, A., Chen, L.Y., Binder, W.: Understanding the dark side of big data clusters: an analysis beyond failures. In: DSN (2015)
20. Maimó, L.F., Gómez, Á.L.P., Clemente, F.J.G., Pérez, M.G., Pérez, G.M.: A self-adaptive deep learning-based system for anomaly detection in 5G networks. IEEE Access **6**, 7700–7712 (2018)

21. Tang, T.A., Mhamdi, L., McLernon, D., Zaidi, S.A.R., Ghogho, M.: Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 258–263. IEEE (2016)
22. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017)
23. Javaid, A., Niyaz, Q., Sun, W., Alam, M.: A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 21–26 (2016)
24. Roy, S.S., Mallik, A., Gulati, R., Obaidat, M.S., Krishna, P.V.: A deep learning based artificial neural network approach for intrusion detection. In: Giri, D., Mohapatra, R.N., Begehr, H., Obaidat, M.S. (eds.) ICMC 2017. CCIS, vol. 655, pp. 44–53. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-4642-1_5
25. Li, Z., Qin, Z., Huang, K., Yang, X., Ye, S.: Intrusion detection using convolutional neural networks for representation learning. In: Liu, D., Xie, S., Li, Y., Zhao, D., El-Alfy, E.-S.M. (eds.) ICONIP 2017. LNCS, vol. 10638, pp. 858–866. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70139-4_87

A Feedback Prediction Model for Resource Usage and Offloading Time in Edge Computing

Menghan Zheng[1], Yubin Zhao[1(✉)], Xi Zhang[1], Cheng-Zhong Xu[1], and Xiaofan Li[2]

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China
{zhengmh,zhaoyb,xi.zhang,cz.xu}@siat.ac.cn
[2] Shenzhen Institute of Radio Testing and Technology, Shenzhen 518048, China
lixiaofan@srtc.org.cn

Abstract. Nowadays, edge computing which provides low delay services has gained much attention in the research filed. However, the limited resources of the platform make it necessary to predict the usage, execution time exactly and further optimize the resource utilization during offloading. In this paper, we propose a feedback prediction model (FPM), which includes three processes: the usage prediction process, the time prediction process and the feedback process. Firstly, we use average usage instead of instantaneous usage for usage prediction and calibrate the prediction results with real data. Secondly, building the time prediction process with the predicted usage values, then project the time error to usage value and update the usage values. Meanwhile, our model re-executes the time prediction process. Thirdly, setting a judgment and feedback number to the correction process. If prediction values meet the requirement or reach the number, FPM stops error feedback and skips to the next training. We compare the testing results to other two model which are BP neural network and FPM without feedback process (NO-FP FPM). The average usage and time prediction errors of BP and NO-FP FPM are 10%, 25% and 16%, 12%. The prediction accuracy in FPM has a great improvements. The average usage prediction errors can reach less than 8% and time error reach about 6%.

Keywords: Computation offloading · Resource optimization
Time prediction · Back propagation · Feedback process

1 Introduction

According to Internet Data Center (IDC), the market prospect of edge computing is very broad. And nearly 40% of IoT data will be processed near the clients

Y. Zhao—This work was supported by the China National Basic Research Program (973 Program, No. 2015CB352400), National Nature Science Foundation of China (Grand No. 61501443), NSFC under grand U1401258.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 235–247, 2018.
https://doi.org/10.1007/978-3-319-94295-7_16

with the use of fog computing and edge computing by 2018 [1]. Depending on the increasing edge devices and users' requirements which include real-time, high efficiency and short delay, edge computing is quite suitable for IoT systems [2,3]. However, power consumption, storage and bandwidth size of the edge platform are limited comparing to the cloud data centers. Therefore, edge computing is not suitable for complex tasks and the importance of resource utilization.

In the process of services providing in edge computing, platform firstly receives the service requests, then determine the service order according to the corresponding rules of the platform. Afterwards, the platform allocates resource for each task [6,7]. Lastly, the platform returns the results to the requesting devices after the computing nodes complete computation process. In general, the response times are regarded as a criterion for evaluating the performance of the platform [5]. On the condition of the limited hardware capability of the platform, we can only improve the response ability through the optimization of resource utilization. In this case, we need to know the resource requirements of each task before the allocation of resources. Then the platform can give the reasonable allocation base on the expected execution time and current state of resource utilization. Meanwhile we obtain the real execution time to help us how to allocate the resources. Therefore the accuracy of execution time prediction and resource requirement are essential in resource allocation.

In this paper, we propose a feedback prediction model (FPM) which is used for resource usage prediction and offloading time prediction. These predictions can be used as references and will benefit resource allocation in edge computing. Time prediction is for the queueing management and division of different categories of tasks. Firstly, we calculate usage by accumulating the resource consumption in the execution time. However curve of resource consumption is difficult to solve, so we give the usage prediction through the average resource consumption and correction base on real usage data. Secondly, we combine the transmission time and predicted usage to give the time prediction model. Thirdly, building a feedback process after we get the time prediction error. The time predicted error will be mapped to the resource usage. During the feedback process, the weights of usage and time prediction model will update until the time error reach our setting threshold or greater than upper bound which is used to prevent the divergence caused by feedback. The main contributions of this paper are listed as follows.

(1) We propose a resource usage prediction model. Because the dynamic change curve of usage is difficult to express, predicted usage is obtained by average predicted usage.
(2) We integrate the predicted usage and transmission time to build the time prediction model, then calculate the time error.
(3) We design the backward propagation method, time error's feedback and usage's correction to update the predicted weights. Then we set the stopping condition of the feedback loop to promise efficiency of feedback process.

The rest of the paper is organized as follows: Sect. 2 introduces some related work. In Sect. 3, we describe the architecture of the prediction model and

introduce the three constituent processes of FPM. In Sect. 4, the simulation results are presented. Finally, the conclusion is given in Sect. 5.

2 Related Work

Research on resource optimization based on cloud computing has been going on for many years. At present, the role which edge computing plays in the IoT has become more and more important. Low network delay make it suitable for applications which demand high performance of real time. In addition, network slicing and edge computing are the key technologies in 5G network [8]. However, the disadvantage is the limited computing power and limited carrying capacity. Meanwhile, for lightweight loads, if you need the platform executes the offloading tasks fast and efficient, the problem of resource optimization has been highlighted. And accurate prediction is an important part of the optimization problem.

In early 2003, AKAMAI which is a provider of content distribution network, CDN and cloud services had been cooperated with IBM on edge computing. Many network companies represented by CISCO are mainly based on fog computing [4]. Strictly speaking, there is no essential difference between fog computing and edge computing, they are both provide a nearby calculation. Up till today, several existing research attempts to find the best resource matching, in order to achieve better resource utilization. The common practice is establishing a prediction model from hundreds of thousands of test data. And they always choose the Artificial Neural Network (ANN) or other training models of artificial intelligence as prototype of the prediction model.

In [9], Markov chain model and linear regression are used to predict the CPU and memory utilizations, moreover, Markov chain model in [10,11] considers both current and future resource utilization to predict the future utilization and give the usage prediction. In [12], they develop a heuristic algorithm which is based on Artificial Fish Swam Algorithm (AFSA) to solve the energy optimization problem. The prediction model in [13] is built by Fuzzy Information Granulation (FIG) and Support Vector Regression (SVR). The main work of [14] focus on resource prediction engines (RPEs) and epidemic models. The authors in [15] establish the prediction algorithm based on BP neural network. Besides, there are another methods used to predict the resource usage and optimal allocation, such as Simulated Annealing Algorithm (SA) in [16], GEO-CLUSTERING ALGORITHM in [17], Deep Learning Approach in [18] and so on.

3 Prediction Model

3.1 Architecture Description

In order to give accurate prediction of execution time, we proposed a prediction model which is called feedback prediction model (FPM). In this model, it combines most affected factors which may bring strong influence to execution time,

such as CPU's current status α, ram's current status β, the jitter of CPU and memory, network bandwidth B, length of data l and so on. Firstly, we divide the prediction model into two parts, part one is for cpu usage and ram usage prediction, part two is for time prediction Secondly, the division always bring error accumulation. So we build a feedback process for predicted correction after comparing the training data to real-time data. Thirdly, we set a threshold as a judgment. When prediction error of time is less than the threshold, model stops the feedback process and continues the next training process. Defining t_p, t_e are the predicted time,expected time. Prediction process is shown in Fig. 1 as follows.

$$\begin{aligned} &X = [x_1, .x_i., x_n], Y = [y_1, .y_j., y_m], \\ &m = 3, n = 6 \end{aligned} \tag{1}$$

Where X and Y are input layer and middle layer.

$$\begin{aligned} y_j &= \mathbf{W}_j \mathbf{F}_j(\mathbf{X}) \\ t &= \mathbf{V}\mathbf{G}(\mathbf{Y}) \end{aligned} \tag{2}$$

Where t is output, $\mathbf{W}_j = [w_j^1, ., w_j^i, .w_j^n]^T, \mathbf{V} = [v_1, .v_j, .v_m]^T$ are weight vectors. If middle layer node y_j and input layer node x_i are independent to each other, $w_j^i = 0$. So this network is not all connected. As the training process goes on, we use the predicted error Δz to correct y_j and update weight vectors $\mathbf{W}_j, \mathbf{V}$. When $||\Delta z||_2^2 < \eta$, the model stops feedback process and finish training.

$$\begin{aligned} &\Delta t_k \longrightarrow (\Delta(\mathbf{Y}_k)) \\ &y_j + \Delta y_j = \mathbf{W}_j^k \mathbf{F}_j(\mathbf{X}) \\ &t^k = \mathbf{V}^k \mathbf{G}(\mathbf{Y}_k) \end{aligned} \tag{3}$$

where k is number of iterations.

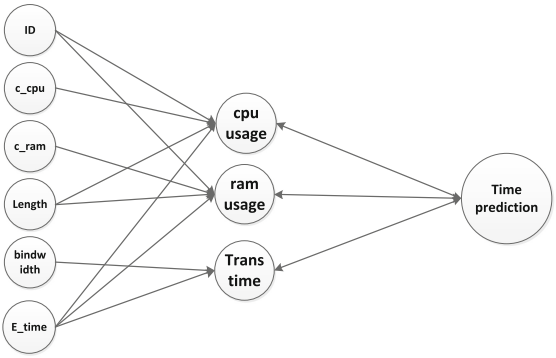


Fig. 1. Prediction process, ID is the number of offloading method, c_cpu(%) is the current cpu status, c_ram(Mb) is the current ram status, length(kb) is the string length of data, E_time(ms) is the expected time t_e.

3.2 Prediction Process

Usage Prediction. In this part, as followed in Fig. 1, we transform the input parameters $(id, \alpha, \beta, B, l, t_e)$ to three outputs cpu usage U_c, ram usage U_r and transmission time t_t. For t_t, it relates to computing data l and bindwidth B directly. And assuming that t_t is independent with computing machine's status. So we can define the fitting function as

$$t_t = \frac{l}{8B} + \upsilon_t. \tag{4}$$

Where υ_t is transmission noise, and $\upsilon_t \sim N(0, \delta_t^2)$.

Excepting for the measurement values above, we have used multi threading method to get the real-time resource usage data during the offloading process. In general neural network, predicted values can have continuous corrections during data training process. But measurement values of hidden-layers' outputs are always unknown and the corrections always depend on other nodes. It may make the hidden-layers nodes' unsensitive and bad performance of network. Comparing to neural network, the proposed prediction model set usage prediction as middle-layer and the real-time data helps to correct the training result of each layer. Usage computing function can express as follows.

$$y_j = \int_{t_{start}}^{t_{end}} F_j(t, \alpha, l, id)dt + \upsilon_j. \tag{5}$$

Where $t_{end} - t_{start} = t_e$, υ_j respect for the shaking influence of current CPU and RAM, $\upsilon_j \sim N(0, \delta_j^2)$. The id means the complexity of each offloading method. F_j is the changing curve of resource j during the execution time. When method is known, prediction function F can be changed to

$$F_j(t, \alpha, l, id) = F_j(t, \alpha, \lambda(l, id)). \tag{6}$$

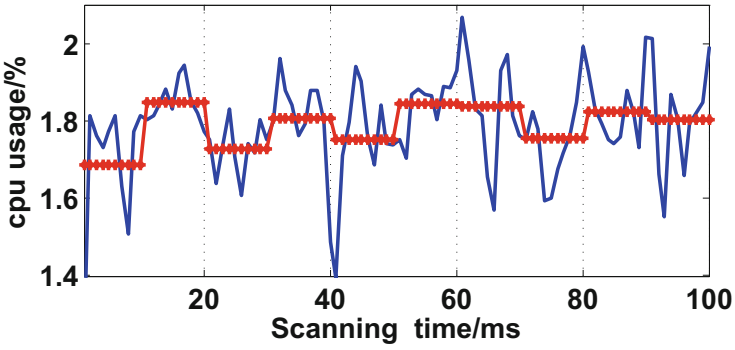


Fig. 2. Average value $\hat{F}, \hat{H}$ prediction. The blue line is the real-time changing curve, and the red line means the average value of CPU or RAM usage in a expected execution time. (Color figure online)

Where λ is the computation function. When the training process start, F_j will be given. Meanwhile the usage can be evaluated through definite integral. However, because of the shaking characteristics and changing regular of CPU, F_j are difficult to training or the rule of change curves is not obvious in unit time as depicted in Fig. 2. Thus, we use average value $\hat{F}$ of cpu status instead of F during each offloading process. The predicted value are given as

$$\tilde{y_j} = \hat{F}_j \bullet t_e + \upsilon_j. \tag{7}$$

Where $\hat{F}_j$ are the average value of cpu status and ram status, $\tilde{y_j}$ are the predicted values. And we transform the training targets from y_j to $\hat{F}_j$. For training number N, the training inputs and outputs are the $\frac{(y_j - \upsilon_c)}{t_e}, \hat{F}$. Conversely, real execution time probably not equals the expected time. Therefore $\hat{F}_j$ need corrections ΔF_j to reach the expected value. Defining predicted average usage values $\tilde{F}_j = \hat{F}_j + \Delta F_j$. According to test, the cpu usage and ram usage follow the Gauss distribution $N(y_j^0, (\sigma_j^c)^2)$. So the best training result F_j^o can be given as

$$\begin{aligned} &E(y_j) = y_j^0 = t_e * E(F_j^o) \\ &D(F_j^o) = \frac{D(y_j) + (\sigma_j^c)^2}{t_e^2} \\ &F_j^o \sim N(\frac{y_j^0}{t_e}, \frac{D(y_j) + (\sigma_j^c)^2}{t_e^2}) \end{aligned} \tag{8}$$

Where y_j^0 are the expected value of measurement values y_j. Defining predicted values as $\tilde{y_j}$, cpu usage predicted error $\Delta y_j = \tilde{y_j} - y_j, E(\Delta y_j) = 0, D(\Delta y_j) = (\delta_j^{\Delta})^2$.

$$\begin{aligned} &\Delta y_j = g(t_e) \\ &\Delta F_j = \frac{\Delta y_j}{t_e} \end{aligned} \tag{9}$$

The final predicted function can be given as

$$\tilde{y_j} = (\hat{F}_j + \Delta F_j) \bullet t_e + \upsilon_j. \tag{10}$$

Besides, $E(\Delta F_j) = 0$, performance evaluation are given like

$$\begin{aligned} &E(\tilde{y_j}) = y_j^0, \\ &D(\tilde{y_j}) = t_e^2 * D(\hat{F}_j) + (\delta_j^{\Delta})^2 + \delta_j^2 \end{aligned} \tag{11}$$

Time Prediction. When the model finish the usage training, we can use resource usage to predict offloading time t_o. The offloading process includes two independent parts, one is data transition time t_t, another is the execution time t_x. So

$$t_o = t_x + t_t + \upsilon_t \tag{12}$$

Where υ_t is the transition noise. t_x relates to $\mathbf{Y} = [\tilde{y_1}, \tilde{y_j}, \tilde{y_m}]$.

$$\begin{aligned} &\tilde{t_x} = \mathbf{VG}(\mathbf{Y}) \\ &\Delta t = \tilde{t_x} - t_x \\ &\Delta t = \Delta \mathbf{VG}(\mathbf{Y}) \end{aligned} \tag{13}$$

Where $\tilde{t_x}$ is the predicted time, Δt is prediction error, t_x is the real execution time, $\Delta \mathbf{V}$ is adjust weights vector.

Feedback Process. When forward propagation process finish, feedback process start back propagation. From part two, we can obtain the predicted error Δt. In part one, predicted usage $\tilde{y_j^k}$ change to

$$\tilde{y_j^k} = (\hat{F}_j + \Delta F_j) \bullet (t_e + \Delta t_j) \tag{14}$$

Then feedback process start back propagation and correct the usage predictions. Meanwhile weights of usage prediction model update by the corrections. So the usage correction can be given like

$$\Delta \tilde{y_j^k} = (\hat{F}_j + \Delta F_j) \bullet \Delta t_j \tag{15}$$

Time prediction process will be re-execute after usage correction. Therefore time prediction error Δt will be updated. If $\Delta t > \eta$, η is the given training threshold, model will go on back propagation and error feedback. Otherwise, when feedback process satisfy the addition of $\Delta t < \eta$ or reach the setting max training number, model will skip to next training. In addition, we give a max training threshold ρ to avoid the training divergence, if $\Delta t > \rho$, model stop back propagation.

The iterative process is as follows. Training flow chart is shown in Fig. 3.

Algorithm 1

Input: $\mathbf{X}, \mathbf{Y}$
Output: $\mathbf{W}, \mathbf{V}$

for $i \leq P$ **do**
  $\tilde{y_j^i} = (\hat{F_j^i} + \Delta F_j^i) \bullet t_e^i$
  $\tilde{t_x^i} = \mathbf{V}^i \mathbf{G}(\mathbf{Y}^i)$
  $\Delta t^i = \tilde{t_x^i} - t_x^i$
  $\Delta t^i = \mathbf{\Delta V G}(\mathbf{Y^i})$
  $\mathbf{V}^i = \mathbf{V}^i + \mathbf{\Delta V}$
  while $||\Delta t^i||_2^2 > \eta \quad \&\& \quad k < N$ **do**
    $\tilde{y_j^{ik}} = (\hat{F_j^{ik}} + \Delta F_j^{ik}) \bullet (t_e^i + \Delta t^i)$
    $y_j^i + \Delta y_j^{ik} = \mathbf{W}_j^k \mathbf{F}_j^i(\mathbf{X}^i)$
    $t^k = \mathbf{V}^k \mathbf{G}(\mathbf{Y}^k)$
    $\Delta t^{ik} = t_e^i - t^k$
    if $||\Delta t^{ik}||_2^2 < \rho$ **then**
      ccontinue
    else
      break
    end if
    $k++$
  end while
end for

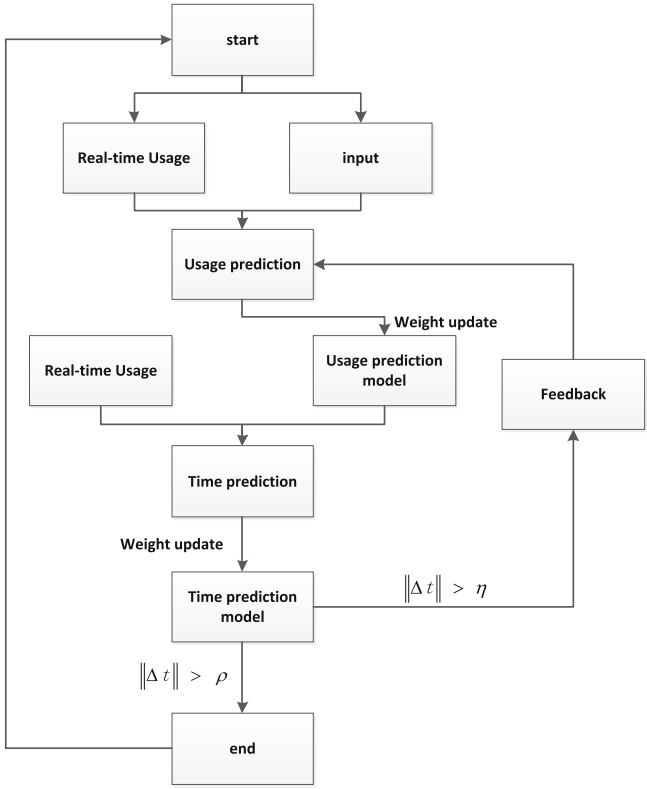


Fig. 3. Flow chat of FPM.

4 Experiment and Analysis

In order to test the performance of FPM, we set up FPM experiment system to obtain the needing training data. Meanwhile the system include five test offloading methods which are Optical Character Recognition (OCR), AES, SHA, MD5 (three encryption algorithms) and indoor localization method (GEO). We measured all parameters mentioned in FPM and give performance comparison with BP neural network and FPM without feedback process (NO-FP FPM).

4.1 Performance of Usage Prediction

Usage predictions include cpu usage prediction and ram usage prediction. From Figs. 4 and 5, we can see the predicted performance with three prediction models which are FPM, NO-FP FPM, BP neural network. Comparing to the real data, three models' prediction errors are shown in Table 1. Besides, each method has different requirements for memory and CPU. It always depends on its own algorithm complexity and the length of the processed data.

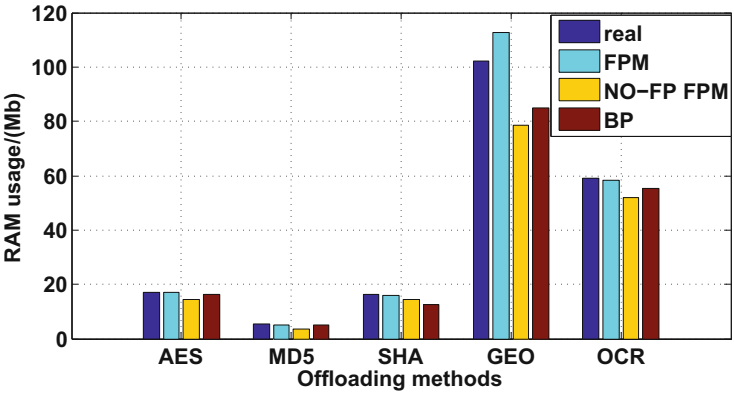


Fig. 4. Comparison of RAM usage prediction.

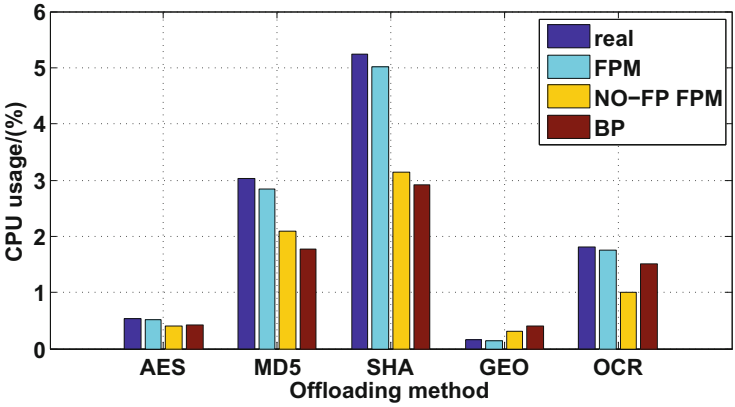


Fig. 5. Comparison of CPU usage prediction.

Table 1. Comparison of usage prediction errors.

Kind	Method	AES (%)	MD5 (%)	SHA (%)	GEO (%)	OCR (%)
CPU	FPM	0.87	3.56	3.54	10.30	1.31
CPU	NO-FP FPM	15.34	33.90	12.90	22.89	12.48
CPU	BP	5.43	6.21	24.35	16.82	6.57
RAM	FPM	5.56	6.27	4.38	6.25	3.31
RAM	NO-FP FPM	24.07	30.69	40.07	93.75	44.19
RAM	BP	20.37	41.58	44.47	156.25	16.57

Table 2. Comparison of time prediction.

Method	AES (ms)	MD5 (ms)	SHA (ms)	GEO (ms)	OCR (ms)
Real	331.9	239.8	205.6	16.2	1332.8
FPM	349.5	231.0	215.4	15.0	1375.3
NO-FP FPM	297.7	259.8	192.1	22.6	1262.5
BP	270.5	208.7	234.4	19.9	1398.8

In Figs. 4 and 5, without feedback process, NO-FP FPM has a large prediction error which can reach nearly 100%. Thus, average value prediction $\hat{F}$ has a large deviation and it is necessary to give a correction. In Table 1, FPM's prediction accuracy can be raised at least 10 times after feedback process which uses output error to finish the backward propagation process. Similarly, we also tested the BP neural network with the same backward propagation process. It is better than NO-FP FPM. However, prediction errors still reach about 5 times of FPM. And the large prediction errors will accumulate to the time prediction.

4.2 Performance of Time Prediction

After finish usage prediction process, we can use them for time prediction. Then, we have tested 500 times for each model. Likewise, we give the prediction performances of three models in Fig. 6 and comparison between predictive values and real values in Table 2.

In Fig. 6(a), (b), (c), (d) and (e) represent the comparison of predicted results of three training models (FPM, NO-FP FPM, BP) respectively. And the results are shown by their probability density function (PDF). In general, we hope that the predicted results are as close to the true distribution of real-time data as possible. In addition to predicted expectations, predicted errors of each model is affected by predicted variances. In each comparison figures, probability density curve of time prediction of FPM is closest to the real curve. And it's hard to choose a sub-optimal model between NO-FP FPM and NP neural network.

In Fig. 6(f), we can see the comparisons of prediction accuracy with the use of three training model. We can clearly see that FPM is the best model and the best accuracy can reach about 1%. In the estimation of AES and GEO, BP neural network model is better than NO-FP FPM. But for MD5, SHA and OCR NO-FP FPM is better.

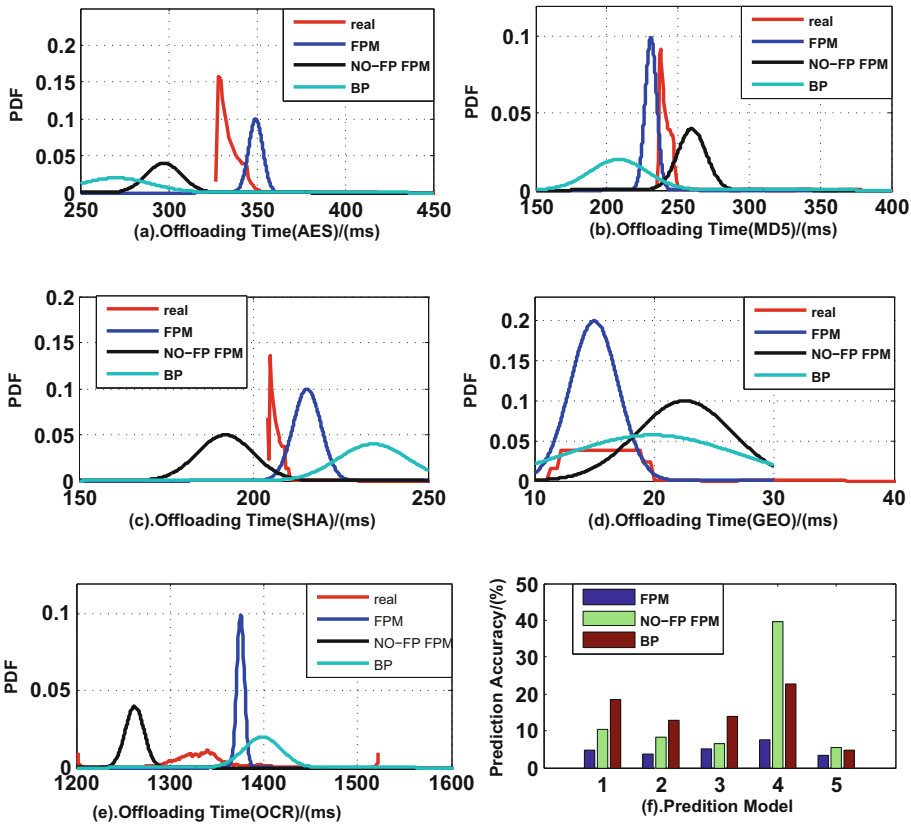


Fig. 6. PDF comparation of three models.

5 Conclusion

In this paper, we propose a time prediction model (FPM) during the computation offloading in edge computing platform. In FPM, we divide the model into two parts, resource usage prediction and time prediction. In usage prediction part, we transfer the predicted target from usage to average usage. In time prediction part, we utilize the training error to correct the predicted model and complete the back propagation until the error reach the set threshold. Then we use the proposed model to do the prediction process and give the comparisons of predicted results with the other two models which are BP neural network and NO-FP FPM. It shows a good performance and best accuracy can be less than 6%.

References

1. Yang, B., Chai, W.K., Xu, Z., Katsaros, K.V., Pavlou, G.: Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications. IEEE Trans. Netw. Serv. Manag. **15**(1), 475–488 (2018)
2. Barik, R.K., Dubey, H., Mankodiya, K.: SOA-FOG: secure service-oriented edge computing architecture for smart health big data analytics. In: 2017 IEEE Global Conference on Signal and Information Processing (Global SIP), pp. 477–481. IEEE Press, New York (2018). https://doi.org/10.1109/SC2.2017.11
3. Premsankar, G., Di Francesco, M., Taleb, T.: Edge computing for the Internet of Things: a case study. IEEE Internet Things J. **5**, 1275–1284 (2018)
4. Baike for Edge Computing. http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E8%AE%A1%E7%AE%97/9044985?fr=aladdin
5. Zhang, J., Xia, W., Yan, F., Shen, L.: Joint computation offloading and resource allocation optimization in heterogeneous networks with moble edge computing. IEEE Access **pp**(99), 1 (2018)
6. Hao, Y., Chen, M., Hu, L., Hossain, M.S., Ghoneim, A.: Energy efficient task caching and offloading for mobile edge computing. IEEE Access Spec. Sect. Mob. Edge Comput. **6**, 11365–11373 (2018)
7. Luo, C., Salinas, S., Li, M., Li, P.: Energy-effecient autonomic offloading in mobile edge computing. In: 15th International Conference on Dependable, Autonomic and Secure Computing, 15th International Conference on Pervasive Intelligence and Computing, 3rd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress, pp. 581–588. IEEE Press, New York (2017). https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2017.104
8. Huang, S.-C., Luo, Y.-C., Chen, B.-L., Chung, Y.-C., Chou, J.: Application-aware traffic redirection: a mobile edge computing implementation toward future 5G networks. In: IEEE 7th International Symposium on Cloud and Service Computing, pp. 17–23. IEEE Press, New York (2017). https://doi.org/10.1109/SC2.2017.11
9. Xia, Y., Ren, R., Cai, H., Vasilakos, A.V., Lv, Z.: Daphne: a flexible and hybrid scheduling framework in multi-tenant clusters. IEEE Trans. Netw. Serv. Manag. **15**(1), 330–343 (2018)
10. Melhem, S.B., Agarwal, A., Goel, N., Zaman, M.: Markov prediction model for host load detection and VM placement in live migration. IEEE Access **pp**(21), 7190–7205 (2018)
11. Yang, X., Chen, Z., Li, K., Sun, Y., Liu, N., Xie, W., Zhao, Y.: Communication-constrained mobile edge computing systems for wireless virtual reality: scheduling and tradeoff. IEEE Access 1–13 (2018)
12. Yang, L., Zhang, H., Li, M., Guo, J., Ji, H.: Mobile edge computing empowered energy efficient task offloading in 5G. IEEE Trans. Veh. Technol. **pp**(11), 1–12 (2018)
13. Zhao, P., Tian, H., Fan, S., Paulraj, A.: Information prediction and dynamic programming based RAN slicing for mobile edge computing. IEEE Wirel. Commun. Lett. 1–4 (2018)
14. Kryftis, Y., Mastorakis, G., Mavromoustakis, C.X., Batall, J.M., Rodrigues, J.J.P.C., Dobre, C.: Resource usage prediction models for optimal multimedia content provision. IEEE Syst. J. **11**(4), 2852–2863 (2017)
15. Wang, X., Wang, X., Che, H., Li, K., Huang, M., Gao, C.: An intelligent economic approach for dynamic resource allocation in cloud services. IEEE Trans. Cloud Comput. **3**(3), 275–289 (2015)

16. Cao, Z., Lin, J., Wan, C., Song, Y., Zhang, Y., Wang, X.: Optimal cloud computing resource allocation for demand side management in smart grid. IEEE Trans. Smart Grid **8**(4), 1943–1955 (2017)
17. Bouet, M., Conan, V.: Mobile edge computing resources optimization: a geo-clustering approach. IEEE Trans. Netw. Serv. Manag. (2018)
18. Yu, S., Wang, X., Langar, R.: Computation offloading for mobile edge computing: a deep learning approach. In: IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), pp. 1–6 (2017)

Application and Industry Track: Cloud Service System

cuCloud: Volunteer Computing as a Service (VCaaS) System

Tessema M. Mengistu(✉), Abdulrahman M. Alahmadi, Yousef Alsenani, Abdullah Albuali, and Dunren Che

Department of Computer Science, Southern Illinois University at Carbondale, Carbondale, USA
{tessema.mengistu,aalahmadi,yalsenani,aalbuali,dche}@siu.edu

Abstract. Emerging cloud systems, such as volunteer clouds and mobile clouds, are getting momentum among the current topics that dominate the research landscape of Cloud Computing. Volunteer cloud computing is an economical, secure, and greener alternative solution to the current Cloud Computing model that is based on data centers, where tens of thousands of dedicated servers are setup to back the cloud services. This paper presents cuCloud, a Volunteer Computing as a Service (VCaaS) system that is based on the spare resources of personal computers owned by individuals and/or organizations. The paper addresses the design and implementation issues of cuCloud, including the technical details of its integration with the well-known open source IaaS cloud management system, CloudStack. The paper also presents the empirical performance evidence of cuCloud in comparison with Amazon EC2 using a big-data application based on Hadoop.

1 Introduction

Cloud Computing, the still fast growing and evolving technology, has kept drawing significant attention from both the computing industry and academia for nearly a decade. Extensive researches in Cloud Computing have resulted in the fulfillment of many of the promised characteristics, such as utility computing, pay-as-you-go, flexible (unlimited) capacity, etc. Emerging cloud systems such as ad hoc and mobile clouds, volunteer clouds, federation of clouds, and hybrid clouds are among the research topics that may jointly help to reshape the landscape of future Cloud Computing – to better satisfy computing needs.

Currently, more than 4 billion users use the Internet[1]. Supercomputing sites and large cloud data centers provide the infrastructures that host applications like Facebook and Twitter, which are accessed by millions[2] of users concurrently. The current cloud infrastructures that are based on tens (if not hundreds) of thousands of dedicated servers are expensive to setup; running the infrastructure needs expertise, a lot of electrical power for cooling the facilities, and redundant

[1] http://www.internetworldstats.com/.
[2] http://www.internetlivestats.com/.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 251–264, 2018.
https://doi.org/10.1007/978-3-319-94295-7_17

supply of everything in a data center to provide the desired resilience. Cloud servers, specialized infrastructure for fault tolerance, and the electricity consumption account for 45%, 25%, and 15% of the total amortized cost of a cloud data center, respectively [1]. The servers and the cooling system make up about 80% of all the electricity consumption in a typical data center [2]. The high energy consumption in data centers contributes to the environmental impact of global warming. Currently, the ICT industry contributes 2% of the total greenhouse gases emission and the contribution of data centers to this emission is expected to increase to 18% by 2020 [3]. On top of the costs and environmental factors, the current "datacenter based" cloud infrastructures, even though well-provisioned and well-managed, are not suitable for some application scenarios. For instance, the inevitable concerns of cloud clients for "no control" on their confidential data and business logic can prohibit them from migrating their applications and datasets to the clouds that are in the hands of third-parties. Self-provisioned, on-premise, private cloud can be a suitable solution to these users. However, the initial investment for setting up such a private cloud infrastructure based on a dedicated data center can be prohibitively expensive.

In addition to the vast number of dedicated servers setup in cloud data centers, there are billions of Personal Computers (PCs) owned by individuals and organizations worldwide. These PCs are underutilized, usually used only for a few hours per day [4]. The computing power of these computers can be consolidated as a huge cloud fabric and utilized as an alternative Cloud Computing solution, i.e., volunteer cloud computing. Volunteer cloud computing based on consolidating the spare capacities of edge computing resources (within an organization or community) and delivering on-premise private clouds is a rather economical and greener alternative to the conventional "data center based" Cloud Computing solutions.

Volunteer clouds come with multi-folds of benefits: no upfront investment for procuring a large number of servers that are otherwise inevitable for data center hosting, no maintenance costs such as electricity consumption for cooling and running the servers in a conventional cloud data center, and boosting the utilization of edge computing resources (such as individually owned PCs). In the meantime, volunteer cloud computing introduces its own technical challenges that are centred on the high dynamics and high heterogeneity of the volunteer computers that are shared not only among the cloud users but also between the cloud users and the local users of the machines. The key issues in Cloud Computing such as availability, reliability, security, and privacy all need to be readdressed in a more serious and critical way in volunteer cloud computing. Nevertheless, exploitation of the untapped excessive capacity of the numerous edge computers owned by individuals and/or organizations for volunteer cloud computing is a worthwhile endeavour [5]. We believe "no data center based" volunteer cloud computing is one of the most promising future research directions of Cloud Computing [6,7].

This paper discusses cuCloud, a Volunteer Computing as a Service (VCaaS) system that is based on the spare resources of personal computers owned by

individuals and/or organizations. The paper addresses the design and implementation issues of cuCloud, including the technical details of its integration with the well-known open source IaaS cloud management system, CloudStack. The paper also presents the empirical performance evidence of cuCloud in comparison with Amazon EC2 using a big-data application based on Hadoop. To the best of our knowledge, cuCloud is the first volunteer cloud system that is implemented using an existing IaaS system with empirical evidence on its performance in comparison with data center based cloud infrastructure.

The rest of the paper is organized as follows. Section 2 discusses cuCloud and its client/server architecture in detail. It also elaborates on the internal details of CloudStack together with the implementation of cuCloud using CloudStack as a base support. Section 3 presents empirical performance comparison between cuCloud and Amazon EC2 using a big-data application. Section 4 reviews related works in regard to volunteer cloud computing and Sect. 5 concludes the paper.

2 Design and Implmentation of cuCloud

cuCloud is a Volunteer Computing as a Service (VCaaS) system that runs over an existing computing infrastructure, which is not specifically setup for cloud purposes. The system thus can be considered as an opportunistic private/public cloud system that executes over scavenged resources of member PCs (also referred to as member nodes or volunteer hosts) within an organization (or community). Being non-dedicated, the PCs have other primary purposes, such as word processing, data entry, web browsing, etc. Only the residual resource capacity that is not used by the local processes is going to be extracted and utilized by cuCloud. This setting decides numerous appealing advantages of cuCloud: affordability, on-premise, self-provision, and greener computing. On the other hand, full-fledged implementation of cuCloud raises unique technical challenges: efficient management of highly dynamic and heterogeneous compute resources, QoS assurance, and security/trust, which are all made more difficult due to the high dynamic availability and heterogeneity of the non-dedicated volunteer hosts. The following subsections discuss the design and implementation issues of cuCloud in detail.

2.1 Design of cuCloud

The cuCloud adopts a client/server architecture with the volunteer hosts being the clients and the dedicated management machine(s) being the server(s). The server has various components as depicted in Fig. 1. The following are descriptions for each of the server side components.

– **Interface:** is the first port of communication between users and cuCloud. cuCloud has different types of users: *admin users*, *cloud users*, and *local users*. *Admin users* use the *Interface* to manage the cuCloud system. The *cloud users* (a.k.a clients or customers) specify their resource requirements and manage

the allocated resources via the *Interface*. *Local users* (native users) are the ones who contribute their spare computing resources to the cuCloud system and use the *Interface* to check their resource contribution details, credits earned, etc.

- **Authentication and Authorization:** component handles the user level authentication, authorization, and security issues of cuCloud. *Admin users* and *cloud users* should be preregistered and should be authenticated and authorized before start using the system.
- **Resource Manager and Allocator (RMA):** component is the one that manages all the volunteered computing resources possessed by cuCloud. It communicates with the cuCloud member nodes and periodically updates the availability of the resources from each member node. The component keeps the reliability and availability profiles of all volunteer hosts. The profiles together with the Service Level Agreement (SLA) are the basis of selecting volunteer hosts for deploying Virtual Machines (VMs) to best satisfy *cloud users'* resource requirements and cuCloud's system scheduling criteria.
- **Scheduler:** component receives *cloud users'* requests and decides whether to admit or deny the requests in consultation with the RMA and the Virtual Machine Manager components. The Scheduler makes the admission decision and allocation of VM(s) to host(s) based on the dynamically updated resource profiles maintained by the RMA. The Scheduler also handles the (re)allocation of VMs in cooperation with the Virtual Machine Manager.
- **Virtual Machine Manager (VMM):** component handles the deployment of VMs on volunteer hosts. More specifically, it performs the following tasks regarding VM management: create a VM, migrate (live and cold) a VM, suspend and resume a VM, kill and restart a VM, etc. The component also monitors all deployed VMs and keeps track of their progress information.
- **Security Module (SM):** component handles the security of the VMs deployed in cuCloud. There is no assumption of trust relationship between the volunteer hosts and the VMs in cuCloud. Therefore, the SM secures VMs from malicious volunteer hosts and vice versa.
- **Monitoring and Management:** component provides fine-grained information about the aggregate computing resources of the cuCloud. This includes, but not limited to, details of donated resource from each individual volunteer host, e.g., the total number of CPU cores, aggregated memory size, total storage capacity, used/available CPU cores, etc.

In the system architecture (Fig. 1) of the cuCloud, member nodes are subsidiary to the server. There is a critical piece of software, called Membership Controller (MC), that resides on each member node that contributes resources to the cuCloud system. Membership Controller monitors the resource utilization of processes at each volunteer node and decides the node's membership status, which is highly dynamic. It also decides the availability and reliability of the node based on the member's historical data and the current resource availability. The MC collects and sends information to the server about the types and quantities of the available resources (CPU, RAM, Hard Disk) for contribution

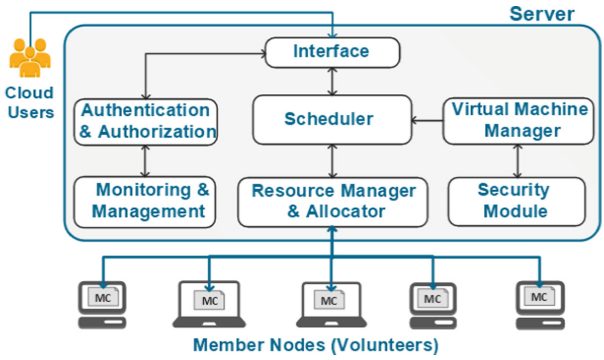


Fig. 1. System architecture of cuCloud

to the resource pool of cuCloud. It also allows the owner of a volunteer host to adjust the proportions of the node's resources for contribution to the cuCloud system. The MC has the following components (Fig. 2):

- **Sensor:** component periodically monitors the resource utilization of processes (both local and guest) on a volunteer host and sends that information to the *Node Profiler*. The total resource capacity of the host such as the number of CPU cores, RAM, and Hard Disk capacity as well as the used and idle resources are monitored periodically. The *Sensor* also collects other pieces of data that are used by the *Node Profiler*.
- **Node Profiler:** is the heart of the Membership Controller. It periodically accepts the resource utilization information from the *Sensor* and calculates the residual computing resource availability information. It also predicts the future availability and reliability of the volunteer host using the historical data it receives from the *Sensor*. The historical data includes time to failure, mean time between failures, mean time to repair, etc. This reliability and availability profile will be used by the RMA for further processing that may result in actions taken by the VMM, for example migration of VM(s).
- **Reporter:** The availability and reliability profile of a volunteer host is reported to the *Resource Manager and Allocator* of the server of cuCloud via the *Reporter*. The *Reporter* periodically accepts the profile information from the *Node Profiler* and pushes the information to the server of cuCloud.
- **Virtual Environment Monitor:** is the component that manages the VMs that are deployed on a volunteer node. It monitors the deployed VMs on a host and passes information regarding their progress and status to the *Node Profiler* for building the availability and reliability profile of the host.
- **Policy:** component stores the *native user's* preferences regarding donated resources, such as the time the resource should not be used, the number of cores or the RAM capacity donated, etc. This policy information is used by the *Node Profiler* for building the profile of the volunteer host.

The cuCloud, a "No Data Center" cloud system [8], is expected to fulfil all the characteristics of Cloud Computing systems such as elasticity, metered services, resource pooling, networked access, and automated on-demand service provisioning. Besides, service provisioning using non-dedicated, highly heterogeneous, and dynamically available volunteer hosts are idiosyncratic to cuCloud. The following are the design goals of the cuCloud.

- **Volunteer based:** Participation in cuCloud is purely voluntary. No host will be forced to join the virtual infrastructure base of cuCloud.
- **Unobtrusive:** The local processes of a volunteer host always have higher priority than the guest processes (cuCloud's VMs).
- **Scalable:** It should be scalable to tens of thousands of volunteer hosts.
- **Easy to use and deploy:** Usage and deployment of cuCloud should be as simple as possible.
- **Security:** The volunteer hosts should securely execute the deployed VMs without tampering them and vice versa.

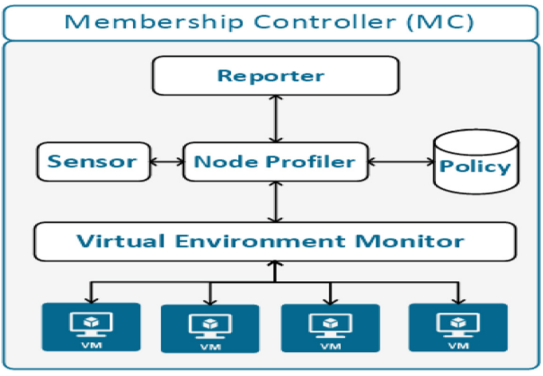


Fig. 2. Membership Controller of cuCloud

2.2 CloudStack Based Implementation of cuCloud

We chose CloudStack[3], an open-source Infrastructure-as-a-Service (IaaS) management platform, to implement cuCloud. Apache CloudStack manages and orchestrates pools of storage, network, and compute resources to build a public or private IaaS cloud. It is a cloud system that supports multiple hypervisors, high availability, complex networking, and VPN configurations in a multitenant environment. According to a survey [9], CloudStack has a 13% adoption rate as a private cloud solution for enterprises. CloudStack is chosen for the implementation of cuCloud because of its modular design, rich documentation, active online development community, and highly extensible architecture. By using

[3] http://cloudstack.apache.org/.

CloudStack for the implementation of cuCloud, we can make use of its rich set of functionalities such as Account Management, Virtual Machine Management, etc., with little or no modification. This will increase the adoption of cuCloud and avoid reinventing the wheel.

CloudStack follows a distributed client-server architecture where the Management Server(s) acts as a server and compute nodes act as clients. CloudStack Management Server can manage tens of thousands of physical servers (compute nodes) installed in geographically distributed datacenters. The Management Server (MS) communicates with the compute nodes through the hypervisors on the machines. Type I hypervisors such as Xen, Hyper-V, and VMWare are supported in CloudStack.

As depicted in the basic layered architecture of CloudStack Management Server (Fig. 3), the business logic is the one that accepts the user/admin requests and applies the necessary checks such as authentication or resource availability before sending it to the next layer. The Orchestration layer is responsible for configuring, provisioning, and scheduling any operations such as VM creation or storage allocation. Controllers are the one that directly communicate with the underlining computing resources such as computing units and networking devices, for the actual provisioning of resources. CloudStack is designed to be extensible in such a way that plugin APIs are provided for extending the functionality or modifying its behaviour [10]. Thus, a new plugin can be defined and integrated with CloudStack as needed.

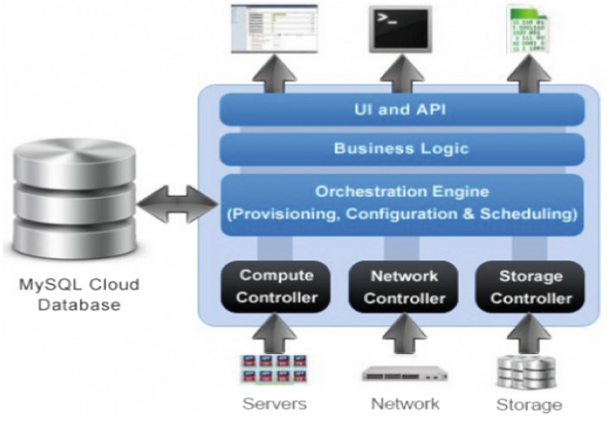


Fig. 3. Layered architecture of CloudStack [10]

CloudStack has *Resource Manager* component that is responsible for managing the compute, storage, and network resources. The addition of a new resource (compute node or storage) in CloudStack is manual, i.e., the admin should give the resource's attributes such as IP address, root password, hypervisor type, etc., via a web-based form. The resource information will then be stored in a MySQL based database. Since CloudStack is an IaaS system that is developed

to manage dedicated servers in data centers, we need to modify the Management Server of CloudStack in such a way that it can handle the non-dedicated volunteer nodes that we use as cloud infrastructure in cuCloud. This needs a fundamental change of the *Resource Manager* component of the MS for two reasons. First, the non-dedicated and high churn rate of the volunteer hosts coupled with the unobtrusive nature of cuCloud require an automatic addition of resources instead of the usual manual addition. Second, once a compute node is added manually to the CloudStack database, the MS pulls servers to check their availability and updates the status of the node in the database. This pull mechanism is not scalable for a highly dynamic resource base of cuCloud. Therefore, a push mechanism should be deployed in cuCloud. Volunteer hosts push availability and reliability information periodically to the MS and if the MS doesn't receive this information for a certain duration, it will mark the volunteer host as unavailable. A new plugin, called *AdHoc component*, is developed to handle the volunteer hosts in coordination with the *Resource Manager* of CloudStack.

The Membership Controller (MC) (see Fig. 2) on the volunteer hosts continuously monitors the resource utilization of processes on the host. This resource utilization information will be used, together with the policy information, by the *Node Profiler* to decide the availability and reliability of the volunteer host [16]. Based on this information, the MC sends "active" availability message to the *AdHoc component* of the CloudStack Management Server, if the volunteer node is available to host Virtual Machine(s), "inactive" otherwise. Algorithm 1 describes the actions the *AdHoc component* takes after receiving the message from the volunteer node.

The host (compute node) is the smallest organizational unit within CloudStack deployment. Every host should have a hypervisor such as Xen or KVM, which is abstracted as *ServerResource* in CloudStack. The Management Server cannot directly communicate with the hypervisor of a compute node, instead it communicates with *ServerResource* through the Agent. *ServerResource* is a translation layer between the hypervisor on a compute node and the commands of the Management Server. The *ServerResources* makes the inclusion of a new hypervisor support in CloudStack simple.

Due to the full self-autonomy of volunteer hosts and the non-intrusiveness of cuCloud, we cannot use type I hypervisors like Xen or Hyper-V that takes full control of the underline hardware of a volunteer host. Instead, we need a virtualization solution that runs along with local processes on member nodes. Therefore, for the virtualization environment at member nodes, we can use Type II hypervisors like VirtualBox or KVM. The current version of CloudStack (4.11) supports KVM but not VirtualBox. Adding a support for VirtualBox in CloudStack is one of the entries in the to-do list of cuCloud. With the inclusion of *AdHoc component* at the Management Sever and based on our client architecture, we extend CloudStack in such a way that hosts (compute nodes) can be added and removed from the resource pool dynamically, which is in high contrast with the dedicated nature of the compute nodes in the original CloudStack. Moreover, the VMs are now able to run along with non-cloud (local) tasks/processes on the volunteer hosts.

Algorithm 1. Member Node Addition Pseudo Code

```
1: message ← null;
   status ← null;
   message ← reciveFromHost();
2: if message is "active" then
3:    if host in DB of CloudStack then
4:       checkStatus();
5:       if status is "maintenance"  then
6:          status ← enabled;
7:       end if
8:    else
9:       addNewHost();
10:   end if
11: else
12:   if host in DB of CloudStack then
13:      checkStatus();
14:      if status is "enabled"  then
15:         status ← maintenance;
16:      end if
17:   end if
18: end if
```

3 Experimentation

In order to test the suitability of cuCloud for real applications, we run a big-data workload using Hadoop over the system. To gain a more detailed comparative performance, the same workload was run on Amazon EC2 with similar VMs specifications. We used BigDataBench[4] software for the benchmarking. The BigDataBench, which is a multi-discipline research and engineering effort from both industry and academia, is an open-source big data and AI benchmark [11]. The current version of BigDataBench provides 13 representative real-world data sets and 47 benchmarks. We used an offline analytics workload of PageRank calculation using Hadoop MapReduce. Three VMs were deployed on cuCloud for the experimentation (one master and 2 slaves). The master is deployed on a machine with 8 GB of RAM, Intel 8 Core i7 2.4 GHz CPU, and a hard disk of 250 GB. Each slave node has 8 GB of RAM, intel 4 Core i3 3.1 GHz CPU, and 250 GB hard disk capacity. All the machines run Ubunu 14.04, are connected to a 16 Gbps switch, and support intel hardware virtualization (VT-x). The experimentation for cuCloud was done in the computer labs at Southern Illinois University Carbondale. The PCs were being used by local users while the PageRank calculation was computed, i.e., they were not dedicated only for the cloud task. General-purpose instances with the same specifications as cuCloud VMs were configured on Amazon EC2. The availability zone of all the EC2 instances was us-east-2b of Ohio region. The details of the VMs for both the cuCloud and EC2 is given in Table 1.

[4] http://prof.ict.ac.cn/.

Table 1. VMs specification

	cuCloud			Amazon EC2		
Master	medium			t2.medium		
	CPU	RAM	HD	CPU	RAM	HD
	2	4	75	2	4	75
Slaves	small			t2.small		
	CPU	RAM	HD	CPU	RAM	HD
	1	2	50	1	2	50

The PageRank calculation was performed for different data sizes (numbers of nodes with the corresponding number of edges). Table 2 gives the dataset used in our experimentations. We run the experimentation for each data set 10 times at different time of the day and week (morning vs. evening and weekend vs. weekdays). Figure 4 depicts the average time required to run the application in both infrastructures.

Table 2. Experimental data size

	Nodes	Edges
Run1	16	21
Run2	256	462
Run3	65,536	214,270
Run4	1,048,576	4,610,034

The experimental results showed that cuCloud outperforms Amazon EC2 on the Hadoop MapReduce based PageRank calculation for all runs. This is mainly due to the comparative advantage of cuCloud over EC2 with regard to the network latency, as cuCloud runs over a LAN. The average round trip time (rtt) between the master and slaves for 10 packets is 10.5 ms for EC2 vs. 0.5 ms for cuCloud. Moreover, the runs on the general-purpose instances of EC2 showed a very large processing time variation. For instance, calculating the PageRank for Run3 took 352 min (during the afternoon) vs. 16 min (during early in the morning). On the other hand the processing times in the cuCloud were more or less uniform. Figure 5 depicts the variability of processing times of the 10 experiments performed on both systems for Run3 dataset. In general, the workloads deployed on EC2 run faster during night times and were slower during afternoons. From the preliminary experimental results we can conclude that for applications with a heavy inter-task network communications requirement like MapReduce, cuCloud performs well. This indicates that volunteer clouds have comparative advantages over public clouds, typically backed by remote data centers, in reducing the round trip time latency of applications. Therefore, volunteer

clouds are a perfect fit to the concept of edge computing since most applications naturally happen at the edge (of the Internet) and volunteer clouds can be most conveniently deployed to directly serve these applications – edge applications.

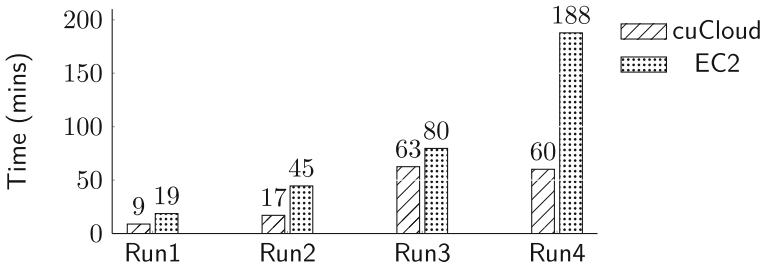


Fig. 4. Average processing time of workloads cuCloud vs. EC2

4 Related Works

The next generation of Cloud Computing needs fundamental extension approaches, to go beyond the capabilities of data centers. This means to include resources at the edge of the network or voluntarily donated computing resources, which are not considered in the existed conventional Cloud Computing model [6]. The concept of using edge resources for computing has been investigated in Cloud Computing as AdHoc Cloud [12], Volunteer Cloud Computing [13–15], and Opportunistic Cloud Computing [17].

In order to make use of and derive the benefit from the preexisting computing resources within an institution/organization scope, Rosales *et al.* built an opportunistic IaaS model through exploiting idle computing resources available in a university campus called UnaCloud [17]. The UnaCloud follows a client/server model, where the server hosts a web application that works as the main interface for all the UnaCloud services and the client is a lightweight application, based on the design concept of SETI@Home agent, installed on each node in the system. All the resources are homogeneous where each resource can handle one VM at a time in order to avoid resources competition. The UnaCloud doesn't provide any explanation on how to handle the associated challenges of non-dedicated computing resources such high churn rate, unreliability, and heterogeneity. On the other hand, the model lacks Cloud Computing basic features such as scalability, interoperability, and QoS. Another research work similar to cuCloud is the AdHoc Cloud Computing system proposed in [12]. The idea of AdHoc Cloud is to transform spare resource capacity from an infrastructure owned locally, but non-exclusive and unreliable, into an overlay cloud platform. The implemented client/server AdHoc cloud system is evaluated for its reliability and performance. In comparison with Amazon EC2, the author claimed

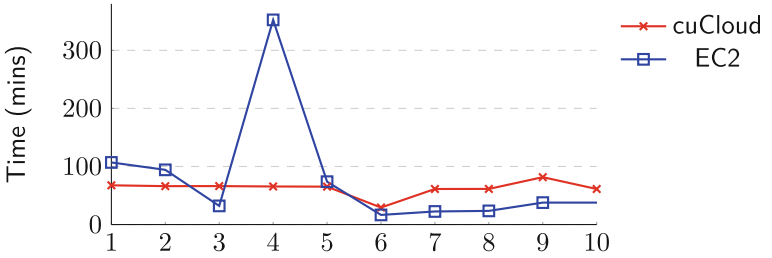


Fig. 5. Workload processing time variability cuCloud vs. EC2

a good comparable performance and concluded that AdHoc cloud is not only feasible, but also a viable alternative to the current data center based Cloud Computing systems. However, the authors mentioned nothing about the elasticity, multitenancy, etc. characteristics of the system. Nebula, proposes to use a widely distributed edge resources over the Internet [14]. It is a location and context-aware distributed cloud infrastructure that provides data storage and task computation. Nebula consists of volunteer nodes that donate their computation and storage resources, along with a set of global and application-specific services that are hosted on dedicated, stable nodes [13]. The authors used Planetlab based simulation to test their system. One typical characteristic of cuCloud that differentiates it from all the above-mentioned works is that by extending the already existing IaaS cloud system, it full-fills all the basic characteristics of a cloud system such as elasticity, multitenancy, etc.

Cloud@home is a volunteer cloud system that provides similar or higher computing capabilities than commercial providers' data centers, by grouping small computing resources from many single contributors [18]. In order to tackle the node churn problem that occurs due to the random and unpredictable join and leave of contributors, an OpenStack[5] based Cloud@home architecture is proposed [15]. Every contributing node has a *virtualization layer* and a *Node Manager* that handle the allocation, migration, and destruction of a virtual resource on the node. The authors detailed the mapping of the reference architecture of Cloud@home to Openstack as well as a possible technical implementation. Compare to our work, the proposed prototype is only a blueprint of implementing Cloud@Home where the authors gave no such evaluation or validation of the performance of the proposed system.

5 Conclusion

The cuCloud, an opportunistic private/public cloud system that executes over scavenged resources of member PCs within an organization (or community), is a cheaper and greener alternative Cloud Computing solution for organizations/communities. In this paper, we discussed cuCloud together with its architecture and its implementation based on the open-source IaaS CloudStack. We

[5] https://www.openstack.org/.

also described the architecture and relevant components of CloudStack with regard to cuCloud's implementation. The paper presented the empirical performance evidence of cuCloud in comparison with Amazon EC2 using a big-data application based on Hadoop. From the results of the experimentation, we concluded that cuCloud can handle big-data applications that need heavy communications well. However, we plan to do an experimentation at a larger scale and with different workloads as well as applications in order to give general conclusions. Moreover, the cuCloud system is still under development. In general, the cuCloud is a system that can be called a genuine volunteer cloud computing system, which manifests the concept of "Volunteer Computing as a Service" (VCaaS) that finds particular significance in edge computing and related applications. In a highly dynamic and heterogeneous resource environment assumed by cuCloud, research and re-investigation on scheduling algorithms, VM migration, QoS, Security, unobtrusiveness, resource management, etc. immediately find new meanings and momentum.

References

1. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The cost of a cloud research problems in data center networks. SIGCOMM Comput. Commun. Rev. **39**(1), 68–73 (2009)
2. Brown, R.: Report to congress on server and data center energy efficiency: public law 109–431. Lawrence Berkeley National Laboratory (2008)
3. The Climate Group: SMART 2020: enabling the low carbon economy in the information age. The Climate Group on behalf of the Global eSustainability Initiative (2008)
4. Domingues, P., Marques, P., Silva, L.: Resource usage of windows computer laboratories. In: IEEE, pp. 469–476 (2005)
5. Che, D., Hou, W.-C.: A novel "Credit Union" model of cloud computing. In: Cherifi, H., Zain, J.M., El-Qawasmeh, E. (eds.) DICTAP 2011. CCIS, vol. 166, pp. 714–727. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21984-9_59
6. Varghese, B., Buyya, R.: Next generation cloud computing: new trends and research directions. Future Gener. Comput. Syst. **79**, 849–861 (2018). https://doi.org/10.1016/j.future.2017.09.020
7. Petcu, D., Fazio, M., Prodan, R., Zhao, Z., Rak, M.: On the next generations of infrastructure-as-a-services, pp. 320–326 (2016)
8. Mengistu, T., Alahmadi, A., Albuali, A., Alsenani, Y., Che, D.: A "No Data Center" solution to cloud computing. In: 10th IEEE International Conference on Cloud Computing, pp. 714–717 (2017)
9. RightScale: State of the Cloud Report (2016). https://www.rightscale.com/lp/2016-state-of-the-cloud-report
10. Sabharwal, N.: Apache Cloudstack Cloud Computing. Packt Publishing Ltd, Birmingham (2013)
11. Gao, W., Wang, L., Zhan, J., Luo, C., Zheng, D., Jia, Z., Xie, B., Zheng, C., Yang, Q., Wang, H.: A Dwarf-based Scalable Big Data Benchmarking Methodology. CoRR (2017)

12. McGilvary, G.A., Barker, A., Atkinson, M.: Ad hoc cloud computing. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), pp. 1063–1068 (2015)
13. Ryden, M., Chandra, A., Weissman, J.: Nebula: data intensive computing over widely distributed voluntary resources. Technical report (2013)
14. Chandra, A., Weissman, J.: Nebulas: using distributed voluntary resources to build clouds. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing series, HotCloud 2009 (2009)
15. Distefano, S., Merlino, G., Puliafito, A.: An openstack-based implementation of a volunteer cloud. In: Celesti, A., Leitner, P. (eds.) ESOCC Workshops 2015. CCIS, vol. 567, pp. 389–403. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33313-7_30
16. Mengistu, T.M., Che, D., Alahmadi, A., Lu, S.: Semi-markov process based reliability and availability prediction for volunteer cloud systems. In: 11th IEEE International Conference on Cloud Computing (IEEE CLOUD 2018) (2018)
17. Rosales, E., Castro, H., Villamizar, M.: Unacloud: opportunistic cloud computing infrastructure as a service. In: Cloud Computing 2011: The Second International Conference on Cloud Computing, GRIDs, and Virtualization. IARIA, pp. 187–194 (2011)
18. Cunsolo, V.D., Distefano, S., Puliafito, A., Scarpa, M.: Cloud@Home: bridging the gap between volunteer and cloud computing. In: Huang, D.-S., Jo, K.-H., Lee, H.-H., Kang, H.-J., Bevilacqua, V. (eds.) ICIC 2009. LNCS, vol. 5754, pp. 423–432. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04070-2_48

CloudsStorm: An Application-Driven Framework to Enhance the Programmability and Controllability of Cloud Virtual Infrastructures

Huan Zhou[1,2(✉)], Yang Hu[1], Jinshu Su[2], Cees de Laat[1], and Zhiming Zhao[1]

[1] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
{h.zhou,y.hu,delaat,z.zhao}@uva.nl
[2] School of Computer Science, National University of Defense Technology, Changsha, China
sjs@nudt.edu.cn

Abstract. Most current IaaS (Infrastructure-as-a-Service) clouds provide dedicated virtual infrastructure resources to cloud applications with only limited programmability and controllability, which enlarges the management gap between infrastructures and applications. Traditional DevOps (development and operations) approaches are not suitable in today's cloud environments, because of the slow, manual and error-prone collaboration between developers and operations personnel. It is essential to involve the operation into the cloud application development phase, which needs to make the infrastructure able to be controlled by the application directly. Moreover, each of these cloud providers offers their own set of APIs to access the resources. It causes the vendor lock-in problem for the application when managing its infrastructure across federated clouds or multiple data centers. To mitigate this gap, we have designed CloudsStorm, an application-driven DevOps framework that allows the application directly program and control its infrastructure. In particular, it provides multi-level programmability and controllability according to the applications' specifications. We evaluate it by comparing its functionality to other proposed solutions. Moreover, we implement an extensible TSV-Engine, which is the core component of CloudsStorm for managing infrastructures. It is the first to be able to provision a networked infrastructure among public clouds. At last, we conduct a set of experiments on actual clouds and compare with other related DevOps tools. The experimental results demonstrate our solution is efficient and outperforms others.

1 Introduction

The purpose of DevOps is to put application development and infrastructure runtime operations together to deliver good quality and reliable software. It encompasses continuous integration, test-driven development, build/deployment

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 265–280, 2018.
https://doi.org/10.1007/978-3-319-94295-7_18

automation, etc. However, its main focus often is the application itself. Especially with the trend of migrating applications onto IaaS clouds, operations are not limited to fixed private physical servers any more. The virtual infrastructure provided by clouds is dynamic and on-demand, but often lacks sufficient programmability and controllability to fully satisfy complex application requirements. From the application perspective, cloud applications have become complex and large-scale, no longer being just simple web services. Some applications need to consider the geographical distribution of their components to achieve better performance in edge computing. Moreover, some dynamic applications, like IoT (Internet of Things) applications, even need to control their underlying infrastructure during runtime, for instance to perform auto-scaling. Hence, it has become vital to fill the gap between the cloud application and its virtual infrastructure in order to better migrate more complex and demanding applications onto clouds.

Traditional DevOps approaches are slow, manual and error-prone, which are difficult to make the infrastructure suitable for the application. In order to settle this problem, some DevOps tools have been constructed to automate the provisioning of virtual infrastructure. For example AWS CloudFormation[1] provided by the Amazon Web Service, is a useful tool to create and manage AWS resources. They also provide AWS Lambda[2] to build serverless applications recently. Nevertheless, both of them mainly work for web applications. The other limitation is that it is a vendor lock-in solution, which can only be used on Amazon EC2 infrastructure. There are also tools to manage the infrastructures from different clouds, avoiding vendor lock-in, such as Libcloud[3], jclouds[4] and fog[5]. However, all the above tools are API-centric. They do not provide high-level programmability and controllability for the application to manage its infrastructure during the entire lifecycle.

To address this, there are some environment-centric [1] tools, including Puppet[6], Chef[7], Ansible[8], JuJu[9] and Nimbus [2]. They help developers to orchestrate the applications running on virtual infrastructures, but more concentrate on deployment and configuration, which is the operation phase. In academic research, there are also some DevOps systems, such as CodeCloud [3], CloudPick [4] and CometCloud [5]. Some of them leverage the concept of managing "Infrastructure as Code". The code here is preferred to be used for describing the infrastructure or configuration, such as Ansible using playbook to unify the configuration process on different machines. However these static codes can hardly describe how the infrastructure dynamically adapts to the application.

[1] https://aws.amazon.com/es/.
[2] https://aws.amazon.com/lambda/.
[3] http://libcloud.apache.org/.
[4] https://jclouds.apache.org/.
[5] http://fog.io/.
[6] https://puppet.com/.
[7] https://www.chef.io/.
[8] https://www.ansible.com/.
[9] https://jujucharms.com/.

Moreover, these tools cannot provision the networked infrastructure, especially not able to manage cloud virtual infrastructures among different data centers or clouds.

In this paper, we propose, CloudsStorm[10], an application-driven DevOps framework to enhance the programmability and controllability of the cloud virtual infrastructure. CloudsStorm framework consists of three components. The first one is the cloud resource metadata, which is the static resource description retrieved from each cloud provider, including resource types, prices, credentials, etc. It contains the essential information needed by each cloud application. Besides these existing metadata information, our main contributions focus on following two components.

DevOps Framework. We design the DevOps framework, CloudsStorm. It leverages YAML (YAML Ain't Markup Language) based language to describe all the applications' requirements on their infrastructures. Via this, the application can not only design its infrastructure network topology but also define the control policy for its infrastructure, including failure recovery, auto-scaling, etc. It brings the infrastructure into the application development phase and enables the applications to directly program and control their virtual infrastructures instead of just describing what they desire. It also provides multiple levels of programmability and controllability according to developers' knowledge of infrastructure. This is helpful to narrow the gap mentioned above. Furthermore, there is no centralized third party to control the infrastructure for all users in CloudsStorm, avoiding the existence of a single point of failure and a possible privacy risk.

TSV-Engine. We implement a TSV-Engine to manage our partition-based infrastructures, that are distributed across different data centers or clouds. It is the key engine of CloudsStorm. It is also the first one to be able to provision a networked infrastructure among public clouds. With these three types of engine at different levels, there are four advantages: (i) Fast. It can control several sub-infrastructures simultaneously to reduce the management overhead, such as provisioning, failure recovery and scaling, etc. (ii) Extensible. It is easy for the application to derive its own engine to control the infrastructure in its private cluster. (iii) Reliable. It partitions the entire infrastructure into small sub-infrastructures in multiple data centers. Even if some data center is down or not accessible, it may not influence the whole application.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents the DevOps framework and model of CloudsStorm. Then we describe the implementation of TSV-Engine in Sect. 4. Section 5 evaluates CloudsStorm from two aspects, functionality and performance. Finally, we conclude and discuss some application scenarios in Sect. 6.

[10] https://github.com/zh9314/CloudsStorm.

2 Related Work

In order to mitigate the difficulty of virtual infrastructure maintenance for cloud applications [6], there have been substantial academic research and industrial tools developed in recent years. However many DevOps tools only focus on some specific steps in the infrastructure management and provide limited controllability. For example, Libcloud, jclouds and fog just unify several clouds' provisioning APIs. They automate the provisioning, but still need manual configuration and management on the infrastructure. Conversely, tools such as Puppet, Chef and Ansible all try to manage infrastructures by turning them into code. However, they are infrastructure-centric, focusing on configuration and installation. They standardize the configuration commands among different systems to make the code reusable, such as the cookbook of Chef and playbook of Ansible. The Nimbus [2] team develops a Context Broker and cloudinit.d to create the "One-Click Virtual Cluster". Furthermore, they offer scaling capability allowing users to automatically scale across multiple distributed clouds [7]. Juju is application-centric which more focuses on application and services. CodeCloud [3] consists of a IM [8] (Infrastructure Manager). IM provides some specific REST APIs to control each individual VM (Virtual Machine). Based on this, CodeCloud leverages CJDL (Cloud Job Description Language) to describe the application and the elasticity of the infrastructure. CloudPick [4] is a system that considers the high-level constraints of the application on the infrastructure, including deadline and budget. However, these systems work as centralized services asking users to upload their cloud credentials, which requires trust in a third party. CometCloud [5] provides a heterogeneous cloud framework to deploy several programming models, such as master/worker, map/reduce and workflow. But it needs provisioned resources in advance to set up a cloud agent for each cloud. In general, most work focuses on either the infrastructure planning phase [9,10] or the application deployment [11] and execution [12] phase. The key phase of provisioning for applications to interact with the actual clouds is still missing. For example, some distributed applications such as real-time interactive simulation often require dedicated environment for system performance [13].

Most of above tools mentioned support multiple clouds, meaning no vender lock-in for configuration and deployment; however this does not mean that they are capable of provisioning a federated infrastructure for running applications. On the other hand, none of above tools consider about provisioning a networked infrastructure, which the VMs can be inter-connected with end-to-end private connections. This is difficult to realize, especially in federated cloud environments. ExoGENI [14] proposes Networked Infrastructure-as-a-Service (NIaaS) architecture based on SDN to permit the customization of network used by the infrastructure. SAVI [15] builds up a test bed for IoT. It leverages OpenFlow to consider the network topology of the virtual infrastructure. However, all these are established on private data centers, which means the data centers in the federation must be totally under the control of the proposed solutions. For instance, these solutions must have direct access to the switch in the data center to control the network. All of these solutions are not feasible to be applied on public clouds.

Recently, Amazon even proposes AWS Lambda, which is a serverless solution. They advocate that the infrastructure is not needed and developers should only focus on their function development. Then the function is thrown onto cloud and executed. It is useful to free the application from the cloud infrastructure for most web applications, which are event-driven. However, for many time-critical scientific applications or IoT applications in edge computing, the geographical distribution of the application components must be considered. It means that the application needs to spread across regions. Therefore, controlling on the cloud infrastructure is inevitable in these cases.

3 DevOps Framework and Model

In this section, we start by introducing the overview of the DevOps framework, CloudsStorm. The goal of this framework is to manage applications in the environment of federated clouds. Subsequently, we describe its core models in detail.

3.1 Framework Overview

Figure 1 illustrates the overview of the DevOps framework we propose. With this framework, cloud applications can achieve the goal of "Code as Infrastructures". It means that cloud application developers are not only able to develop their own applications but also able to program on the virtual infrastructures. The infrastructure management can be brought into the application development phase.

In this framework, there are three kinds of code, including infrastructure code, application code and runtime control policy. Cloud applications need only leverage these to control the whole lifecycle of the infrastructure their applications rely on, including phases of provisioning, deployment, auto-scaling and destruction. Among these code types, the infrastructure code is the core of the framework. It first allows application developers to describe the infrastructure they wish, which also includes the network topology. Then it indicates which execution code should be running on which node of the infrastructure. The execution code is the part that the application developer should initially focus on. It is the main logic for the cloud application. The difference here is that the controlling logic can be embedded in the execution code to allow the application to directly control the infrastructure on demand. This is explained in Sect. 3.2. At last, the infrastructure code defines the runtime control policy for dynamically managing the infrastructure at runtime.

The networked infrastructure designed by the application developer can be automatically provisioned through executing the infrastructure code. The runtime environment for the cloud application is configured immediately afterwards. Then, the specified execution code is uploaded to the corresponding instance to execute. There is a control agent in the whole infrastructure. It can be explicitly assigned by the developer or generated by CloudsStorm in an autonomous way. It manages the infrastructure descriptions and has the whole view of the

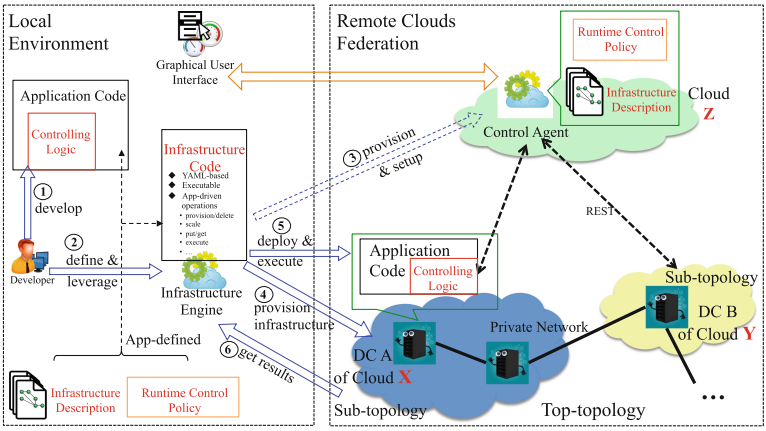


Fig. 1. Overview of CloudsStorm framework

infrastructure. On one aspect, it affords REST APIs to interact with the execution code to receive control requests and combines the runtime control policy defined by the application to control the infrastructure, including failure recovery and auto-scaling, etc. On the other hand, it provides a graphical user interface for developers to manually check the status of their infrastructure and control it in a visual way. All the outward connections of this control agent use public IP addresses. For federated clouds, we propose partition-based infrastructure management. The whole topology of the infrastructure is called the "top-topology". It is partitioned into small pieces of infrastructure, each referred to as a "sub-topology". Each sub-topology belongs to a data center domain of a cloud provider. It describes how the nodes are connected in one data center and the top-topology describes how the sub-topologies are connected. It is worth mentioning that all the nodes are connected with private network IP addresses. The implementation technique and advantage of this private networked infrastructure are demonstrated in following sections.

All these related descriptions and the infrastructure code are based on YAML format, which is human readable and easy to learn. The detailed syntax can be checked from the manual[11] of CloudsStorm.

3.2 Runtime Controlling Model

After executing the infrastructure code, all the infrastructures are provisioned and different components of the application run on the desired nodes. The infrastructure description is generated by the infrastructure code and uploaded to the control agent. Then control agent takes over the responsibility to manage the infrastructure. Here, the control agent is placed in a separate sub-topology, $subNI$ and its public IP is configured into all the other nodes for communication.

[11] https://cloudsstorm.github.io/.

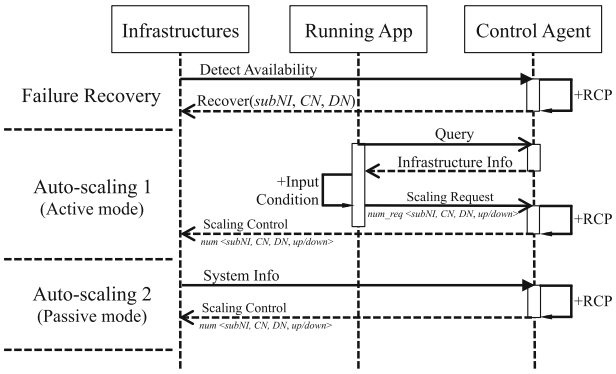


Fig. 2. Sequence diagram of runtime controlling model

Figure 2 illustrates the sequence diagram of runtime controlling model. It consists of three controlling scenarios. In the failure recovery scenario, we assume that there is an auto failure recovery mechanism provided by cloud provider for each individual node. Therefore, we more focus on the failure that the data center is not accessible. The control agent detects the availability of each sub-topology, $subNI$. If it is not available, the control agent sets up a new $subNI$ from another data center DN of cloud CN to replace the failed one according to the runtime control policy, RCP. Meanwhile, the private network topology among the infrastructures is preserved. In the auto-scaling scenario, there are two different modes for the application to control the infrastructure. One is an active mode, which means the application actively controls the infrastructure. This is responsible by the controlling logic embedded in the application code shown in Fig. 1. It queries infrastructure information from the control agent, for example how many nodes in this data center. Then according to input conditions such as the input data size, the application decides whether to scale up or down and sends the request to the control agent. After receiving the request, the control agent takes the application-defined RCP into account, for instance budget, and finally takes control of the infrastructures to scale up or down by the calculated number num. The other mode is passive mode. The infrastructures are passively controlled by the control agent based on RCP and the infrastructure's system information. In this model, the application invokes REST APIs to communicate with the control agent.

4 Implementation

To demonstrate this framework, we implement a prototype[12] of CloudsStorm. In this section, we introduce fundamental techniques to realize the core component

[12] https://youtu.be/Frc8VYjT51Q; Note: this is only to demonstrate some key features.

of CloudsStorm, TSV-Engine, which is a partition-based infrastructure control engine. Then we describe the infrastructure lifecycle management based on TSV-Engine. In addition, TSV-Engine is also responsible for connecting the VMs to provision a networked infrastructure. To implement this, we adopt the IP tunnel technique to connect VMs, which is proposed in our previous work [16].

4.1 TSV-Engine

TSV-Engine is the core engine of CloudsStorm. "TSV" is short for "Top-topology", "Sub-topology" and "VM" as shown in Fig. 3. A T-Engine is responsible for "Top-topology" management. In CloudsStorm, every application has one T-Engine. It helps the application to control the whole infrastructure and manage the connections among sub-topologies, $subNI$. During runtime, the application can generate different requests to dynamically change the infrastructure, including provisioning, recovering, scaling, deleting or stopping. The T-Engine takes these requests and queries the user database, UD, to set up a specified S-Engine for each cloud, for example, "S-Engine-EC2" for cloud "EC2". Meanwhile, the T-Engine queries the user credential, UC, to configure the S-Engine with a proper credential, which makes the S-Engine able to access that cloud. Then, the S-Engine manages each individual VM and its connections via the specified V-Engine. The V-Engine is responsible for the VM lifecycle from creation to stopping or deleting VMs. It also controls the connection between the VM and other VMs. These connections are based on the IP tunnel mechanism proposed by us [16] to connect the VMs in different federated clouds with application-defined private network. After provisioning, the V-Engine can run the application-defined script to configure the runtime environment and deploy the application. The V-Engine is a basic engine. Different customized V-Engines can be derived from it depending on the VM's features, such as "V-Engine-ubuntu" for ubuntu VM, etc. If the application has specific operations on some VM, it can also customize its own V-Engine. Meanwhile, this design

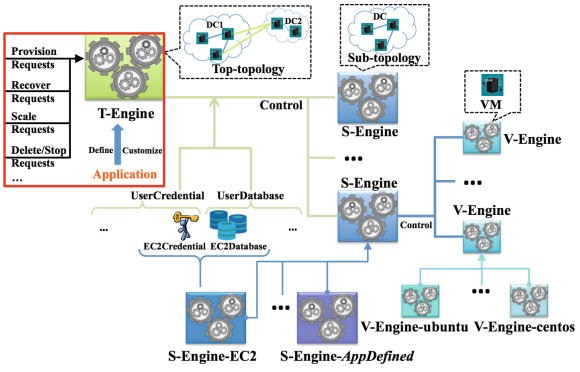


Fig. 3. Architecture of TSV-Engine

makes our framework very extensible. For example, in order to take Cloud Azure into account, we therefore first derive "S-Engine-Azure" to manage Azure cloud. Afterwards, we add the credential and database information in "UserCredential" and "UserDatabase". Then, the T-Engine can take the new cloud into account when doing partitioning and manage the whole lifecycle of the resource from that cloud.

In addition, all the S-Engines and V-Engines are running in multi-thread. It means that the T-Engine can start several S-Engines at the same time. If the sub-topologies managed by these S-Engines belong to different data centers, there will be no contention among them. It is the same for V-Engine. In other words, TSV-Engine accelerates the controlling process. This engine has also been integrated in the software release of EU project, SWITCH [17], to be the fundamental provisioning engine.

4.2 Implementation of Control Mechanism

In order to control infrastructures, we define 5 statuses for the sub-topology in CloudsStorm. These statuses are recorded in the infrastructure descriptions shown in Fig. 1. Figure 4 is the status transition graph. It begins with the "Fresh" status, which means the infrastructure is in the design phase and the public IPs are not assigned. When the T-Engine controls the S-Engine to do provisioning, the status of the sub-topology infrastructure can transit into two statuses, "Running" or "Failed", depending on whether there are errors during provisioning. If becoming "Running", the public IPs of VMs in the running sub-topology must have been assigned. Afterwards, if the control agent detects some running sub-topology is not accessible or failed, the T-Engine marks its status as "Failed" and controls the corresponding S-Engine to identify the failed sub-topology. Meanwhile, all the sub-topologies which originally are connected with this failed one should be detached. The failed sub-topology then can be recovered within another data center according to the recovery requests. "Stopped" status is circled with dashed line, because some clouds do not provide the stopping function. If the cloud doesn't support this function, there is no "Stopped" status in its lifecycle. The reason for designing the status "Stopped" in the lifecycle is that the stopped VM is faster to bring up again than provisioning new

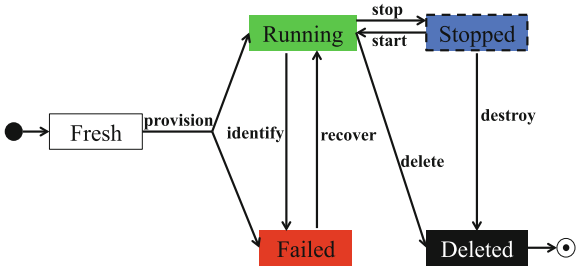


Fig. 4. Status transition of infrastructure lifecycle in CloudsStorm

one from "Fresh" when scaling up. Finally, "Deleted" is the terminated status of the lifecycle. All the "Stopped" and "Running" infrastructures can be deleted to release resources.

5 Evaluation

In this section, we compare CloudsStorm with other related DevOps tools and evaluate it from two aspects, functionality and performance.

5.1 Functionality Evaluation

Table 1 shows the functionality comparison among different related tools and frameworks. We classify these DevOps tools into three categories. They are API-centric, infrastructure-centric and application-centric. The latter two both belong to the category of environment-centric. In this table, × and √ means supported and not supported respectively. − indicates partial support. For instance, some tools partially support auto-scaling and failure recovery, because some manual work is still needed to actually trigger the process. Another case is where the auto-scaling refers to configuration scaling on pre-existing resources. In the programmability comparison, "Abstract description" indicates whether the tool allows the application to describe high level QoS constraints on the infrastructure. For controllability comparison, "Multi-Mode" is explained in Sect. 4.2. "Decentralized" demonstrates the infrastructure management is in a decentralized way. If it is not, it means all the application developers' infrastructures are managed by one administration, which requires everyone's cloud credentials. If it is partial, it means an application developer can set up his own server for managing all his infrastructures instead for one application. Table 1 demonstrates CloudsStorm is application-driven and enhances the application's programmability and controllability on its infrastructure.

5.2 Performance Evaluation

We conduct some experiments on actual clouds to evaluate CloudsStorm's performance on auto-scaling and failure recovery. We also compare its performance with some other related tools.

Auto-Scaling and Failure Recovery. These two are the key controllability of CloudsStorm. We design the experiment on ExoGENI to test the auto-scaling performance. In this experiment, there are two sub-topologies in the beginning, $subNI_1$ containing 1 VM and $subNI_2$ containing 8 "XOMedium" VMs. Each VM in $subNI_2$ is connected with the VM in $subNI_1$ via a private network link. This is a typical "Master/Slave" distributed framework. $subNI_2$ is defined as a scaling group. According to the scaling request, the infrastructure can scale up to other data centers based on one or multiple copies of $subNI_2$. At the

Table 1. Functionality comparison among different DevOps tools and frameworks

		Programmability					Controllability					
		Language	Abstract Description	Federated Clouds	Networked Infrastructure	Public Cloud	Automated Provision	Automated Configuration	Auto-Scaling	Failure Recovery	Multi Mode	Decen-tralized
API-centric	jclouds	Java	×	-	×	√	√	√	×	×	×	×
	Libcloud	Python	×	-	×	√	√	√	×	×	×	×
Infrastructure-centric	Puppet/Chef	Ruby	-	√	×	√	√	√	-	-	×	-
	Ansible	YAML	×	√	×	√	×	√	-	-	×	-
	SAVI	Not Known	×	√	√	×	√	×	-	-	×	×
App-centric	Juju	YAML	-	√	×	√	√	√	-	-	×	×
	cloudinit.d	Python	×	√	×	√	√	√	-	-	×	×
	CodeCloud(IM)	CJDL(XML)	-	√	×	√	√	√	√	×	×	×
	CloudPick	GUI	√	√	×	√	√	√	-	√	×	×
	CometCloud	Java	√	√	×	√	×	×	×	×	×	-
App-driven	CloudsStorm	YAML	√	√	√	√	√	√	√	√	√	√

same time, all the network links between the scaled copies and $subNI_1$ are connected. These connections leverage private addresses, which can be defined before actual provisioning. Hence, the "Master" in $subNI_1$ can always know where are the scaled resources are. Figure 5 illustrates that we scale up the 8 VMs of $subNI_2$ accordingly at 1, 2, 3, 4, 8 and 16 times. Each scaled $subNI_2$ is provisioned in independent data centers simultaneously. Therefore, the flat dashed line is the ideal scaling performance in theory. However, the provisioning performances of different data centers are not the same. This is demonstrated by the varied dashed line, which is the average value of the maximum provisioning overhead among the scaled $subNI_2$. Moreover, the end-to-end connections need to be set up. The more copies of $subNI_2$ requested, the more connections need to be configured. The solid line in the figure shows total cost. For each scale, we conduct 10 repeated experiments. The error bar denotes the standard deviation. It demonstrates that the scaling overhead does not grow at the same proportion as the number of VMs being created. Therefore, it is able to complete large-scale auto-scaling in a short time. In addition, most clouds have limitations on the resource allocation. For instance, ExoGENI only allow one user to apply a maximum of 10 VMs from one data center. The limitation for EC2 is 20. Nevertheless, with CloudsStorm, we can break through these limits to realize large-scale scaling by combining resources from different data centers and even clouds.

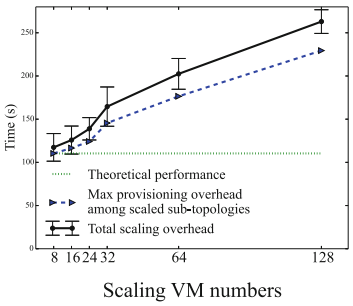


Fig. 5. Auto-scaling performance

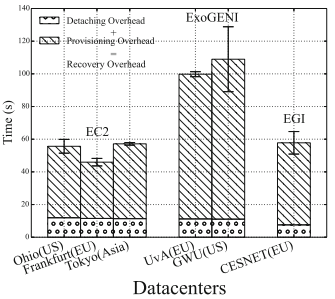


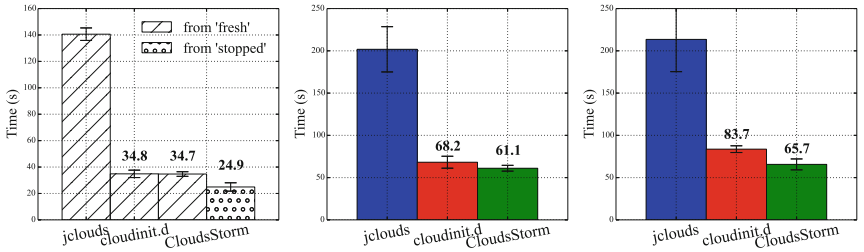
Fig. 6. Failure recovery performance

Figure 6 shows the experimental result on failure recovery. In this experiment, there are still two sub-topologies in the beginning, $subNI_1$ and $subNI_2$. Each of them contains only one VM, n_1 and n_2. These two nodes are connected with the private network. Then we simulate the case where the data center of $subNI_2$ is not available. CloudsStorm recovers the same sub-topology from another data center or cloud. Finally, the private network is resumed. Hence, the application is not aware of this infrastructure modification. We get the detaching overhead from CloudsStorm, which is the time for $subNI_1$ to disconnect the original link. It is illustrated by the bar covered with dots. On the other aspect, we continually

test the private link from n_1 of $subNI_1$ to n_2 of $subNI_2$ and record the time from lost connection to the time that the link is resumed. This is the total recovery overhead. We conduct this experiment on 3 clouds currently supported and pick 6 data centers from them. In order to compare, n_2 always has 2 cores and around 8G memory with "Ubuntu 14.04". Correspondingly, they are "t2.large" of EC2, "XOLarge" of ExoGENI and "mem_medium" of EGI. The results show ExoGENI has a relative higher recovery overhead and some of its data centers are not stable. The performance of EC2 and EGI are close, however, most data centers of EC2 are more stable. These information are important to decide where to recover to satisfy the application QoS, considering about the recovery overhead and data center geographic information.

Comparison. Finally, we conduct a set of experiments to compare CloudsStorm with other DevOps tools. We pick jclouds from API-centric tools. It is adopted by a lot of environment-centric tools to be the basic provisioning engine, such as CloudPick [4]. From environment-centric tools, we pick Nimbus team's cloudinit.d. Other tools like Juju and IM provide graphical interfaces, which makes it difficult to measure performance. Both of jclouds and cloudinti.d do not support networked infrastructure. The ones which support networked infrastructure can only be applied in private data centers, which CloudsStorm cannot have the access permission, like SAVI. We pick EC2 to do these experiments, because this is the most popular cloud provider and commonly supported by these tools. First, we compare the scaling performance. The scaling request is to add 5 more "t2.micro" VMs in EC2 California data center. However, jclouds and cloudinit.d cannot directly support auto-scaling behavior, we use them to provision 5 new VMs in California data center to simulate this scenario. For each operation, we repeated 10 times. Figure 7(a) illustrates the results. For jclouds, the provisioning process proceeds in sequence, hence, its scaling overhead is much more bigger than the other two. If only considering the scaling performance from "Fresh" (defined in Sect. 4.2) state, cloudinit.d and CloudsStorm have similar performance, demonstrated by the bars covered with slashes. CloudsStorm is a little bit stable than cloudinit.d. Moreover, EC2 supports stopping a instance. CloudsStorm can perform auto-scaling from "Stopped" status. It reduces the overhead, shown by the bars covered with dots. It is worth to mention that we do not consider deployment overhead in this experiment. Scaling from "Stopped" status can even omit the deployment. Through this way, CloudsStorm outperforms cloudinit.d much better, reducing the scaling overhead by more than half referring to Fig. 7(b) and (c).

The second experiment is to compare the provisioning performance including deployments. All of these three allow users to define a script to deploy applications immediately after provisioning. In this experiment, we choose California data center to provision 5 "t2.micro" VMs and install tomcat on each of them. Each test is repeated 10 times. Figure 7(b) shows the results. With jclouds, the application are installed one by one, which costs plenty of time. For CloudsStorm, there is a V-Engine responsible for each individual VM to provision and deploy.



(a) Scaling (b) Including Deployment (c) Deployment with Dependency

Fig. 7. Performance comparison

Therefore, it achieves the best performance according to the overhead and stability. The last experiment is based on the second experiment considering about the deployment dependency. In this experiment, 4 out of 5 VMs install tomcat and the remaining one installs a mysql database. In this case, there is a dependency when using jclouds and cloudinit.d, because they do not provision networked infrastructure and use public addresses. Tomcat can only be deployed after provisioning mysql VM to know the server address. Hence, jclouds needs to provision mysql VM first in its sequence. Cloudinit.d defines different levels to realize the dependency. In this scenario, the first level is the mysql VM and the second level contains four tomcat VMs. The difference for CloudsStorm is that it can provision networked infrastructure. The nodes are connected with application-defined private network links. The mysql server address is pre-defined before actual provisioning. Therefore, all the deployments can proceed simultaneously even with the dependency. Figure 7(c) demonstrates that the deployment dependency makes smaller influence on CloudsStorm's performance comparing to that on jclouds and cloudinit.d. We can reason out that if there are more dependencies, CloudsStorm has a greater advantage over others.

6 Discussion

There are plenty of application scenarios for CloudsStorm. To benefit from the networked infrastructure, a lot of programming models are easy to deploy, such as Master/Slave (including docker cluster), Map/Reduce and workflow. These models are usually leveraged by big data processing. To benefit from the sufficient and efficient controllability, it is useful for IoT applications and sensor-cloud. To benefit from the application-driven design, we can bring the infrastructure into the application incremental development phase. For example, testing the application with several VMs in the beginning and final release running on a large-scale infrastructure. Therefore, CloudsStorm narrows the DevOps gap and we are going to try to migrate more applications onto clouds through CloudsStorm in the future work.

Acknowledgment. This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIPLUS project) and 676247 (VRE4EIC project). The research is also partially funded by the COMMIT project. The author, Huan Zhou, is also sponsored by China Scholarship Council.

References

1. Wettinger, J., et al.: Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. FGCS **56**, 317–332 (2016)
2. Keahey, K., Freeman, T.: Contextualization: providing one-click virtual clusters. In: IEEE Fourth International Conference on eScience 2008, pp. 301–308 (2008)
3. Caballer, M., de Alfonso, C., Moltó, G., Romero, E., Blanquer, I., García, A.: Codecloud: a platform to enable execution of programming models on the clouds. J. Syst. Softw. **93**, 187–198 (2014)
4. Dastjerdi, A.V., Garg, S.K., Rana, O.F., Buyya, R.: CloudPick: a framework for QoS-aware and ontology-based service deployment across clouds. Softw.: Pract. Exp. **45**, 197–231 (2015)
5. Diaz-Montes, J., AbdelBaky, M., Zou, M., Parashar, M.: CometCloud: enabling software-defined federations for end-to-end application workflows. IEEE Internet Comput. **19**(1), 69–73 (2015)
6. Jeferry, K., Kousiouris, G., Kyriazis, D., Altmann, J., Ciuffoletti, A., Maglogiannis, I., Nesi, P., Suzic, B., Zhao, Z.: Challenges emerging from future cloud application scenarios. Procedia Comput. Sci. **68**, 227–237 (2015)
7. Marshall, P., Tufo, H.M., Keahey, K., La Bissoniere, D., Woitaszek, M.: Architecting a large-scale elastic environment-recontextualization and adaptive cloud services for scientific computing. In: ICSOFT (2012)
8. Caballer, M., Blanquer, I., Moltó, G., de Alfonso, C.: Dynamic management of virtual infrastructures. J. Grid Comput. **13**(1), 53–70 (2015)
9. Zhao, Z., Grosso, P., Van der Ham, J., Koning, R., De Laat, C.: An agent based network resource planner for workflow applications. Multiagent Grid Syst. **7**(6), 187–202 (2011)
10. Wang, J., Taal, A., Martin, P., Hu, Y., Zhou, H., Pang, J., de Laat, C., Zhao, Z.: Planning virtual infrastructures for time critical applications with multiple deadline constraints. Future Gener. Comput. Syst. **75**, 365–375 (2017)
11. Hu, Y., Wang, J., Zhou, H., Martin, P., Taal, A., de Laat, C., Zhao, Z.: Deadline-aware deployment for time critical applications in clouds. In: Rivera, F.F., Pena, T.F., Cabaleiro, J.C. (eds.) Euro-Par 2017. LNCS, vol. 10417, pp. 345–357. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64203-1_25
12. Ouyang, X., Garraghan, P., Primas, B., McKee, D., Townend, P., Xu, J.: Adaptive speculation for efficient internetware application execution in clouds. ACM Trans. Internet Technol. (TOIT) **18**(2), 15 (2018)
13. Zhao, Z., Van Albada, D., Sloot, P.: Agent-based flow control for HLA components. Simulation **81**(7), 487–501 (2005)
14. Baldin, I., et al.: ExoGENI: a multi-domain infrastructure-as-a-service testbed. In: McGeer, R., Berman, M., Elliott, C., Ricci, R. (eds.) The GENI Book, pp. 279–315. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33769-2_13

15. Kang, J.-M., Lin, T., Bannazadeh, H., Leon-Garcia, A.: Software-defined infrastructure and the SAVI testbed. In: Leung, V.C.M., Chen, M., Wan, J., Zhang, Y. (eds.) TridentCom 2014. LNICST, vol. 137, pp. 3–13. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13326-3_1
16. Zhou, H., Wang, J., Hu, Y., Su, J., Martin, P., de Laat, C., Zhao, Z.: Fast resource co-provisioning for time critical applications based on networked infrastructures. In: IEEE International Conference on Cloud Computing, pp. 802–805 (2016)
17. Zhao, Z., Taal, A., Jones, A., Taylor, I., Stankovski, V., Vega, I.G., Hidalgo, F.J., Suciu, G., Ulisses, A., Ferreira, P.: A software workbench for interactive, time critical and highly self-adaptive cloud applications (SWITCH). In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 1181–1184. IEEE (2015)

A RESTful E-Governance Application Framework for People Identity Verification in Cloud

Ahmedur Rahman Shovon, Shanto Roy, Tanusree Sharma, and Md Whaiduzzaman(✉)

Institute of Information Technology, Jahangirnagar University, Dhaka, Bangladesh
shovon.sylhet@gmail.com, sroy.iitju@gmail.com, tanusreesharma207@gmail.com, wzaman@juniv.edu

Abstract. An effective application framework design for e-governance is definitely a challenging task. The majority of the prior research has focused on designing e-governance architecture where people identity verification takes long time using manual verification system. We develop an efficient application framework that verifies peoples identity. It provides cloud based REST API using deep learning based recognition approach and stores face meta data in neural networks for rapid facial recognition. After each successful identity verification, we store the facial data in the neural network if there is a match between 80–95%. This decreases the error rate in each iteration and enhance the network. Finally, our system is compared with the existing system on the basis of CPU utilization, error rate and cost metrics to show the novelty of this framework. We implement and evaluate our proposed framework which allows any organization and institute to verify people identity in a reliable and secure manner.

Keywords: E-Governance · Cloud computing · RESTful API
ID verification

1 Introduction

E-governance applications are based on Government to Government (G2G), Government to Business (G2B), Government to Citizens (G2C) and Government to Enterprise (G2E) [1]. It is nothing but a service that act as a medium of exchange of information, transactions and services. Cloud Computing is the delivery of computing services servers, storage, databases, networking, software, analytics and more over the Internet.

A number of issues have risen in e-governance application. The lack of appropriate framework, content development, citizen access are the main problems. Every government needs to record the data of its citizens including national ID (NID) information and photo which are not updated regularly. In the mean time

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 281–294, 2018.
https://doi.org/10.1007/978-3-319-94295-7_19

a person's look changes with his age. Thus verification can be problematic in that case. Here, updating photo is necessary which might be costly and requires more space that remains idle mostly. Again large amount of data can make chaos and redundancy. For that an effective E-Governance system should be cost effective, reliable and easy to maintain. Unfortunately,current technologies are not enough to meet the overall requirements of E-Governance [1].

To overcome the scenario, Cloud technology or mobile computing [2] is the solution to the problems. E-governance is facing failure in meeting requirements of public services, high IT budgets compared to low utilization of existing resources, poor security mechanisms, costly operations and maintenance (O & M) etc. [3]. Hence cloud based E-Governance application framework provides many benefits in particular to government. It provides a comparatively better platform for efficient deployment of E-governance System [1] than the traditional manual processes.

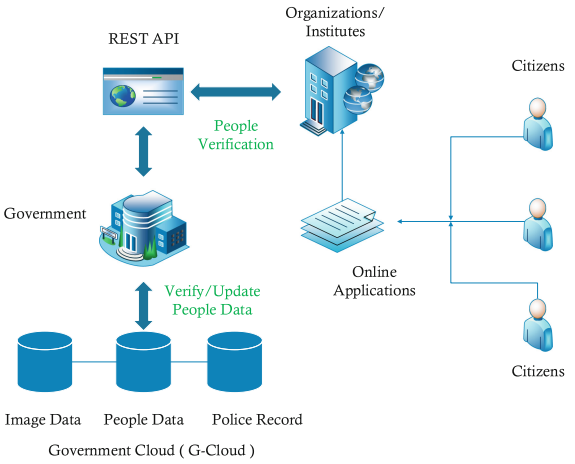


Fig. 1. Block diagram of the proposed system

In this paper, we implemented a smart approach that the government can keep only the vector co-ordination information in cloud for further verification. By adding images of a unique face and updating it with AWS AI approaches decrease the error rate of people identity verification. The feature of deep learning based image recognition technique make the system efficient for authorized access. The increase of facial data of the citizens will open the door of future study. Our system provides better CPU utilization, scope for migration to new technologies.Here, we used police verification data which eliminate manual verification process and reduce the delay. Moreover, REST approach confirms the instant response from verification framework that can be integrated to any authorized application. Overall, the primary contributions of this work are a people

verification framework, utilization of cloud for E-Governance application, RESTful approach for better and secure service provision. For better understanding, the block diagram of the proposed system is showed on Fig. 1.

The remainder of the paper is organized as follows. Section 2 describes different E-governance system proposed by several researchers. Section 3 describes our proposed methodology. Section 4 presents the architecture of the framework, Sect. 5 shows the results. Finally, Sect. 6 presents the conclusion and future work of this study.

2 Related Work and Motivation

In this section, we discuss several existing E-governance system and also describe our motivation for this study.

2.1 Related Work

Several researchers have developed and surveyed some system for E-governance. For example Witarsyah, et al. in [4] studied on adding some important variables: trust, information quality, system quality of e-government adoption in improving service to the citizen. In [5] Rao and Krishna proposed a system comparing with existing system on the basis of browsing performance, response time, and security to show that novelty nature of the software architecture to support e-governance of India on highly distributed evaluation environment. Hence, information quality and data scaling are ignored here. In [6] Zissis and Lekkas proposed a high level electronic governance and electronic voting solution, supported by cloud computing architecture and cryptographic technologies. It is quite clear from this paper that there is a large scope for further research.

Zhang and Chen in [7] introduced a new concept of C-government that utilizes cloud computing in e-governance. Authors designed the architecture and discussed about the implementation and deployment of the c-government system. The paper also explained how citizen engagement is maximized with enhanced collaboration towards government productivity and service delivery. Hashemi et al. In [8], authors discussed about the benefits of using cloud computation in e-governance including improvement and optimization providing public services, governments enhanced ability to interact and collaborate, achieving transparency and accountability as well as overall improvement of the relationship between the government and its citizens etc. How developed countries like Singapore, USA and UK are adapting to the cloud technology is also mentioned here. In [9] Cellary and Strykowski researched on two levels of aggregation: organizational units and functional modules. Some mentionable advantages of cloud are: dynamic load of resources, proper maintenance and administration, higher performance etc. Authors also claims that REST based web technology is gaining a lot of attractions.

Face verification across ages has importances in some potential applications such as passport photo verification, image retrieval and post processing, surveillance etc. [10]. As human faces change in course of time and still the face image

may need some verification. Photo identification means the matching accuracy in percentage using 3D modeling [11], discriminative methods [10], feature extraction and modeling through deep learning etc [12–14].

In [15] Christensen et al. implies to the integration of REST-based web service and cloud computing in order to create futuristic web and mobile applications along with optimization and security. As REST architecture ensures the data security as well as maintains a secure communication [16]; that is the reason behind choosing this technology in our work.

2.2 Motivation

From our observation, as discussed in Subsection A of Sect. 2 we can come to a conclusion that we needed to develop an framework that includes digitization, proper citizen access, increase facial data of each citizen, lowest cost, lower error rate of face recognition. To get the best performance of E-governance system we need to focus on this trade-off. Therefore, in our proposed policy we take a more holistic approach by considering a RESTful API can be accessed from any device that is connected with internet.

3 Methodology

We designed the architecture to verify people identity and personal information in G-cloud using the secure RESTful API. The process starts with searching information in the database using NID; matches the facial vector data with the stored data mapped with the NID. The system flowchart and algorithm are shown in Fig. 2 and Algorithm 1 accordingly.

4 Architecture of the Framework

Our proposed system developed an E-Governance application framework which includes hardware and software requirements and it uses cloud based artificial neural network.

4.1 Structural Description

The proposed framework uses the Amazon Web Service (AWS) for all operations. The REST API is hosted in EC2 instance and can be accessible via API calls from any other systems. The REST API is built using Python Flask Framework and it abides by the prescribed standard of HTTP status codes. JSON Web Token (JWT) which is a JSON based open standard for creating access tokens is used to authenticate the API calls. For storing the newly compared faces, the framework uses Rekognition and Elastic Load Balancer (ELB) from Amazon Web Services. For storing the public people data it utilizes the functionalities of SQLAlchemy, Python SQL Toolkit and Object Relational Mapper (ORM) that

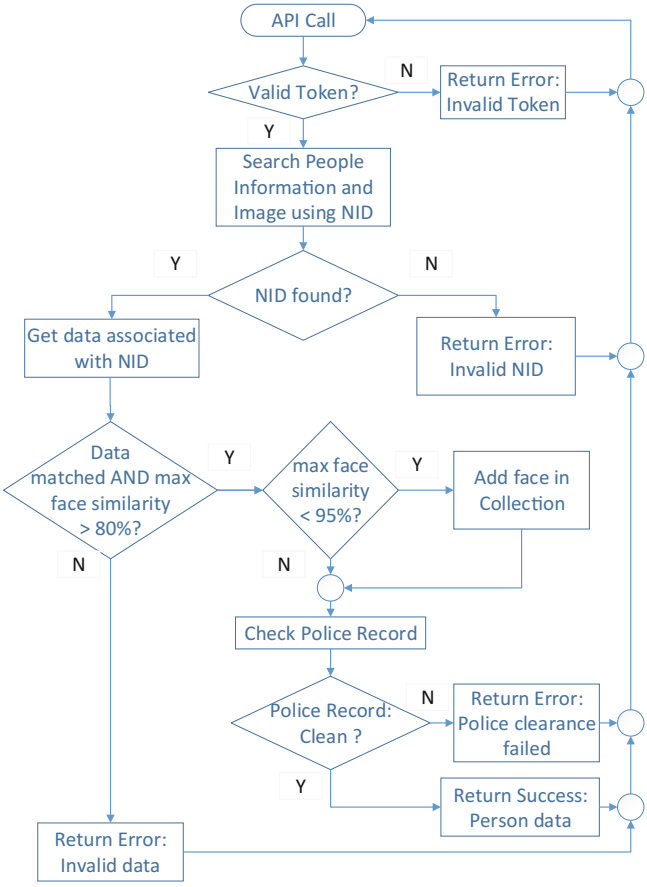


Fig. 2. System flowchart

provides full power and flexibility of SQL. The ORM is connected with MySQL database for handling the queries.

In this scenario, there are 4 subnets in the (Virtual Private Cloud) VPC. Three of them are private and one public. In each subnets the proposed framework used Application server, Database server, API Server which are in the private subnets. In the public subnet there is a bastion host which is used by the developer to SSH on each server in the VPC.

4.2 Functionalities

There are two Classic Load Balancers are used in this application. One is public facing load balancer in which users can connect to its port 80 and internally it connects to the web application. Other is internal load balancer by which the application communicates with the API. Each load balancer there is an

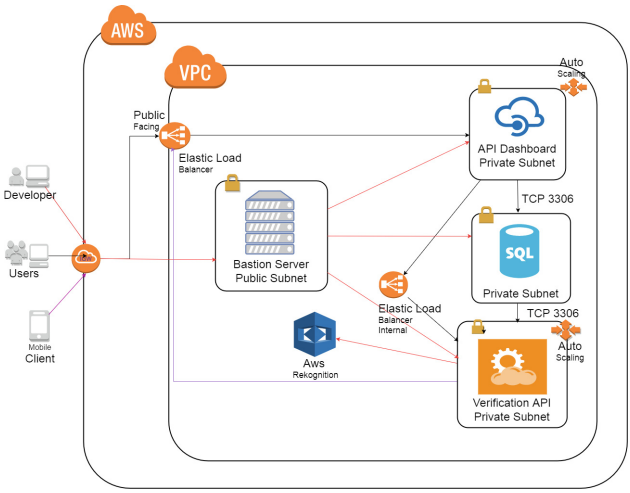


Fig. 3. System architecture

auto scaling group which scales the application and API servers. Mobile users can directly hit the API server through the public facing elastic load balancer (Fig. 3).

4.3 Requirement Analysis

The necessary software and hardware tools needed to complete this work are described in Tables 1 and 2 accordingly.

4.4 Security

The instances in the public subnet can send outbound traffic directly to the Internet, whereas the instances in the private subnet can't. Instead, the instances in the private subnet can access the Internet by using a network address translation (NAT) gateway that resides in the public subnet. The database servers can connect to the Internet for software updates using the NAT gateway, but the Internet cannot establish connections to the database servers. Web application, API and database cannot accessible from the Internet. Each subnet has its own security groups. In the security groups inbound and outbound rules are configured for the security and application needs.

5 Implementation and Results

We used Machine Learning approaches in E- governance system and developed a framework with RESTful API for citizen access.

5.1 Implementation Environment

The deployment procedure of our proposed framework in started with creating new VPC which includes three public and one private subnet. Later we create an Internet gateway and a NAT gateway from which NAT gateway is placed in public subnet and associate an EIP to it Then finally create route by adding IGW to the public subnet and NAT Gateway to the private subnet.After that we initialize security group for each subnet for security purpose and create dashboard-sec for dashboard subnet, db-sec for db-subnet, api-sec for api segment and jump sec for jump subnet. Then we create an EC2 instance for dashboard application. We deployed the code base and create an Internet facing classic load balancer.After creating a launch configuration with dashboard application, we also create auto scaling group and add load balancer that previously created. We add the scaling policy on CPU utilization which scales the servers and send alerts through emails.

We designed four tables for our database named authenticated_organization, people, aws_face, and police_record. The fields needed for the tables are listed in Table 3. People table has relation with aws_face and police_record using the national_id field. To create JSON Web Token for API call authenticity, a combination of organization's id, email address and password will be used.

5.2 Performance Evaluation

Primarily the neural network contains one picture per person mapped to the persons NID. For this reason, falsification of face recognition is occurred in some cases. We used the face recognition technology (FERET) database [17] which is a dataset used for facial recognition system evaluation to measure the error rate. We mapped one individual photo with one national id. Then we tested the system using same individual's other photos available in the FERET database.

We calculate the error rate of the framework by measuring number of false results returned by the API. At first, the neural network has only one image per person. We mark this stage as single image stage. In each verification request if there is 80–95 % similarity then the framework adds the new image to neural network. By using this machine learning approach provided by AWS, it becomes easy and efficient to verify people by the increasing number of iterations. As the number of iteration is increases, the error rate is decreases. Even for this deep learning approach, this image data can be used further for video verification.

Error rate in this single image per person scenario is 1.60%. After this iteration, each searched persons have two facial data (on average) available. The error rate in this scenario is 1.13%. In third iteration, there are three facial data (on average) is available for each person. The error rate in this step is 0.27%. The improvement of each iteration is shown in Fig. 4.

5.3 Scalability Evaluation

To evaluate the scalability of our framework we chose CPU utilization as primary metric. The auto load testing script which is developed is used to generate

Iteration	Error Rate	Improvement
1	1.60%	N/A
2	1.13%	29.38%
3	0.27%	76.11%

Fig. 4. Performance chart in each verification iteration

artificial hit on the API endpoints. The CPU utilization of EC2 instances of People Identity Verification API is shown in Fig. 5. The Y axis reflects the CPU utilization and is measured in percentage which go up by 5 percent at each level. The X axis represents time frame. The instances are configured to add multiple instances instantly in case of high CPU utilization.

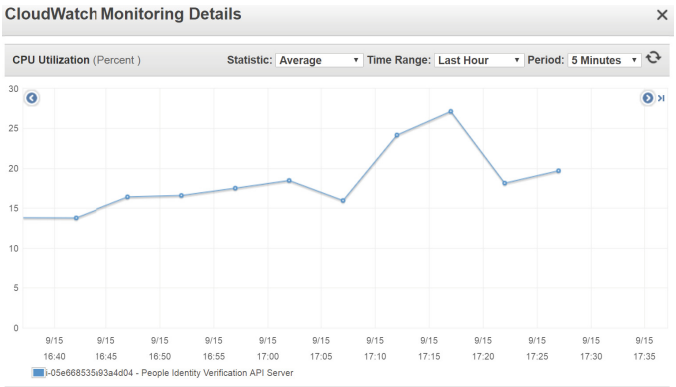


Fig. 5. CPU utilization of People Identity Verification API

Furthermore, we generate excessive hit on the API endpoints using test script and exceeds the maximum limit of CPU power. Our system is configured to generate alert for low and excessive CPU utilization within a time range and add additional computing resources as needed. The additional resources are detached from the system when the CPU utilization becomes normal. Figure 6 shows a scenario of live monitoring the instances.

5.4 Cost Evaluation

For any application framework, cost assumption is important to evaluate the efficiency and usability. We deployed our application in Amazon Web Services

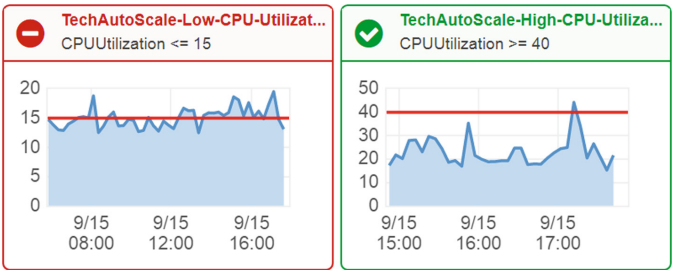


Fig. 6. Auto scaling alert monitoring

(AWS) which offers a pay-as-you-go approach. With AWS the application needs to pay only for the individual services it needs, for as long as it uses them. It requires to pay for the services the application framework consumes, and once it stop using them, there are no additional costs or termination fees. If we used non cloud hosting for our application framework neither it would not scalable nor efficient. The charges of various services utilization are shown in Fig. 7. This figure proves that costing is reliable and efficient for this application framework.

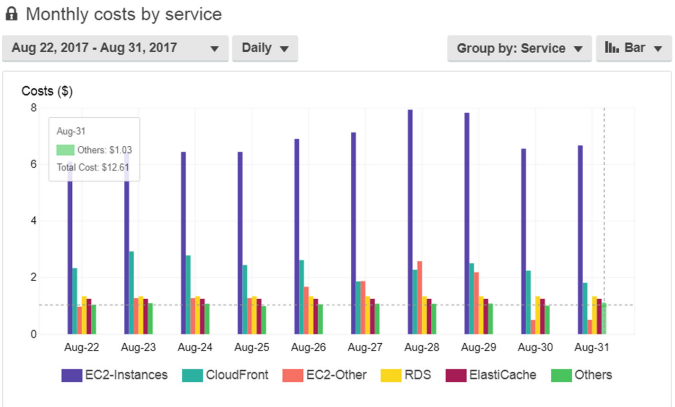


Fig. 7. People Identity Verification API costing graph

6 Discussion

In this section, we discuss the advantages, application areas, scopes and limitations of our framework.

Algorithm 1: Algorithm for people verification

```
API call;
if TOKEN in Request = True then
    if Validity_TOKEN = True then
        if NID, personal information & image in Request then
            if NID_found = True then
                if personal information & image matches with database then
                    Get Face_List from Database;
                    Set Max_Score = 0 ;
                    foreach Face in Face_List do
                        Calculate similarity score;
                        Set Max_Score = max(score, max_score);
                    end
                    if Max_Score > 80 then
                        if Max_Score < 95 then
                            Store image in Collection;
                        end
                        Get Records_Police ;
                        if Clearence = True then
                            Return: Verification Success with all data;
                        end
                        else
                            Return Error with case record details;
                        end
                    end
                    else
                        Return Error: Image match failed;
                    end
                end
                else
                    Return Verified = True;
                end
            end
            else
                Return Error: NID not found;
            end
        end
        else
            Return Error: Missing mandatory fields;
        end
    end
    else
        Return: TOKEN not valid;
    end
end
else
    Return: TOKEN not found;
end
```

6.1 Advantages

The work is useful for updating and verifying public data as well as for utilizing the e-government processes along with better government to citizen (G2C) relationship. The REST-based web API approach secures the access to the government cloud. It also enhance the artificial neural network of citizens facial data in each use of the verification API. So, our work ensures the shortening of time updating and verifying public citizen data by digitalizing the processes in a secure way.

6.2 Potential Application Areas

Government and different organizations or institutions verifies applicant's data while recruiting them. They can easily verify people by accessing the g-cloud where all the citizen's data are stored. As the access mechanism is completely secure due to the RESTful API where all of them can access to the database using JWT (JSON Web Token); government need not to worry about data security. In the mean time, the appearance and other variable information can be changed; now, face image or personal profile can be updated alongside verification. Also, government can construct a secure, scalable and highly available hybrid g-cloud utilizing this framework.

6.3 Scope and Limitations

While updating the data still there is a possibility of fraud. Government must ensure that the request comes from authenticated organization as well as prioritize the previous data primarily stored in the g-cloud. Although, our work eradicate the possibility of updating wrong images as we are collecting meta-data of that face and considering the maximum similarities before storing new meta-data.

Table 1. Software description

Software	Version
Python	Python 3.6.1
Flask	Flask version 0.12.2
MySQL	MySQL version 5.7.19
SQLAlchemy	Flask-SQLAlchemy version 2.2
JSON Web Tokens	PyJWT version 1.5.2 and Flask-JWT 0.3.2
AWS CLI	awscli version 1.11.112, boto3 version 1.4.4 and botocore version 1.5.75

Table 2. Hardware description

Hardware	Description
VPC	A VPC with three private subnets and one public subnet is created to host the whole application
Bastion node	Bastion server is needed to access the others servers as it includes the public inbound rules and private outbound rules. We have chosen m1-large EC2 instance for this node
API and API dashboard node	The main Flask API and its dashboard are deployed in EC2 instances. For both operation, we have chosen m1-large instance which include Auto scaling groups. The AWS services like Rekognition, AMI, AWSCLI are performed from these nodes
Database node	The database is hosted in this node. It is also a m1-large EC2 instance and has only private access from the other nodes
Load balancers	Two Elastic Load Balancers ensure to handle the inbound traffic for both API access and API dashboard access
Security groups	Proper security groups are created for handing the inbound and outbound traffic in each node and services to restrict unauthenticated access of the application

Table 3. Fields of database

Table name	Fields
authenticated_organization	organization_id (PK), organization_email_address, organization_password, organization_name, organization_address, organization_phone_number, organization_email_address, organization_registration_id, organization_registration_address, organization_details
people	national_id (PK), full_name, father_name, mother_name, date_of_birth, gender, birth_district, permanent_address, present_address, email_address, phone_number, blood_group, nationality, religion, marital_status, spouse_name, tin_number, passport_number, driver_license_number, photo_url, created_timestamp, updated_timestamp
aws_face	face_id (PK), face_created_time, face_url, national_id (FK)
police_record	case_id (PK), case_status, case_description, case_location, case_outcome, case_created_time, case_updated_time, case_complainer_national_id (FK), case_investigator_national_id (FK), case_defendant_national_id (FK)

7 Conclusion

We developed an efficient approach towards the verification of people through securely accessing the public database supervised by government. We utilized the benefits of RESTful approach for ensuring security. All governments can utilize the hybrid architecture to ease the entire process thereby. Overall, this system provides public services in a faster way eradicating long time-wasting analog processes with its' features of digitization, citizen access, developed content. Here,

the deep learning based people verification approach improves the performance rapidly in each iteration. It is proved by the improvement of identity verification by 29.38% and 76.11% in first and second iteration. Again application pay only for the individual services it needs by using AWS which ultimately reduce the cost. We are storing the meta-data of every updated face image of people in a neural network. Increasing number of stored meta-data opens door to track all the people in a country or within a particular region boundary. The framework can be integrated with any authorized application to verify people's identity with ease.

References

1. Smitha, K.K., Thomas, T., Chitharanjan, K.: Cloud based E-Governance system: a survey. Procedia Eng. **38**, 3816–3823 (2012)
2. Whaiduzzaman, Md., Naveed, A., Gani, A.: MobiCoRE: mobile device based cloudlet resource enhancement for optimal task response. In: IEEE Transactions on Services Computing (2016)
3. Liang, J.: Government cloud: enhancing efficiency of E-Government and providing better public services. In: 2012 International Joint Conference on Service sciences (IJCSS), pp. 261–265. IEEE (2012)
4. Witarsyah, D., Sjafrizal, T., Fudzee, M.D., Farhan, M., Salamat, M.A.: The critical factors affecting E-Government adoption in Indonesia: a conceptual framework. Int. J. Adv. Sci. Eng. Inf. Technol. **7**(1), 160–167 (2017)
5. Rao, M.N., Krishna, S.R.: Efficient and ubiquitous software architecture of E-Governance for Indian Administrative Services. Int. J. Adv. Res. Comput. Sci. **4**(11), 66–74 (2013)
6. Zissis, D., Lekkas, D.: Securing E-Government and E-Voting with an open cloud computing architecture. Gov. Inf. Q. **28**(2), 239–251 (2011)
7. Zhang, W.J., Chen, Q.: From E-Government to C-Government via cloud computing. In: 2010 International Conference on E-Business and E-Government (ICEE), pp. 679–682. IEEE (2010)
8. Hashemi, S., Monfaredi, K., Masdari, M.: Using cloud computing for E-Government: challenges and benefits. Int. J. Comput. Inf. Syst. Control Eng. **7**(9), 596–603 (2013)
9. Cellary, W., Strykowski, S.: E-Government based on cloud computing and service-oriented architecture. In: Proceedings of the 3rd International Conference on Theory and Practice of Electronic Governance, pp. 5–10. ACM (2009)
10. Ling, H., Soatto, S., Ramanathan, N., Jacobs, D.W.: A study of face recognition as people age. In: 2007 IEEE 11th International Conference on Computer Vision, ICCV 2007, pp. 1–8. IEEE (2007)
11. Park, U., Tong, Y., Jain, A.K.: Age-invariant face recognition. IEEE Trans. Pattern Anal. Mach. Intell. **32**(5), 947–954 (2010)
12. Ling, H., Soatto, S., Ramanathan, N., Jacobs, D.W.: Face verification across age progression using discriminative methods. IEEE Trans. Inf. Forensics Secur. **5**(1), 82–91 (2010)
13. Sun, Y., Wang, X., Tang, X.: Deep learning face representation from predicting 10,000 classes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1891–1898 (2014)

14. Taigman, Y., Yang, M., Ranzato, M.A., Wolf, L.: Deepface: closing the gap to human-level performance in face verification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1701–1708 (2014)
15. Christensen, J.H.: Using RESTful web-services and cloud computing to create next generation mobile applications. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, pp. 627–634. ACM (2009)
16. Adamczyk, P., Smith, P.H., Johnson, R.E., Hafiz, M.: Rest and web services: in theory and in practice. In: REST: From Research to Practice, pp. 35–57. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9_2
17. Color FERET Database. https://www.nist.gov/itl/iad/image-group/color-feret-database. Accessed 03 Oct 2017

A Novel Anomaly Detection Algorithm Based on Trident Tree

Chunkai Zhang[1(✉)], Ao Yin[1], Yepeng Deng[1], Panbo Tian[1], Xuan Wang[1], and Lifeng Dong[2]

[1] Department of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China
ckzhang812@gmail.com, yinaoyn@126.com, erpim@qq.com, panbo_tian@qq.com, wangxuan@cs.hitsz.edu.cn

[2] Department of Physics, Hamline University, 1536 Hewitt Avenue, St. Paul, MN 55104, USA

Abstract. In this paper, we propose a novel anomaly detection algorithm, named T-Forest, which is implemented by multiple trident trees (T-trees). Each T-tree is constructed recursively by isolating the data outside of 3 *sigma* into the left and right subtree and isolating the others into the middle subtree, and each node in a T-tree records the size of datasets that falls on this node, so that each T-tree can be used as a local density estimator for data points. The density value for each instance is the average of all trees evaluation instance densities, and it can be used as the anomaly score of the instance. Since each T-tree is constructed according to 3 *sigma* principle, each tree in TB-Forest can obtain good anomaly detection results without a large tree height. Compared with some state-of-the-art methods, our algorithm performs well in AUC value, and needs linear time complexity and space complexity. The experimental results show that our approach can not only effectively detect anomaly points, but also tend to converge within a certain parameters range.

Keywords: Anomaly detection · Isolation · Forest · 3 *sigma* Gaussian

1 Introduction

Anomaly points are the data points that deviate from most data and do not obey the distribution of most data points [1–3]. Anomaly detection has been a widely researched problem in several application domains such as system health management, intrusion detection, healthy-care, bio-informatics, fraud detection, and mechanical fault detection. For these applications, anomaly detection as an unsupervised learning task is very important. The significance of anomaly detection is due to the fact that anomaly data can be translated into important operational information in various application domains. For example, with the

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 295–306, 2018.
https://doi.org/10.1007/978-3-319-94295-7_20

development of cloud computing [4–7], it is very important to detect abnormal traffic in the cloud in a timely manner. And with the development of the cloud storage [8,9], timely detection of abnormal disk reads and writes in cloud storage can greatly reduce the potential risk of cloud storage.

Many unsupervised anomaly detection approaches, including classification-based [10,11], clustering-based [12,13], density-based [14–17], and angle-baed [18], calculate the distance between data points to determine their similarity, and then determine whether the data points are abnormal. There are many ways to calculate the distance, such as Euclidean distance, DTW, and so on. By analyzing the characteristics of these distance calculation formulas, we can get that the result can be easily affected by the data values and the number of data attributes. And many of the above algorithms are constrained to low dimensional data and small data size due to the high computational complexity of its original algorithm. In order to solving the above problems, Liu *et.al* proposed a different approach that detects anomalies by isolating instances, without relying on any distance or density measure [19,20]. In this approach, since the attribute and the split value of each node in the isolation tree are randomly selected, the built isolation tree can also be called a completely random tree. Consequently, there is often a certain degree of randomness in the anomaly detection results by using such a model.

In response to these challenges, we propose a novel anomaly detection algorithm on the basis of isolation-forest [19]. The key insight of our proposed algorithm is a fast and accurate local density estimator implemented by multiple trident trees(T-trees), called T-Forest. The proposed approach can isolate anomaly data faster and more accurately. To achieve this, we extend the binary tree structure of the original isolation tree to the structure of the trident tree. And instead of selecting the split value randomly, we take advantage of *sigma* principle to select two split values at a time to split the inconsistent attribute data as soon as possible. In this paper, we will show that trident trees can effectively isolate anomaly points.

The main contributions of this paper are summarized as follows:

- We propose a novel anomaly detection algorithm, called TB-Forest. Data points which have short average path on the T-tree, may be seen as anomaly points.
- We propose a local density assessment method, which is used to estimate the anomaly degree of data points. We perform many experiments on benchmark dataSets. The experiment results show that our proposed approaches outperforms the competing methods on most of the benchmark dataSets in AUC score.

The remainder of this paper is organized as follows. In Sect. 1, we review the related work. In Sect. 2, we present the proposed anomaly detection algorithm. In Sect. 3, we perform some empirical experiments to demonstrate the effectiveness of our algorithm. Lastly, the conclusion will be shown in Sect. 4.

2 Anomaly Detection by Our Method

In this section, we will show the detailed steps of our proposed detection algorithm. We first show some definitions used in this paper. Then we present the implementation details. Table 1 summarizes the symbols used in this paper.

2.1 Definitions

In this section, we will present the definition of T-tree and introduce the attributes of each node in a T-tree. And we definite the formula for calculating the anomaly score.

Table 1. Symbols and descriptions

Symbols	Description
N	Number of instances in a dataset
x	An instance
X	A dataset of N instances
Q	A set of attributes
q	An attribute
u_q	The mean of the attribute q
σ_q	The standard deviation of the attribute q
T	A tree
T_r	A right tree of a node τ
T_l	A left tree of a node τ
T_m	A middle tree of a node τ
p_l	The split value between left tree and middle tree, $u_q - 3\sigma_q$
p_r	The split value between middle tree and left tree, $u_q + 3\sigma_q$
t	Number of trees
ψ	Sample ratio
$den(x)$	The number of contained instances of the external node that x belongs to
$hlim$	Height limit
$slim$	Size limit in training a tree

Definition 1. T-tree: *Let T denote a T-tree and τ be a node in this T-tree. It is either an external-node with no child, or an internal-node with a set and exactly three child trees(T_l,T_m,T_r). Each node in a T-tree contains the following elements:(1) variables p_l and p_r, which are used to divide data points into T_l, T_m, and T_r subtree; (2) variable size, which is used to record the instances number located in this node; (3) three node pointers, which are used to the left subtree, the right subtree, and the middle subtree.*

Given a sample of data X=$\{x_1, x_2, ..., x_n\}$ of n instances, in which each x_i has d attributes, to build a trident tree(T-tree). Figure 1 is used as an example

to illustrate the structure of a T-tree. We recursively divide X by randomly selecting an attribute q and calculating the left split value p_l and the right split value p_r, until either: (1) the tree height reaches limit, $hlim$, (2) the number of instances at a node τ is less than the size limit, $slim$ or (3) all points of the selected attribute have the same value in X. An T-tree has the property as BST, that is, the value of an attribute in left child tree is less than the middle side and the middle size is less than the right side. Assuming all instances are distinct, the number of instances in left child tree is equal to the number in the right side. Since each leaf node in the MB-Tree contains no fewer than one instance and each internal node contains exactly three children, the total number of nodes in a T-tree is less than $\frac{3N}{2} - 1$. Hence, an MB-Tree is only linear storage overhead.

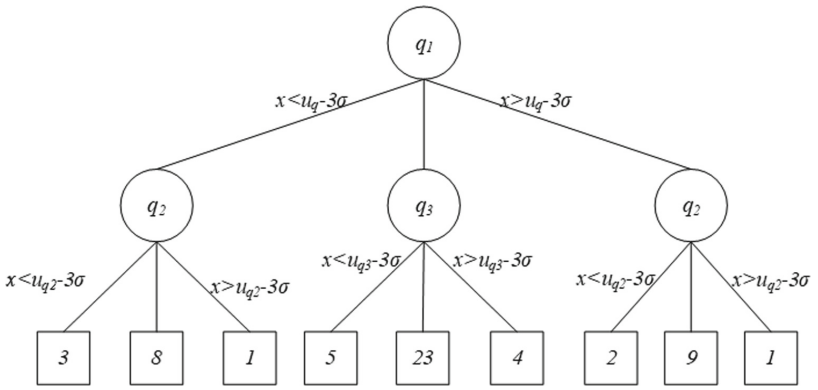


Fig. 1. This figure is used as an example to illustrate the structure of a T-tree and show the process of the division. Round nodes represent internal nodes, and square nodes represent external nodes.

In this paper, calculating a score for each data point that reflects the anomaly degree of each data point. Since all instances in the dataSets fall on different external nodes after being divided several times according to different attribute values, each node in a T-tree forms a local subspace and instances located on each node have similar data characteristics. Instances on each node become K-nearest-neighbors to each other. Hence, we calculate the local density of the instance by counting the number of instances in each node where the instance falls, to determine the anomaly degree of the instance.

Anomaly Score: The anomaly score of each point x is measured by the average local density, $den(x)$. $den(x)$ denotes the number of instances contained in the terminational node of x. Since we have adopted the idea of ensemble learning and the number of instances used to train each T-tree may be different, we need to normalise the value $den(x)$ in order to calculate the final anomaly score by using all results in the TB-Forest. Therefore, the anomaly score of an instance can be defined as follows:

$$score(x) = \frac{1}{t}\sum_{i=1}^{t}\frac{den_i(x)}{N_i} \tag{1}$$

where $den_i(x)$ is the local density in a T-tree and N_i is the instances number in T-tree$_i$. Because of the characteristics "less and different" of anomaly points, the value of $score(x)$ will be less than the normal points. Therefore, the smaller the value of $score(x)$, the more abnormal it is. In other words, the closer of the value of $score(x)$ is to 0, the more likely the corresponding instance is an anomaly point.

2.2 Training for T-Forest

In this section, we will introduce the concrete steps of building a T-Forest, which is composed of many T-trees. Each T-tree is built on the dataset sampled by using the variable bagging technique.

Building a T-tree: A T-tree can be constructed by recursively dividing the training datasets until the tree reaches height limit or the instances number is less than the size limit. During the construction of each node of a T-tree, an attribute will be randomly selected from the attributes of the datasets as the splitting attribute. And these two splitting values, p_l and p_r, will be obtained by calculating the mean value and the standard derivation of this attribute, and then are used to divide the training datasets. The concrete steps to build a T-tree are described in Algorithm 1. The time complexity of this algorithm is only $O(log_3(N))$, and the space complexity is liner $O(N)$.

Building a T-Forest: There are three parameters that need to be set manually to the TB-Forest. They are the height limit of a T-tree $hlim$, the size limit of each node $slim$, and the tree number t. There is also a variable, sample ratio ψ, which is generated uniformly from the range between 0.2 and 0.8. This parameter may be different in each T-tree, so it can increase the diversity of samples in each T-tree. The changes of these parameters will affect the evaluation effect. For example, many trees or a high T-tree can increase the accuracy of the result. However, after increasing to a certain value, the result will tend to converge. If these two parameters are set too large, it not only did not improve the detection performance, but increase the model's runtime and memory consumption. The size limit is always set to 15, because if the number of samples is less than 15, the criteria 3 $sigma$ will no longer apply. The concrete steps of TB-Forest algorithm are described as Algorithm 2. The time complexity of building a TB-Forest is $O(t * \psi * log_3(N))$. The space complexity of building a TB-Forest $O(t * \psi * N)$.

2.3 Evaluation Stage

In this evaluation stage, an anomaly score of instance x can be estimated by the average local density $E(den(x))$. $den(x)$ can be calculated by Algorithm 3. After getting all $den(x)$ of instance x by TB-Forest, we can use the formula

Algorithm 1: Building a T-tree($Data$, cur, $hlim$, $slim$)

Input: A dataSet $Data$, a current height cur, a height limit $hlim$, a size limit $slim$.

Output: A T-tree

if $|Data| \leq slim$ *and* $cur \geq hlim$ **then**
    return node(size=|Data|,exteral=TRUE);
end
Randomly select an attribute $q \in Q$.
Calculate the mean value, u_q, of the attribute q.
Calculate the standard derivation, σ, of the attribute q.
left-split $\leftarrow u_q - 3\sigma$
right-split $\leftarrow u_q + 3\sigma$
$Data_l \leftarrow$ data-filter($Data$, $x \leq$ left-split)
$Data_m \leftarrow$ data-filter($Data$, $x >$left-split and $x <$right-split)
$Data_r \leftarrow$ data-filter($Data$, $x \geq$ right-split)
Left-Tree $\leftarrow$ Building-Tree($Data_l$,$cur + 1$, $hlim$, $slim$)
Middle-Tree $\leftarrow$ Building-Tree($Data_m$,$cur + 1$, $hlim$, $slim$)
Right-Tree $\leftarrow$ Building-Tree($Data_r$,$cur + 1$, $hlim$, $slim$)
return node(Left-Tree,Middle-Tree,Right-Tree,size=|Data|,p_l =left-split,p_r =right-split,exteral=FALSE);

1 to calculate the average local density $E(den(x))$. The closer of the value of $E(den(x))$ is to 0, the more likely the instance x is an anomaly point. For the test dataSet, we can sort the anomaly scores of these instances to get the top K anomaly points. The time complexity of getting the average local density $E(den(x))$ of this test dataSet is $O(M * t * log_3(N * \varphi))$, where N is the instances number of training dataSets, and M is the instances number of this test dataSet.

3 Experimental Evaluation

In this section, we will present the performance of our proposed algorithm from many experiments using the public dataset from UCI. For comparability, we implemented all experiments on our workstation with 2.5 GHz, 6 bits operation system, 4 cores CPU and 16 GB RAM, and the algorithms codes are built in Python 2.7.

3.1 Experimental Metrics and Experimental Setup

Metrics: In our experiment, we use Area Under Curve(AUC) as the evaluation metric with other classic anomaly detection algorithms. AUC denotes the area under of Receiver Operating Characteristic(ROC) curve and illustrates the diagnostic ability of a binary classifier system. AUC is created by plotting the true positive rate against the false positive rate at various threshold settings[1].

[1] https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

Algorithm 2: Building a TB-Forest

Input: A dataSet *Data*,number of trees t, height limit *hlim*, size limit *slim*
Output: A set of TB-Trees, TB-Forest

```
Forest ← set();
sample-size ← uniform(min())
for i=1 to t do
    ψ ← uniform(0.2,0.8)
    Data~ ← sample(Data, ψ).
    tree ← Building-Tree(Data~,0,hlim,slim)
    Forest ← [Forest, tree]
end
return Forest;
```

Algorithm 3: Local density of an instance($den(x)$)

Input: An instance x, a height limit *hlim*, A T-tree *root*
Output: The anomaly score of this instance

```
if root is external or cur >= hlim then
    return root→size / N;
end
if x_q ≤ root.q_l then
    return Local-density(x,cur + 1,hlim, root → left);
else if x_q ≥ root.q_l then
    return Local-density(x,cur + 1,hlim, root → right);
else
    return Local-density(x,cur + 1,hlim, root → middle);
end
```

The detection algorithm with larger AUC has the better detection accuracy, otherwise, the detection algorithm is less effective.

Experimental Setup: There are two types of experiments. First, we compare the differences in the AUC value between our proposed algorithm and other classic algorithms. Second, we compare the effect of parameters on our proposed algorithm and iForest. All above experiments are performed on the selected twelve datasets from public UCI datasets [21], which are summarized in Table 2. Most of these datasets include two labels, and we use the most class as the normal class and the less class as the anomaly class, for example, Http which is from KDD CUP 99 network intrusion data [22] includes two classes 0, 1. For other datasets that include multiple classes, we need process these into two labels. Arrhythmia has 15 classes, and we choose 3, 4, 5, 7,8,9, 14, 15 as the anomaly class and other classes as the normal class. We choose *NON-MUSK-252*, *NON-MUSK-j146*, and *NON-MUSK-j147* as the normal class, and choose *MUSK-213*, *MUSK-211* as the anomaly class in musk dataset. In HAPT dataset, we choose

the class 5 as the normal class and the class 11 as the anomaly class to form the *hapt511* dataset.

In order to present the efficiency of our proposed algorithm, we choose four representative anomaly detection algorithms, which include LOF [23], iForest [19], HS-Forest [24], RS-Forest [25]. LOF is the same as our proposed algorithm to determine the abnormality of data points by calculating the local density of data points, but LOF calculate the local density of data points by calculating the similarity between data points and it needs $O(N^2)$ time complexity. HS-Forest, RS-Forest are the same as our proposed algorithm to calculating the local density of data points by counting the instances number of a terminal node in model tree, but each model tree in these two algorithms is constructed without training datasets. And these two algorithms can be used to stream data, but we only use their function on static datasets. Because our algorithm is an improvement on iForest, we choose it as a comparision algorithms. As for LOF, we set $k = 10$ in our experiment. As for HS-Forest, RS-Forest, and iForest, we set height limit $hlim = 6$ and the trees number $t = 25$. For our proposed algorithm, we set the height limit $hlim = 4$ and the trees number $t = 25$.

Table 2. Benchmark data sets, where n denotes the size of datasets, and d denotes the number of attributes of datasets.

Datasets	n	d	Anomaly ratio
Http	567497	3	0.4%
Satellite	6435	37	31.6%
ann_throid	7200	6	7.4%
Cardiotocography	1831	20	9.6%
Musk	5682	165	1.7%
Epileptic	11500	178	20%
hapt511	1513	561	5.9%
Breast	569	10	35.7%
Arrhythmia	452	273	14.6%
Shuttle	14500	10	5.9%
Pima	768	9	35%
Ionosphere	351	34	35.8%

3.2 Performance on AUC

This experiment is to compare our proposed algorithm with other algorithms in term of AUC. Table 3 presents the results of all compared algorithms on benchmark datsets. From this table, we can find that our algorithm outperforms other algorithms on most of this benchmark datasets. The AUC performance of our algorithms are approximative to RS-Forest, HS-Forest, and iForest, but it is much better than LOF algorithm on all these datasets. From this table,

we can find that our proposed algorithm outperforms iForest on nine of twelve datasets, which can illustrate that our improvement is successful. And we can observe that our algorithms performs well on these datasets, which contain a small percentage of anomaly class from Table 2.

As we all known, the Http dataset is a network traffic data set provided by KDD99. There is a small amount of traffic data in this dataset as abnormal traffic, and we treat these small amount of abnormal data as data with *class 0*. Our detection algorithm can efficiently detect such abnormal network traffic, and our algorithm only requires a logarithmic level of runtime. So our algorithm can be used to detect whether there is abnormal traffic in the public cloud of private cloud.

Table 3. Performance comparison of different methods on different benchmark data sets. AUC score is measured, and the bold font indicates that the algorithm performs significantly better.

Data sets	**TB-Forest***	iForest	LOF	RS-Forest	HS-Forest
Http	0.9925	**1.00**	NA	0.999	0.996
Satellite	0.68	**0.71**	0.52	0.7	0.59
ann_thyroid	**0.85**	0.81	0.72	0.68	0.8
Cardiotocography	**0.93**	0.92	0.539	0.88	0.74
Musk	**0.77**	0.64	0.531	0.64	0.66
Epileptic	**0.984**	0.98	0.57	0.88	0.94
hapt511	**0.999**	0.998	0.595	0.997	0.982
Breast	0.88	0.84	0.6293	0.518	**0.94**
Arrhythmia	**0.836**	0.80	0.69	0.695	0.686
Shuttle	0.992	**1.00**	0.55	0.998	0.999
Pima	**0.71**	0.67	0.513	0.49	**0.71**
Ionosphere	**0.94**	0.85	0.546	0.89	0.78

Parameters Analysis: In this experiment, we show the effect of two parameter (*hlim* and t) values on the detection results of all compared algorithms on benchmark datasets. Due to the space limitations, we only show the experiment results on *Http* and *shuttle* datasets.

Figure 2 shows that the value of AUC changes as the tree height changing in the *Http* dataset, when the number of trees is set to 25. This figure shows that our proposed algorithm performs better when the height limit is in the range 2 to 6. Since *Http* dataset has only three attributes, it does not require a very high tree to get good performance. Figure 3 shows that the value of AUC changes as the tree height changing in the *shuttle* dataset, when the number of trees is set to 25. In this figure, with the increasement of height limit, all comparable algorithms perform better and better and tend to converge. From these two

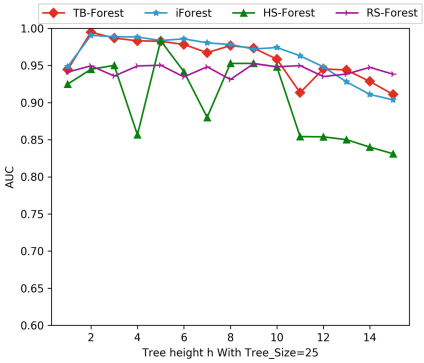


Fig. 2. AUC changes with *hlim*, when fixed $t = 25$, on http data.

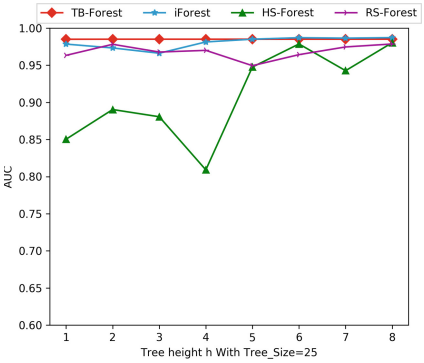


Fig. 3. AUC changes with *hlim*, when fixed $t = 25$, on shuttle data.

examples, we can find that the tree height limit of each T-tree in T-Forest can be in the range of 4 to 6, in our proposed algorithm.

Therefore, we run two experiments to detect the effect of changes in the number of trees on the AUC value with height limit $hlim = 6$. Figure 4 shows that the value of AUC changes as the number of trees changing in the *http* dataset, when the height limit of trees is set to 6. Figure 5 shows that the value of AUC changes as the number of trees changing in the *shuttle* dataset, when the height limit of trees is set to 6. From these two figures, we can observe that changes in the number of trees have little effect on the effectiveness of our algorithm. To accommodate most data sets, the parameters t can be in the range of 10 to 25.

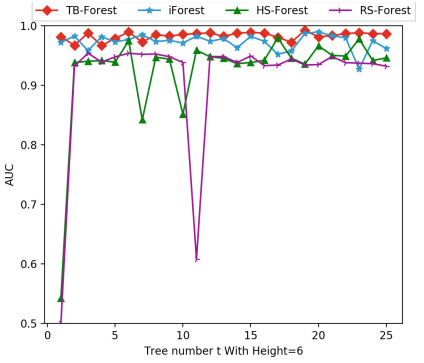


Fig. 4. AUC changes with t, when fixed $hlim = 6$, on http data.

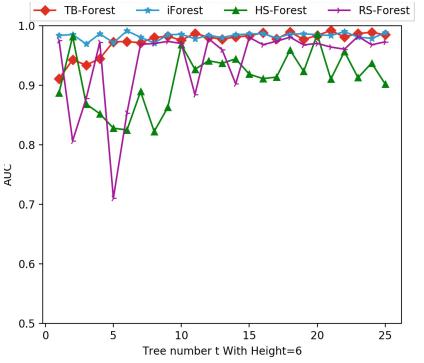


Fig. 5. AUC changes with t, when fixed $hlim = 6$, on shuttle data.

4 Conclusions

In this paper, we propose a novel anomaly detection, T-Forest, based on isolation principle. Our algorithm constructs many TB-Trees using sampled datasets by variable sample technique, and each T-tree is a trigeminal tree which is built by recursively segmenting dataset to map dataset to different subtrees by 3 *sigma* principle. Then, we have performed some experiments to illustrate the detection effect of our algorithm. The experiment results show that our algorithm can detect anomaly data points effectively and efficiently. In the future, we will focus on how to improve the detection accuracy of our algorithm on the datasets, in which normal and anomaly points are mixed distributions.

Acknowledgment. This study was supported by the Shenzhen Research Council (Grant No. JSGG20170822160842949, JCYJ20170307151518535).

References

1. Yu, X., Tang, L.A., Han, J.: Filtering and refinement: a two-stage approach for efficient and effective anomaly detection. In: IEEE International Conference on Data Mining, pp. 617–626 (2009)
2. Xie, M., Han, S., Tian, B., Parvin, S.: Anomaly detection in wireless sensor networks: a survey. J. Netw. Comput. Appl. **34**(4), 1302–1325 (2011)
3. Liu, S., Chen, L., Ni, L.M.: Anomaly detection from incomplete data. ACM Trans. Knowl. Discov. Data (TKDD) **9**(2), 11 (2014)
4. Baucke, S., Ali, R.B., Kempf, J., Mishra, R., Ferioli, F., Carossino, A.: Cloud atlas: a software defined networking abstraction for cloud to WAN virtual networking. In: IEEE Sixth International Conference on Cloud Computing, pp. 895–902 (2014)
5. Sayeed, Z., Liao, Q., Grinshpun, E., Faucher, D., Sharma, S.: Cloud analytics for short-term LTE metric prediction-cloud framework and performance. In: IEEE CLOUD (2015)
6. Kauffman, R.J., Ma, D., Shang, R., Huang, J., Yang, Y.: On the financification of cloud computing: an agenda for pricing and service delivery mechanism design research. Int. J. Cloud Comput. Featur. Article (2015)
7. An, B., Zhang, X., Tsugawa, M., Zhang, Y., Cao, C., Huang, G., Fortes, J.: Towards a model-defined cloud-of-clouds. In: Collaboration and Internet Computing, pp. 1–10 (2016)
8. Bellini, P., Cenni, D., Nesi, P.: A knowledge base driven solution for smart cloud management. In: IEEE International Conference on Cloud Computing, pp. 1069–1072 (2015)
9. Chen, W.-S.E., Huang, M.-J., Huang, C.-F.: Intelligent software-defined storage with deep traffic modeling for cloud storage service. Int. J. Serv. Comput. (IJSC), 1–14 (2016)
10. Abe, N., Zadrozny, B., Langford, J.: Outlier detection by active learning. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 504–509. ACM (2006)
11. Shi, T., Horvath, S.: Unsupervised learning with random forest predictors. J. Comput. Graph. Stat. **15**(1), 118–138 (2006)

12. He, Z., Xiaofei, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24**(9–10), 1641–1650 (2003)
13. Niu, K., Huang, C., Zhang, S., Chen, J.: ODDC: outlier detection using distance distribution clustering. In: Washio, T., et al. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4819, pp. 332–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77018-3_34
14. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J.: LOF: identifying density-based local outliers. In: ACM Sigmod Record, vol. 29, pp. 93–104. ACM (2000)
15. Fan, H., Zaïane, O.R., Foss, A., Wu, J.: A nonparametric outlier detection for effectively discovering top-N outliers from engineering data. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 557–566. Springer, Heidelberg (2006). https://doi.org/10.1007/11731139_66
16. Salehi, M., Leckie, C., Bezdek, J.C., Vaithianathan, T., Zhang, X.: Fast memory efficient local outlier detection in data streams. IEEE Trans. Knowl. Data Eng. **28**(12), 3246–3260 (2016)
17. Yan, Y., Cao, L., Kulhman, C., Rundensteiner, E.: Distributed local outlier detection in big data. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1225–1234. ACM (2017)
18. Kriegel, H.-P., Zimek, A., et al.: Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 444–452. ACM (2008)
19. Liu, F.T., Ting, K.M., Zhou, Z.-H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, ICDM 2008, pp. 413–422. IEEE (2008)
20. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. ACM Trans. Knowl. Discov. Data **6**, 1 (2012)
21. Dheeru, D., Taniskidou, E.K.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
22. Yamanishi, K.: On-line unsupervised outlier detection using finite mixture with discounting learning algorithms. Data Min. Knowl. Discov. **8**(3), 275–300 (2004)
23. Breunig, M.M., Kriegel, H.P., Ng, R.T.: LOF: identifying density-based local outliers. In: ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
24. Tan, S.C., Ting, K.M., Liu, T.F.: Fast anomaly detection for streaming data. In: Proceedings of the International Joint Conference on Artificial Intelligence, Barcelona, IJCAI 2011, Catalonia, Spain, pp. 1511–1516, July 2011
25. Wu, K., Zhang, K., Fan, W., Edwards, A., Philip, S.Y.: RS-forest: a rapid density estimator for streaming anomaly detection, pp. 600–609 (2014)

Application and Industry Track: Cloud Environment Framework

Framework for Management of Multi-tenant Cloud Environments

Marek Beranek[1], Vladimir Kovar[1], and George Feuerlicht[1,2,3(✉)]

[1] Unicorn College, V Kapslovně 2767/2, 130 00 Prague 3, Czech Republic
marek.beranek@unicorncollege.cz,
vladimir.kovar@unicorn.eu,
[2] Prague University of Economics, W. Churchill Square. 4,
130 67 Prague 3, Czech Republic
[3] University of Technology, Sydney,
PO. Box 123, Broadway, NSW 2007, Australia
george.feuerlicht@gmail.com

Abstract. The benefits of using container-based microservices for the development of cloud applications have been widely reported in the literature and are supported by empirical evidence. However, it is also becoming clear that the management of large-scale container-based environments has its challenges. This is particularly true in multi-tenant environments operating across multiple cloud platforms. In this paper, we discuss the challenges of managing container-based environments and review the various initiatives directed towards addressing this problem. We then describe the architecture of the Unicorn Universe Cloud framework and the Unicorn Cloud Control Centre designed to facilitate the management and operation of containerized microservices in multi-tenant cloud environments.

Keywords: Cloud computing · Multi-tenancy · Resource management

1 Introduction

Forrester Research estimates that the cloud computing market could reach 236 billion by 2020 [1]. This rapid rate of adoption of cloud computing is likely to have a dramatic impact on how organizations develop and operate enterprise applications. Public cloud platforms (e.g. AWS [2], Microsoft Azure [3], etc.) offer highly elastic and practically unlimited compute power and storage capacity, allowing flexible acquisition of IT (Information Technology) resources in the form of cloud services, avoiding many of the limitations of on-premises IT solutions. This new technology environment is creating opportunities for innovative solutions at a fraction of the cost of traditional enterprise applications. However, to take full advantage of these developments, organizations involved in the production of enterprise applications must adopt a suitable enterprise architecture and application development frameworks. The architecture should reduce the complexity of application development and maintenance and facilitate effective reuse of application components and infrastructure services. These requirements demand a revision of existing architectural principles and application

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 309–322, 2018.
https://doi.org/10.1007/978-3-319-94295-7_21

development methods. Enterprise IT architecture needs to reflect current technology trends and provide framework services that support cloud deployment of applications, mobile computing and integration with IoT devices. This allows application developers to concentrate on the functionality that directly supports business processes and adds value for the end users. Framework services should enable a single sign-on and user authentication regardless of the physical location of the user and the device used, and the application should run on different end user devices such as mobile phones, tablets, notebooks, etc. without the need for excessive modifications. Reusing standard framework services across all projects saves programming effort and improves the reliability of the applications.

Unlike traditional enterprise applications that store data on local servers on-premises, most mobile applications store data and applications in the cloud to allow applications to be shared by very large population of users. Furthermore, given the requirements of modern business environments, the architecture needs to facilitate fast incremental development of application components and ensure rapid cloud deployment. The need for continuous delivery and monitoring of application components impacts on the structure and skills profile of IT teams, favoring small cross-functional teams leading to the convergence of development and operations (DevOps).

There is now increasing empirical evidence that to effectively address such requirements, the architecture needs to support microservices and container-based virtualization. A major advantage of using containers to implement microservices is that the microservices architecture hides the technical details of the underlying hardware and operating systems, and allows developers to focus on managing applications using application level APIs (Application Programming Interfaces). It also allows the replacement of hardware and operating systems upgrades without impacting on existing applications. Finally, as the unit of deployment is the application, monitoring information (e.g. metrics such as CPU and memory usage) is tied to applications rather than machines, which dramatically improves application monitoring and introspection [4]. It can also be argued that using containers for virtualization improves isolation in multi-tenant environments [5]. However, it is also becoming clear that the management of large-scale container-based environments has its own challenges and requires automation of application deployment, auto-scaling and predictability of resource usage. At the same time, there is a requirement for portability across different public and private clouds. While the use of public cloud platforms is economically compelling, an important function of the architecture is to ensure independence from individual cloud providers, avoiding a provider *lock-in*.

Platforms and frameworks that support the development and deployment of container-based applications have become an active area of research and development with a number of open source projects, including Cloud Foundry [6], OpenShift [7], OpenStack [8] and Kubernetes [9] recently initiated. These projects share many common concepts and in some cases common technologies. A key idea of such open source platforms is to abstract the complexity of the underlying cloud infrastructure and present a well-designed API that simplifies the management of container-based cloud environments. An important requirement that is often not fully addressed by these frameworks concerns the management of multi-tenancy. Management of multiple tenants across multiple public cloud platform presents a number of significant

challenges. Each tenant has to be allocated separate resources that are separately monitored and billed. Moreover, the access to tenants' resources must be controlled so that only authorized users are able to deploy applications on specific resources (i.e. physical or virtual servers) and access these applications.

In this paper, we describe the Unicorn Universe Cloud Framework (uuCloud) - framework designed to manage multi-tenant cloud environments. Unicorn is group of companies based in Prague, Czech Republic that specializes in providing information systems solutions for private and government organizations across a range of industry domains including banking, insurance, energy and utilities, telecommunications, manufacturing and trade (https://unicorn.com/). Unicorn Universe Cloud (uuCloud) is an integral part of the Unicorn Application Framework (UAF), a framework developed by Unicorn that comprises a reference architecture and fully documented methods and tools that support the collaboration of teams of developers during the entire development life-cycle of enterprise applications. A key UAF architectural objective is to support various types of mobile and IoT devices and to facilitate deployment of containerized cloud-based applications using standard framework services [10].

In the next section (Sect. 2) we review related literature focusing on frameworks for the management of container-based cloud environments. Section 3 describes the main components of uuCloud framework and Sect. 4 includes our conclusions and directions for further work.

We note that all diagrams in this paper are drawn using uuBML (Unicorn Universe Business Modeling Language: https://unicornuniverse.eu/en/uubml.html), UML-like graphical notation used throughout Unicorn organizations. Figure 1 illustrates basic relationships and their graphical representation. The prefix *uu* abbreviates *Unicorn Universe*, and is a naming convention used throughout the paper to indicate UAF objects.

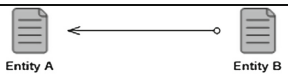
Type of relationship	Graphical representation
Association One-to-Many	
Association One-to-One	
Association Many-to-Many	

Fig. 1. uuBML relationships

2 Related Work

Claus Pahl [11, 12] reviews container virtualization principles and investigates the relevance of container technology for PaaS clouds. The author argues that while VMs are ultimately the medium to provision a PaaS platform and application components at the infrastructure layer, containers appear to be more suitable for application packaging and management in PaaS clouds. The paper compares different container models and concludes that container technology has a huge potential to substantially advance PaaS technology, but that significant improvements are required to deal with data and network management aspects of container-based architectures.

Brewer [13] in his keynote "Kubernetes The Path to Cloud Native" argues that we are in middle of a great transition toward *cloud native* applications characterized by highly available unlimited "ethereal" cloud resources. This environment consists of co-designed, but independent microservices and APIs, abstracting away from machines and operating systems. Microservices resemble objects as they encapsulate state and interact via well-define APIs. This allows independent evolution and scaling of individual microservices. The Kubernetes project initiated by Google in 2014 as an open source cluster manager for Docker containers has its origins in an earlier Google container management systems called Borg [14] and Omega [15]. The Kubernetes project is hosted by the Cloud Native Computing Foundation (CNCF) [16] that has a mission "to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self-healing multi-tenant nodes". The objective is to facilitate *cloud native* systems that run applications and processes in isolated units of application deployment (i.e. software containers). Containers implement microservices which are dynamically managed to maximize resource utilization and minimize the cost associated with maintenance and operations. CNCF promotes well-defined APIs as the main mechanism for ensuring extensibility and portability. The state of objects in Kubernetes is accessed exclusively through a domain-specific REST API that supports versioning, validation, authentication and authorization for a diverse range of clients [4]. A basic Kubernetes building block is a *Pod* - a REST object that encapsulates a set of logically connected application containers with storage resources (Volumes) and a unique IP address. Pods constitute a unit of deployment (and a unit of failure) and are deployed to *Nodes* (physical or logical machines). Lifetime of a Volume is the same as the lifetime of the enclosing Pod allowing restart of individual containers without the loss of data, however restarting the Pod results in the loss of Volume data. Pods are externalized as *Services*; Kubernetes service is an abstraction that defines a logical set of Pods and a policy for accessing the Pods. *Replication Controller* is used to create replica Pods to match the demand of the application and provide auto-scaling. Kubernetes uses *Namespaces* to avoid object name conflicts and to partition resources allocated to different groups of users. Although namespaces can be used to enable multi-tenancy in Kubernetes, this approach appears to have limitations. The application of Docker and Kubernetes container architecture to multi-tenant SaaS (Software as a Service) applications has been investigated and assessed using SWOT (Strength, Weakness, Opportunities and Threats) analysis and contrasted with developing multi-tenant SaaS

applications using middleware services [17]. The authors conclude that more research is needed to understand the true potential and risks associated with container orchestration platforms such as Kubernetes. Other authors have considered multi-tenancy from the viewpoint of security challenges and proposed a model for identifying suspicious tenant activities [18].

Other efforts in this area include OpensStack [19] - a *cloud operating system* designed to control large pools of compute, storage, and networking resources. Administrators manage resources using a dashboard, and provision resources through a web interface. In [20], the authors argue that many existing frameworks have a high degree of centralization and do not tolerate system component failures; they present the design, implementation and evaluation of a scalable and autonomic virtual machine (VM) management framework called *Snooze*.

Li et al. [21] describe a REST service framework based on the concept of Resource-Oriented Network (RON). The authors discuss the advantages of container-based virtualization over VMs (Virtual Machines) that include lower CPU and memory consumption, faster reboot time, and significantly higher deployment density (6–12 times), requiring fewer physical servers for the same workload and resulting in significant savings in capital expenditure. They identify a trend towards RaaS (Resource-as-a-Service) that allows consumers to rent fine-grained resources such as memory, CPU and storage on demand for short periods of time. The authors point out advances in Network Functions Virtualization (NFV) that make it possible to control network elements, and advances in computer architecture that make it possible to disaggregate CPU cores from the internal memory. The authors argue that a new service abstraction layer is required to control and monitor fine-grained resources and to overcome the heterogeneity of the underlying Linux resource control models. They and note that RaaS cloud brings new challenges, in particular the ability to efficiently control a very large number of fine-grain resources that are rented for short periods of time. According to Ben-Yehuda [22], the trend toward increasingly finer resource granularity is set to continue resulting in increased flexibility and efficiency of cloud-based solutions. Resources such as compute, memory, I/O, storage, etc. will be charged for in dynamically changing amounts, not in fixed bundles. The authors draw an analogy between cloud providers and phone companies that have progressed from billing landlines per several minutes to billing cellphones by the minute, or even per second. They conclude that the RaaS cloud requires new mechanisms for allocating, metering, charging for, reclaiming, and redistributing CPU, memory, and I/O resources among multiple clients every few seconds.

In another publication [23] the authors explore the potential of vertical scalability, and propose a system for autonomic vertical elasticity of Docker containers (ElasticDocker). ElasticDocker performs live migration when the resources on the hosting machine are exhausted. Experimental results show improved efficiency of live migration technique, with a gain of 37.63% when compared to Kubernetes. As a result of increase or decrease of workload, ElasticDocker scales up and down CPU and memory assigned to containers, modifying resource limits directly in the Linux control groups (cgroups) associated with Docker containers. When the host resource limits are reached, the container is migrated to another host (i.e. the container image is transferred from a source to a target host). The authors argue that vertical scalability has many

benefits when compared to horizontal scalability, including fine-grained scaling and avoiding the need for additional resources such as load balancers or replicated instances and delays caused by starting up new instances. Furthermore, horizontal scalability is only applicable to applications that can be replicated and decomposed into independent components. The experimental verification was performed using Kubernetes platform running CentOS Linux 7.2 on 4 nodes compared to running identical workload on ElasticDocker, showing an improvement in the average total execution time of almost 40%.

While Kubernetes appears to be gaining momentum as a platform for the management of cloud resources with support for major public cloud platforms including Google Cloud Platform, Microsoft Azure, and most recently Amazon AWS, there is a number of other projects that aim to address the need for universal framework for the development and deployment of cloud applications. This rapidly evolving area is of research interest to both academia and industry practitioners, but currently there is lack of agreement about a standard application framework designed specifically for cloud development and deployment. This is particularly true in the context of multi-tenant cloud applications where the framework needs to support the management of resources allocated to a large number of individual tenants, potentially across multiple public cloud platforms. Moreover, some proposals lack empirical verification using large-scale real-life applications. Further complicating the situation is the rapid rate of innovation in this area characterized by the current trend toward increasingly finer resource granularity with corresponding impact on the complexity of cloud resource management frameworks.

3 Unicorn Universe Cloud (uuCloud)

uuCloud framework facilitates the provisioning of elastic cloud services using containers and virtual servers, and consist of two basic components: the uuCloud Operation Registry (uuOR) - a database that maintains active information about uuCloud objects, and the uuCloud Control Centre (uuC3) - a tool that is used to automate deployment and operation of container-based microservices. To ensure portability and to reduce overheads, uuCloud uses Docker [24] container-based virtualization. Docker containers can be deployed either to a public cloud infrastructure (e.g. AWS or Microsoft Azure) or to a private (on-premises) infrastructure (e.g. Plus4U – Unicorn Universe cloud infrastructure). uuCloud supports the deployment of virtualized Unicorn Universe Applications (uuApps). uuApp implements a cohesive set of functions, and in general is composed of sub-applications (uuSubApps); each uuSubApp is an independent unit of functionality that implements a specific business use case (e.g. booking a visit to a doctor's surgery). We made an architectural decision to implement each uuSubApp as a containerized microservice using a Docker container.

Access to sub-applications is controlled using identities that are derived from user Roles (typically derived from the organizational structure), and are assigned to users and system commands. The uuCloud security model is based on a combination of Application Profiles (collection of application functions that the application executes) and Application Workspaces (collection of application objects). Access to Application

Workspaces is granted according to identities associated with the corresponding Application Profile.

In the following sections, we describe the uuCloud Operation Registry (Sect. 3.1) and the Cloud Control Centre (Sect. 3.2). In Sect. 3.3 we describe the process of application deployment in the context of a multi-tenant cloud environment.

3.1 uuCloud Operation Registry

uuCloud Operation (uuOR) is a component of the uuCloud environment that maintains active information about uuCloud objects (i.e. tenants, resource pools, regions, resource groups, hosts, nodes, etc.). Table 1 contains a list of uuCloud objects and their brief descriptions, and Fig. 2. shows the attributes of uuCloud objects and relationships between these objects.

Table 1. uuCloud objects

Object	Description
Cloud	Public or private cloud platform (e.g. AWS or Plus4U)
Host	Physical or virtual server
Node	Unit of deployment - (a containerized application, e.g. Docker container) created on a Host within a given Resource Pool that has free capacity
Node Image	Binary image of the Node
Node Set	Set of Nodes with identical functionality, i.e. based on the same Node Image
Node size	Size of the Node, based on the number of CPUs, size of RAM, and size of ephemeral storage
Region	IT infrastructure with co-located compute, storage and network resources with low latency and high bandwidth connectivity
Resource	Service allocated to the application, e.g. MongoDB or a gateway
Resource Group	Logical grouping of Hosts in a given Region
Resource Lease	Mechanism for allocating resources to a resource pool. Resource Lease specifies the usage constraints, i.e. the contracted capacity of resources
Resource Pool	Resources assigned to a tenant, i.e. Nodes that belong to a single Resource Group
Runtime Stack	Archives and components needed to run the application (i.e. system tools, system libraries, etc.) that are used to create the Node Image
Tenant	A tenant is assigned Resources from a Resource Pool using the mechanism of Resource Lease; Tenant typically represents a separate organization

Each *Host* is allocated to a single logical unit called *Resource Group*. Resource groups can be allocated additional *Resources* (e.g. MongoDB [25], gateways, etc.); Resource attributes are not indicated on the diagram as different attributes apply to different types of resources. Each Resource Group belongs to a *Region* – a separate IT infrastructure from a single provider with co-located compute, storage and network resources with low latency and high bandwidth connectivity (e.g. Azure North

(EU-N-AZ)). Regions are grouped into a (hybrid) *Cloud* that can be composed of multiple public (e.g. AWS, Microsoft Azure) and private clouds (e.g. Plus4U).

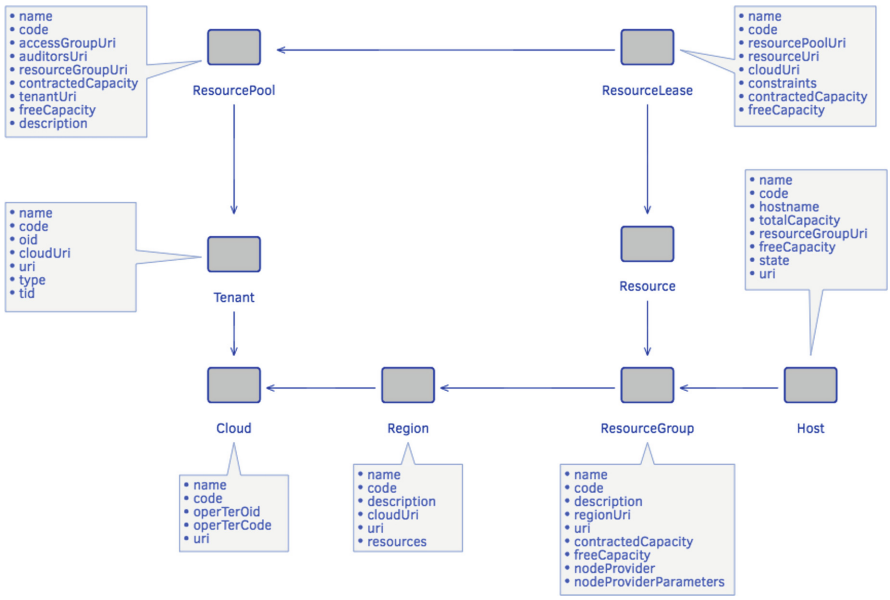


Fig. 2. Structure of the uuCloud Operation Registry

uuCloud supports multi-tenant operation; each *Tenant* typically represents a separate organization (e.g. a doctor's surgery). Tenants are assigned resources from R*esource Pools* using the mechanism of *Resource Lease* that specifies the usage constraints, such as the contracted capacity, free capacity, etc. A Resource Pool defines the maximum amount of resources (i.e. number of vCPUs, RAM size, size of storage, etc.) that are available for the operation of a specific Tenant. A Tenant can be allocated several Resource Pools, for example separate Resource Pool for production and development, preventing application development from consuming production resources. Each Tenant consumes its own resources and is monitored and billed separately. Applications can be shared among multiple Tenants using the *Share* command (see Table 2), with the Tenant in whose Resource Pool the application is deployed being responsible for the consumed resources.

3.2 Cloud Control Center

The Operation Registry described in the previous section stores and manages uuCloud metadata and is used by the Cloud Control Center (uuC3) to automate the management and operation of container-based microservices deployed on public or private cloud infrastructure. The Operation Registry database is managed using REST commands

that include commands for creating and deleting repository objects and commands for generating various repository reports. uuC3 defines an API that abstracts the complexity of the proprietary cloud infrastructures and supports node deployment and host management operations listed in Table 2.

Table 2. uuC3 REST API

Operation	Description
Deploy	Deploys a specified application (uuSubApp) to a specified Resource Pool
Redeploy	Redeploys a specified application (uuSubApp) to a specified Resource Pool
Undeploy	Undeploys a specified application (uuSubApp) from the ResourcePool, deletes its Nodes, and releases all associated resources
Reinit	Reinitializes the specified Host to a consistent state as per uuOR
Scale	Increases or decreases the number of deployed Nodes
Share	Shares the specified application (uuSubApp) with specified tenants
Unshare	Terminates sharing of the specified application (uuSubApp) with the specified tenant
getAppDeploymentList	Returns a list of applications (uuSubApps) and their Nodes deployed to a specified Resource Pool
getAttributes	Returns the attributes for a specified application (uuSubApps) deployment
getResourcePoolCapacity	Returns information about Resource Pool Capacity
getStatus	Returns the status of a specified host
getNodeImageList	Returns the list of node images

3.3 Application Deployment Process

The first step in the application deployment process is to create a *Node Image*. As illustrated in Fig. 3, Node Image is created from a uuSubApp and a *Runtime Stack* that contains all the related archives and components needed to run the application (i.e. system tools, system libraries, etc.). During deployment, the uuC3 application deployment command (Deploy) reads the corresponding Deployment Descriptor (JSON file that contains parameter that define the application runtime environment). The resulting Node Image constitutes a unit of deployment. Node Image metadata are recorded in the uuOR and the Node Image is published to the private Docker Image Registry.

The JSON code fragment in Fig. 4 shows the structure of the deployment descriptor, and the Table 3 below contains the description of the Deployment Descriptor attributes.

Figure 5 shows the relationship between objects used during the process of deployment of containerized applications. Node (i.e. Docker container) is a unit of deployment with characteristics that include virtual CPU (vCPU) count, RAM size and

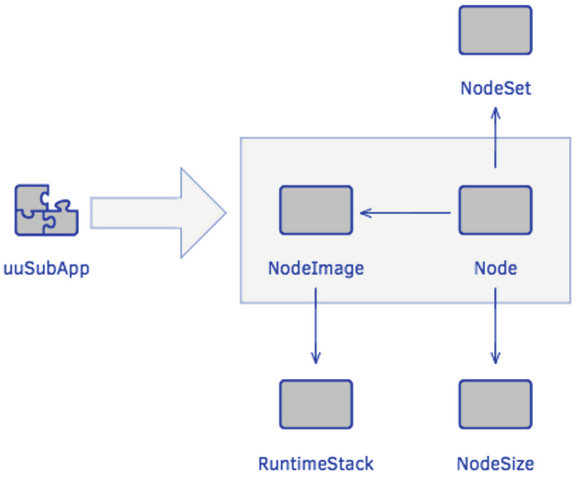


Fig. 3. Process of creating a Node Image

```
{
  "schemaUri": "ues:SYSTEM:UU.OS/UUAPP-DEPLOY-SCHEMA-
V2.1",
  "data": {
    "code": "<vendor>.<app>.<subApp>",
    "name": "<app name>",
    "version": "<app version>",
    "multitenant": true,
    "packs": [{
      "code": "<vendor>.<app>.<subApp>/SERVER-PACK",
      "targetNodeSets": [
      "<vendor>.<app>.<subApp>/CMD-CONTAINER"]}],
    "nodeSets": [{
      "code": "<vendor>.<app>.<subApp>/CMD-CONTAINER",
      "nodeSize": "G1_M",
      "runtimeStack": "UU.OS9/UC_RUBY_STACK-V1.0"}]}
}
```

Fig. 4. Deployment descriptor

the size of ephemeral storage. A Node can be deployed to an available *Host* (physical or virtual server with computational resources, i.e. CPU, RAM, and storage). Nodes are classified according to *NodeSize* (Small: 1x vCPU, 1 GB of RAM, 0.5 GB of disk storage, Medium: 1x vCPU, 1 GB of RAM, 1 GB of disk storage, and Large: 2x vCPU, 2 GB of RAM, 1 GB of disk storage). uuCloud Nodes are stateless and use external resources (e.g. MongoDB) to implement persistent storage. Nodes are further classified as synchronous or asynchronous depending on the behavior of the application that the node virtualizes. Nodes are grouped into *NodeSets* - sets of Nodes with

Table 3. Deployment descriptor attributes

Attribute	Description
code	Unique identifier of the sub-application to be deployed to the Node
name	Application name
version	Application version
multitenant	Determines if application can be shared across different tenants
packs	List of packages that make up the application
packs:code	Package code
packs:targetNodeSets	List of Node Sets where the applications is to be deployed
nodeSets	Defines the requirements for individual Node Sets
nodeSets:code	Unique identifier of the Node Set
nodeSets.nodeSize	Defines the minimum size of the Node where the application can be deployed (e.g. G1_M)
nodeSets. runtimeStack	Runtime environment that is deployed to the Node

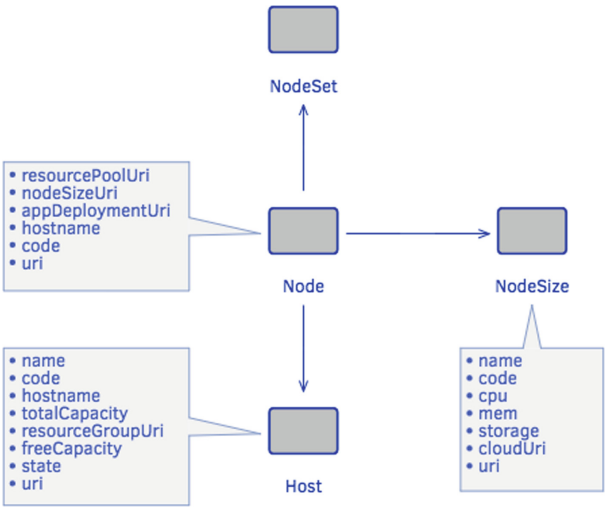


Fig. 5. Application deployment scenario

identical functionality (i.e. nodes that virtualize the same applications). NodeSets support elasticity by increasing or decreasing the number of available nodes. At runtime, a gateway (uuGateway) forwards client requests to a router that passes each request to a load balancer. The load balancer selects a Node from a NodeSet of functionally identical nodes, optimizing the use of the hardware infrastructure and providing a failover capability (if the Node is not responsive the request is re-directed to an alternative Node).

The deployment of the application is performed using the uuC3 Deploy command. The Deploy command searches for an existing application deployment object in

uuOperation Registry, which can be identified by a composite key (ResourcePool URI, ASID, uuAppBox URI). If the deployment already exists, it will be updated, else a new application deployment is created for each instance of a particular application (uuSubApp).

```
UU::C3::AppDeployment.deploy
('<DEPLOY_POOL_URI>',
  appBoxUri: '<APPBOX_URI>',
  asid: '<ASID>',
  uuEEs: ['<SYS_OWNER_UID>'],
  config: {
    tid: '<TID>',
    asid: '<ASID>',
    uuSubAppDataStoreMap: {
      primary: '<OSID>'
    },
    privilegedUserMap: {
      uuSubAppInstanceSysOwner: '<SYS_OWNER_UID>'
    }}
)
```

Fig. 6. uuC3 Deploy command

The input parameters of uuC3 Deploy command in the code fragment illustrated in Fig. 6 include: *DEPLOY_POOL_URI* (URI of the Resource Pool that the application is to be deployed into), *appBoxUri* (URI of the application distribution package), *ASID* (Application Server Identifier) – an application identifier generated by uuCloud to uniquely identify each sub-application, *uuEEs* (list of identities that the application uses), *TID* (Tenant Identifier of the Tenant whose resources the application uses), *uuSubAppDataStoreMap* (mapping between the logical SubApplication Data Store and the physical Data Store identifier values), *privilegedUserMap* (mapping between logical identity names and the actual user identities that are associated with access rights), *uuSubAppInstanceSysOwner* (identity of the system user who is authorized to deploy the application).

4 Conclusions

Wide-spread adoption of cloud computing and extensive use of mobile and IoT devices have impacted on the architecture of enterprise applications with corresponding impact on application development frameworks [26]. Microservices architecture has evolved from SOA (Service Oriented Architecture) and is currently the most popular approach to implementing cloud-based applications. However, the management of large-scale container-based microservices requires automation to ensure fast and predictable application deployment, auto-scaling and control of resource usage. Platforms and

frameworks that support the development and deployment of container-based applications have become an active area of research and development, and a number of open source projects have been recently initiated. Kubernetes project, in particular, appears to be gaining acceptance by major public cloud providers including Google, Microsoft and most recently Amazon AWS, as well as other important industry players. However, currently there is a lack of agreement about a standard application development framework designed specifically for cloud development and deployment. This situation is further complicated by the high rate of innovation in this area. It is quite likely that the present approach of implementing applications as containerized microservices will evolve to take advantage of much finer grained cloud resources with dramatic impact on the design, implementation and operation of cloud-based applications.

In this paper, we have described the uuCloud framework that is designed to facilitate the management of large-scale container-based microservices with specific extensions to handle multi-tenancy. An important benefit of the uuCloud framework is its ability to manage complex multi-tenant environments deployed across multiple (hybrid) cloud platforms, hiding the heterogeneity of the underlying infrastructures (i.e. hardware, operating systems, virtualization technologies, etc.). uuCloud manages cloud metadata, supports automatic application deployment and autonomic scaling of applications.

We are continuously monitoring the rapidly evolving landscape of cloud management platforms and frameworks, and we may consider alignment of the uuCloud framework with Kubernetes in the future as both frameworks mature and the direction of cloud standardization becomes clearer.

References

1. Forester: The Public Cloud Services Market Will Grow Rapidly to $236 Billion in 2020 (2018). https://www.forrester.com/. Accessed 10 Feb 2018
2. AWS: Amazon Web Services (AWS) - Cloud Computing Services (2018). https://aws.amazon.com/. Accessed 2 Feb 2018
3. Microsoft: Windows Azure Platform. http://www.microsoft.com/windowsazure/. Accessed 10 Feb 2018
4. Burns, B., et al.: Borg, omega, and kubernetes. Queue **14**(1), 10 (2016)
5. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Softw. **33**(3), 42–52 (2016)
6. CloudFoundry: Cloud Application Platform - Devops Platform—Cloud Foundry (2017). https://www.cloudfoundry.org/. Accessed 28 Sept 2017
7. OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes (2017). https://www.openshift.com/. Accessed 28 Sept 2017
8. OpenStack: OpenStack Open Source Cloud Computing Software (2018). https://www.openstack.org/. Accessed 2 Feb 2018
9. Kubernetes (2017). https://kubernetes.io/. Accessed 25 Aug 2017
10. Beránek, M., Feuerlicht, G., Kovář, V.: Developing enterprise applications for cloud: the unicorn application framework. In: International Conference on Grid, Cloud and Cluster Computing, GCC 2017, CSREA Press, Las Vegas, USA (2017)

11. Pahl, C., et al.: Cloud container technologies: a state-of-the-art review. IEEE Trans. Cloud Comput. 1 (2017). https://doi.org/10.1109/TCC.2017.2702586
12. Pahl, C.: Containerization and the PAAS cloud. IEEE Cloud Comput. **2**(3), 24–31 (2015)
13. Brewer, E.A.: Kubernetes and the path to cloud native. In: Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM (2015)
14. Burns, B., et al.: Borg, omega, and kubernetes. Queue **14**(1), 70–93 (2016)
15. Schwarzkopf, M., et al.: Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems. ACM (2013)
16. Home - Cloud Native Computing Foundation (2017). https://www.cncf.io/. Accessed 27 Sept 2017
17. Truyen, E., et al.: Towards a container-based architecture for multi-tenant SaaS applications. In: Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware 2016. ACM, Trento, Italy, pp. 1–6 (2016)
18. AlJahdali, H., et al.: Multi-tenancy in cloud computing. In: 2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE). IEEE (2014)
19. Kumar, R., et al.: Open source solution for cloud computing platform using OpenStack. Int. J. Comput. Sci. Mob. Comput. **3**(5), 89–98 (2014)
20. Feller, E., Rilling, L., Morin, C.: Snooze: a scalable and autonomic virtual machine management framework for private clouds. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society (2012)
21. Li, L., Tang, T., Chou, W.: A rest service framework for fine-grained resource management in container-based cloud. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD). IEEE (2015)
22. Agmon Ben-Yehuda, O., et al.: The rise of RaaS: the resource-as-a-service cloud. Commun. ACM **57**(7), 76–84 (2014)
23. Al-Dhuraibi, Y., et al.: Autonomic vertical elasticity of docker containers with elasticdocker. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). IEEE (2017)
24. Docker: What is Docker, 14 May 2015. https://www.docker.com/what-docker. Accessed 21 Aug 2017
25. MongoDB for GIANT Ideas (2017). https://www.mongodb.com/index. Accessed 21 Aug 2017
26. Raj, P., Venkatesh, V., Amirtharajan, R.: Envisioning the cloud-induced transformations in the software engineering discipline. In: Mahmood, Z., Saeed, S. (eds.) Software Engineering Frameworks for the Cloud Computing Paradigm, pp. 25–53. Springer, London (2013). https://doi.org/10.1007/978-1-4471-5031-2_2

Fault Tolerant VM Consolidation for Energy-Efficient Cloud Environments

Cihan Secinti(✉) and Tolga Ovatman

Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey
{secintic,ovatman}@itu.edu.tr

Abstract. Cloud computing applications are expected to guarantee the service performance which is determined by the Service Level Agreement (SLA) between cloud owner and client. While satisfying SLA, Virtual Machines (VMs) may be consolidated based on their utilization in order to balance the load or optimize energy efficiency of physical resources. Since, these physical resources are prone to system failure, any optimization approach should consider the probability of a failure on a physical resource and orchestrate the VMs over the physical plane accordingly. Otherwise, it is possible to experience unfavorable results such as numerous redundant application migrations and consolidations of VMs that may easily cause in SLA violations. In this paper, a novel approach is proposed to reduce energy consumption and number of application migration without violating SLA while considering the fault tolerance of the cloud system in the face of physical resource failures. Simulation results show that proposed model reduce energy consumption by approximately 37% and number of application migration by approximately 9%. Besides, in case of faults the increase of energy consumption is less than 11% when the proposed approach is used.

Keywords: VM consolidation · Fault tolerance · Cloud computing
Green cloud · Live-migration · Application migration

1 Introduction

Cloud computing has many trending features such as on-demand services, virtualization and automatic scaling which leads remarkable increase in cloud usage. Data centers that host major cloud services contain enormous servers which cause tremendously high energy consumption levels [1]. Along with the increase in the number of cloud users and powerful data centers, energy consumption of cloud systems gained significant importance [2,3] over the last decade.

Growing importance of energy efficiency in cloud systems caused recent academic studies to concentrate on energy efficiency on cloud systems as well. In this manner, virtual machine (VM) consolidation is a widely used method to promote more efficient energy consumption and utilization. On the other hand,

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 323–333, 2018.
https://doi.org/10.1007/978-3-319-94295-7_22

in cloud computing, service performance is leveraged mostly with automatic scaling that may frequently led over-provisioned infrastructure and high energy consumption.

Automatic scaling can be achieved in terms of vertical scaling and horizontal scaling. In this paper, horizontal scaling is focused where VM creation and VM consolidation policies becomes important aspects of energy consumption trade-offs.

VM consolidation can be used as a means to reduce energy consumption and optimize the VM utilization in horizontal scaling process. Consolidating applications in multiple VMs into a single VM may cause high utilization and reduced fault tolerance. Fault tolerance can be achieved either using a reactive approach or a proactive approach. Reactive fault tolerance policies aim to reduce the damage of failures after the failure whereas, proactive fault tolerance policies try to predict the faults and reacts before the failure [4].

In modern cloud systems, three aspects; fault tolerance, energy efficiency and performance requirements form a trade-off. High performance and fault tolerant cloud environments require horizontal scaling, and energy efficient cloud systems need VM consolidation for preventing unnecessary energy consumption. An approach is necessary to keep fault tolerance, energy efficiency and performance requirements in an optimized balance.

In this paper, a reactive fault tolerance approach is proposed for cloud environments which try to minimize the energy consumption by using VM consolidation. There are three different cases in the study; no-consolidation, consolidation by threshold and consolidation according to fault tolerant consolidation algorithm (FTC). No-consolidation case is presented as the control experiment for the proposed approaches; when a new VM is created it will not be consolidated even if it runs no applications. In the second case, consolidation by threshold, VMs would be consolidated when their utilization is less then defined threshold value. In the last case, FTC algorithm is used to accept or reject consolidation by threshold requests using the measure defined latter in the paper.

Major contributions of this paper are:

1. A novel measure to evaluate fault tolerance vulnerabilities.
2. An algorithm (FTC) to orchestrate energy efficiency and fault tolerance together.
3. Simulating application placement, VM consolidation and fault tolerance together to analyze trade-offs among them.
4. Reduced number of application migration and energy consumption in horizontal scaling.

The rest of this paper is organized as follows. The related work in this research field is presented in Sect. 2. Then, the simulation system features and structural details are described in Sect. 3. In Sect. 4, experiment details and results are presented, finally, followed by conclusion and planned future work in the last section.

2 Related Work

VM consolidation has become a significant technique for data center energy and resource management. For consolidation of VM several methods are carried on already such as Ant Colony Optimization, K-Nearest Neighbor, Greedy Heuristics [5–7]. On the other hand, VM consolidation requires live migration of the applications which are hosted by the corresponding VM [8]. Besides in [9], authors analyzed CO_2 emissions of the data centers by considering their energy consumption in network level. In [3], the authors are proposed a model which migrates highly utilized virtual machines to the low utilized hosts while keeping the energy efficiency of the data center by realizing firefly algorithm. However, these techniques do not consider the fault toleration of the VMs.

In the real world, cloud systems could encounter system failures by their dynamic nature [10] which must be prevented to achieve guaranteed Quality of Service (QoS). Hence, various fault tolerant cloud environment approaches are investigated. In [11], fault occurrences are more similar to Weibull Distribution (WD).

The authors proposed an active replication model to provide fault tolerance in [12]. Then, in [13], the replication approach is further enhanced by using byzantine fault tolerance gain to optimize the overall cost. However, all of these studies require user experience and knowledge for configuration and application preparation phase. Since most of the approaches utilize VM level live-migration among data centers to create fault tolerant cloud environment, the techniques represented in [14] distinguishes these approaches by using application level live-migration.

In [15], the authors emphases the gap of fault tolerant, energy efficient cloud environment approaches by using VM consolidation.

3 Simulation System Structure of the Proposed Approach

Simulation software[1] used in this study is developed in Java 8 and tested on PC which has Intel Core i7 7700HQ @ 2.80 GHz, 16 GB DDR4-2400, 4 GB NVidia GTX1050, on a Windows 10 64-bit environment. Typically, a simulation run takes 15 s. The application creation time and resource usage data are generated randomly by using uniform distribution. The simulation framework works for only one physical machine and all the VMs are created and consolidated on the same physical machine.

The main structure of the simulation is shown in Fig. 1. The simulation consists of two main phases; application assignment of VM and VM consolidation on physical machine. The application placement [16] is a divergent and huge area for covering in this paper hence Round Robin algorithm is used to ease implementation of this part. So, it is same for all three test cases (no-consolidation, consolidation by threshold, consolidation by FTC). The second phase of the simulation is the main focus of enhancement.

[1] https://github.com/secintic/CloudSimulation.git.

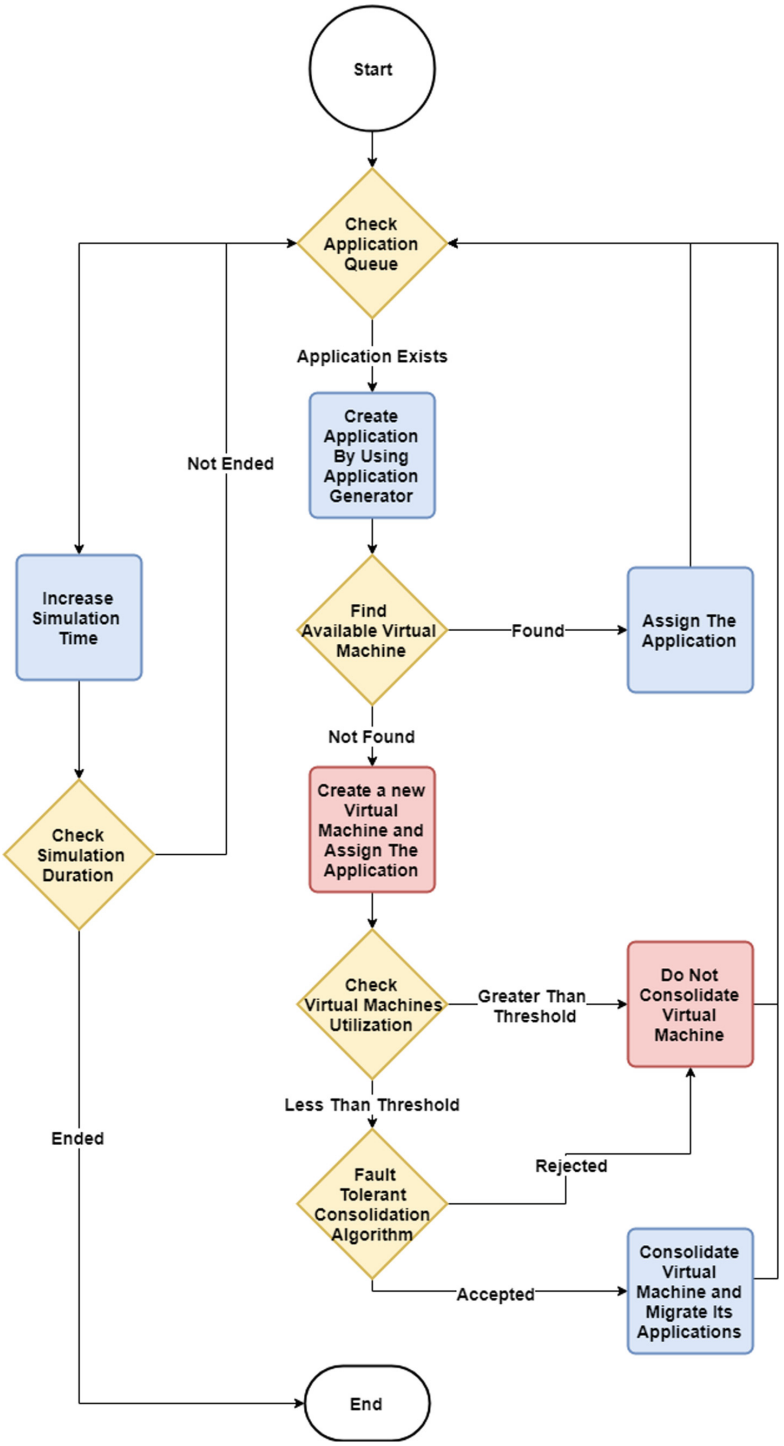


Fig. 1. Simulation flow chart

In the first phase, basically, the simulation engine checks the simulation end period of time and continue if it has more period to work. Then generate certain number of applications according to Gaussian Distribution and search for available VM if it is able to find the VM then assign corresponding application to VM and return to the simulation counter. If it cannot find any available VM for assignment, then it will create a new VM and assign application to currently created VM.

In the second phase, after all application assignments are completed on corresponding period, simulation framework checks the VMs utilization if their utilization is less then defined threshold. Depending on that, it will consolidate VM on consolidation by threshold case. But in consolidation by FTC case it sends a consolidation request to FTC algorithm and the algorithm determines the request is accepted or rejected. On no consolidation test case this step will be skipped because in that scenario there is no consolidation checks.

3.1 Fault Tolerant Consolidation Algorithm

The FTC Algorithm works on every period of simulation lifetime. As shown in Algorithm 1, it checks every VMs running on the physical machine and compare their utilization with the defined threshold value.

Algorithm 1: Fault Tolerant Consolidation Algorithm

```
1  n: number of virtual machines V: list of virtual machines
2  for i ← 1 to n do
3      if util(V_i) < threshold then
4          V_s ←findByPolicy(util(V));
5          if isMigratable(V_s) then
6              consolidate(V_i);
7          end
8      else
9          continue;
10     end
11 end
```

After the threshold control on *line 3*, *findByPolicy* function works to find guaranteed VM according to its policy value which can be defined as minimum utilized, median utilized or maximum utilized. The effects of these policies are also demonstrated on the Sect. 4. In *line 5*, *isMigratable* function checks all VMs except VM which has less utilization than threshold and is selected by previous line to guarantee all applications of the selected VM could be migrated to the rest of VMs.

In consolidate function of the algorithm, the applications which are hosted by the VM should be migrated to other VMs before consolidation process of the VM. However, live-migration of applications causes overhead on resources

consumption of the VMs which may easily result in higher energy consumption by 10% [17]. Therefore, the key factor of the consolidation is to reduce number of application migration to reduce energy consumption and it is not significant to migration application to which VM.

In general, the main purpose of the algorithm is to prevent new VM creation when a failure occurrence on any of the VMs. Therefore, it determines consolidation request approval by checking the other VMs are capable of to manage all its applications. If they are capable then the algorithm will accept the consolidation request and VM will be consolidated. Otherwise, the request will be rejected and consolidation would not be happened.

Energy consumption of VM_i is calculated in Eq. 1.

$$E_i = k + \sum_{j=1}^{m} cpu(a_j) + memory(a_j) \tag{1}$$

where variable k represents idle energy consumption, a denotes for application and m is the total number of applications hosted by VM_i. *cpu* and *memory* functions return the memory and CPU consumption of a_j. The equation sums the idle energy consumption and all utilized resources (CPU and Memory in case of suggested model) which is used by applications.

In Eq. 2, total energy consumption is calculated by using VM energy consumption and application migration' energy overhead.

$$TE = \sum_{i=1}^{n} (1 + \frac{s(p)}{10}) \times E_i \tag{2}$$

where variable *s(p)* represents the number of the application migration among the VMs on period p.

4 Simulation Results

In the simulation, energy consumption and number of the application migrations are compared for three different cases (no-consolidation, consolidation by threshold and FTC) which are mentioned in Sect. 1.

Number of generated applications are shown in Fig. 2. To test model on a fluctuating application generated environment several Gaussian Distributions are overlapped independently from fault occurrence distribution.

In the first experiment, energy consumption on various threshold values are checked for all cases. The threshold values are defined as utilization percentage of the VM and shown in Table 1. In the other experiments, number of application migrations and energy consumption changes on simulation time are analyzed by using a constant 50% threshold value.

Defined simulation variables are shown in Table 1. For resources types CPU and Memory are used and application resources consumption are selected from uniform distribution on the interval of 0.1 and 0.3. The variables are defined

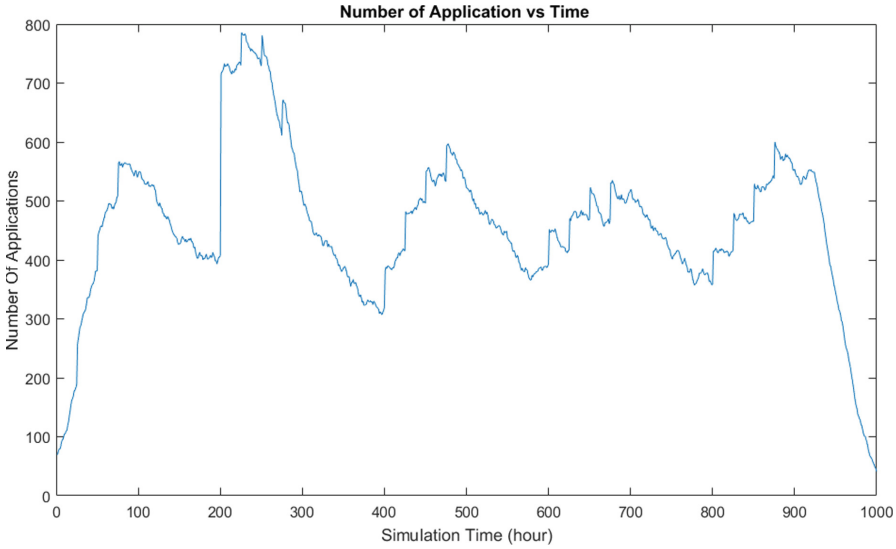


Fig. 2. Application generation distribution

Table 1. Simulation variables

Variables	Values
Resources	{CPU, Memory}
VM Capacities	{CPU: 20, Memory: 20}
Application resource consumption	{CPU: 0.1 – 0.3, Memory: 0.1 – 0.3}
Fault Occurrence times	WD random points period: {49, 72, 88, 137, 157, 337, 364, 475, 733, 891}
Simulation duration	1000 periods
Threshold (%)	{30, 40, 50, 60, 70}

according to Google Cluster Data [18,19]. The VM capacities defined as 20 core CPU and normalized 20 units of memory. Faults occurrences are gathered from Weibull Distribution and it happens 10 times in a simulation lifetime. The simulation runs for 1000 period to collect results.

In Fig. 3, total energy consumption of the simulation lifetime has calculated by using Eq. 2 for different threshold values. In No-consolidation case, model do not consolidate any VM and if it is necessary it creates new one so energy consumption values are higher than other two case. In normal conditions, the threshold value changes would not affect no-consolidation energy consumption but in this simulation model, faulty VM is selected randomly so it can be increase when occurred on VMs with high utilization or vice-versa. In Consolidation case, keeping the threshold value low gives the advantage of reduced energy consumption. Hence, consolidation of VMs with high utilization causes costly

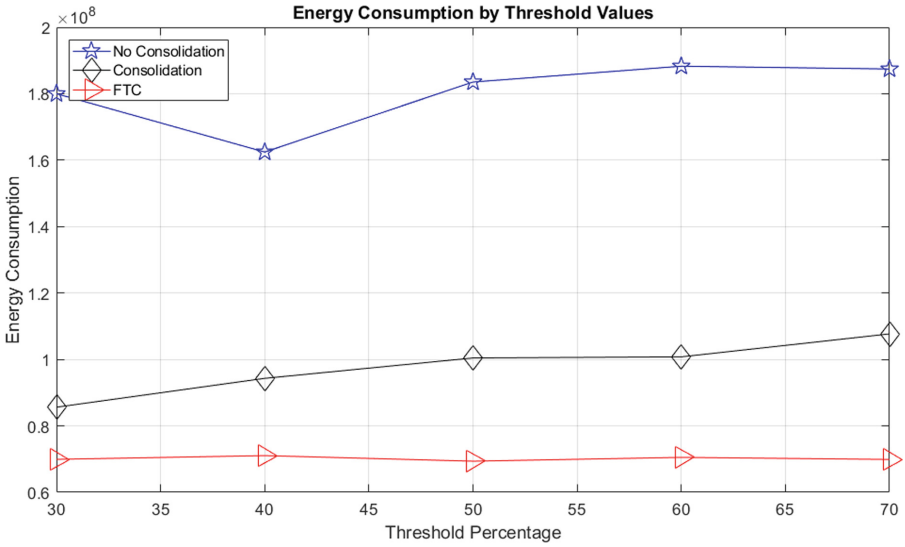


Fig. 3. Energy consumption comparison for different threshold values

application migrations because of having more number of applications also fault occurrences affect Consolidation case worse than the other cases. The FTC model is not affected from threshold value changes by the advantage of its rejection capability. But its slightly more robust on higher threshold values. Instead of consolidating low threshold VMs by considering fault possibility, consolidation of VMs with high threshold results the better energy efficiency. In results, the FTC approach has reduced energy consumption by 30% to 50%.

In Fig. 4, number of application migration and energy consumption of whole system are compared for all three test cases. the FTC algorithm has the minimum number of application migration and also 50% less energy consumption according to consolidation by threshold case. When an error occurred in all cases, applications are migrated from faulty VM to other VMs which causes to increase of energy consumption on fault occurrence times of the simulation. The FTC model degrades impacts of fault occurrence and prevents the unnecessary application migrations for fault tolerance so it keeps energy consumption more stabilized.

In Fig. 5, three different guaranteed VM finder policies are compared; maximum, minimum and median utilized as mentioned in Sect. 3. The results show that if it is selected as minimum it will converge to the consolidation case, otherwise it is selected as two times maximum it will converge to the no-consolidation case. The best results are found when the maximum utilized VM is selected and other VMs are checked by hosting the maximum utilized VM applications.

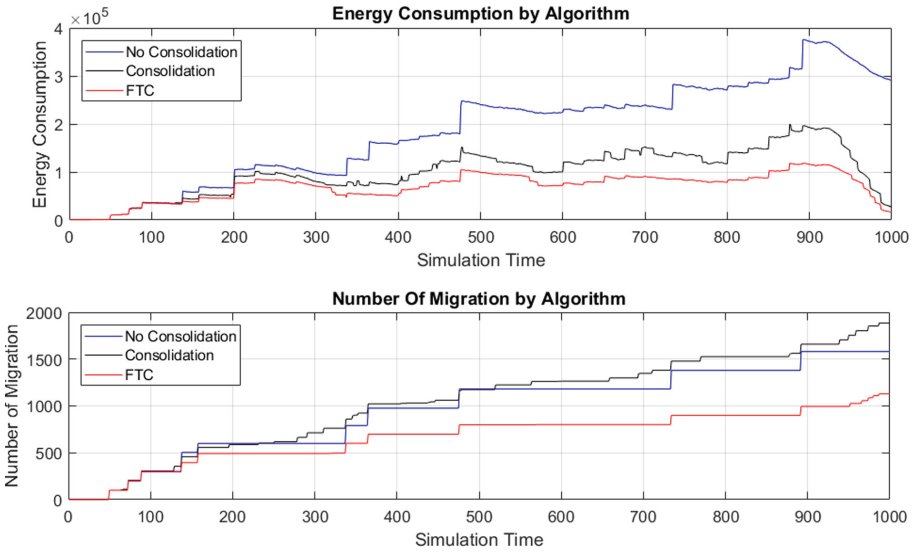


Fig. 4. Threshold value is selected as 50%

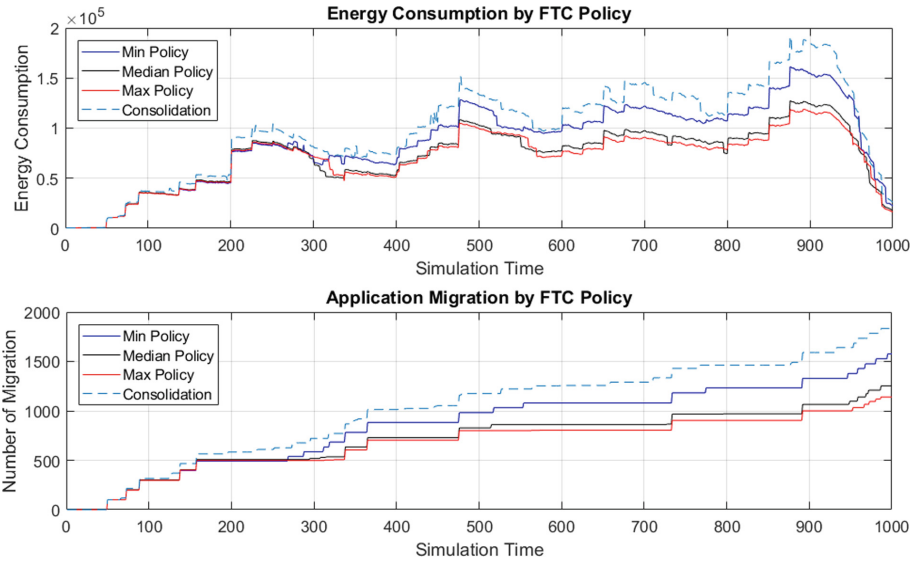


Fig. 5. Max, Min, Median FTC Algorithm policies on 50% threshold value

5 Conclusion and Future Works

In this paper, a novel approach is proposed to satisfy fault tolerance and energy efficiency together. Hence, two different experiment are created to prove FTC model and its advantages. Energy consumption and number of application migra-

tion results are compared under various threshold values. The experiments show that with the FTC approach the energy consumption ca be reduced by 30% to 50% and number of application migrations can be reduced by %10 in a faulty cloud environment. The proposed approach also reduces the impact of faults on VMs.

In the future, the proposed approach is planned to be validated using different real-world data sets. An additional contribution to proposed approach can be to adapt machine learning techniques in order to perform time series analysis and smart prediction of future fluctuations on VM resource demands.

References

1. Ghribi, C., Hadji, M., Zeghlache, D.: Energy efficient VM scheduling for cloud data centers: exact allocation and migration algorithms. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp. 671–678, May 2013. https://doi.org/10.1109/CCGrid.2013.89
2. Rodero, I., et al.: Energy-efficient thermal-aware autonomic management of virtualized HPC cloud infrastructure. J. Grid Comput. **10**(3), 447–473 (2012). https://doi.org/10.1007/s10723-012-9219-2. ISSN 1572–9184
3. Kansal, N.J., Chana, I.: Energy-aware virtual machine migration for cloud computing - a firey optimization approach. J. Grid Comput. **14**(2), 327–345 (2016). https://doi.org/10.1007/s10723-016-9364-0. ISSN 1572–9184
4. Bala, A., Chana, I.: Fault tolerance-challenges, techniques and implementation in cloud computing. **9**, January 2012
5. Farahnakian, F., et al.: Using Ant colony system to consolidate VMs for green cloud computing. IEEE Trans. Serv. Comput. **8**(2), 187–198 (2015). https://doi.org/10.1109/TSC.2014.2382555. ISSN 1939–1374
6. Farahnakian, F., et al.: Utilization prediction aware VM consolidation approach for green cloud computing. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 381–388, June 2015. https://doi.org/10.1109/CLOUD.2015.58
7. Ferdaus, M.H., Murshed, M., Calheiros, R.N., Buyya, R.: Virtual machine consolidation in cloud data centers using ACO metaheuristic. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 306–317. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09873-9_26
8. Ferdaus, M.H., et al.: Multi-objective, Decentralized Dynamic Virtual Machine Consolidation using ACO Metaheuristic in Computing Clouds. In: CoRR abs/1706.06646 (2017). arXiv: 1706.06646
9. Cavdar, D., Alagoz, F.: A survey of research on greening data centers. In: 2012 IEEE Global Communications Conference (GLOBECOM), pp. 3237–3242, December 2012. https://doi.org/10.1109/GLOCOM.2012.6503613
10. Amin, Z., Singh, H., Sethi, N.: Review on Fault tolerance techniques in cloud computing. Int. J. Comput. Appl. **116**(18), 11–17 (2015)
11. Liu, J., et al.: A Weibull distribution accrual failure detector for cloud computing. PLOS ONE **12**(3), 1–16 (2017). https://doi.org/10.1371/journal.pone.0173666
12. Santos, G.T., Lung, L.C., Montez, C.: FTWeb: a fault tolerant infrastructure for web services. In: Ninth IEEE International EDOC Enterprise Computing Conference (EDOC 2005), pp. 95–105, September 2005. https://doi.org/10.1109/EDOC.2005.15

13. Wood, T., et al.: ZZ and the art of practical BFT execution. In: Proceedings of the Sixth Conference on Computer Systems, EuroSys 2011, pp. 123–138. ACM, Salzburg (2011). https://doi.org/10.1145/1966445.1966457
14. Cully, B., et al.: Remus: high availability via asynchronous virtual machine replication. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2008, pp. 161–174. USENIX Association, San Francisco (2008). http://dl.acm.org/citation.cfm?id=1387589.1387601
15. More, N.S., Ingle, R.B.: Challenges in green computing for energy saving techniques. In: 2017 International Conference on Emerging Trends Innovation in ICT (ICEI), pp. 73–76, February 2017. https://doi.org/10.1109/ETIICT.2017.7977013
16. Seçinti, C., Ovatman, T.: On optimizing resource allocation and application placement costs in cloud systems. In: Proceedings of the 4th International Conference on Cloud Computing and Services Science, CLOSER 2014, pp. 535–542. SCITEPRESS - Science and Technology Publications, LDA, Barcelona, (2014). https://doi.org/10.5220/0004849605350542. ISBN: 978-989-758-019-2
17. Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R.: Cost of virtual machine live migration in clouds: a performance evaluation. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 254–265. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10665-1_23
18. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, November 2011. Version 2.1. https://github.com/google/cluster-data. Accessed 17 Nov 2014
19. Wilkes, J.: More Google cluster data. Google research blog. http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html, November 2011

Over-Sampling Algorithm Based on VAE in Imbalanced Classification

Chunkai Zhang[1(✉)], Ying Zhou[1], Yingyang Chen[1], Yepeng Deng[1], Xuan Wang[1], Lifeng Dong[2], and Haoyu Wei[3]

[1] Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
ckzhang812@gmail.com
[2] Hamline University, 1536 Hewitt Avenue, St. Paul, USA
[3] Sichuan University, Chengdu, Sichuan, China

Abstract. The imbalanced classification problem is a problem that violates the assumption of uniform distribution of samples, classes differ in sample size, sample distribution and misclassification cost. The traditional classifiers tend to ignore the important minority samples because of their rarity. Oversampling, the algorithm uses various methods to increase the minority samples in the training set to increase the recognition rate of them. However, these over-sampling methods are too coarse to improve the classification effect of the minority samples, because they can't make full use of the information in the original samples, but increase the training time because of adding extra samples. In this paper, we propose to use the distribution information of the minority samples, use the variational auto-encoder to fit the probability distribution function of them without any prior assumption, and reasonably expand the minority class sample set. The experimental results prove the effectiveness of the proposed algorithm.

Keywords: Imbalanced classification · Variational auto-encoder Oversampling

1 Introduction

The classification problem is a very important part of machine learning, and it is also the first step for artificial intelligence to understand human life. At present, most classifiers assume that the samples of different classes are evenly distributed, and the classification costs are the same. However, in reality, the data people are more concerned about is often scarce, such as the detection of credit card fraud and medical disease diagnosis. In the medical disease diagnosis, most of the results are normal while only a small proportion of the results are diagnosed as diseases, which indicates the different distribution in different classes samples. Second, if healthy people are misdiagnosed as diseases, they can be removed by other inspection methods, errors do not cause very serious accidents, but if the disease people are diagnosed as healthy, it may cause the patients to miss the best treatment time and cause serious consequences. This is the second feature of the imbalanced classification problems: different classes of misclassification costs are inconsistent. At the same time, if samples are classified as

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 334–344, 2018.
https://doi.org/10.1007/978-3-319-94295-7_23

diseases as much as possible because they are afraid to miss the disease samples, it will cause a huge waste of medical resources and intensify conflicts between doctors and patients. Therefore, it is not feasible to determine all samples as disease, and the best way is to try to separate these two results as correct as possible. Due to the scarcity of the minority samples and the definition of global accuracy, the classifier pays less attention to the minority class, so the recognition performance is unsatisfying. Imbalanced classification problems arise in many fields, such as bioinformatics [1, 2], remote sensing image recognition [3], and privacy protection in cybersecurity [4–6]. The imbalanced problems cover widely and have a very important practical significance.

The traditional solutions to the imbalanced problems are divided into two parts: the algorithm-level methods and the data-level methods. The algorithm-level methods mainly focus on the different misclassification costs, such as improved neural network [7]: it uses the approximation of F1 value of the minority class as the cost function; the bagging algorithm [8] continues to enhance the misclassified the minority samples, and improve the recognition rate of the minority samples; structured SVM [9] uses the F1 value of the minority samples as the optimization function, and has a better performance in the classification of the minority samples.

The data-level methods focus on the imbalance of sample size, which mainly adjust the data sample size through resampling to reduce the impact on classification performance. The data-level methods can be divided into over-sampling, under-sampling and hybrid sampling. Over-sampling adds the minority samples in the training process, Over-sampling can effectively improve the classification performance of the minority class but it has no idea of the rationality. Under-sampling [10] removes the majority samples before training, which can quickly reach equilibrium, but may take a risk of losing valuable samples.

The oversampling method can be divided into random sampling and informed sampling. Random sampling means repeating the known samples, which includes simple repetition [11], linear interpolation [12], nonlinear interpolation [13], etc.; SMOTE [12], as a classic over-sampling algorithm, interpolates linearly in the minority samples, will increase the amount of information and rationality of synthesized samples in random oversampling, which improves the classification effect. Border-line-smote [14], to reduce the risk of overfitting, it selects the minority samples needing to be interpolated called boundary samples. The above oversampling methods only consider the influence of the sample size and the local sample distribution on the classification performance, ignoring the overall distribution of the sample, which is more informative for classification performance.

Informed sampling [15] uses the distribution information in the sample to fit its probability distribution function (PDF) and sample it according to the PDF. Chen [16] proposed a normal distribution based oversampling approach, and this approach assumes the minority class distribution as the Gaussian normal distribution, the parameters are calculated from the minority samples with EM algorithm, the experimental results are better than SMOTE and random oversampling. Different scholars have proposed oversampling algorithms based on various distributions, such as the Gaussian distribution [16, 17], Weibull distribution [18], etc. Due to the distribution information, these algorithms have made greater progress than random oversampling method. However, the problems are also obvious: there is a prior assumption about the

real distribution and all the features are dependent from each other. If the real distribution meets this hypothesis, it will get better results, otherwise, the improvement is limited, so it is inconsistent in their effect on different datasets.

Data level methods are of great matter in imbalanced classification, as it can be regarded as a step in data preprocessing, it will have a positive effect on the final classification results. Since the factors that affect the datasets classification include not only the sample size, but also the sample distribution, while the current over-sampling methods do not make full use of distribution information and cannot guarantee the rationality of the generated samples.

In this paper, we propose a oversampling method based on the variational auto-encoder [19] (VAE) model to generate the minority samples. The proposed method is motivated that the distribution information plays an important role in oversampling methods, and aims at the rationality of the generated samples, we use VAE to increase minority instances, to our knowledge, first, the output dimension of the neural network is not limited so it can generate data of any dimension; second, the strong fitting ability of the neural network can simulate any distribution function without any prior knowledge in advance. We use this model to model the distribution of minority samples and oversample according to the model, the proposed method shows the superiority that it doesn't need any prior distribution assumption nor the dependent features assumption, the experimental results prove the effectiveness of the algorithm.

We organize the paper as follows. Section 2 describes related work of this paper. Section 3 presents the proposed algorithm and analyze it. Section 4 shows the experimental results. Section 5 concludes the paper.

2 Related Work

In 2013, KM [19] proposed VAE: add variational inference to auto-encoder and use parameterization trick to make the variational inference combined with stochastic gradient descent. The overall structure of vae network is shown in Fig. 1, while it assumes the hidden variables to be a Gaussian standard distribution, it is easy to sample and the final probability distribution function is uncertain, coincides with the characteristics of distribution-based oversampling.

In VAE, we assume the variables are determined by the hidden compression code z, the encoder can map z to X, which makes z obey a particular distribution (such as Gaussian distribution, etc.). Knowing the possibility distribution function and its mapping function, we can sample z and encode z, to get new x to generate infinite sample theoretically. The structure of vae as shown below:

Assume z is a latent variable, and its distribution function is p(z), use Bayesian conditional probability formula to calculate P(X):

$$p(X) = \int p(X|z)p(z)\,dz \tag{1}$$

However, in z's prior distribution, most of z cannot generate reliable samples, that is p(X|z) tends to 0, so p(X|z)p(z) tends to 0. To simplify the calculation, only p(X | z) need

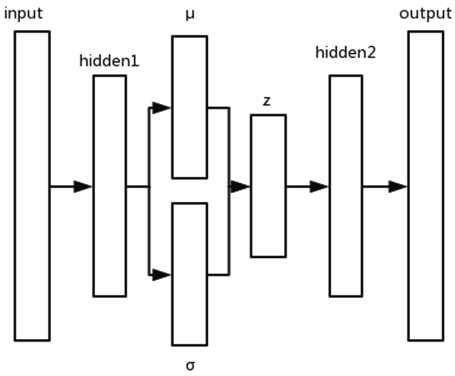


Fig. 1. Structure of variational auto-encoder.

to be calculated. Considering the z with larger P(X|z), which is represented by P(z|X) form the encoder, but only considering this part of z cannot generate samples that are not in original data, so we need to assume the distribution of P(z | X) and complete the error through the decoder.

Q(z) is the assumption of the real distribution, we use KL divergence to calculate the difference between the real distribution and the assumption:

$$D(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{2}$$

Formula (2) shows that if two distribution is close, KL divergence will tend to 0. And the loss function of VAE model is

$$argmin\ D(Q(z)||P(z|X)) \tag{3}$$

Apply the formula (2) to the formula (3)

$$D[Q(z)||P(z|X)] = E_{z\sim Q}[\log Q(z) - \log P(z|X)] \tag{4}$$

Apply Bayes rule to P(z|X), we can get both P(X) and P(X|z)

$$D(Q(z)||P(z|X)) = E_{z\sim Q}[\log Q(z) - \log P(z)] + \log P(X) \tag{5}$$

Apply the D[Q(z)||P(z|X)] into it, note that X is fixed, and Q can be any distribution, not just a distribution which does a good job at mapping X to the z's to produce X. since we're interested in inferring P(X), it makes sense to construct a Q which does depend on X, and in particular, one which makes D(Q(z)||P(z|X)) small: Because P(X) is fixed, the minimum D(Q(z)||P(z|X)) will transform to maximize the value of the right side of the equation, and log P(X|z) is the probability of X decoded by z. It is calculated as the cross-entropy or mean-squared error of the original sample. The latter

can be regarded as the difference between the assumption and the distribution of z in the encoder.

$$\log P(X) - D[Q(z)||P(z|X)] = E_{z\sim Q}[\log P(X|z)] - D[Q[z]||P(z)] \tag{6}$$

3 The Proposed Method

In this paper, an oversampling method based on VAE is proposed, motivated by the idea that the distribution information is important in oversampling method. Without any prior assumption of the real PDF of the minority samples nor the independent assumption in the features, the proposed method can automatically model the PDF with the oral data. However, there is also a trick in the proposed, there might have discrete features in the data, while the features generated by the stochastic gradient descent must be continuously differentiable, so this part of the features must be selected before vae training using formula (9), and after generating the continuous features, use 1-NN to classify the generated continuous and combine the continuous features with the discrete features of the nearest original sample into a new composite sample.

We don't have enough information about whether a feature is discrete or not, so we assume that it is a discrete feature if there are no more than 2 distinct values in all the feature values. In fact, it is useless in classification if there is only one distinct value among the whole dataset.

Given training dataset $X = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, $x_i \in R^d$ is the sample of d dimension, $y_i \in \{0, 1\}$ is the labels represent negative and positive. We use P and N to represents a positive class sample subset and a negative class sample subset, where P contains N_+ positive samples, N contains N_- negative samples, and $N_+ + N_- = N$.

During the training of the VAE model, $nelements_j$ represents the number of distinct feature values in j_{th} dimension in the positive subset, the formula is shown as (7):

$$nelements_j = \sum_i^{N_+} distinct\{x_{ij}\},\ 1 \le j \le d \tag{7}$$

$$x_i = \{x_{i1}, x_{i2}, \cdots, x_{ik}\} \cup \{\ x_{i(k+1)},\ \cdots,\ x_{id}\} \tag{8}$$

$$\text{s.t.} \begin{cases} nelements_j > 2, & 1 \le j \le k \\ nelements_j \le 2, & k+1 \le j \le d \end{cases} \tag{9}$$

If $nelements_j$ is no more than 2, the feature j is discrete, otherwise, the feature is continuous. Divide the features in the positive subset into continuous and discrete features in order and the continuous features are used as the final training set.

$$Xtrainvae = \begin{bmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{N_+1} & \cdots & x_{N_+k} \end{bmatrix} \tag{10}$$

Train a VAE model with *Xtrain* and randomly sample it, assume *Xnew* is synthetic a sample:

$$\begin{cases} Xfinal_{ij} = Xnew_{ij} \cup X_{lm}, & k+1 \le m \le d \\ s.t.\ agrmin \sum \left|\left| Xnew_{ij} - X_{lj} \right|\right|^2, & 1 \le j \le k \end{cases} \tag{11}$$

X_{final} is the final synthetic sample, and $X \cup X_{final}$ is the final training set called X_{ov}.

Algorithm 1: Oversampling using VAE

Input: A dataSet X,sampling rate k
Output: x_{ov} after oversampling

$X \leftarrow \frac{X-\overline{X}}{s}$
$X_{train}, X_{test} \leftarrow \text{divide}(X)$
for *each feature* j *in* X **do**
    $nelements_j \leftarrow \sum_1^{N_+} distinctX_j$
end
Decide $X_{trainvae}$ with formula (10)
$vae \leftarrow \text{trainvae}(X_{trainvae})$
$X_{new} \leftarrow \text{sample}(vae)$
Synthesize X_{final} with formula (11)
$X_{ov} \leftarrow X_{final} \bigcup X_{train}$
return X_{ov}

The whole process is described as Algorithm 1, firstly, normalize the dataset to scale the range of data, and divide(X) is a function which can split the dataset as training set and testing set, in imbalanced classification, to keep the distribution unchanged in these subsets, the positive and negative samples are split separately. Secondly, choose the features with over two distinct values and use them as the $X_{trainvae}$. Thirdly, train a VAE model and sample from the trained model, suppose the generated samples as X_{new}. Finally, add the discrete features for the generated samples using their nearest neighbors' discrete features, and these are X_{final}(Table 1).

Table 1. Dataset.

Index	Dataset	Samples	Attributes	Minority	Imbalance ratio
1	breast-w	699	9	241	1.90
2	vehicle	846	18	199	3.25
3	segment-challenge	1500	19	205	6.32
4	Diabetes	768	8	268	1.87
5	Ionosphere	351	34	126	1.79

4 Experiment

4.1 Dataset and Evaluation

In this paper, all datasets are from UCI [20] Machine Learning Repository, and some of them are multi-label datasets, so we select one class as the minority class and the remaining samples as majority class. The missing values are supplemented by the most frequent value. After that we use normalization, the formula is shown in (12):

$$x_{inew} = \frac{x_i - \bar{x}}{s} \tag{12}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,\ s = \sqrt{\left(\frac{1}{n-1}\sum_{i}^{n}(x_i - \bar{x})\right)^2} \tag{13}$$

In traditional classification method, global accuracy is used as the evaluation, but in the imbalanced problem, this evaluation will mask the classification performance of the minority class. In extreme conditions, assume the dataset only contain 1% minority class, if the classifier decides all samples as majority class, the accuracy still can reach 99%, and however, the recognition rate of the minority samples is 0. In binary imbalanced classification, the confusion matrix in Table 2. Confusion metrics. is often used to evaluate the performance of the classifier.

Table 2. Confusion metrics.

	Positive prediction	Negative prediction
Positive class	True positive(TP)	False negative(FN)
Negative class	False positive(FP)	True negative(TN)

Among them, FN is the number of the positive samples misclassified as negative, and FP is the number of the negative samples misclassified as positive. There are some new evaluation metrics based on confusion matrix to calculate the accuracy and recall of imbalance data such as F-value, G-mean [21].

$$precision = \frac{TP}{TP + FP} \tag{14}$$

$$recall = \frac{TP}{TP + FN} \tag{15}$$

$$F - value = \frac{(1 + \beta^2) \times recall \times precision}{\beta^2 \times recall + precision} \tag{16}$$

Where $\beta \in [0, +\infty]$.

$$Gmean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \tag{17}$$

In this experiment, we choose $\beta = 1$ for F-value, it is the average between recall and precision. Gmean is the geometric mean of the classification accuracy of the minority class and majority class. Only when the precision of minority class and precision of majority are high at the same time, gmean will be maximum.

4.2 Experiment Results

In this paper, we compare different oversampling algorithms such as NDO-sampling [17] and random interpolation algorithm SMOTE [12] (SMO). The classifier is naïve Bayes to reduce the impact of classifier's parameters on classification performance. To reduce the randomness in the final results, each algorithm calculates the average of 10 times with 10-fold cross-validation. The results of NDO are from the corresponding paper, and k = 5 in SMOTE, the structure of the proposed method is shown in Fig. 1, we use the random sample in generating new samples.

The result shown in the Table 3 indicate that vae performs better in generating samples than NDO and SMOTE when the number of oversampling is the same, as the VAE can generate more reasonable samples with more information. With the growth of oversampling rate, all the sampling methods can help to improve the classification performance, which indicate that the original minority samples don't contain enough information for a classifier to recognize them correctly from the negative samples.

Table 3. F1-min of different algorithms and oversampling rate.

	100%			200%			300%		
	VAE	NDO	SMO	VAE	NDO	SMO	VAE	NDO	SMO
1	**94.65**	94.38	94.38	**94.63**	94.38	94.38	**94.67**	94.38	94.38
2	**58.22**	55.66	56.26	**58.12**	56.63	56.45	**58.78**	56.27	56.45
3	**66.88**	65.47	62.44	**69.45**	66.55	61.15	**71.21**	61.90	61.15
4	65.51	65.93	**66.27**	**66.96**	66.74	66.33	**66.34**	65.59	66.33
5	**87.00**	82.34	80.54	**85.89**	82.63	82.71	**84.26**	81.44	82.71

In the meanwhile, from the result in Table 4, compared with the traditional oversampling algorithms which sacrifice some majority samples to ensure the classification performance of minority, the proposed method can guarantee the rational distribution of synthetic samples and improve the classification performance of the majority samples, which indicates a stronger classifier.

The proposed method can produce more reasonable samples, which can be concluded form the result shown in Table 5, as the classifier trained with the samples generated by the proposed method can get a better overall classification performance, as the *Gmean* is the geometric mean of the accuracy of the minority samples and the majority samples, and with a higher oversampling rate, the classifier gets a best result with the proposed oversampling method.

Table 4. F1-maj of different algorithms and oversampling rate.

	100%			200%			300%		
	VAE	NDO	SMO	VAE	NDO	SMO	VAE	NDO	SMO
1	**96.95**	96.89	96.89	**96.94**	96.89	96.89	**96.96**	96.89	96.89
2	**74.50**	72.06	72.35	**75.90**	71.98	71.87	**77.82**	71.73	71.90
3	91.68	**91.79**	89.69	**92.78**	92.22	89.41	**93.43**	92.47	89.03
4	79.78	79.73	**80.25**	77.44	**77.78**	76.71	**76.29**	74.95	74.48
5	**93.70**	89.62	87.79	**93.42**	89.84	88.56	**92.82**	90.07	89.45

Table 5. Gmean of different algorithms and oversampling rate.

	100%			200%			300%		
	VAE	NDO	SMO	VAE	NDO	SMO	VAE	NDO	SMO
1	**96.50**	96.35	96.35	**96.50**	96.35	96.35	**96.55**	96.35	96.35
2	**75.14**	72.71	73.27	**75.19**	73.50	73.18	**75.79**	73.30	73.34
3	**90.34**	89.55	88.91	**91.40**	89.23	89.36	**91.88**	89.41	89.01
4	73.15	73.57	**73.84**	74.04	**74.07**	73.01	**73.35**	73.24	73.03
5	**88.54**	86.47	85.32	**87.46**	86.66	85.99	86.07	86.86	**86.98**

The experimental results also show that for all oversampling methods, a higher oversampling rate can lead to a better classification performance, when the minority samples after oversampling are equal to the majority ones in size, the best classification performance is reached, this indicates that the size has limited effect on the classification performance, more informative samples and stronger classifier play a bigger role.

5 Conclusion

In this paper, we propose an oversampling algorithm based on VAE, in order to make full use of the distribution information in the dataset, it can generate more reasonable samples with no prior assumption of the real distribution nor the assumption that the features are independent, what's more, we separate the features into discrete and continuous ones, use the nearest discrete features as the features of the generated samples, to generate samples with real meaning as can as possible. The experiment results prove the effectiveness of the proposed method, it can improve the overall performance rather than the minority samples. The sampling is still too rough to guarantee the generated samples' impact on the classifiers, and the future work is to overcome this drawback.

Acknowledgements. This study was supported by the Shenzhen Research Council (Grant No. JSGG20170822160842949, JCYJ20170307151518535, JCYJ20160226201453085, JCYJ 20170307151831260).

References

1. Wang, Y., Li, X., Tao, B.: Improving classification of mature microRNA by solving class imbalance problem. Scientific reports (2016)
2. Stegmayer, G., Yones, C., Kamenetzky, L., Milone, D.H.: High class-imbalance in pre-miRNA prediction: a novel approach based on deepSOM. IEEE/ACM Trans. Comput. Biol. Bioinf. **14**, 1316–1326 (2016)
3. Leichtle, T., Geiß, C., Lakes, T., Taubenböck, H.: Class imbalance in unsupervised change detection – a diagnostic analysis from urban remote sensing. Int. J. Appl. Earth Obs. Geoinf. **60**, 83–98 (2017)
4. Li, C., Liu, S.: A comparative study of the class imbalance problem in Twitter spam detection. Concurr. Comput. Pract. Exp. **30**(4), e4281 (2018)
5. Singh, S., Liu, Y., Ding, W., Li, Z.: Empirical Evaluation of Big Data Analytics using Design of Experiment: Case Studies on Telecommunication Data (2016)
6. Hale, M.L., Walter, C., Lin, J., Gamble, R.F.: A Priori Prediction of Phishing Victimization Based on Structural Content Factors (2017)
7. Zhang, C., Wang, G., Zhou, Y., Jiang, J.: A new approach for imbalanced data classification based on minimize loss learning. In: IEEE Second International Conference on Data Science in Cyberspace, pp. 82–87 (2017)
8. Provost, F.: Machine learning from imbalanced data sets 101 (extended abstract). In: 2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR), pp. 435–439 (2008)
9. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: International Conference on Machine Learning, p. 104 (2004)
10. Donoho, D.L., Tanner, J.: Precise undersampling theorems. Proc. IEEE **98**(6), 913–924 (2010)
11. Olken, F., Rotem, D.: Random sampling from databases: a survey. Stat. Comput. **5**(1), 25–42 (1995)
12. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**(1), 321–357 (2002)
13. Zhang, C., Guo, J., Lu, J.: Research on classification method of high-dimensional class-imbalanced data sets based on SVM. In: IEEE Second International Conference on Data Science in Cyberspace, pp. 60–67 (2017)
14. Han, H., Wang, W.-Y., Mao, B.-H.: Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In: Huang, D.-S., Zhang, X.-P., Huang, G.-B. (eds.) ICIC 2005. LNCS, vol. 3644, pp. 878–887. Springer, Heidelberg (2005). https://doi.org/10.1007/11538059_91
15. Gao, M., Hong, X., Chen, S., Harris, C.J.: Probability density function estimation based over-sampling for imbalanced two-class problems. In: International Joint Conference on Neural Networks, pp. 1–8 (2012)
16. Chen, S.: A generalized Gaussian distribution based uncertainty sampling approach and its application in actual evapotranspiration assimilation. J. Hydrol. **552**, 745–764 (2017)
17. Zhang, H., Wang, Z.: A normal distribution-based over-sampling approach to imbalanced data classification. In: Tang, J., King, I., Chen, L., Wang, J. (eds.) ADMA 2011. LNCS (LNAI), vol. 7120, pp. 83–96. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25853-4_7

18. Li, D.C., Hu, S.C., Lin, L.S., Yeh, C.W.: Detecting representative data and generating synthetic samples to improve learning accuracy with imbalanced data sets. Plos One **12**(8), (2017)
19. Diederik, P.K., Max, W.: Auto-Encoding Variational Bayes
20. Amini, M.R., Usunier, N., Goutte, C.: http://archive.ics.uci.edu/ml/datasets.html. Accessed 22 Mar 2018
21. Menardi, G., Torelli, N.: Training and assessing classification rules with imbalanced data. Data Min. Know. Discov. **28**(1), 92–122 (2014)

Application and Industry Track: Cloud Data Processing

A Two-Stage Data Processing Algorithm to Generate Random Sample Partitions for Big Data Analysis

Chenghao Wei, Salman Salloum, Tamer Z. Emara, Xiaoliang Zhang, Joshua Zhexue Huang(✉), and Yulin He

Big Data Institute, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China
{chenghao.wei,zx.huang}@szu.edu.cn

Abstract. To enable the individual data block files of a distributed big data set to be used as random samples for big data analysis, a two-stage data processing (TSDP) algorithm is proposed in this paper to convert a big data set into a random sample partition (RSP) representation which ensures that each individual data block in the RSP is a random sample of the big data, therefore, it can be used to estimate the statistical properties of the big data. The first stage of this algorithm is to sequentially chunk the big data set into non-overlapping subsets and distribute these subsets as data block files to the nodes of a cluster. The second stage is to take a random sample from each subset without replacement to form a new subset saved as an RSP data block file and the random sampling step is repeated until all data records in all subsets are used up and a new set of RSP data block files are created to form an RSP of the big data. It is formally proved that the expectation of the sample distribution function (*s.d.f.*) of each RSP data block equals to the *s.d.f.* of the big data set, therefore, each RSP data block is a random sample of the big data set. Implementation of the TSDP algorithm on Apache Spark and HDFS is presented. Performance evaluations on terabyte data sets show the efficiency of this algorithm in converting HDFS big data files into HDFS RSP big data files. We also show an example that uses only a small number of RSP data blocks to build ensemble models which perform better than the single model built from the entire data set.

Keywords: Big data analysis · Random sample partition · RSP
HDFS · Apache Spark

The work of this paper was supported by National Natural Science Foundations of China (61503252 and 61473194), China Postdoctoral Science Foundation (2016T90799), Scientific Research Foundation of Shenzhen University for Newly-introduced Teachers (2018060) and Shenzhen-Hong Kong Technology Cooperation Foundation (SGLH20161209101100926).

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 347–364, 2018.
https://doi.org/10.1007/978-3-319-94295-7_24

1 Introduction

In the era of big data, large files of millions of objects with thousands of features are frequently encountered in many applications [1]. For example, millions of customers in a telecom company are required to be segmented into a large number of small groups for targeted marketing and product promotion [2]. Millions of people in a social network are analyzed to understand their opinions on different policies in different social groups [3]. Analyzing such an ever increasing big data is a challenging problem, even on computing clusters, especially when data volume goes beyond the available computing resources. In this regard, divide-and-conquer has become a common and necessary computing strategy to handle big data [4] by distributing both data and computation tasks to the nodes of clusters using distributed file systems (e.g., Hadoop distributed file system (HDFS)) [5] and big data processing frameworks (e.g., Hadoop's MapReduce [6,7], Apache Spark [8,9] and Microsoft R Server[1]). Nevertheless, all distributed data blocks of a data set must be loaded and processed when running data analysis algorithms with current big data frameworks. If the data set to be analyzed exceeds the memory of the cluster system, the algorithms will be inefficient or may not be able to analyze the data. Therefore, the size of memory is an important factor to the ability and performance of the current big data processing and analysis platforms. However, memory may never be enough considering the speed of the increase of big data in size. Can we have a technology with which analysis of big data will not be limited to the size of memory? The answer is affirmative.

If we only analyze some small distributed data blocks without considering the entire big data set and use the results of these blocks to estimate or infer the results of the entire big data set, then the above problem can be solved. In our previous empirical study [10], we found that it is possible if the distributed data blocks have similar probability distributions to the probability distribution of the big data set. We also found that if the records of a big data set are randomly organized, i.e., satisfying the *i.i.d.* condition, sequentially chunking the big data set into a set of data block files will make each data block a random sample of the big data, i.e., making the probability distribution of each data block similar to the probability distribution of the big data set. In statistical analysis, using random samples is a basic strategy to analyze a large data set or an unknown population [11]. The statistical properties of a big data set can be estimated and inferred from the results of random samples of the big data [12]. However, having a naturally randomized data set can only be taken as an exception rather than a rule in big data because it depends on the data generation process. Moreover, the data partitioning methods in current distributed file systems do not take the statistical properties of data into consideration. Consequently, data blocks produced by such methods may not necessarily be random samples of the big data set and using these data blocks as random samples to directly estimate statistics of the big data or build analytical models will lead to biased or incorrect

[1] https://www.microsoft.com/en-us/cloud-platform/r-server.

results. Therefore, a new big data representation model is required to support scalable, efficient and effective big data analysis.

Recently, we have proposed the random sample partition (RSP) data model for big data analysis. The RSP model represents a big data set as a set of non-overlapping subsets which forms a partition of the big data set. Each subset is saved as an RSP data block file that is used as a random sample of the big data set. In this way, analysis of a big data set approximately equals to analysis of one or more RSP data blocks which can be easily read in the nodes of a cluster and computed without the need of big memory. Therefore, the memory limitation on big data analysis is removed in the new RSP data representation model. However, current big data sets are usually saved as HDFS files on computing clusters. Efficient and scalable algorithms are required to convert existing big HDFS files into RSP representations in the scale of Terabyte to support big data analysis. Currently, such algorithms are not available.

In this paper, we propose a two-stage data processing (TSDP) algorithm that is used to convert a big HDFS data set into an RSP representation which ensures that each RSP data block is a random sample of the big data, therefore, it can be used to estimate the statistical properties of the big data. The first stage of this algorithm is to sequentially chunk a big data set into non-overlapping subsets and distribute these subsets as data block files on the nodes of a cluster. This operation is provided in HDFS. The second stage is to take a random sample from each subset without replacement to form a new subset saved as an RSP data block file. The random sampling step is repeated until all data records in all subsets are used up and a new set of RSP data block files are created to form an RSP of the big data. We formally prove that the expectation of the probability distribution of each RSP data block equals to the probability distribution of the big data set, therefore, each RSP data block is a random sample of the big data set.

We present the implementation of the TSDP algorithm on Apache Spark and HDFS and the performance evaluations of this algorithm on terabyte data sets. The experiments were conducted on a computing cluster with 29 nodes. The evaluation results have shown that the TSDP algorithm is efficient in converting HDFS big data files into HDFS RSP big data files. We also show an example that uses only few RSP data blocks to build ensemble models which perform better than the single model built from the entire data set.

The remainder of this paper is organized as follows. Section 2 presents preliminaries used in this research, including two basic definitions. The proposed TSDP algorithm is introduced in Sect. 3. Section 4 presents the TSDP implementation on Apache Spark. Experimental results are discussed in Sect. 5. Further discussions on the advantages of the RSP model for big data analysis and the TSDP algorithm are given in Sect. 6. Finally, the conclusions are given in Sect. 7.

2 Preliminaries

This section briefly reviews big data representations in Hadoop distributed file system (HDFS) and Apache Spark, and gives a formal data definition of the random sample partition (RSP) representation model.

2.1 Big Data Representations in HDFS and Spark

The current big data frameworks use divide-and-conquer as a general strategy to analyze big data on computing clusters. A big data set is chunked into small data blocks and distributed on the nodes of a computing cluster using distributed file systems such as Hadoop distributed file system (HDFS) [13]. To tackle a big data analysis task, the data blocks of a big data set are processed in parallel on the cluster. The intermediate results from the local data blocks processed on local nodes are integrated to produce the final result for the entire data set. To save a big data set as an HDFS file, HDFS operation sequentially chunks the big data set into small data blocks of fixed size (e.g., 64 MB or 128 MB) and distribute the data blocks randomly on the local nodes of the cluster. For data safety, 3 copies of the same data block are usually stored on three different nodes. Apache Spark provides processing-level distributed data abstractions as the resilient distributed data set (RDD) which is held in memory to facilitate the data processing and analysis. The RDDs APIs are provided to import HDFS files or other data files to Spark RDD, control RDD partitioning and manipulate RDD using a rich set of operators. An RDD can be partitioned and repartitioned using different partitioning methods such as hash partitioner and range partitioner.

However, these data partitioning methods, whether HDFS or Spark RDD, do not consider the statistical properties of the big data set. As a result, the data blocks in HDFS files and Spark RDDs cannot be used as random samples for statistical analysis of the big data set. Using these data blocks to estimate the big data tends to produce statistically biased results.

2.2 RSP Data Representation Model

To enable HDFS distributed data blocks to be used as random samples for estimation and analysis of the entire big data set, we propose to use RSP distributed data representation model to represent big data, which ensures that each data block in the RSP model is a random sample of the big data set. The main properties of the RSP model are given in the following two definitions.

Definition 1 (Partition of Data Set): Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a data set containing N objects. Let $\mathbb{T}$ be an operation which divides $\mathbb{D}$ into a family of subsets $\mathrm{T} = \{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$. T is called a *partition* of data set $\mathbb{D}$ if

(1) $\bigcup_{k=1}^{K} \mathrm{D}_k = \mathbb{D}$;
(2) $\mathrm{D}_i \cap \mathrm{D}_j = \emptyset$, when $i, j \in \{1, 2, \cdots, K\}$ and $i \neq j$.

Accordingly, $\mathbb{T}$ is called a partition operation on $\mathbb{D}$ and each $\mathrm{D}_k (k = 1, 2, \cdots, K)$ is called a data block of $\mathbb{D}$.

An HDFS file is a partition of data set $\mathbb{D}$ where data blocks $\{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$ are generated by sequentially cutting the big data. Usually, these data blocks in HDFS files do not have similar distribution properties as the big data, therefore, cannot be used in general as random samples for analysis of the big data set. However, the data blocks in the RSP as defined below can be used as random samples of the big data set.

Definition 2 (Random Sample Partition): Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a big data set which is a random sample of a population and assume $F(x)$ to be the sample distribution function (*s.d.f.*) of $\mathbb{D}$. Let $\mathbb{T}$ be a partition operation on $\mathbb{D}$ and $\mathrm{T} = \{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$ be a partition of data set $\mathbb{D}$ accordingly. T is called a *random sample partition* of $\mathbb{D}$ if

$$E[\tilde{F}_k(x)] = F(x) \quad \text{for each} \quad k = 1, 2, \cdots, K,$$

where $\tilde{F}_k(x)$ denotes the sample distribution function of $\mathrm{D_k}$ and $E[\tilde{F}_k(x)]$ denotes its expectation. Accordingly, each $\mathrm{D_k}$ is called an RSP block of $\mathbb{D}$ and $\mathbb{T}$ is called an RSP operation on $\mathbb{D}$.

3 Two-Stage Data Processing Algorithm

In this section, we present a two stage data processing algorithm for generating RSP from a big data set. Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a data set with N objects and M features. To generate RSP data blocks (i.e., random samples) from $\mathbb{D}$, if N is not big, we can easily use the following steps to convert $\mathbb{D}$ into Q RSP data blocks.

- Step 1: Generate N unique random integer numbers for N objects from a uniform distribution;
- Step 2: Sort N objects on the random numbers to reorganize $\mathbb{D}$;
- Step 3: Sequentially cut N reordered objects into Q small data blocks, each with N/Q objects.

Corollary 1. *Let* $\{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_Q\}$ *denote the above* Q *small blocks. Each* $\mathrm{D_k}$ *is an RSP data block of* $\mathbb{D}$.

Proof: Set $\mathrm{D}_k = \left\{\mathrm{x}_1^{(k)}, \mathrm{x}_2^{(k)}, \cdots, \mathrm{x}_{N_k}^{(k)}\right\}$, $k \in \{1, 2, \cdots, Q\}$ and N_k is the number of objects in D_k. It is obvious that

$$P\left\{\mathrm{D}_k = \left\{\mathrm{x}_{s_1}, \mathrm{x}_{s_2}, \cdots, \mathrm{x}_{s_{N_k}}\right\}\right\} = \frac{1}{C_N^{N_k}}, \tag{1}$$

where $x_{s_1}, x_{s_2}, \cdots, x_{s_{N_k}}$ are s_{N_k} objects selected arbitrarily from $\mathbb{D}$.

Assume $F(x)$ is the *s.d.f.* of $\mathbb{D}$. On one hand, for each real number $x \in R^1$, the number of samples whose values are not greater than x is $F(x) \cdot N$. On the other

hand, for each data block $\mathrm{D_k}$ and object $x_i \in \mathbb{D}$, $P\{x_i \in \mathrm{D}_k\} = C_{N-1}^{N_k-1} \cdot \frac{1}{C_N^{N_k}} = \frac{N_k}{N}$. Thus the number of objects in D_k whose values are not greater than x is $F(x) \cdot N \cdot \frac{N_k}{N} = F(x) \cdot N_k$. We therefore obtain that the expectation of the *s.d.f.* of D_k is $\frac{1}{N_k} \cdot F(x) \cdot N_k = F(x)$. According to Definition 2, D_k is an RSP data block of $\mathbb{D}$.

When N is not large and $\mathbb{D}$ can be sorted in memory, the above algorithm works well. However, when N is large and $\mathbb{D}$ is big, e.g., in terabyte, if Q is also large, e.g., 100000 data blocks, the execution of this algorithm on the cluster becomes impractical, e.g., running extremely long time or bringing the cluster down due to many processes competing for resources. To deal with that situation, a general two-stage data-processing (TSDP) algorithm is designed for converting a big data set to a set of RSP data blocks below. Firstly, we give the following theorem before we present the TSDP algorithm.

Theorem 1. *Let $\mathbb{D}_1$ and $\mathbb{D}_2$ be two big data sets with N_1 and N_2 objects respectively. Assume that D_1 with n_1 objects is an RSP data block of $\mathbb{D}_1$ and D_2 with n_2 objects is an RSP data block of $\mathbb{D}_2$. Then, $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is an RSP data block of $\mathbb{D}_1 \bigcup \mathbb{D}_2$ under the condition that $\frac{n_1}{n_2} = \frac{N_1}{N_2}$.*

Proof: Let $F_1(x)$ and $F_2(x)$ denote the *s.d.f.*s of $\mathbb{D}_1$ and $\mathbb{D}_2$ respectively. Assume that the *s.d.f.*s of D_1 and D_2 are $\tilde{F}_1(x)$ and $\tilde{F}_2(x)$, respectively. According to Definition 2, we have $E[\tilde{F}_1(x)] = F_1(x), E[\tilde{F}_2(x)] = F_2(x)$. For any real number x, the number of objects in $\mathrm{D}_1 \bigcup \mathrm{D}_2$ whose values are not greater than x is $n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)$. Therefore, the *s.d.f.* of $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is:

$$\tilde{F}(x) = \frac{n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)}{n_1 + n_2}.$$

Similarly, the *s.d.f.* of $\mathbb{D}_1 \bigcup \mathbb{D}_2$ is:

$$F(x) = \frac{N_1F_1(x) + N_2F_2(x)}{N_1 + N_2}.$$

The expectation of $\tilde{F}(x)$ is

$$\begin{aligned} E[\tilde{F}(x)] &= E[\frac{n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)}{n_1 + n_2}] = \frac{n_1E[\tilde{F}_1(x)] + n_2E[\tilde{F}_2(x)]}{n_1 + n_2} \\ &= \frac{n_1F_1(x) + n_2F_2(x)}{n_1 + n_2} = \frac{N_1F_1(x) + N_2F_2(x)}{N_1 + N_2} \\ &= F(x). \end{aligned}$$

Therefore, $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is an RSP data block of $\mathbb{D}_1 \bigcup \mathbb{D}_2$.

Remark: With a subtle modification, the proof of Corollary 1 can be extended to data in multiple dimensions and the proof of Theorem 1 can also be extended to the multiple dimensions and more than two data sets.

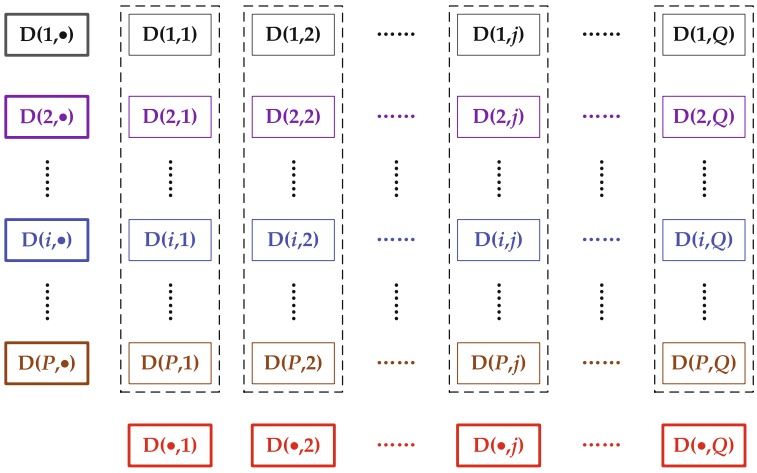


Fig. 1. Illustration of TSDP algorithm.

Let $\mathbb{D}$ be a data set of N objects and M features where N is too big for statistical analysis on a single machine. The following TSDF algorithm is designed to generate an RSP with Q RSP data blocks from $\mathbb{D}$ on a computing cluster.

Stage 1 (Data-Chunking): $\mathbb{D}$ is sequentially chunked into P data blocks $\{\mathrm{D}_i\}_{i=1}^{P}$ which form a partition of $\mathbb{D}$. HDFS provides functions to perform this operation. We call these blocks as HDFS data blocks which are not necessarily random samples of $\mathbb{D}$.

Stage 2 (Data-Randomization): Each RSP data block is built using random samples from P HDFS data blocks. This requires a distributed randomization operation where a slice of samples is selected randomly without replacement from each HDFS data block to form a new RSP data block. This operation is repeated Q times to produce Q RSP data blocks. The main steps in this stage are summarized as follows:

- Step 1: Randomizing D_i in parallel for $1 \leq i \leq P$;
- Step 2: Cutting each D_i in parallel into Q subsets $\{\mathrm{D}(i,j)\}_{j=1}^{Q}$ which form an RSP of D_i, for $1 \leq i \leq P$;
- Step 3: From $P \times Q$ subsets $\{\mathrm{D}(i,j)\}$ for $1 \leq i \leq P$ and $1 \leq j \leq Q$ which form a partition of data set $\mathbb{D}$, merge the subsets of $\{\mathrm{D}(i,j)\}_{i=1}^{P}$ for $1 \leq j \leq Q$ to form Q subsets $\{\mathrm{D}(\cdot,j)\}_{j=1}^{Q}$ which form an RSP of $\mathbb{D}$.

TSDP algorithm is illustrated in Fig. 1. The first stage generates the first level partition $\{\mathrm{D}(1,\cdot), \mathrm{D}(2,\cdot), \cdots, \mathrm{D}(P,\cdot)\}$ of $\mathbb{D}$. The second stage randomizes each subset $\mathrm{D}(i,\cdot)$ independently, generates the second level partitions $\{\mathrm{D}(i,1), \mathrm{D}(i,2), \ldots, \mathrm{D}(i,Q)\}$ for $1 \leq i \leq P$ and merges the column partitions into $\{\mathrm{D}(\cdot,1), \mathrm{D}(\cdot,2), \ldots, \mathrm{D}(\cdot,Q)\}$. By repeatedly applying Theorem 1, we can deduce that the new data block $\mathrm{D}(\cdot,j), (j = 1, \cdots, Q)$ is an RSP data block of $\mathbb{D}$.

4 TSDP Implementation on Apache Spark

In this section, we present an implementation of the TSDP algorithm on Apache Spark with HDFS. This implementation is called the *round-random partitioner* which is a hybrid between the range partitioner and the hash partitioner in Spark. Algorithm 1 shows the Pseudocode of this TSDP implementation in which the two stages are fulfilled as follows:

Algorithm 1. Round-Random Partitioner

Input:
- $\mathbb{D}_{HDFS}$: The HDFS file of a big data set $\mathbb{D}$;
- P: the number of RDD partitions in $\mathbb{D}_{HDFS}$;
- Q: the number of RSP data blocks in the RSP output HDFS file;

Method:
$\mathbb{D}_{RDD} \leftarrow$ read $\mathbb{D}_{HDFS}$; load HDFS file into an RDD
for all block i in $\mathbb{D}_{RDD}$ **do**
  $K = Seq$ (1 to n)
  shuffle K;
end for
bind K with $\mathbb{D}_{RDD}$ partitions to generate $pairRDD(K, V)$;
repartition $pairRDD$ using HashPartitioner(Q)
$\mathbb{D}_{HDFS-RSP}$ = values of $pairRDD$; ignore the random numbers in $\mathbb{D}_{HDFS-RSP}$

Output:
- $\mathbb{D}_{HDFS-RSP}$: the final RSP; Generate the RSP HDFS file.

Stage 1 (Data-Chunking): This stage is performed using HDFS. A given big data set $\mathbb{D}$ is converted to an HDFS file with P HDFS data blocks. The HDFS file of $\mathbb{D}$ is loaded to a Spark RDD using `SparkContext.textFile()` function. By default, each HDFS data block is mapped into an RDD partition and the Spark RDD consists of P RDD partitions. The number of RDD partitions to be generated can also be specified in the Spark data loading function. This step is basically a range partitioner which produces P HDFS data blocks[2].

Stage 2 (Data-Randomization): This stage randomizes all Spark partitions in parallel, randomly takes samples from each partition to form a set of Q RDD partitions and saves these RDD partitions as RSP data blocks in an HDFS file. This stage is completed with the following operations:

– Key generation: Generate an array of a sequence of integers K_i $(1 \leq i \leq P)$ for each RDD partition. The length of the sequence equals to the number of records in the RDD partition. This operation is carried out in parallel.

[2] Note: In Spark's terminology, an RDD is equal to a partition of the big data set. A partition is equal to a data block of the big data set. In this section, we use partition to indicate a data block of the big data set loaded to an Spark RDD in order to be consistent with Spark' terminology in this Spark implementation.

- Key Randomizing and Binding: Use Spark shuffle operation to randomize the integer sequences in the arrays Ks in parallel. Use the randomized integers in each array as random numbers and bind them to its corresponding RDD partition in a Spark partition pair named as pairRDD.
- Repartitioning: Repartition the pair RDD by hashing the random numbers in the pairRDD to generate Q new RDD partitions in the pairRDD.
- RSP HDFS file generation: Ignore the random numbers in the pairRDD and save the rest partitions as RSP data blocks in an HDFS file. This HDFS file is an RSP representation of the big data set $\mathbb{D}$.

Algorithm 2. Synthetic Data Generation

Input:
- M: number of features;
- P: number of data blocks;
- N: number of objects;
- H: number of clusters;

Method:
for all $h = 1$ to H **do in parallel**
  $\mu(M)$ =Fill $(\mu_m)|\mu_m \in U\,[0, 10]$
  $\sigma(M)$ =Fill $(\sigma_m)|\sigma_m \in U\,(0, 1)$
  $D\_RDD \leftarrow GenerateRandomRDD(N/H, M, P, \mu, \sigma)$
  save D_RDD on HDFS
end for
Collect all files under one directory using HDFS APIs

5 Experiments and Results

In this section, we demonstrate the performance of Spark implementation of the TSDP algorithm on a computing cluster consisting of 2 name nodes and 27 data nodes, all running Centos 6.8. Each node has 16 cores (32 with Hyper-threading), 128 GB RAM and 1.1 TB disk storage. Several synthetic data sets were used in the experiments. We first present the method used to generate synthetic data sets. Then, we present the time performance of the algorithm in generating RSP data blocks from HDFS data sets of different sizes, and the time performance in generating different numbers of RSP data blocks from 1 terabyte HDFS data set in 1000 HDFS data blocks, and the time performance in generating the same number of RSP data blocks from 1 terabyte HDFS data set with different numbers of HDFS data blocks. After that, we illustrate the changes of probability distributions of HDFS data blocks and RSP data blocks. Finally, we use a real data example to show how the RSP data blocks are used to build ensemble models from only few RSP data blocks with accuracies equivalent to or even better than the model built from the entire data set.

5.1 Synthetic Data Generation

Different synthetic data sets with different numbers of objects N, different numbers of features M and different numbers of clusters H were generated and saved as HDFS files with different numbers of data blocks P. The clusters in the data sets have normal distributions with different means μ and standard deviations σ. Table 1 shows the characteristics of the synthetic data sets. The data generation algorithm is illustrated in Algorithm 2 which was implemented in Apache Spark. The data generation is performed in the following steps:

- Mean Generation: For each cluster, create a mean array for all features $\mu(M)$ and fill the array with values generated from the uniform distribution $\mathrm{U}(0, 10)$.
- Standard Deviation Generation: For each cluster, create a standard deviation array for all features $\sigma(M)$ and fill this array with values generated from the continuous uniform distribution $\mathrm{U}(0, 10)$.
- Cluster RDD Generation: Randomly generate a cluster RDD, D_RDD, for each cluster from the normal distribution with the following parameters, the number of objects $\frac{N}{H}$, the number of features M, the number of data blocks P, the array of features' means μ, and the array of features' standard deviations σ.
- Save and Collect: Save D_RDD on HDFS. After saving the RDDs for all clusters, they are collected under the same directory using HDFS APIs.

Table 1. Characteristics of synthetic data sets: N, M and H are the numbers of objects, features and clusters, respectively.

DS name	Total size	N	M	H	Associated tasks
DS001	≃100 GB	100,000,000	100	100	Classification
DS002	≃200 GB	200,000,000	100	100	Classification
DS003	≃300 GB	300,000,000	100	100	Classification
DS004	≃400 GB	400,000,000	100	100	Classification
DS005	≃500 GB	500,000,000	100	100	Classification
DS006	≃600 GB	600,000,000	100	100	Classification
DS007	≃700 GB	700,000,000	100	100	Classification
DS008	≃800 GB	800,000,000	100	100	Classification
DS009	≃900 GB	900,000,000	100	100	Classification
DS010	≃1 TB	1,000,000,000	100	100	Classification

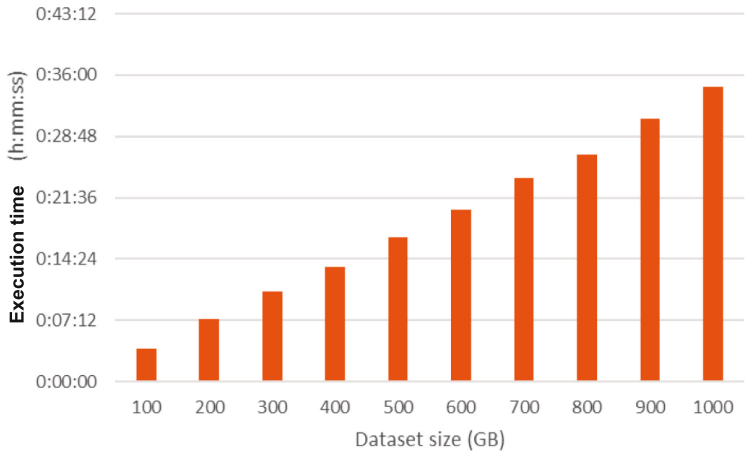


Fig. 2. Execution time for different data sets. The size of data set is changed from 100 GB to 1 TB, with 100 GB increase in each run.

5.2 Time Performance of TSDP

Three experiments were conducted with the synthetic data sets in Table 1 on the computing cluster. In the first experiment, all 10 synthetic data sets from 100 GB to 1 TB were used to convert HDFS data blocks to RSP data blocks of the these data sets. The size of both HDFS and RSP data blocks was fixed to 100 MB. The execution time for converting all HDFS data blocks to RSP data blocks for each data set by the TSDP algorithm on the computing cluster was recorded. Figure 2 shows the result of this experiment. We can see that when the size of data blocks is fixed, the time to generate RSP data blocks increases linearly to the size of the data set. We can observe that the execution time for 1 TB data set is about 35 min. Since we only need to transform each data set once, this time is quite acceptable in real applications. With this transformation, analysis on terabyte data becomes easier.

In the second experiment, only the data set in 1 TB was used. The number of HDFS data blocks was fixed to 1000. The number of RSP data blocks to be generated was increased from 10000 to 100000 with 10000 increase in each step. Figure 3(a) shows the execution time of ten runs on the computing cluster. We can see that the execution time increases linearly as the increase of the number of RSP data blocks to be generated when the number of HDFS data blocks was fixed to 10000. However, the time increase is not significant in each step, which indicates that the number of RSP data blocks is not an important factor to the execution time in this algorithm.

In the third experiment, only the data set in 1 TB was used. The number of RSP data blocks was fixed to 10000. The number of HDFS data blocks was changed from 10000 to 100000 with 10000 increase in each step. Figure 3(b) shows the execution time of ten runs on the computing cluster. Again, we can see the

linear increase in execution time as increase in the number of HDFS data blocks in the HDFS file. However, the increase is more significant in each step. This indicates that the number of HDFS blocks in the HDFS file is an important factor to the execution time of this algorithm. We can see that converting 1 terabyte HDFS data set with 10000 HDFS blocks to 10000 RSP data blocks took about 30 min where converting the same data with 100000 HDFS blocks to 10000 RSP data blocks took more than 1 h. The reason is that the HDFS data blocks need to be randomized in this algorithm, which requires more memory resource to execute in parallel when the size and number of HDFS data block become large. These three experiments show that the TSDP algorithm is scalable to terabyte data, which facilitates the analysis of big data in terabyte size.

5.3 Distribution of Data Blocks

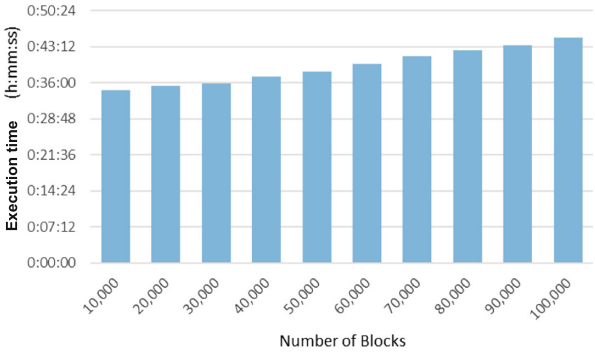
In this section, we show the probability distributions of features in HDFS data blocks and RSP data blocks and demonstrate that RSP data blocks of the same data set have similar probability distributions among themselves whereas the HDFS data blocks do not have similar probability distributions among themselves. Given a sequence of values of one feature from an HDFS data block, the probability density function (*p.d.f.*) is represented as a kernel density function which can be estimated by Parzen window method [14]. The density estimation algorithm is used to disperse the mass of the empirical *p.d.f.* over a regular grid with at least 512 points. The fast Fourier transform is used to convolve the approximation with a discredited version of the kernel. Gaussian basis functions are used to approximate univariate data. The rule-of-thumb is used to determine the bandwidth h of Parzen window [14] as

$$h = \left(\frac{4\sigma^5}{3n}\right)^{1/5} \approx 1.06\sigma n^{-1/5} \tag{2}$$

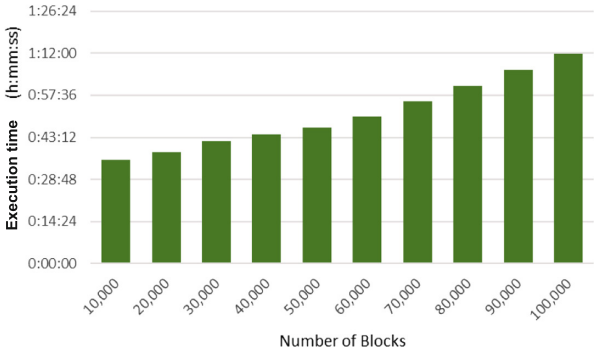
where σ is the standard deviation of the feature and n is the number of points. With this method, we plot the *p.d.f.* of any continuous feature in an HDFS data block and an RSP data block.

Figure 4(a) and (c) show the *p.d.f.* of two features from 6 randomly chosen HDFS data blocks in data set DS010. We can see that different HDFS data blocks have different probability distributions of two features. This indicates that these data blocks cannot be used to estimate and analyze the big data set because they are not representatives of the entire data set. To analyze the big data set, all HDFS data blocks must be taken into consideration, therefore, all data blocks have to be loaded to memory and compute. If the computing cluster has limited resource, then, the ability of analyzing big data will be limited.

Figure 4(b) and (d) show the *p.d.f.* of two features from 6 randomly chosen RSP data blocks in data set DS010. We can see that all RSP blocks have the same probability distribution on the same feature. Theorem 1 shows that the expectation of the probability distribution of each RSP data block is the probability distribution of the entire data set. Therefore, these RSP data blocks



(a) The number of RSP data blocks Q is changed from 10000 to 100000 blocks, while the number of HDFS data blocks P is fixed to 10000.



(b) The number of HDFS data blocks P is changed from 10000 to 100000 blocks, while the number of RSP data blocks Q is fixed to 10000.

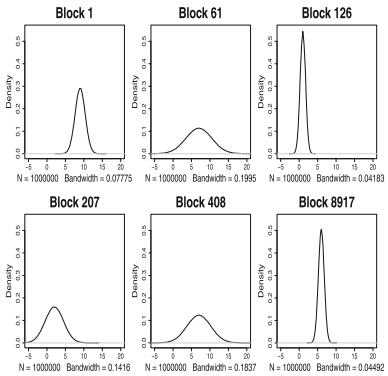
Fig. 3. Execution time against the numbers of both HDFS and RSP data blocks in 1 terabyte data set.

are good representatives of the entire data set and can be used to estimate and analyze the big data set. This is the rational to use RSP data blocks as random samples to analyze the big data set. After converting an HDFS big data set to RSP data blocks, analyzing the big data set is transferred to analysis of RSP data blocks. The big data set no longer needs to compute directly. Therefore, RSP representation significantly facilitates the analysis of big data and enables terabyte data sets to be easily analyzed.

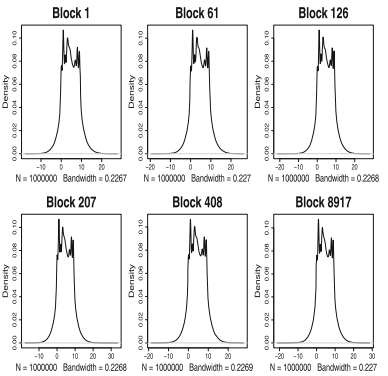
5.4 RSP Block-Based Ensemble Classification

In this section, we use a real data example to show the use of few RSP data blocks to build ensemble classification models which perform equally good or even better than the model built from the entire data set. In this experiment,

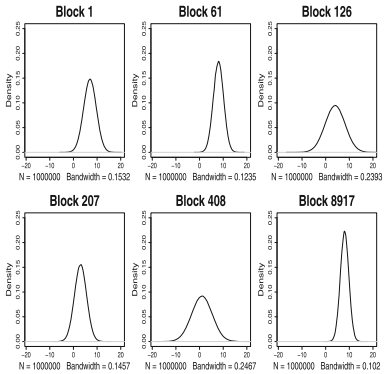
we used the asymptotic ensemble framework (called Alpha framework) [10] to build ensemble models from RSP data blocks.



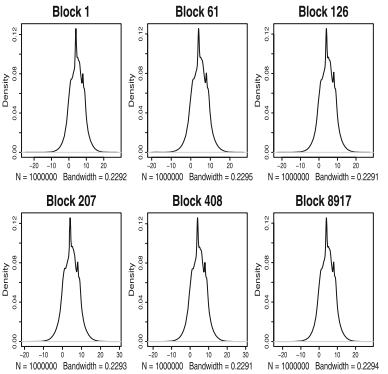
(a) *p.d.f.* without TSDP method on feature 1



(b) *p.d.f.* with TSDP method on feature 1



(c) *p.d.f.* without TSDP method on feature 2



(d) *p.d.f.* with TSDP method on feature 2

Fig. 4. *p.d.f.* comparison between HDFS data blocks and RSP data blocks in 1 TB synthetic data.

Alpha framework is illustrated in Fig. 5. A big data set $\mathbb{D}$ is first converted into RSP data blocks using the TSDP algorithm. Then, a subset of RSP data blocks is randomly selected without replacement, e.g., $\mathrm{D}_5, \mathrm{D}_{120}, \mathrm{D}_{506}$ and D_{890} in Fig. 5. After that, a base model is built from each of these data blocks, e.g., four classifiers $\pi_1, \pi_2, \pi_3, \pi_4$ built in parallel from 4 selected RSP data blocks. The next operation is to update the ensemble model Π with the newly built classifiers and evaluate Π. If Π satisfies the termination condition(s), then output Π, otherwise, go back to RSP representation and randomly select a new batch of RSP data blocks to build a second batch of new classifiers, update the ensemble

model Π with the second batch of classifiers and evaluate it again. This process continues until a satisfactory ensemble model Π is obtained or all RSP data blocks are used up.

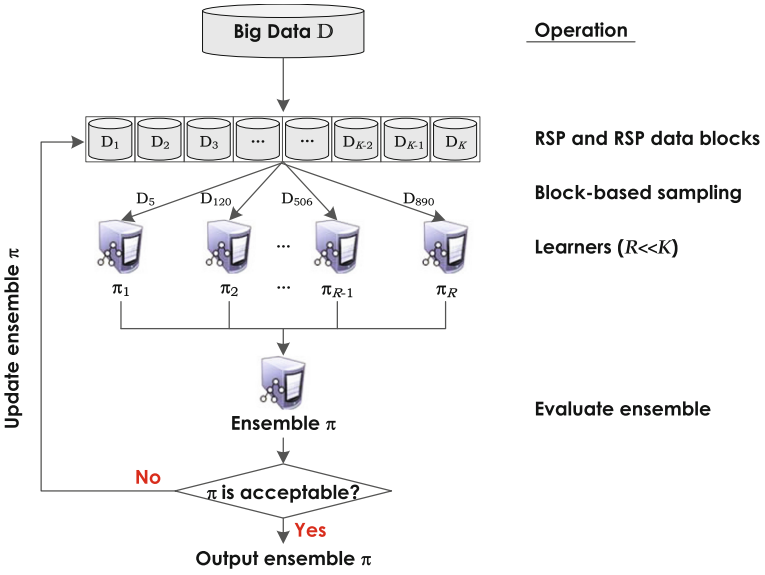


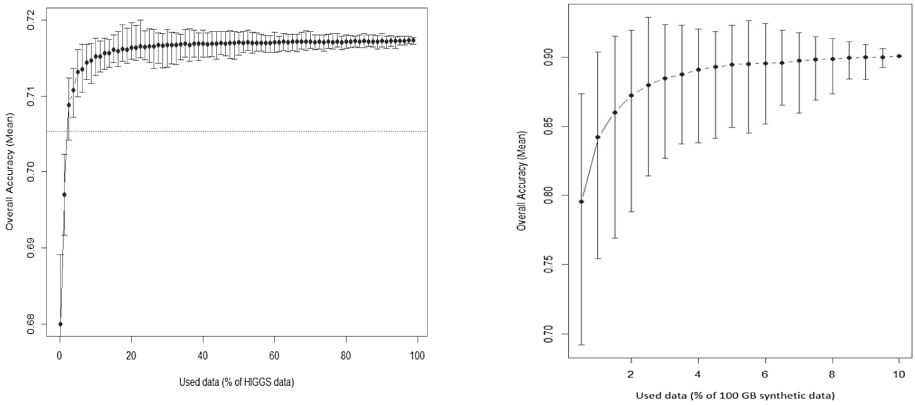
Fig. 5. Alpha framework: an asymptotic ensemble framework to build ensemble models from RSP data blocks.

The real data HIGGS with $N = 11,000,000$ objects and $M = 28$ features was used in this experiment. 400 RSP data blocks were generated with the TSDP algorithm. Figure 6(a) shows the accuracy of the ensemble models after each batch of data blocks. We can see that there is no significant change after using about 15% of the data and the accuracy value converges to a value which is approximately the same as the accuracy of a single model built from the whole data set (the dotted line in the figure). In addition, we also used the 100 GB synthetic data set DS001 to test the performance of the ensemble models from RSP data blocks. We found that only 10% of this data is enough to build an ensemble model with 90% accuracy as shown in Fig. 6(b). In such cases, we do not need to continue building further models from the remaining data blocks.

6 Discussions

Two computational costs are significant in cluster computing: one is the cost of communications among the computing nodes and the other is the cost of i/o operations on read/write data from/to the disks of local nodes. The two costs limit the ability of cluster computing on analysis of big data with complex

algorithms. The in-memory computing technology alleviates the i/o cost but the communication cost can still be significant in analysis of big data. One solution is to use less sample data to estimate the big data. However, conducting random sampling on distributed big data sets is still very expensive because of the i/o and communication costs in cluster computing. RSP big data presentation model on cluster computing offers a fresh new solution to analysis of a big data set if the big data set is converted to a set of distributed RSP data blocks. In this new approach, analysis of big data becomes analysis of few randomly selected RSP data blocks from the RSP data representation. Three advantages can be summarized as follows:



(a) HIGGS data (b) Synthetic data

Fig. 6. Ensemble classification using RSP data blocks. Each point represents the overall ensemble accuracy calculated after each batch of blocks (averaged from 100 runs of the ensemble process).

- RSP block based analysis can estimate the statistical results of the big data set without computing the entire data set. Therefore, after a big data set is converted to a set of RSP data blocks, the size of the big data is no longer a barrier in big data analysis. This increases the ability of cluster computing when data volume exceeds the available resources.
- Since each RSP data block is processed independently on a local node with the same algorithm, sequential algorithms can be used on RSP data blocks without need of parallelization in the asymptotic ensemble learning framework. This approach makes many existing sequential algorithms useful in analysis of big data.
- Theorem 1 provides a theoretical foundation for analysis of a big data set distributed in multiple computing clusters. For example, if a big data set A is divided into three parts, (A_1, A_2, A_3) stored in three computing clusters (C_1, C_2, C_3), respectively. The TSDP algorithm can be used to convert each A_i to RSP blocks on C_i for $1 \leq i \leq 3$. To analyze A, we can randomly select

the same number of RSP data blocks from each computing cluster and merge the three sets of RSP data blocks into one set of RSP data blocks for A where each new RSP block is formed by merging three RSP blocks, each from one computing cluster. According to Theorem 1, the new blocks are RSP data blocks of A and can be used to estimate the statistical properties of A or build ensemble models for A. This new approach provides an opportunity to analyze big data sets cross data centers. However, detail technology needs to be further developed to facilitate analysis of big data sets cross multiple data centers.

7 Conclusions and Future Work

In this paper, we have proposed a two-stage data processing (TSDP) algorithm to generate the random sample partitions (RSPs) from HDFS big data sets for big data analysis. This algorithm is an enabling technology for RSP data block based analysis of big data sets, which does not need to compute the entire big data set. We have presented an implementation of the TSDP algorithm on Apache Spark and demonstrated the time performance of this algorithm on 1 terabyte data. The experiment results show that conversion of an HDFS data set in 1 terabyte was completed in less one hour which is well acceptable in real applications. We have also shown a real data example to demonstrate that the ensemble models built using few RSP data blocks performed better than the model built from the entire data set. Our future work is to extend the current algorithm to work on bigger data sets, e.g., 10 terabytes. We will also develop a distributed big data management system on computing clusters based on the RSP representation model.

References

1. Fan, J., Fang, H., Han, L.: Challenges of big data analysis. Nat. Sci. Rev. **1**(2), 293–314 (2014)
2. Zhao, J., Zhang, W., Liu, Y.: Improved K-means cluster algorithm in telecommunications enterprises customer segmentation. In: IEEE International Conference on Information Theory and Information Security, pp. 167–169 (2010)
3. Michael, B.: Uncovering online political communities of Belgian MPs through social network clustering analysis. In Proceedings of the ACM 2015 2nd International Conference on Electronic Governance and Open Society, pp. 150–163 (2015)
4. Ahmad, A., Paul, A., Rathore, M.M.: An efficient divide-and-conquer approach for big data analytics in machine-to-machine communication. Neurocomputing **174**(86), 439–453 (2016)
5. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: IEEE 26th Symposium Mass Storage Systems and Technologies, pp. 1–10 (2010)
6. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)

7. Elteir, M., Lin, H., Feng, W.C.: Enhancing mapreduce via asynchronous data processing. In: IEEE International Conference on Parallel and Distributed Systems, pp. 397–405 (2010)
8. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: HotCloud 2010, p. 10 (2010)
9. Salloum, S., Dautov, R., Chen, X., Peng, P.X., Huang, J.Z.: Big data analytics on apache spark. Int. J. Data Sci. Anal. **1**(3–4), 145–164 (2016)
10. Salloum, S., Huang, J.Z., He, Y.L.: Empirical analysis of asymptotic ensemble learning for big data. In: Proceedings of the 2016 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, pp. 8–17 (2016)
11. Cormode, G., Duffield, N.: Sampling for big data: a tutorial. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 1975 (2014)
12. Garcia, D., Lubiano, M.A., Alonso, M.C.: Estimating the expected value of fuzzy random variables in the stratified random sampling from finite populations. Inf. Sci. **138**(4), 165–184 (2001)
13. Leo, S., Zanetti, G.: Pydoop: a python mapreduce and HDFS API for hadoop. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 819–825 (2010)
14. Sheather, S.J., Jones, M.C.: A reliable data-based bandwidth selection method for kernel density estimation. J. Roy. Stat. Soc. **53**(3), 683–690 (1991)

An Improved Measurement of the Imbalanced Dataset

Chunkai Zhang[1(✉)], Ying Zhou[1], Yingyang Chen[1], Changqing Qi[1], Xuan Wang[1], and Lifeng Dong[2]

[1] Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
ckzhang8l2@gmail.com
[2] Hamline University, 1536 Hewitt Avenue, St. Paul, USA

Abstract. Imbalanced classification is a classification problem that violates the assumption of uniform distribution of samples. In such problems, traditional imbalanced datasets are measured in terms of the imbalance of sample size, without considering the distribution information, which has a more important impact on the classification performance, so the traditional measurements have a weak relation with the classification performance. This paper proposed an improved measurement for imbalanced datasets, it is based on the idea that a sample surrounded by more same class samples is easier to classify, for each sample of different classes, the proposed method calculates the average number of the k nearest neighbors in the same class in different subsets under the weighted k-NN, after that, the product of these average values is regarded as the measurement of this dataset, and it is a good indicator of the relationship between the distribution of samples and the classification results. The experimental results show that the proposed measurement has a higher correlation with the classification results and shows the difficulty of classification of data sets more clearly.

Keywords: Imbalanced classification · Measurement · Imbalance ratio

1 Introduction

The classification problem is a very important part of machine learning. In the traditional classification problem, the model training is based on the assumption that the sample distribution is uniform, so the classification cost of each sample is consistent. However, in realistic data sets, the assumption of uniform distribution of samples is difficult to satisfy, in order to pursue the global accuracy, the traditional classifier can easily get an unsatisfying classification performance of the minority samples, causing them to be hard to recognize. The imbalanced classification problem has appeared in many fields, such as bioinformatics [1, 2], remote sensing image recognition [3], and privacy protection in cybersecurity [4–6]. The wide coverage of the imbalance problem has very important practical significance.

The traditional imbalanced classification problem has two features: the difference in sample size; the difference in misclassification cost for different classes. Scholars have proposed the data-level methods for feature 1 and the algorithm-level methods for

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 365–376, 2018.
https://doi.org/10.1007/978-3-319-94295-7_25

feature 2. As the data-level methods can be regarded as a preprocessing before training the classifier, so they have been popular for many years. The data-level methods can be divided into the oversampling to add the minority samples, the under-sampling to reduce the majority samples and the hybrid sampling methods, they aim to get a balanced dataset, which is defined by the imbalanced dataset measurement.

Measurement of the imbalanced datasets can be divided into two types: local measurements and global measurements. The local measurements refer to these methods which need traversing each sample in a data set [7, 8], calculating a measurement usually accompanied by the k-NN algorithm for each sample, the overall measurement is defined by the mean value of measurement of all the samples in the dataset. Because this kind of measurement contains the calculation for each sample, and it can be used in the sampling algorithm to find a simpler dataset to model with enough information with the original dataset. Global measurement [9] refers to a result calculated for a sample in the entire data set, or a variety of indicators derived from statistical analysis. It is usually accompanied by a variety of calculations for the separate results of the positive and negative subsets. Such measurements are difficult to achieve in a single implemented on the sample, it can only be used as a measure of the dataset, and it is difficult to play a role in the sampling algorithm because the movement of a single sample can hardly affect the original measurement result.

As the number of samples has had a noticeable effect on the classification results. Therefore, the imbalance ratio [8–10] (IR) of the number of samples in different classes has been popular for many years as a measurement of imbalanced datasets. Based on IR, scholars have proposed many sampling algorithms to balance the datasets to release effect of the imbalance in sample size on the classification performance, so the measurement plays a very important role in imbalanced classification. However, the IR is not informative enough to measure a specific dataset overall, as it is a global measurement, studies [10] have shown that when the number of samples is relatively large, it does not cause a reduction in the classification performance of the minority class, but when the number of samples is seriously insufficient, the rarity of the minority samples will cause a low recognition rate of the minority samples. The local measurements develop the global ones as they take the distribution into consideration, meanwhile, with the understanding of the classifier and data, the distribution based sampling methods [12–14] are taking the distribution information into consideration, and also encourage the new measurements to contain the distribution information.

This paper proposes a measurement containing the distribution information, it is motivated that the nearer a sample is with the same labeled samples, the easier it can be classified correctly. The proposed method calculates the average number of the k nearest neighbors in the same class in different subsets under the weighted k-NN, after that, the product of these average values is regarded as the measurement of this dataset. It improves the correlation between the measurement and the final classification performance, which indicate that the proposed is more informative. This paper is arranged as follows: Sect. 2 describes the related work in measurement of the imbalanced dataset, Sect. 3 shows the proposed measurement improved generalized imbalanced ratio (IGIR), and Sect. 4 describes the experimental results and analysis, the final section concludes the proposed method and the future work.

2 Related Work

There are many different factors which have effects on the imbalanced classification, resulting in various kinds of measurements considering different factors. For example, the Imbalance Ratio (IR) is based on the difference in sample size, the maximum Fisher's discriminant ratio (F1) is based on the overlap in feature values from different classes, and the Complexity Measurement (CM), Generalized Imbalance Ratio (GIR), and the proposed method Improved Generalized Imbalance Ratio (IGIR) are based on the idea that data distribution plays important role in imbalanced classification. These measurements are used in such two ways: indicate whether a dataset is easy to classify, and measure the sampled subset in sampling methods. Therefore, in order to achieve a better performance, the measurement should have a relatively high correlation with the classification results.

Given dataset X, which contains N_+ positive samples (the minority class), N_- negative samples (the majority class), and the total number of samples is $N = N_- + N_+$.

2.1 IR

Imbalance ratio [11–13] the definition is as follows, it is defined as the size sample ratio:

$$IR = \frac{N_-}{N_+} \tag{1}$$

When samples with different labels have the same distribution, the sample size is able to reflect whether the samples are easy to classify, otherwise, the IR is not so informative to indicate whether the dataset is easy to classify. For example, in the Fig. 1, the IR of data in (a) is 4 and in (b) is 1, but the two classes in (a) have a clear linear boundary while there is not in (b), so we can get 100% accuracy in (a) but cannot in (b) with a same linear model, which is contrary to the comparison result of IR, since IR is the proportion of sample size and does not contain any sample distribution information, complexity of the data distribution cannot be represented in IR.

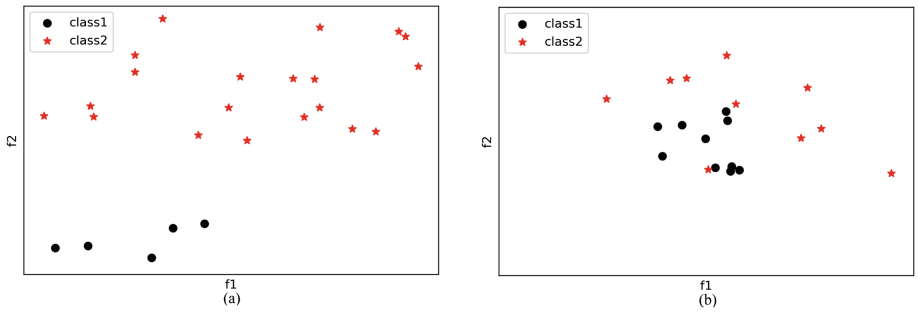


Fig. 1. The dilemma of IR.

2.2 F1

A classical measure of the discriminative power of the covariates, or features, is Fisher's discriminant ratio [9], and F1 is the maximum Fishers discriminant ratio:

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{2}$$

Where μ_1, μ_2, σ_1^2, σ_2^2 are the means and variances of the positive and negative subsets, respectively. For multidimensional problems, the maximum f over all features can be used. However, if f gets 0, it does not necessarily mean that the classes are not separable, as it could just be that the separating boundary is not parallel to an axis in any of the given features.

2.3 CM

CM [7] focuses on the local information for each data point via the nearest neighbors, and uses this information to capture data complexity.

$$\mathrm{CM}_{k(j)} = I\left(\frac{\begin{array}{c}Number\ of\ patterns\\ j'in\ N_j\ with\ y'_j = y_j\end{array}}{k} \leq 0.5\right) \tag{3}$$

Where I(.) is the indicator function. The overall measurement is

$$\mathrm{CM}_k = \frac{1}{n}\sum\nolimits_{j=1}^{n} CM_{k(j)} \tag{4}$$

CM is determined by the label of its neighbors. If the neighbors of a sample contain more samples in the same class, the sample will be easier to classify. On the contrary, if the samples are surrounded by samples with different labels, then the sample is difficult to classify correctly, and the average number of different classes samples contained in the k nearest neighbors is used as the measurement. The higher the CM, the more difficult the dataset is to learn.

2.4 GIR

The GIR [8] is an improvement of cm, it focuses on the differences in the difficulty of classifying the samples in different classes. A dataset with a larger GIR is more difficult to get a good performance of the minority class, as the classifier tends to be trained with the easier samples according to the Occam shaver principle. Because we tend to use a most simple classifier to fit the whole dataset, while the more difficult samples need a more complex classifier, which may cause overfitting with single classifier, so this is the reason why ensemble can be effective in the imbalance classification, as they have

the different classifiers corresponding to different levels of sample classification difficulty.

$$T_+ = \frac{1}{N_+}\sum\nolimits_{x\in P}\frac{1}{k}\sum\nolimits_{r=1}^{k} IR(x,X) = \frac{1}{N_+}\sum\nolimits_{x\in P} t_k(x) \tag{5}$$

$$T_- = \frac{1}{N_-}\sum\nolimits_{x\in N} t_k(x) \tag{6}$$

$$\mathrm{GIR} = \mathrm{T}_- - \mathrm{T}_+ \tag{7}$$

Where IR(x, X) is an indicator function. For a sample x, if its k-nearest neighbor's label is the same as x, the result is 1, otherwise, it gets 0.

GIR considers the different measurements of the positive and negative subsets, which is an improvement of CM, GIR is defined as the difference between positive and negative sample subsets, and paper [8] successfully applies GIR to oversampling and under-sampling algorithms. The experimental results show that GIR-based resampling algorithm can effectively improve the classification performance.

However, there are two problems in GIR, first, in the classification process, besides the label of k nearest neighbors, their distance from the sample will also affect the classification result. Second, the GIR of the data set is calculated by the measurement of the negative class minus positive class, so GIR is a relative measurement. As shown in Table 1, the final result shows that the two data sets have the same GIR, but it is clear that the dataset (b) is more difficult to classify than (a). Therefore, GIR is not so sufficient to fully interpret the complexity of the dataset distribution.

Table 1. The dilemma of GIR

	(a)	(b)
T_-	0.9	0.5
T_+	0.7	0.3
GIR	0.2	0.2

3 The Proposed Method

We proposed an improved measurement called IGIR in this paper, it is based on the idea that the sample distribution plays an important role in the classification result, the motivation of IGIR is that if there are many samples with the same label around the sample, the sample is easily classified, and on the contrary, the sample is hard to classify. Different distances of the k nearest neighbors have different effects on the classification results of the sample.

$$weight_r = \frac{k-r}{k}, r = 0, 1, 2 \cdots k-1 \tag{8}$$

$$\text{wei-T}_+ = \frac{1}{\text{N}_+}\sum\nolimits_{\text{x}\in\text{P}} 1/\text{k}\sum\nolimits_{\text{r}=1}^{\text{k}} weight_r * \text{IR}(\text{x},\text{X}) = \frac{1}{\text{N}_+}\sum\nolimits_{\text{x}\in\text{P}} \text{t}_\text{k}(\text{x}) \tag{9}$$

$$\text{wei-T}_- = \frac{1}{\text{N}_-}\sum\nolimits_{\text{x}\in\text{N}} \text{t}_\text{k}(\text{x}) \tag{10}$$

$$\text{wei-IGIR} = \sqrt{\text{wei-T}_- * \text{wei-T}_+} \tag{11}$$

In the calculation of IGIR, the k-nearest neighbors of each sample in the dataset are calculated at first, and their neighboring class labels are retained. First, according to the calculation method in formula (8), the weights of the k nearest neighbors are gradually reduced to 0. The main reason for not using distance is that the distances between different samples and its neighbors are inconsistent. This will result in the inconsistency of the weights of each sample in the calculation process, therefore, it is not possible to have a comparative standard for the overall results; second, to describe the dataset reasonably with an absolute measurement, and to avoid the relativity in the original GIR and spired by the definition of geometric mean, IGIR is defined as the compound measurements of positive and negative subsets. In this case, to ensure that the order of magnitude is unchanged, it is processed by prescribing to better measure the difficulty of classification of the dataset.

Algorithm 1: Computing the Improved Generalized Imbalance Ratio IGIR

Input: A dataSet X,label Y,number of nearest neighbors k in k-NN
Output: IGIR

compute *weight* according to formula (8)
for x *in* X *with label* y_x **do**
    $M \leftarrow$ the k nearest neighbors of x
    $t_k(x) \leftarrow \frac{1}{k}\sum weight * IR(x, M)$
end
$wei - T_- \leftarrow \frac{1}{N_-}\sum t_k(x) * sgn(y_x == 0)$
$wei - T_+ \leftarrow \frac{1}{N_+}\sum t_k(x) * sgn(y_x == 1)$
IGIR $\leftarrow \sqrt{wei - T_- * wei - T_+}$
return IGIR

In the proposed method, firstly, calculate the *weight* according to formula (8), secondly, compute the k nearest neighbors of each sample and $t_k(x)$ for each sample, thirdly, compute the average $t_k(x)$ of the positive and negative subsets, finally, compute the IGIR according to the formula (11).

IGIR can be regarded as the average classification accuracy under a weighted k-NN. That is, the more neighbors of the same class in the sample, the more likely the sample is to be classified as the original classifier, then IGIR has the nature to be related to the final classification performance.

4 Experimental Results

4.1 Datasets

The experimental data in this paper comes from the UCI machine learning database [14]. Some of them are multi-class datasets, in order to obtain a harder dataset to classify, we select one of the class as the minority class, and the rest of classes are regarded as the majority, and the details are shown as Table 4.

4.2 Evaluation

In the binary imbalanced classification, the confusion matrix is often used to evaluate the performance of the classifier, which is defined in Table 2:

Table 2. Confusion metrics

	Positive prediction	Negative prediction
Positive class	True positive (TP)	False negative (FN)
Negative class	False positive (FP)	True negative (TN)

FN represents the number of positive samples that are incorrectly classified as negative, and FP is the number of samples that are incorrectly classified as positive, there have been compound evaluations, such as F-value and Gmean [15].

$$\text{sensitivity} = \frac{TP}{TP + FP} \tag{12}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{13}$$

$$\text{F-value} = \frac{(1 + \beta^2) \times \text{recall} \times \text{precision}}{\beta^2 \times \text{recall} + \text{precision}} \tag{14}$$

$$\text{Gmean} = \sqrt{\frac{\text{TP}}{\text{TP} + \text{FN}} \times \frac{\text{TN}}{\text{TN} + \text{FP}}} \tag{15}$$

4.3 Experimental Settings and Results

Set $\beta = 1$ in F-value called F1_min, all involved k-NN are set with $k = 5$, the classifier is C4.5, all results are the average of 10 times of 10-fold cross-validation.

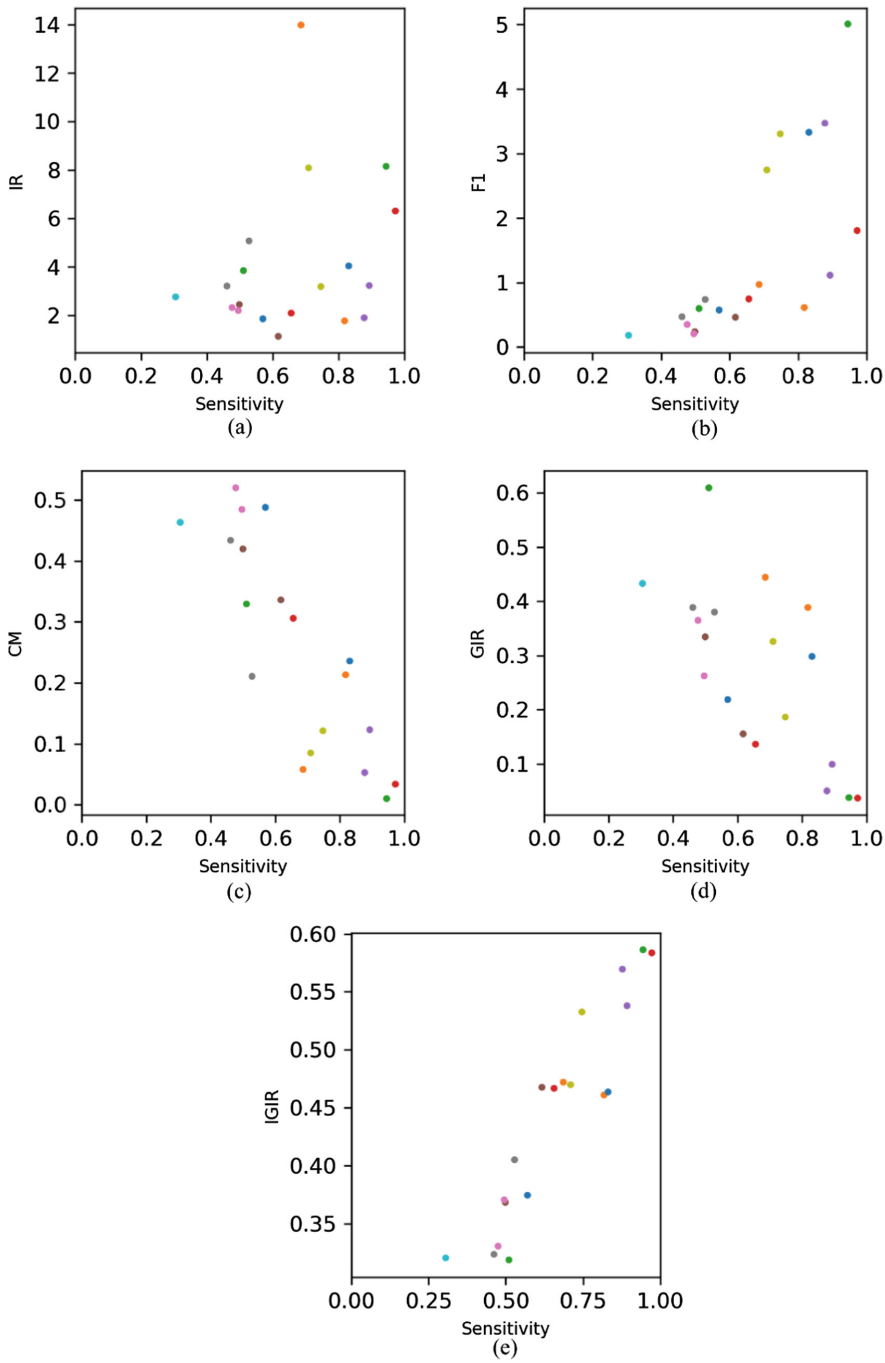


Fig. 2. Measurements and sensitivity

The classification results and measurements of different datasets are shown in Table 5, as we can see from the table, the IR and F1 have no value restriction, so the value can be very huge, while the CM, GIR and IGIR are limited in [0, 1], the scatter plots are used to show a more clear relation between the measurements and classification as shown in Fig. 2. Taking the sensitivity of minority class as an example, the Fig. 2 shows the relationship between different measurements and classification results. It can be seen that the correlation between CM and IGIR have a stronger linear relation with sensitivity as the measurement while there is no obvious trend in the rest measurements. In addition, the points in CM are more dispersed and the ones in IGIR are more concentrated, which means datasets with the same IGIR are more likely to have the same degree of classification difficulty than those with the same CM.

4.4 Analysis

In order to quantitatively analyze the relationship between different measurements and the classification results, the results are further analyzed by the determination coefficient R^2. R^2 reflects how many percentages of the fluctuation of Y can be described by the fluctuation of X. That is to say, what percentage of the variance of the representation variable Y can be explained by the controlled variable X.

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{16}$$

Where SST = SSR + SSE, SST represents the total sum of squares, SSR represents the regression sum of squares, and the SSE represents the error sum of squares.

The R^2 in Table 3 also shows the superiority of IGIR. The IGIR proposed in this paper is more capable to indicate the classification results, and it has a stronger relevance with the final classification performance and can be a better indicator of the sampled subset in resampling methods.

In IGIR, we calculate the number of samples of the average k-nearest neighbors by each sample, so the calculated value can be considered as the probability that the sample is classified as its own class. To a certain extent, this measurement can be regarded as Gmean under the k-NN classifier, and it is reasonable to indicate the classification performance of other classifiers.

Table 3. R^2 of measurements and classification results.

	F1_min	Gmean	Sensitivity
IGIR	**0.92**	**0.88**	**0.93**
IR	0.18	0.34	0.28
GIR	−0.70	−0.58	−0.67
CM	−0.80	−0.85	−0.84
F1	0.70	0.70	0.71

Table 4. Datasets

Datasets	Samples	Attributes	Target	Minority
breasttissue	106	9	carcinoma	21
breastw	699	9	malignant	241
diabetes	768	8	0	268
german	1000	24	2	300
glass	214	9	1 2 3	51
haberman	306	3	1	81
ionosphere	351	34	G	126
movement	360	90	1	24
satimage	6435	36	4	703
segment-challenge	1500	19	brick face	205
sonar	208	60	R	97
spect	267	22	1	55
vehicle	846	18	van	199
vertebral	310	6	AB	100
wpbc	198	33	1	47
yeast0	1484	8	0	244
yeast1	1484	8	1	429
yeast2	1484	8	2	463
yeast6	1484	8	6	163

Table 5. Measurements and classification results

	IR	GIR	CM	F1	IGIR	F1_min	Gmean	Sensitivity
breasttissue	4.05	0.30	0.24	3.33	0.46	0.78	0.83	0.83
breastw	1.90	0.05	0.05	3.47	0.57	0.91	0.92	0.88
diabetes	1.87	0.22	0.49	0.58	0.37	0.57	0.66	0.57
german	2.33	0.37	0.52	0.35	0.33	0.47	0.60	0.48
glass	3.20	0.19	0.12	3.31	0.53	0.75	0.80	0.75
haberman	2.78	0.43	0.46	0.18	0.32	0.24	0.35	0.30
ionosphere	1.79	0.39	0.21	0.61	0.46	0.84	0.87	0.82
movement	14.00	0.44	0.06	0.98	0.47	0.59	0.77	0.69
satimage	8.15	0.04	0.01	5.01	0.59	0.95	0.97	0.94
Segment*	6.32	0.04	0.03	1.81	0.58	0.97	0.98	0.97
sonar	1.14	0.16	0.34	0.46	0.47	0.59	0.58	0.62
spect	3.85	0.61	0.33	0.60	0.32	0.50	0.65	0.51
vehicle	3.25	0.10	0.12	1.12	0.54	0.88	0.92	0.89
vertebral	2.10	0.14	0.31	0.75	0.47	0.67	0.71	0.66
wpbc	3.21	0.39	0.43	0.47	0.32	0.42	0.56	0.46
yeast0	5.08	0.38	0.21	0.74	0.41	0.49	0.68	0.53
yeast1	2.46	0.34	0.42	0.24	0.37	0.48	0.62	0.50
yeast2	2.21	0.26	0.48	0.21	0.37	0.49	0.61	0.49
yeast6	8.10	0.33	0.08	2.75	0.47	0.69	0.82	0.71

5 Conclusion

In this paper, an improved measurement for imbalanced datasets is proposed, it takes the distribution information into consideration and it is based on the idea that a sample surrounded by more same class samples is easier to classify, for each sample of different classes, the proposed method calculates the average number of the k nearest neighbors in the same class in different subsets under the weighted k-NN, after that, the product of these average values is regarded as the measurement of this dataset. The experimental results show that the proposed measurement has a higher correlation with the classification results and can be used in the sampling algorithm. The future work will be sampling algorithms based on this measurement to improve the classification results.

Acknowledgment. This study was supported by the Shenzhen Research Council (Grant No. JSGG20170822160842949, JCYJ20170307151518535, JCYJ20160226201453085, JCYJ20170307151831260).

References

1. Wang, Y., Li, X., Tao, B.: Improving classification of mature microRNA by solving class imbalance problem. Sci. Rep. **6**, 25941 (2016)
2. Stegmayer, G., Yones, C., Kamenetzky, L., Milone, D.H.: High class-imbalance in pre-miRNA prediction: a novel approach based on deepSOM. IEEE/ACM Trans. Comput. Biol. Bioinf. **14**(6), 1316–1326 (2017)
3. Leichtle, T., Geiß, C., Lakes, T., Taubenböck, H.: Class imbalance in unsupervised change detection – a diagnostic analysis from urban remote sensing. Int. J. Appl. Earth Obs. Geoinf. **60**, 83–98 (2017)
4. Li, C., Liu, S.: A comparative study of the class imbalance problem in Twitter spam detection. Concurr. Comput. Pract. Exp. **30**(4) (2018)
5. Singh, S., Liu, Y., Ding, W., Li, Z.: Empirical evaluation of big data analytics using design of experiment: case studies on telecommunication data (2016)
6. Hale, M.L., Walter, C., Lin, J., Gamble, R.F.: A priori prediction of phishing victimization based on structural content factors (2017)
7. Anwar, N., Jones, G., Ganesh, S.: Measurement of data complexity for classification problems with unbalanced data. Stat. Anal. Data Min. **7**(3), 194–211 (2014)
8. Tang, B., He, H.: GIR-based ensemble sampling approaches for imbalanced learning. Pattern Recogn. **71**, 306–319 (2017)
9. Ho, T.: A data complexity analysis of comparative advantages of decision forest constructors. Pattern Anal. Appl. **5**(2), 102–112 (2002)
10. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**(1), 321–357 (2002)
11. Han, H., Wang, W.-Y., Mao, B.-H.: Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In: Huang, D.-S., Zhang, X.-P., Huang, G.-B. (eds.) ICIC 2005. LNCS, vol. 3644, pp. 878–887. Springer, Heidelberg (2005). https://doi.org/10.1007/11538059_91
12. Zhang, M.: Foundations of Modern Analysis. Academic Press, London (1960)

13. Weiss, G.M.: Learning with rare cases and small disjuncts. In: Twelfth International Conference on Machine Learning, pp. 558–565 (1995)
14. Zhang, H., Wang, Z.: A normal distribution-based over-sampling approach to imbalanced data classification. In: International Conference on Advanced Data Mining and Applications, pp. 83–96 (2011)
15. Li, D.C., Hu, S.C., Lin, L.S., Yeh, C.W.: Detecting representative data and generating synthetic samples to improve learning accuracy with imbalanced data sets. PLoS ONE **12**(8), e0181853 (2017)
16. Moreo, A., Esuli, A., Sebastiani, F.: Distributional random oversampling for imbalanced text classification, pp. 805–808 (2016)
17. Amini, M.R., Usunier, N., Goutte, C.: http://archive.ics.uci.edu/ml/datasets.html. Accessed 22 Mar 2018
18. Menardi, G., Torelli, N.: Training and assessing classification rules with imbalanced data. Data Min. Knowl. Disc. **28**(1), 92–122 (2014)

A Big Data Analytical Approach to Cloud Intrusion Detection

Halim Görkem Gülmez, Emrah Tuncel, and Pelin Angin(✉)

Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
{gorkem.gulmez,emrah.tuncel}@metu.edu.tr, pangin@ceng.metu.edu.tr

Abstract. Advances in cloud computing in the past decade have made it a feasible option for the high performance computing and mass storage needs of many enterprises due to the low startup and management costs. Due to this prevalent use, cloud systems have become hot targets for attackers aiming to disrupt reliable operation of large enterprise systems. The variety of attacks launched on cloud systems, including zero-day attacks that these systems are not prepared for, call for a unified approach for real-time detection and mitigation to provide increased reliability. In this work, we propose a big data analytical approach to cloud intrusion detection, which aims to detect deviations from the normal behavior of cloud systems in near real-time and introduce measures to ensure reliable operation of the system by learning from the consequences of attack conditions. Initial experiments with recurrent neural network-based learning on a large network attack dataset demonstrate that the approach is promising to detect intrusions on cloud systems.

Keywords: Intrusion detection · Cloud · Recurrent neural networks
Big data analytics

1 Introduction

Advances in the networking and virtualization technologies in the past decade have made cloud platforms a popular choice for the data storage and computing needs of many enterprises. Many companies have moved their infrastructures to major cloud platforms for improved reliability. As cloud services have gotten popular and common with decreased costs, they have also become hot targets for attacks. The rise of the Internet of Things (IoT) paradigm, which is closely connected to cloud computing, has increased the attack surface for cloud systems through increased connections and vulnerabilities attackers can exploit. The vulnerability of cloud platforms to attacks calls for security measures to prevent those attacks or detect them and take action when an attack happens. Although some of the existing security methods can be applied to cloud systems, new methods designed truly for the cloud systems would be more preferable.

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 377–388, 2018.
https://doi.org/10.1007/978-3-319-94295-7_26

Cloud systems are prone to zero-day attacks that are malicious activities not observed previously. Even though security mechanisms are updated quickly, even seconds are important to provide high reliability in such systems. Hence, a real-time security system which is not signature-based is more suitable for the cloud. The system should be able to detect both old and new attack methods as fast as possible. As the system will be under new attacks everyday, it is important that the system will easily adapt to those attacks. Unsupervised learning methods will enable the system to handle new attacks and learn from them.

Not unlike cloud computing, big data has become a popular concept with the technological advances in the past decade. Today, many different frameworks and tools exist to handle different types of big data [18]. The ability to store and process data in large volumes and velocity provides a significant means for analyzing cloud data in real time to detect anomalies that could signal presence of attacks on the cloud system. In this work, we propose a big data analytics approach for intrusion detection in cloud systems, based on recurrent neural networks that are capable of incorporating the temporal behavior of the system into the anomaly detection task. Since we want our system to respond in near real-time, we use streaming data from cloud platforms. An initial prototype of the proposed system using Apache Spark and the TensorFlow machine learning framework for recurrent neural networks has been developed and promising results have been achieved with experiments on a large network attack dataset.

2 Related Work

Network intrusion detection has been a well studied topic with many different approaches proposed along the years. While some of these solutions can also be applied in a cloud environment, because of the different characteristics of the cloud environments and comparatively larger network traffic volumes, several new approaches are suggested to solve the problem of intrusion detection in the cloud. In a cloud environment, intrusion detection monitors can be deployed at various locations. With the usage of virtualization techniques, the intrusion detection monitors can be deployed in a guest VM or in virtual machine monitors [12,17]. Beyond host or network device monitoring, distributed collaborative monitoring approaches are also utilized to catch system-wide attacks [4]. In these types of solutions, infrastructure problems need to be solved, since the system must support massive amounts of data gathered from different monitors and these data must be processed quickly to detect possible attacks as soon as possible.

Recent works have proposed using big data processing approaches to solve the problem of intrusion detection in cloud environments [5]. To detect a possible attack using intrusion detection systems (IDS), basically two techniques can be used [13]: In misuse detection, the IDS knows about previous attack patterns and tries to catch an attack by comparing the collected data with previous patterns. In anomaly detection, the IDS does not know about any previous attacks and tries to find anomalies in the network data, which could be possible signs of

attacks. In recent years, machine learning approaches have been used successfully for both of these techniques [9,10,16].

Recent years show many promising results of applying deep learning methods to machine learning problems and intrusion detection is not an exception for this case. In [8,20], the authors propose applying recurrent neural networks to intrusion detection systems and got very promising results. These works only show that RNN could be used while detecting anomalies in related data and they do not propose a complete end-to-end intrusion detection system. Our approach differs from these previous approaches in that it attempts to build a self-healing cloud system through deep learning with recurrent neural networks, which integrates time dependencies between observations (data points) in the system into the learning process to provide a more accurate representation of the attack progression and normal system processes.

3 Proposed Approach

In this work, we propose a cloud intrusion detection system that works with real-time cloud data analytics to detect possible attacks and develop a resilience mechanism through deep learning with recurrent neural networks.

The solution involves the collection of system metrics from cloud platforms and near real-time processing of those metrics using big data analytics to discover anomalies. Metric collection is done by metric collection agents deployed in related parties like guest VMs. These data include network packets and other related metrics like VM usage metrics, HTTP server performance etc. After collection, these metrics are sent to a stream to be processed by a stream processing engine. The stream processing engine gathers the metrics inside the stream, considering their timestamps and processes these data by feeding them to a recurrent neural network trained previously. If the network finds an anomaly in the data, it labels it and triggers an alarm to inform the system administrators. The details of these steps are listed below.

3.1 Metric Collection

Popular cloud system providers such as AWS[1] share the statistics and state of their cloud systems through an API. These statistics contain utilization of CPU, RAM, disks, number of packets/bytes received/transmitted, and many other details about the current state of the system. In this work, we utilize guest VM agents for metric collection, since this approach does not depend on the cloud infrastructure and is more flexible than virtual machine monitor solutions. At the metric collection phase, the agents collect the required metrics from the guest VM like network flow, basic system usage metrics such as CPU utilization, disk read/write metrics etc. and usage metrics of applications that can affect the system performance. The metric collection agent has two components, the

[1] https://aws.amazon.com/.

producer and the consumer. The producer component gathers the system and application metrics from the VM using different interfaces. To achieve this, the producer must have a pluggable architecture that written plug-ins can gather the metrics from, knowing how to get them. The responsibility of the consumer side is to gather metric records from the producer and pass them onto the processing phase.

3.2 Metric Processing

Due to the large volume and velocity of the data collected from cloud systems, big data processing frameworks are needed to analyze the data. Big data can be processed as batches or as streams. Deciding which type of processing is needed is up to the task. If we handle the data as batches, we need to wait for some amount of time to create batches from the given data. After the data become batch, the processing starts. This contradicts with our purpose of near real-time detection in this work, as we need to act in real time in order to prevent or stop attacks before they can harm the cloud system. Stream processing on the other hand involves handling the data in memory as they arrive.

In this work, we utilize Apache Spark [2] to process the stream data collected from cloud systems. Spark has advantages like fault-tolerance, in-memory computation, being faster than similar frameworks, having a wider community, multiple language support etc. The data that streams from our cloud systems are handled by Spark and served to our algorithm in order to detect possible attacks. Multiple networks can be watched by using this framework.

To support stream processing, many tools are available to specifically handle the requirements of this process. Tools like Apache Kafka [1] and Amazon Kinesis [3] streams provide great support for handling stream data in a scalable way. In this solution, we use Apache Kafka to collect the metrics from the guest VM agents and pipe them to the stream processing engine.

3.3 RNN-Based Learning for Anomaly Detection

Signature-based intrusion detection systems rely on detailed information about previously observed attacks. These approaches fail in the case of cloud systems, which are open to attacks that might be novel. On the other hand, unsupervised learning methods enable us to prevent or at least detect changes in the system parameters, i.e. the normal behavior of the system. By this way, the system will be able to detect anomalies and will try to prevent if there is an attack going on. In the mean time, alarms will be created in the system so that if the security system cannot stop the attack, it will warn the user/owner of the cloud system. This is actually the main difference from a signature-based intrusion detection system. If this type of system does not have any information about an attack, it will most likely be missed. On the other hand, for a system with an unsupervised learning algorithm, even a minor anomaly might cause the system to detect if something is wrong. When run on isolated data points/cloud activity logs, unsupervised algorithms may not achieve very high accuracy due to noise

in the data. For instance, observation of a sudden spike in CPU utilization might signal a possible attack even if it was caused by a legitimate process and does not persist for a long period, not causing any degradation in the performance of the system. Precisely for this reason, we need to be able to model the time-based behavior of the system by considering the data points collectively as a time series rather than isolated incidents.

Recent advances in deep neural networks have made it an effective tool for many supervised and unsupervised learning tasks, achieving higher accuracy than competing approaches. Recurrent neural networks (RNN) are machine learning models consisting of nodes that are connected to each other. These nodes can memorize and pass information in a sequence, though they process the data one by one. Therefore they can handle inputs and outputs that are dependent on each other. RNNs have been successful in various tasks such as image captioning, speech synthesis, time series prediction, video analysis, controlling a robot, translating natural language and music generation [11].

Normally, there is only one single network layer in a node of a classic RNN. In conventional neural networks, it is not defined how the network will remember events of the past to use the information about them in the future. Recurrent neural networks aim to solve this issue by using the architecture depicted in Fig. 1:

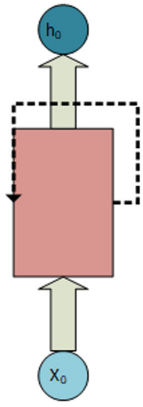


Fig. 1. An RNN loop

As shown in the diagram, the network gets an input x, processes it, and outputs an output h. The outcome of the process is used in the next step. To make it clear, the loop is demonstrated in an open form in Fig. 2.

The equation below shows the network mathematically:

$$h_t = \theta(Wx_t + Uh_{t-1})$$

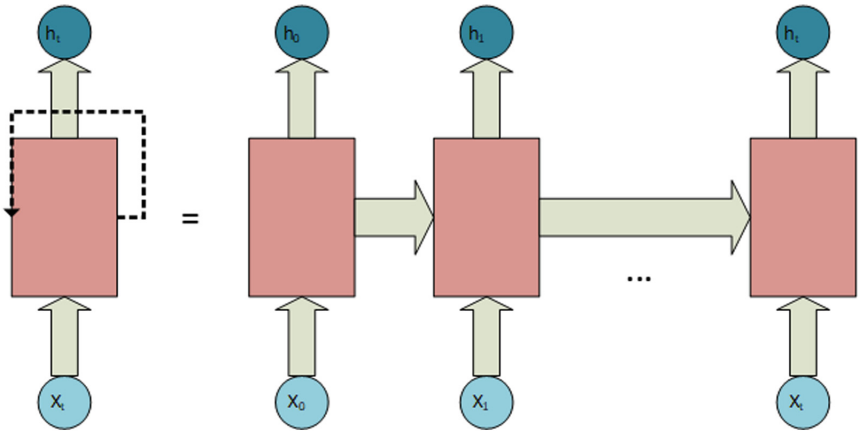


Fig. 2. An unrolled RNN loop

Here W stands for the weight matrix, which is multiplied by the input of the current time. The result is added to the multiplication of the output (hidden state) of the previous time step and its own hidden state and the hidden state matrix (transition matrix) U. These weight matrices are used to define how much of the information both from the current input and past output will be used to determine the current output. If they generate an error, it will be used to update the weights to minimize error. The resulting sum is condensed by the hyperbolic tangent function θ [6].

Some example uses of this standard RNN architecture include predicting the next character in a series of letters, picking the next note after a sequence of notes of a song, deciding where to go when controlling the motion of a robot etc. In our case, we use RNN in order to predict an intrusion, but we use LSTM-RNN because of the reasons explained below.

LSTM stands for Long Short Term Memory. Without it, gradients that are computed in training might get closer to zero (in case of multiplying values between zero and one) or overflow (in case of multiplying large values). In other words, as the time sequences grow, RNN might not connect older inputs to the outputs. LSTM adds additional gates to the architecture to control the cell state. By this modification, training over long sequences is not a problem anymore.

In an LSTM-RNN there are four layers, which interact with each other. First of all, the input is received and copies itself into four. The first one goes into a sigmoid layer. This layer decides whether the output of the previous layer is needed and should be used or it should be thrown away. Then another sigmoid layer decides which values are going to be updated. A tanh layer generates possible values, which might be included in the state. These two layers get combined to update the state. Finally another sigmoid layer picks what we are going to output from our cell state (Fig. 3).

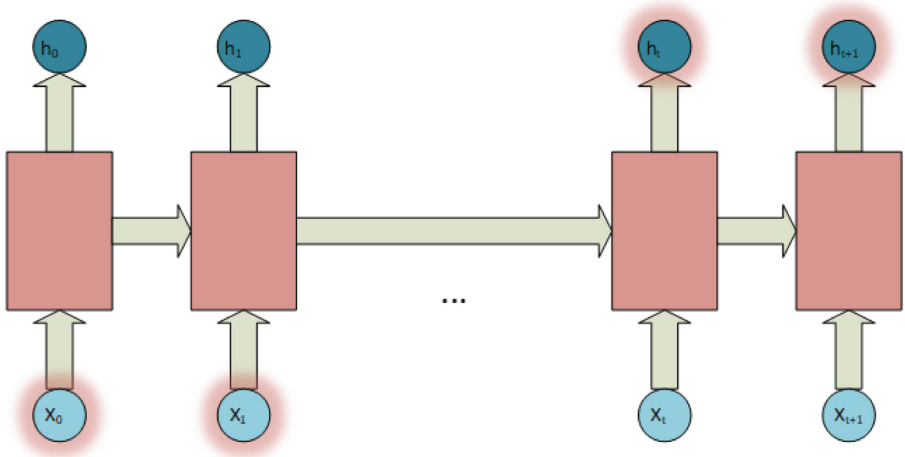


Fig. 3. Disconnected dependencies

In the proposed model, we utilize the LSTM recurrent neural networks (RNN) [7] algorithm to detect deviations from the normal behavior of the cloud system under monitoring. Note that because of the nature of the algorithm, it first needs to learn the normal state of the system. By processing the normal state, the system will detect anomalies when metric values that deviate significantly from the normal behavior of the system are observed. In RNNs, inputs are not independent, every time sequence uses information from the previous ones. This feature perfectly suits our task, as we cannot directly specify if there is an anomaly without analyzing the system's state for the time being (Fig. 4).

The algorithm receives parameters of the system from Spark and uses those parameters as a time series input. The parameters indicate the state of the system's properties for that time series. The algorithm then serves these parameters to its prediction function. The prediction function tries to find out if there is an anomaly in the system. For example, if there is an unrealistic peak in the CPU utilization and number of disk operations and incoming network packets, this might indicate that the system is under a denial of service attack. From this point, the system can create an alarm to warn system administrators or initiate a security action (Fig. 5).

We have used LSTM-RNN in Tensorflow. LSTM is actually handled by Tensorflow itself, but we needed to convert some of the fields in the data as we could not pass them directly to the algorithm. For example, fields like ip addresses, protocol types, service types etc. were converted to data types that LSTM-RNN accepts, as strings are not accepted. There was only one output for our experiment, which is the actual result: whether there was an attack (1) or not (0). How LSTM-RNN works in general is described below step by step [15]:

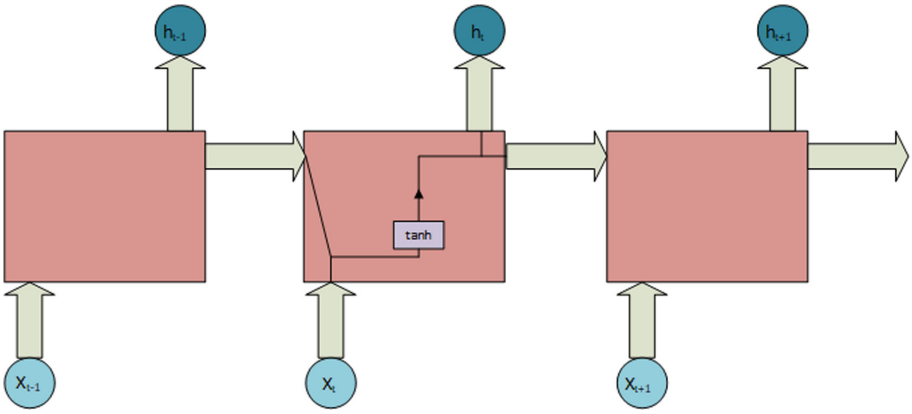


Fig. 4. Single layered structure of standard RNN

1. The first layer gets the current input and output of the past time series, then decides if the previous information is needed now. Actually this layer can be called the *forget* layer. h stands for the output of the past, x stands for the current input, W is the weight of this layer, and b is the bias.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

2. Then we move onto the input layer. This layer is another sigmoid layer, which decides the values that are going to be updated.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

3. A hyperbolic tangent layer creates candidate values, which might be included in the cell state. Cell state is a straight line in our network that flows for entire network. LSTM changes information on this state across the road with the help of the gates.

$$cdt_t = \tanh(W_{cdt}.[h_{t-1}, x_t] + b_{cdt})$$

4. Results of all previous steps are combined in order to create an update to the cell state.

$$c_t = f_t * c_{t-1} + i_t * cdt_t$$

5. Finally, the output is decided. Naturally, the cell state is used in deciding. Another sigmoid layer takes part, and its output is multiplied the by cell state (state will go into tanh first).

$$h_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) * \tanh(c_t)$$

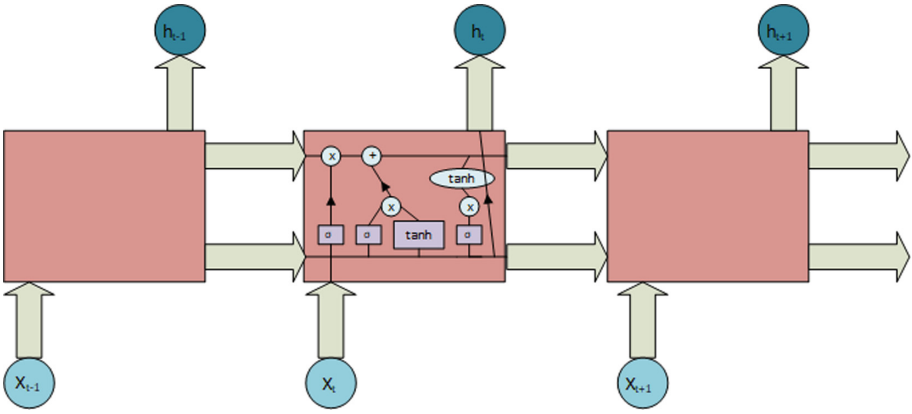


Fig. 5. Four layered structure of LSTM-RNN

4 Evaluation

We have developed an initial prototype of the proposed cloud IDS using Apache Spark collecting data from machine instances on AWS. Figure 6 demonstrates the high-level operation of the developed prototype.

To evaluate the accuracy of the RNN component in the proposed cloud intrusion detection system, we have used the "UNSW-NB15" dataset of UNSW ADFA [14] for experiments. In this dataset, there are attack and non-attack records of network traffic. The total number of records in the dataset is approximately two and a half millions, making it quite comprehensive. The test and training sets as partitions of the original dataset are also provided. The records have 49 fields in total, a subset of which is listed in Table 1 below, including information like source IP address, source port, protocol, number of packets on requests and responses, bytes of those packets, number of losses, attack type if there is one etc.

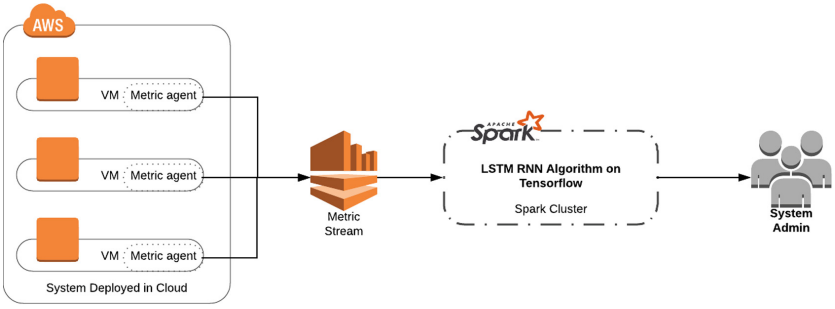


Fig. 6. Big data processing system operation

We have simulated the data so that the system behaves like the data are streaming. At first, a training set is provided for LSTM-RNN, which runs on TensorFlow [19], which is an open-source machine learning framework. With the help of the model that is created on the framework, the system decides whether a record seems like an attack or not. We ran the model using 50000 records for training, and 650000 records for testing. The error rate was below 7%, which is a promising accuracy for intrusion detection.

Table 1. UNSW-NB15 parameters

No.	Name	Type	Description
1	srcip	Nominal	Source IP address
2	sport	Integer	Source port number
3	dstip	Nominal	Destination IP address
4	dsport	Integer	Destination port number
5	proto	Nominal	Transaction protocol
6	state	Nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	Nominal	http, ftp, smtp, ssh, dns, ftp-data, irc and (-) if not much used service
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	Integer	Source to destination packet count
18	Dpkts	Integer	Destination to source packet count
19	swin	Integer	Source TCP window advertisement value
20	dwin	Integer	Destination TCP window advertisement value
21	stcpb	Integer	Source TCP base sequence number
22	dtcpb	Integer	Destination TCP base sequence number
...	...	...	...
48	attack cat	Nominal	The name of each attack category. In this data set, nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
49	Label	Binary	0 for normal and 1 for attack records

5 Conclusion

In this work, we proposed an intrusion detection system for cloud computing that leverages big data analytics with recurrent neural networks. We have implemented an initial prototype of the proposed architecture with Apache Spark and evaluated the performance of recurrent neural networks on a network attack dataset, which provided promising attack detection accuracy results. We have used LSTM-RNN algorithm in Tensorflow to detect attacks in the network. LSTM-RNN has been picked as we were interested with using time series data. Results of this work looked promising about integrating big data analytics to intrusion detection in cloud. Our future work will focus on the integration of a deep reinforcement learning based model, which will provide automated self-healing of cloud systems by learning from the consequences of attacks on the system, thereby avoiding states that are vulnerable to attacks. We will also experiment with real data streams gathered using cloud APIs and quantify the delays of big data processing as well as the mean time to detect attacks and recover in major cloud platforms. Extensive experimentation with various attack types on cloud systems will be conducted to evaluate and optimize the big data analytics-based intrusion detection framework.

Acknowledgement. This work was supported by Middle East Technical University under grant number: YOP-312-2018-2819.

References

1. Apache: Apache kafka, a distributed streaming platform. https://kafka.apache.org/. Accessed Apr 2018
2. Apache: Apache spark, lightning-fast unified analytics engine. https://spark.apache.org/. Accessed Apr 2018
3. AWS: Amazon kinesis. https://aws.amazon.com/kinesis/. Accessed Apr 2018
4. Bharadwaja, S., Sun, W., Niamat, M., Shen, F.: Collabra: a xen hypervisor based collaborative intrusion detection system. In: 2011 Eighth International Conference on Information Technology: New Generations, pp. 695–700, April 2011. https://doi.org/10.1109/ITNG.2011.123
5. Casas, P., Soro, F., Vanerio, J., Settanni, G., D'Alconzo, A.: Network security and anomaly detection with big-DAMA, a big data analytics framework. In: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), pp. 1–7, September 2017. https://doi.org/10.1109/CloudNet.2017.8071525
6. DL4J: A beginner's guide to recurrent networks and LSTMs. https://deeplearning4j.org/lstm.html. Accessed Apr 2018
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
8. Kim, J., Kim, H.: Applying recurrent neural network to intrusion detection with Hessian free optimization. In: Kim, H., Choi, D. (eds.) WISA 2015. LNCS, vol. 9503, pp. 357–369. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31875-2_30

9. Kumar, R.S.S., Wicker, A., Swann, M.: Practical machine learning for cloud intrusion detection: challenges and the way forward. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec 2017, pp. 81–90. ACM, New York (2017). https://doi.org/10.1145/3128572.3140445
10. Li, Z., Sun, W., Wang, L.: A neural network based distributed intrusion detection system on cloud platform. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, vol. 01, pp. 75–79, October 2012. https://doi.org/10.1109/CCIS.2012.6664371
11. Lipton, Z., Elkan, C., Berkowitz, J.: A critical review of recurrent neural networks for sequence learning. Accessed Apr 2015
12. Maiero, C., Miculan, M.: Unobservable intrusion detection based on call traces in paravirtualized systems. In: Proceedings of the International Conference on Security and Cryptography, pp. 300–306, July 2011
13. Mishra, P., Pilli, E.S., Varadharajan, V., Tupakula, U.: Intrusion detection techniques in cloud environment: a survey. J. Netw. Comput. Appl. **77**, 18–47 (2017). https://doi.org/10.1016/j.jnca.2016.10.015. http://www.sciencedirect.com/science/article/pii/S1084804516302417
14. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Proceedings of the Military Communications and Information Systems Conference, pp. 1–6 (2015)
15. Olah, C.: Understanding LSTM networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed Apr 2018
16. Pandeeswari, N., Kumar, G.: Anomaly detection system in cloud environment using fuzzy clustering based ANN. Mob. Netw. Appl. **21**(3), 494–505 (2016). https://doi.org/10.1007/s11036-015-0644-x
17. Sharif, M.I., Lee, W., Cui, W., Lanzi, A.: Secure in-VM monitoring using hardware virtualization. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 477–487. ACM, New York (2009). http://doi.acm.org/10.1145/1653662.1653720
18. Sinanc, D., Demirezen, U., Sagiroglu, S.: Evaluations of big data processing. Serv. Trans. Big Data **3**, 44–54 (2016). https://doi.org/10.29268/stbd.2016.3.1.4
19. TensorFlow: an open source machine learning framework for everyone. https://www.tensorflow.org/. Accessed Apr 2018
20. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017). https://doi.org/10.1109/ACCESS.2017.2762418

Short Track

Context Sensitive Efficient Automatic Resource Scheduling for Cloud Applications

Lun Meng[1] and Yao Sun[2(✉)]

[1] College of Public Administration, Hohai University, Nanjing 210098, China
[2] School of Software Engineering, Jinling Institute of Technology, Nanjing 211169, China
suny2l6@jit.edu.cn

Abstract. Nowadays applications in cloud computing often use containers as their service infrastructures to provide online services. Applications deployed in lightweight containers can be quickly started and stopped, which adapt to changing resource requirements brought from workload fluctuation. Existing works to schedule physical resources often model applications' performance according to a training dataset, but they cannot self-adapt dynamic deployment environment. To address this above issue, this paper proposes a feedback based physical resource scheduling approach for containers to deal with the significant changes of physical resources brought from dynamic workloads. First, we model applications' performance with a queue-based model, which represents the correlations between workloads and performance metrics. Second, we predict applications' response time by adjusting the parameters of the application performance model with a fuzzy Kalman filter. Finally, we schedule physical resources according to the predicted response time. Experimental results show that our approach can adapt to dynamic workloads and achieve high performance.

Keywords: Resource scheduling · Cloud application · Fuzzy logic Kalman filter

1 Introduction

Nowadays, applications in cloud computing are often composed with fine-grained isolated service components that use lightweight communication protocols to cooperate with each other. These autonomous components update independently, which meets the changing requirements of cloud applications. Furthermore, different components of an application have various resource requirements, so dynamic resource scheduling for components is important for cloud applications, especially in the real scenario with significant fluctuate workloads. Cloud applications often use containers (e.g., Docker) as their service infrastructures to provide online cloud-based services. A container is a lightweight stand-alone executable software package that includes everything needed to run it, which isolates applications from its surroundings and avoids conflicts between teams running different applications on the same infrastructure. An application deployed in a container always runs the same regardless of environment. Lightweight

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 391–397, 2018.
https://doi.org/10.1007/978-3-319-94295-7_27

containers can be quickly started and stopped, so it can adapt to changing resource requirements brought from workload fluctuation in cloud computing. Existing works often model an application's performance, and then schedule resources with physical hosts or virtual machines. However, modeling an application's performance often requires the domain knowledge of the application. Moreover, since they set the parameters of the application's performance model according to a training dataset, these approaches cannot self-adapt to dynamic deployment environment. The contributions of this paper are listed as follows:

- We model applications' performance with a Jackson network model, which automatically presents the correlations between workloads and performance metrics without domain knowledge and human intervention.
- We predict workloads with a fuzzy Kalman filter, which can conveniently adapt to dynamic deployment environments by adjusting the parameters of applications' performance model.
- We have implemented a prototype system and conducted extensive experiments in real scenarios to validate the proposed approach in predicting workloads and scheduling resources.

2 Resource Scheduling Approach

2.1 Queueing Based Performance Model

Response time is a key QoS (Quality of Service) metric of cloud applications, which is often decided by dynamic workloads and allocated physical resources. We use a queue-based model based on a Jackson network to model applications' performance, and the model presents the correlations among workloads, resources and response time; use a Kalman filter to predict workloads and response time; use a feedforward controller based on fuzzy logic to rectify the parameters of the filter to precisely predict the response time.

Processing requests includes that a user sends a request to an application, the request is capsuled as an event, the event is processed by many components in sequence, and then a response is sent back to the user. When one component has many instances, the application uses a round robin scheduling strategy to process the request. We model a cloud application composed of many components with a Jackson queueing network, where f is a user's request flow, j is an application component, i is the ith instance of the jth component. Each request flow is processed by n components $j_1, j_2, \ldots, j_n$, each component j has m instances $j(i_k)$ $(1 \leq k \leq m)$, and each component runs in a container. Various components have different resource requirements (e.g., CPU and I/O intensive), and we define the resource preference (rp_i) of component j as the resource type with the highest resource utilization; u_j $(0 \leq u_j \leq 1)$ is the resource utilization of rp_j, and u_{0j} is the resource utilization of rp_j with no workload; r_{ji} is the concurrency number (i.e., the number of requests in a second) of instance i of component j, which fits for the Poisson distribution; T_{ji} is the processing time of instance i of component j, and T_j is the average processing time of component j; d is the

transmitting time of a request flow f; B is the response time of a request flow f; τ_j is the correlation coefficient between workloads and resource utilization. Then, we get a Jackson network flow, and the performance formula is described as follows:

$$u_j = \sum\nolimits_i (u_{0j} + \tau_j \times \gamma_{ji} \times T_{ji}),$$

$$B = d + \sum\nolimits_j \frac{T_j}{1 - u_j/m_j},$$

where we can get u_j, u_{0j}, r_{ji} and B by monitoring applications; get τ_j from experience with historical data; predict T_{ji} and d. We aim at provisioning resources to guarantee B in a predefined range.

2.2 Kalman Filter Based Response Time Prediction

The Kalman filter is an optimal linear state estimation approach, which recursively predicts the next value using the previous predicted value and the current monitoring data instance [1]. The Kalman filter formula are as follows:

$$X(k+1) = F(k)X(k) + Q(k),$$

$$Z(k) = H(k)X(k) + R(k),$$

where $\mathrm{X}(k) = (T_j, d)^T \forall j$ is a prediction matrix representing the processing times of components; $\mathrm{Z}(k)$ is the observed matrix of $X(k)$; $\mathrm{H}(k) = (u_j, u_{0j}, \gamma_{ji}, B)^T \forall i$ represents the workloads, resource utilization and response time; $\mathrm{Q}(k)$ is a process excitation noise covariance matrix, which fits for $Q(k) \sim N(0, Q)$; $\mathrm{R}(k)$ is a measurement noise covariance matrix, which fits for $R(k) \sim N(0, R)$. $\mathrm{Q}(k)$ and $\mathrm{R}(k)$ are generally set as zero mean white noise [2]. However, uncertain workloads change over time, so $\mathrm{Q}(k)$ and $\mathrm{R}(k)$ should adapt to dynamic workloads as follows:

$$Q_k = TQ,$$
$$R_k = UR,$$

where T and U are time-varying adjustment parameters, and then we can get the prediction formula as follows:

$$\overline{X}(k+1|k) = F(k+1|k)\overline{X}(k|k),$$

$$\overline{Z}(k+1|k+1) = H(k)\overline{X}(k+1|k),$$

where $\overline{X}(k+1|k)$ is the predicted time of processing requests, and $\overline{Z}(k)$ is the predicted value of the status, and then we can predict the response time as follows:

$$\overline{X}(k+1|k+1) = \overline{X}(k+1|k) + K(k+1) \times (Z(k+1|k+1) - \overline{Z}(k+1|k+1)),$$
$$K(k+1) = P(k+1|k)H(k)^T (H(k)$$
$$P(k+1|k))H(k)^T + T(k+1)R)^{-1},$$
$$P(k+1|k+1) = (I_n - K(k+1)H(k))P(k+1|k),$$
$$P(k+1|k) = F(k+1)P(k|k)F(k+1|k)^T + U(k+1)Q.$$

The Kalman filter rectifies the predicted value with the last calculated value in each iteration, so it does not require historical data instances. However, the typical Kalman filter cannot process the irregular time-varying data instances, so it requires feedback-based methods to online adjust noises and excitation matrixes. The residual error $r = Z(k) - \overline{Z}(k)$ represents that the deviation between the predicted value and the monitored one, and then we ought to modify the constructed model, if the deviation violates the predefined threshold.

2.3 Resource Scheduling

We online schedule physical resources based on the predicted response time. We initiate the Kalman filter by defining parameters as follows: the noise parameters T and U of the Kalman filter, the maximum and minimum of resource utilization, and the maximum response time of a service instance (Input); a fuzzy logic function adaptively adjusts the parameters of the EKF model; a response time function using EKF's outputs calculates response time; a migration function migrates containers; a expansion function expands containers; a contraction function contracts containers. Then, we predict response time as follows: model an application with a Jackson network queue; online predict response time with a Kalman filter; adjust the Kalman filter with a fuzzy function; estimate the response time based on the prediction model. Finally, we migrate, expand and contract a container as follows:

- Migrate containers: the resource utilization of a container does not reach the upper limit, but the resource utilization of the located physical host reaches to the upper limit. Then, the scheduler migrates the container to the host with sufficient physical resources. The scheduler persists a source container as an image reserving the application's state, destroys the source container, and then is initiates a target container from the image in the target host.
- Expand containers: the resource utilization of a container reaches the upper limit, and the schedule initiates new containers in selected physical hosts.
- Contract containers: the resource utilization of many instances of an application is lower than expected value, and then we reduce the number of instances.

3 Experiment

We set up a cluster composed of six hosts. Each host has an Intel Core i7 CPU 3.4 GHz, an 8 G memory and a gigabit ethernet network interface, and runs CenOS 7 and Docker 1.7. We have a Master node, four Slave nodes (i.e., Slave 1–4), and a

workload generator node running JMeter. We deploy a typical cloud application Web Serving in Cloudsuite (http://cloudsuite.ch//pages/benchmarks/webserving/) that is a benchmark suite for cloud services.

JMeter simulates workloads in a cyclical change model, a stable model, a steady rise model and a sudden decline model. Admission control is a widely used approach of scheduling resources. Since the admission controller takes a long period to converge, the fixed parameters cannot adapt to changing irregular workloads [3]. In the experiment, we compare our approach with the admission control approach. In initialization, we deploy six services on Slave 1 and the rest six services on Slave 2. As shown in Fig. 1, the experimental results are described as follows:

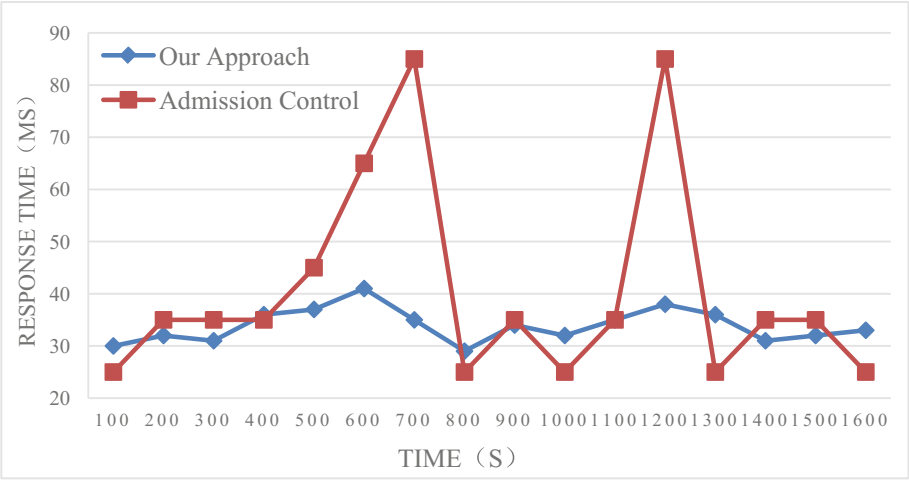


Fig. 1. Responses time

In the period of the 0–200th second, we simulate stable workloads with 300 visits per second to initialize the Kalman filter. In the period of the 200–300th second, we simulate dynamic workloads with 30–60 visits per second, and the response time increases. Since the resource utilization of services does not reach the upper limit, expansion does not occur. As the resource utilization of Slave 1 reaches the upper limit, three services on Slave 1 are migrated to Slave 3 and Slave 4, and the response time fluctuates in a narrow range. In the period of the 300–400th second, we simulate stable workloads with 50 visits per second, and the response time also stable. In the period of the 600–700th second, we simulate sudden rise workloads with 50–100 visits per second in the 650th second, some services on Slave 1 and Slave 2 are expanded, and the response time stably decreases. In the period of the 700–800th second, we simulate sudden decline workloads with 100–30 visits per second in the 750th second, and some services are contracted.

The experimental results show that the response time keeps stable in 30–40 micro seconds, which demonstrates that our approach can adapt to dynamic workloads.

4 Related Works

Existing works of provisioning resources often use machine learning, fuzzy logic and admission control to train models and calculate parameters. Lama and Zhou [4] predict the spike of applications' resource requirements, and provision physical resources in the quality of service (QoS) constraints. Cao et al. [5] estimate the supported workloads according to the allocated resources, and use an admission control method to guarantee QoS by denying overloading requests. Cherkasova and Phaal [6] model the correlations between workloads and throughput, estimate resource requirements, and accept requests with admission mechanism. Robertsson et al. [7] use a linearity model to predict resource utilization for guaranteeing QoS. Xu et al. [8] consider the performance overhead of VM and trace the changes of applications' resources in virtualization environment. These approaches train models and get parameters with a static training dataset, so cannot adapt to dynamic fluctuate workloads.

5 Conclusion

This paper proposes a feedback based physical resource scheduling approach for containers to deal with the significant changes of physical resources brought from dynamic workloads. First, we model applications' performance with a Jackson network model, which presents the correlations between workloads and performance metrics. Second, we predict applications' response time by adjusting the parameters of applications' performance model with a fuzzy Kalman filter. Finally, we schedule physical resources based on the predicted response time. Experimental results show that the maximum error of our approach in predicting response time is only about 10%.

Acknowledgment. This work was supported by the National Social Science Foundation of China (grant number 16CXW027), fundamental research funds for the central universities (grant 2014B00514) and the Ministry of Education of Humanities and Social Science project (grant 15YJCZH117).

References

1. Kalman, R.E.: A new approach to linear filtering and prediction problems. Trans. ASME-J. Basic Eng. **82**, 35–45 (1960)
2. Sinopoli, B., Schenato, L., Franceschetti, M., et al.: Kalman filtering with intermittent observations. IEEE Trans. Autom. Control **49**(9), 1453–1464 (2004)
3. Lu, C., Lu, Y., Abdelzaher, T.F., et al.: Feedback control architecture and design methodology for service delay guarantees in web servers. IEEE Trans. Parallel Distrib. Syst. **17**(9), 1014–1027 (2006)
4. Lama, P., Zhou, X.: Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters. IEEE Trans. Parallel Distrib. Syst. **23**(1), 78–86 (2012)
5. Cao, J., Zhang, W., Tan, W.: Dynamic control of data streaming and processing in a virtualized environment. IEEE Trans. Autom. Sci. Eng. **9**(2), 365–376 (2012)

6. Cherkasova, L., Phaal, P.: Session-based admission control: a mechanism for peak load management of commercial web sites. IEEE Trans. Comput. **51**(6), 669–685 (2002)
7. Robertsson, A., Wittenmark, B., Kihl, M., et al.: Design and evaluation of load control in web server systems. In: Proceedings of IEEE American Control Conference, vol. 3, pp. 1980–1985 (2004)
8. Xu, C.Z., Rao, J., Bu, X.: URL: a unified reinforcement learning approach for autonomic cloud management. J. Parallel Distrib. Comput. **72**(2), 95–105 (2012)

A Case Study on Benchmarking IoT Cloud Services

Kevin Grünberg and Wolfram Schenck(✉)

Faculty of Engineering and Mathematics, Center for Applied Data Science Gütersloh, Bielefeld University of Applied Sciences, Bielefeld, Germany
kevin_gruenberg@web.de, wolfram.schenck@fh-bielefeld.de

Abstract. The Internet of Things (IoT) is on the rise, forming networks of sensors, machines, cars, household appliances, and other physical items. In the industrial area, machines in assembly lines are connected to the internet for quick date exchange and coordinated operation. Cloud services are an obvious choice for data integration and processing in the (industrial) IoT. However, manufacturing machines have to exchange data in close to real-time in many use cases, requiring small round-trip times (RTT) in the communication between device and cloud. In this study, two of the major IoT cloud services, Microsoft Azure IoT Hub and Amazon Web Services IoT, are benchmarked in the area of North Rhine-Westphalia (Germany) regarding RTT, varying factors like time of day, day of week, location, inter-message interval, and additional data processing in the cloud. The results show significant performance differences between the cloud services and a considerable impact of some of the aforementioned factors. In conclusion, as soon as (soft) real-time conditions come into play, it is highly advisable to carry out benchmarking in advance to identify an IoT cloud workflow which meets these conditions.

Keywords: Cloud benchmarking · Internet of Things · IoT
Industrial IoT · Cyberphysical systems · Round-trip time
Microsoft Azure IoT Hub · Amazon Web Services IoT

1 Introduction

During the last decade, cloud computing has become a highly popular approach to provide scalable and flexible hardware and software infrastructure at various service levels (infrastructure as a service [IasS], platform as a service [PaaS], software as a service [SaaS]) [1]. With the Internet of Things (IoT) being on the rise, a new use case for cloud computing emerges. In the IoT, physical objects like sensors, machines, cars, household appliances, and other items are connected via the internet to enable interaction and cooperation of these objects. Application areas of the IoT are, among others, transportation, healthcare, smart homes and industrial environments [2]. IoT devices typically generate and transmit data at regular intervals. This data is integrated, processed, and monitored (e.g., via

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 398–406, 2018.
https://doi.org/10.1007/978-3-319-94295-7_28

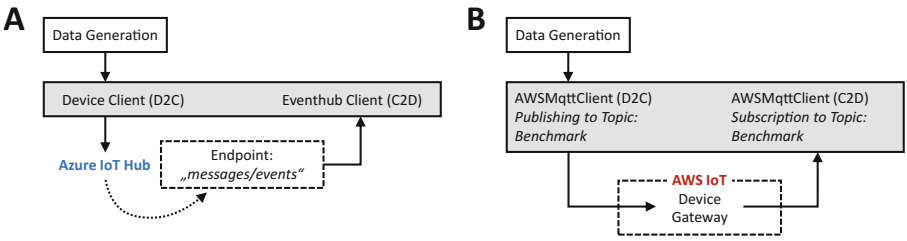


Fig. 1. Message flow between benchmarking applications (gray box incl. data generation) and cloud services. A: For MS Azure IoT Hub. B: For AWS IoT.

mobile devices or web applications), and commands for corrective actions (which are either manually or automatically generated) are sent back to the devices [3]. Furthermore, IoT devices may interact directly with each other.

It is an obvious approach to use the cloud for data integration and processing [4]. The major cloud service providers (CSP) have established meanwhile specialized services for data collection from IoT devices, supporting popular protocols like MQTT or AMQP. However, many application scenarios in the IoT—especially in the industrial area—rely on operations taking place in close to real-time: For example, when manufacturing machines are controlled via apps running in the cloud, or when these machines use the cloud as central instance for data exchange to coordinate their work in an assembly line. Therefore it is important that cloud platforms used for these purposes process and forward messages with small latency and consistent availability and performance.

Although many studies on cloud benchmarking in general exist, we are not aware of any study which addresses especially the performance of IoT cloud services. As a first step in this direction, we present benchmarking data collected from Microsoft Azure IoT Hub (MAIH) and Amazon Web Services IoT (AWSI) at three different locations in North Rhine-Westphalia (NRW; Germany) around Sept./Oct. 2017. This data collection was part of a larger research project for a local mechanical engineering company to integrate their products in the industrial IoT. Despite this local scope, we are convinced that some general conclusions can be drawn from our data.

2 Previous Work

Studies on cloud benchmarking address mostly the PaaS and IaaS levels. Concerning PaaS, Binnig et al. [5] point out that classic benchmarking approaches are not well suited for cloud APIs because resources are usually scaled up automatically with increasing load. They suggest to concentrate on scalability, cost, coping with peak load, and fault tolerance. Agarwal and Prasad [6] carried out a comprehensive benchmarking study on the storage services of the Microsoft Azure cloud platform, especially for high-performance computing. In their view, benchmarking is definitely required to understand the performance capabilities

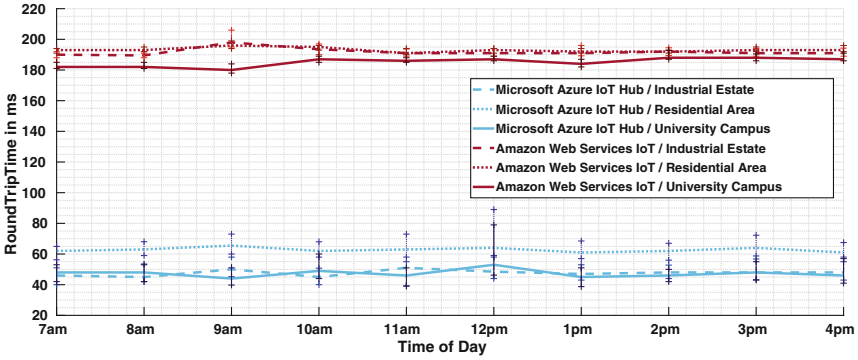


Fig. 2. RTT measurements, varying time of day and location. Each data point depicts the median value and interquartile range of around 100 measurements.

of the underlying cloud platforms because the cloud service offerings are usually only best-effort based without any performance guarantees. In a similar spirit, Kossmann et al. [7] compared the performance of major CSPs in transaction processing. Their results show that the services vary dramatically when it comes to end-to-end performance, scalability, and cost—although the price matrices of the CSPs are very similar in terms of network bandwidth, storage cost, and CPU cost (see also [8] for a holistic evaluation approach). The study by Bermbach and Wittern [9] is related to our work because the authors use response latency as performance measure for web API quality. They observed 15 different web APIs over a longer time period from different locations in the world. They results show that web API quality is very volatile over time and location, and that developers have to take this into account from an architectural and engineering point of view.

High variability is also observed at the IaaS level. Gillam et al. [10] and Zant and Gagnaire [11] carried out a large set of benchmarks on virtual machines (VM) of different specifications and from different CSPs. Even if the virtual resources are of the same specification, the observed variability is substantial (e.g., regarding CPU and memory performance). To help customers to choose the best cloud platform for their needs, benchmarking results could be made available in publicly available databases (as in [10]), or tools for easier benchmarking could be developed. Following the second approach, Cunha et al. [12] implemented "Cloud Crawler" for describing and automatically executing application performance tests in IaaS clouds for different virtual machine configurations.

3 Methods

Performance Measure. The quality of data transfer to and from cloud services can vary along two main dimensions: throughput (number and/or size of messages per time unit) and latency. Using throughput as performance measure,

Table 1. Mean values and standard deviations (in brackets) of the RTT median values (varying over time of day) shown in Fig. 2 to illustrate the systematic differences between locations. **Values given in ms.**

	Location		
	Industrial estate	Residential area	University campus
MS Azure IoT Hub	47.7 (2.0)	62.8 (1.4)	47.2 (2.7)
AWS IoT	191.8 (2.5)	199.1 (1.4)	185.4 (2.8)

however, is problematic because many cloud services automatically scale up the available communication resources with increasing load [13]. For this reason, and because we aim on real-time performance in the context of our study, which requires message transfers with low latency, we use the round-trip time (RTT) in our benchmarking. RTT is defined here as the time difference between sending a message to a communication endpoint (time of sending) and receiving the same message back from this endpoint (time of arrival).

Microsoft Azure IoT Hub (MAIH). The benchmarking application for MAIH was implemented using the .NET SDK provided by Microsoft [14]. On the client side, a so-called "DeviceClient" has to be created and initialized for sending messages to the IoT Hub (see Fig. 1A). Within the Azure cloud, messages are forwarded to an "EventHub" endpoint. Within the client, a subscription to this endpoint is created so that messages are automatically received back. (MAIH server location: Western Europe)

Amazon Web Services IoT (AWSI). The benchmarking application for AWSI has a similar structure but is based on the JAVA SDK provided by Amazon [15]. An important step is to register the device at the AWSI device gateway (see Fig. 1B) as a "thing" with a unique identifier ("topic") under which it is published. The same topic is used for the subscription to receive the messages back from the device gateway. (AWSI server loc.: Frankfurt/Main, Germany)

Messages. The content of the sent messages was a single boolean value[1] which was embedded into a JSON container (size: 250 Bytes). This container also contained the time of sending as timestamp. In most benchmarking tests, messages were sent every two seconds. To measure a single data point, about 100 messages were sent; the median and the interquartile range of this sample are shown in most of the figures in the results section. The benchmarking applications were run on a notebook with a quadcore CPU (Intel i7-4712MQ). Messages retrieved under a CPU load of more than 20% were discarded to eliminate CPU load as a cause of variability [13]. The percentage of discarded messages was in the low single-digit range. Messages were transferred via MQTT.

[1] Pretests did not show any significant RTT differences between different basic data types like `int`, `float`, etc.

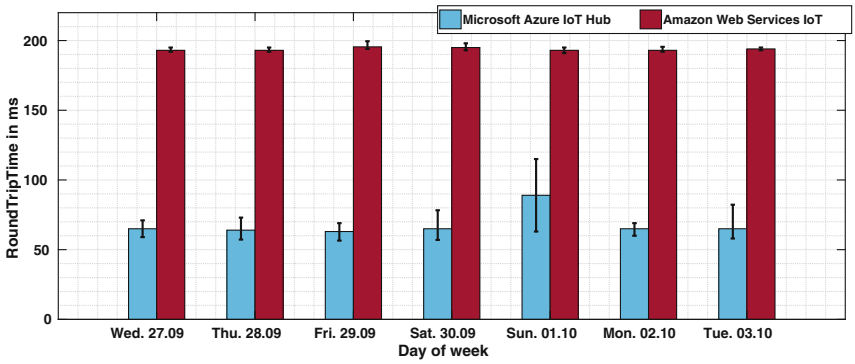


Fig. 3. RTT measurements, varying day of week. Each data point depicts the median value and interquartile range of around 100 measurements.

Locations. Benchmarking took place at three different locations in NRW (Germany): (1) At a mechanical engineering company within an industrial park in Rheda-Wiedenbrück (denoted as "industrial estate" in the following; connection to internet via optic fiber); (2) at a residential building in Herzebrock-Clarholz ("residential area"; connection to internet via DSL over regular copper phone lines); (3) at the main campus of Bielefeld University of Applied Sciences in Bielefeld ("university campus"; connection to internet via the German National Research and Education Network [DFN] [optic fiber]).

4 Results

Time of Day and Location. In the first set of benchmarks, time of day and location were varied systematically. Figure 2 shows considerable RTT differences between MAIH (mean median value: 52.5 ms) and AWSI (mean median value: 192.1 ms) while performance stays nearly constant during the day. Interquartile ranges vary during the day, and sometimes strong outliers with large RTT values were observed (not shown). Location has systematic impact too, with "university campus" exhibiting the smallest RTT values for both CSPs, followed by "industrial estate" and "residential area" (in this order; see Table 1). Measurements were carried out around midweek.

Day of Week. RTT measurements were repeated between Sept. 27th and Oct. 3rd at the residential area to check the impact of the day of the week. The results in Fig. 3 show small variability between the days of the week regarding median values with one notable exception: MAIH on Sunday with an RTT increase of about 30%. Because this effect does not appear in the AWSI measurements carried out at nearly the same time, it can be most likely attributed to the Azure platform or to the connecting network route.

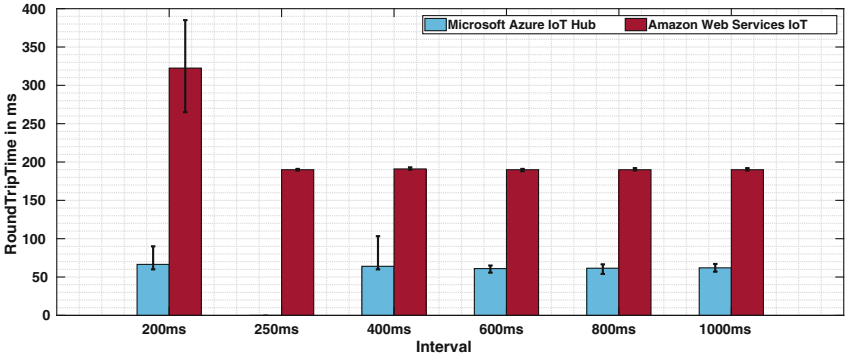


Fig. 4. RTT measurements, varying the time interval between sending messages. Each data point depicts the median value and interquartile range of around 100 measurements. The data point at 250 ms was only measured for AWSI.

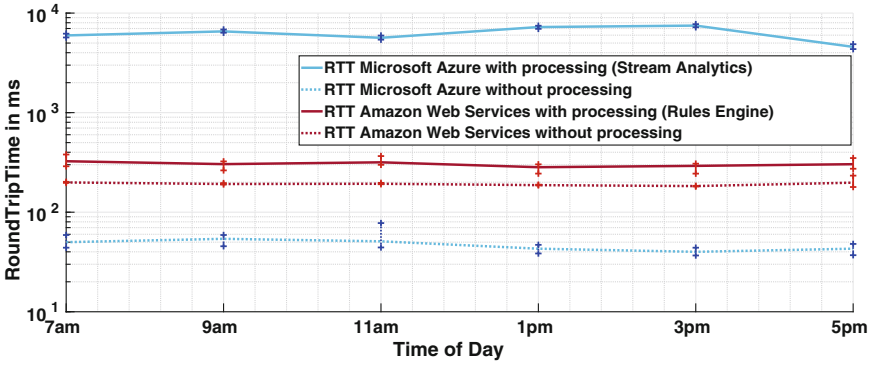


Fig. 5. RTT measurements, with/without processing in the cloud; in addition the time of day was varied. Each data point depicts the median value and interquartile range of around 100 measurements. Note the logarithmic scaling of the y-axis.

Message Interval. In this benchmark, the interval between sending messages was varied between 200 ms and 1000 ms. Measurement location was the residential area. On the whole, performance does not vary significantly depending on the message interval (see Fig. 4). However, RTT values are much larger (and have higher variability) for AWSI when an interval of 200 ms is used; they amount to ca. 320 ms in this case. Since AWSI RTT values amount otherwise to around 190 ms, this shows most likely a detrimental effect of sending and receiving messages at around the same time. We think that the causes of this effect reside mostly on the cloud server side, because the client performs sending and receiving of messages in a concurrent way on a quadcore CPU which is under low load. In such a setting, it is very unlikely that the rather large time difference of 130 ms (320 ms–190 ms) is caused by a resource conflict on the client side alone.

Additional Processing in the Cloud. For the (industrial) IoT, it is a common scenario that device data is processed in the cloud before it is sent back to another endpoint (e.g., data preprocessing in the cloud before data visualization on a mobile device). As a benchmark for this use case, we sent in each message a single float value to the cloud where it was multiplied by two and sent back. For this purpose, the workflows in the cloud had to be modified: Within Microsoft Azure, messages are forwarded from the IoT Hub to the service "Stream Analytics" where processing takes place. Afterwards, the "Service Bus" sends the modified messages back to a so-called "Queue Client" implemented as part of the benchmarking application. In the AWS cloud, received messages are forwarded from the Device Gateway to the "Rules Engine" for processing. The processed messages are made available under a new topic to which the MQTT client in the benchmarking application is subscribed.

Benchmarks were carried out at the industrial estate. The results are shown in Fig. 5. Performance does not vary significantly during the course of the day, but the additional processing in the cloud has a clear impact on the RTT values. For AWSI, the mean value of the median values increases from 192.8 ms (without processing) to 304.3 ms (with processing). For MAIH, an increase from 49.2 ms (without processing) to 6336 ms (with processing) was observed. The latter is a really significant slowdown by a factor of 129.

5 Discussion and Outlook

Results of cloud benchmarking usually have to be taken with a grain of salt because there exist many uncontrollable factors along the route between client and cloud servers [13]. Furthermore, CSPs may upgrade and modify their systems without notice. Therefore, we are well aware that our results only represent a local snap-shot, and that similar performance measurements may be different in other parts of the world and in the future. For this reason, it is definitely not our goal to attribute good or bad performance to MAIH or AWSI in general.

However, our results clearly show that one has to be careful in choosing the right cloud platform for IoT operations in one's local area, especially if we consider the use case of cyber-manufacturing systems (industrial IoT) which are intended to interact in real-time—not in the very strict sense of hard real-time conditions, but in the sense of seamless coordination between machines (and maybe machines and humans). Our data shows a clear performance advantage for one of the two benchmarked CSPs, and how this pattern completely reverses in an extreme fashion as soon as additional processing of the data in the cloud is required. Mean performance and variability is rather stable when varying the day of week, the time of day, or the location. However, the latter factor has a clearly visible and systematic impact on the performance of both CSPs. A considerable performance drop could be observed for AWSI when decreasing the sending interval between messages to the usual RTT. This is a clear warning that performance is subject to factors in an unexpected way (one message every 200 ms does not look like heavy load, does it?).

Although the number of clients in a manufacturing line, which are independently connected to an IoT cloud service, is usually not that large compared to e.g. the millions of smart home devices, it is definitely an important step for further benchmarking to increase the number of clients which send data in parallel to the cloud. Furthermore, we plan to repeat our measurements over a longer time period and, in addition, to vary message size. Message size is important because it is common that programmable logic controllers in manufacturing collect sets of sensory and control variables and transmit them at once, e.g. to a local OPC-UA client which in turn forwards the data in the form of larger packages to the cloud.

Given our already existing benchmarking results, we are convinced that manufacturing companies, when they move forward to cyberphysical production systems, have to carefully plan the data flow to the cloud and to carry out corresponding benchmarks because it may depend strongly on the CSP if specific real-time conditions for IoT data processing are met in the end.

Acknowledgements. We appreciate the general support by Dominik Osthues (Rheda-Wiedenbrück) for our benchmarking work. Furthermore, we thank Martin Kohlhase (Bielefeld Univ. of Applied Sciences) for many helpful discussions.

References

1. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. **39**(1), 50–55 (2008)
2. Jeschke, S., Brecher, C., Meisen, T., Özdemir, D., Eschert, T.: Industrial internet of things and cyber manufacturing systems. In: Jeschke, S., Brecher, C., Song, H., Rawat, D.B. (eds.) Industrial Internet of Things – Cybermanufacturing Systems. SSWT, pp. 3–19. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-42559-7_1
3. Lee, G.M., Crespi, N., Choi, J.K., Boussard, M.: Internet of things. In: Bertin, E., Crespi, N., Magedanz, T. (eds.) Evolution of Telecommunication Services: The Convergence of Telecom and Internet: Technologies and Ecosystems. LNCS, vol. 7768, pp. 257–282. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41569-2_13
4. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
5. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the weather tomorrow?: towards a benchmark for the cloud. In: Proceedings of the 2nd International Workshop on Testing Database Systems, DBTest 2009, pp. 9:1–9:6. ACM, New York (2009)
6. Agarwal, D., Prasad, S.K.: AzureBench: benchmarking the storage services of the Azure cloud platform. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum, pp. 1048–1057 (2012)

7. Kossmann, D., Kraska, T., Loesing, S.: An evaluation of alternative architectures for transaction processing in the cloud. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, pp. 579–590. ACM, New York (2010)
8. Li, Z., O'Brien, L., Zhang, H.: Boosting metrics: measuring cloud services from the holistic perspective. Int. J. Cloud Comput. (IJCC) **2**(4), 12–22 (2014)
9. Bermbach, D., Wittern, E.: Benchmarking web API quality. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 188–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_11
10. Gillam, L., Li, B., O'Loughlin, J., Tomar, A.P.S.: Fair benchmarking for cloud computing systems. J. Cloud Comput.: Adv. Syst. Appl. **2**(1), 6 (2013)
11. Zant, B.E., Gagnaire, M.: Performance and price analysis for cloud service providers. In: 2015 Science and Information Conference (SAI), pp. 816–822 (2015)
12. Cunha, M., Mendonca, N., Sampaio, A.: A declarative environment for automatic performance evaluation in IaaS clouds. In: 2013 IEEE Sixth International Conference on Cloud Computing, pp. 285–292 (2013)
13. Bermbach, D., Wittern, E., Tai, S.: Cloud Service Benchmarking – Measuring Quality of Cloud Services from a Client Perspective. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55483-9
14. Microsoft Corporation: Get started with Azure IoT Hub (.NET) – Microsoft Docs (2017). https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-getstarted. Accessed 08 Aug 2017
15. AWS Inc.: AWS IoT – Developer Guide (2017). http://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf. Accessed 02 Sept 2017

A Comprehensive Solution for Research-Oriented Cloud Computing

Mevlut A. Demir(✉), Weslyn Wagner, Divyaansh Dandona, and John J. Prevost

The University of Texas at San Antonio,
One UTSA Circle, San Antonio, TX 78249, USA
{mevlut.demir,jeff.prevost}@utsa.edu, {weslyn.wagner,ubm700}@my.utsa.edu

Abstract. Cutting edge research today requires researchers to perform computationally intensive calculations and/or create models and simulations using large sums of data in order to reach research-backed conclusions. As datasets, models, and calculations increase in size and scope they present a computational and analytical challenge to the researcher. Advances in cloud computing and the emergence of big data analytic tools are ideal to aid the researcher in tackling this challenge. Although researchers have been using cloud-based software services to propel their research, many institutions have not considered harnessing the Infrastructure as a Service model. The reluctance to adopt Infrastructure as a Service in academia can be attributed to many researchers lacking the high degree of technical experience needed to design, procure, and manage custom cloud-based infrastructure. In this paper, we propose a comprehensive solution consisting of a fully independent cloud automation framework and a modular data analytics platform which will allow researchers to create and utilize domain specific cloud solutions irrespective of their technical knowledge, reducing the overall effort and time required to complete research.

Keywords: Automation · Cloud computing · HPC
Scientific computing · SolaaS

1 Introduction

The cloud is an ideal data processing platform because of its modularity, efficiency, and many possible configurations. Nodes in the cloud can be dynamically spawned, configured, modified, and destroyed as needed by the application using pre-defined application programming interfaces (API). Furthermore, cloud-based systems are highly scalable and can be configured to provide a high degree of availability [1]. For these reasons, industry has fully embraced cloud computing for delivering services and applications, as well as establishing dedicated jobs focused on the maintenance and management of cloud infrastructure. For all the reasons stated, a cloud-based solution would also be optimal for academic

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 407–417, 2018.
https://doi.org/10.1007/978-3-319-94295-7_29

research [3]. Currently, most researchers utilize local compute systems to run calculations and store required data. These calculations are often complex and can take multiple hours to days to converge to a conclusion. A cloud-scaled solution could outperform local compute systems, while also allowing the researcher to quickly change the underlying topology in response to transient parameters or characteristics of the models used.

In academia, however, Infrastructure as a Service (IaaS) solutions have not been adopted for two main reasons: lack of technical expertise and the need for dedicated management. IaaS requires the user to fully design and configure their cloud environment which may consist of runtime, middle-ware, and operating systems [7]. This configuration must then be maintained so that software can run efficiently on top of it. Researchers, who are experts in their own field of research, usually lack the technical skills to create and manage a cloud solution, and become reliant upon their institution's IT department to provide them with assistance. Those who desire to possess the requisite skills are forced to divert time from their research to learn the necessary skills themselves. Cloud providers recognize this issue and have created systems such as one-click installs or automation frameworks to greatly reduce the effort and expertise needed to create cloud infrastructure, but these solutions are not holistic in nature nor geared towards academia.

To address this problem, we propose Research Automation for Infrastructure and Software framework (RAINS). RAINS aims to be an all-in-one infrastructure procurement, configuration, and management framework tailored specifically towards the needs of researchers [2]. The framework is agnostic to the cloud provider giving the researcher the option of hosting with different public providers, or running a private cloud on premises.

RAINS incorporates a crowd-sourced model to allow researchers to share and collaborate their unique infrastructure builds across the world in a seamless manner. This would enable researchers to quickly reproduce the results of ongoing research, and share optimum configurations.

RAINS will make the IaaS model accessible to researchers in a manner that is easy to use and configure, thus allowing them to focus solely on their research and not on extraneous system configuration.

Once a custom infrastructure and software environment has been created, software deployments will need to be connected and synchronized before applications are fully operational in the cloud. There are hundreds of active, open source tools for big data analytics - each one has a custom installation, working environment, and communication method. For a typical researcher, learning all of these tools and their selective APIs can be a massive hurdle. These big data tools often work in a clustered topology consisting of a data ingestion layer, data processing layer, data storage layer, and a data visualization layer.

This further adds complexity to utilizing IaaS because it is not directly apparent which set of tools would be correct for the researcher and what type of environment and dependencies each one has. A researcher would have to have

advanced knowledge of programming, data science, and system administration to connect these individual tools into a cluster formation and run them successfully.

To speed up integration we propose Software Platform for Experimental Analysis and Research (SPEAR). SPEAR is designed to allow for easy integration between various big data and machine learning tools to create a working research cluster atop a RAINS-created infrastructure. The platform takes advantage of the cloud's high availability and scalability to handle numerous concurrent data streams for parallel processing and real time visualization. SPEAR empowers the end user to prototype and execute data processing algorithms in an abstracted environment by independently handling data migration. SPEAR offers plug and play support for popular data acquisition, control, or analysis tools that the end user may wish to integrate into their custom installations.

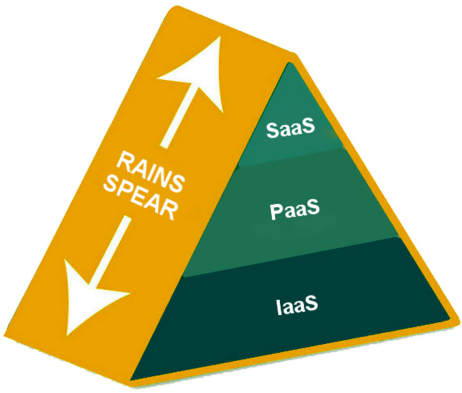


Fig. 1. SPEAR and RAINS integrate IaaS, PaaS, and SaaS

The emergence of cloud-based services in the early 2000's revolutionized Internet-based content delivery. Today, almost every interaction we have with an on-line service is being handled through a cloud-based system [4]. Cloud services can be classified as either Software as a Service (SaaS) [6], Platform as a Service (PaaS) [5], or Infrastructure as a Service. Under, SaaS a user can use the software without having to worry about its underlying hardware, software environment, or session which are all being managed be the service provider. PaaS allows a user to interact with the cloud service at an operating system level. IaaS gives complete control over the underlying virtual or physical hardware stack, to the end user, allowing for a truly complete customizable experience. Together RAINS and SPEAR provide the researcher with solution as a service which integrates IaaS, PaaS, and SaaS as shown in Fig. 1.

2 RAINS Framework

The aim of RAINS is to provide easy-to-use cloud automation methods that can be used standalone, or as a base for software platforms such as SPEAR. RAINS has a modular design in which the back-end and front-end applications are coded separately and communicate through established API calls. The front end is designed to run the browser and allow for ease of access. The back-end consists of a REST API which interfaces with the orchestration engine to procure and deploy the designed infrastructure.

2.1 Dashboard

The RAINS dashboard is a browser-based graphical user interface intended for configuring, designing, managing, and deploying cloud infrastructure. The dashboard consists of a palette and a canvas for visualization. RAINS offers three viewing modes: Machine Level, Application Level, and Network Level. Each view has a custom palette and canvas associated for managing aspects of the infrastructure build. The framework employs a drag-and-drop API in conjunction with form-based wizards to create representational states which can then be rendered to the canvas. Upon initialization the user can drag and drop hardware and software components to create their unique cloud infrastructure build. Once the infrastructure build has been graphically designed, the deploy button realizes the infrastructure.

Machine View. The Machine View focuses on the machine level aspects of the infrastructure. The palette provides virtual machines and containers as draggable components. The canvas renders machine level details such as hostname, IP address, and machine type.

Application View. The Application View focuses on the application and software aspects of the infrastructure. The palette offers a list of supported software as draggable components. The canvas renders application level details such as software name and current status.

Network View. The Network View highlights TCP/IP network configurations for each machine nodes in detail, as well specific ports that applications are attached to.

Dragged and dropped components on the canvas are referred to as droplets. A droplet consists of a machine level component and may also have software and networking components attached. RAINS makes use of forms, models, and templates to prompt the user for basic information needed to satisfy a particular droplet's requirements. In case the user cannot provide basic information, the user is given the choice to use default options or pull a build from trusted repositories. The combination of all the droplets on the canvas represents the

complete infrastructure build the user wishes to deploy. Once an infrastructure build is deployed, the canvas continues to serve as a real-time representation of the active build and can be used to dynamically or manually reconfigure the infrastructure. Each droplet host key-value pairs which describe the infrastructure build, its own canvas styling, and the droplet's current status if it is actively deployed. The revolutionary novelty of RAINS is that infrastructure builds can be further packed into droplet clusters with exposed endpoints, allowing for integration of two complex cloud systems comprised of many droplets. In this manner, systems can increase in complexity while still remaining easy to understand through encapsulation and nesting.

2.2 Canonical Form

When a user builds an infrastructure set, RAINS dynamically describes the build in a JSON-based form. This format encapsulates all droplets and droplet clusters into a stateful text based representation. The JSON format can be shared with others or saved to create a snapshot of the current infrastructure build. This representation is then passed to the back end which uses the JSON format to generate a complete canonical form. This form consists of a human readable task list for orchestration and procurement in YAML format. Using a single canonical form facilitates repeatability and completely decouples the orchestration file from the task list. A text-based markup or object level format is easier to share between peers and is computer-friendly enough for quick parsing, leading to a higher degree of integration with other available systems that understand markup. The canonical form can then undergo translation and mapping to create orchestration files specifically tailored for each major cloud platform including AWS, Google Cloud, and OpenStack.

2.3 REST API

RAINS uses the traditional HTTP-based REST API [13] and database combination to provide service communication. The REST API allows the front-end dashboard to access the database to store session information, get updates, and post JSON forms for translation. RAINS utilizes both a relational and non-relational databases to store data. The REST API can also be used internally to serve as a communication broker between server processes.

2.4 Synthesis

The JSON output from the front end is the most minimal description of the infrastructure build. When the JSON form is POSTed to the backend, hopper scripts parse and sort the information based on whether it is a procurement or orchestration directive. After sorting, a task-based canonical form is generated, which consists of a list of the procurement and orchestration tasks needed to realize the infrastructure. The canonical form gives precedence to procurement

directives while orchestration directives are listed afterwards. The YAML-based canonical form can then be translated into orchestration templates and Ansible [8] execution Playbooks.

2.5 Translation

The canonical form alone is not an inclusive enough input for an orchestration engine. The key-value pairs in the canonical form need to be extended into orchestration scripts that the orchestration engines can use. A translation service uses advanced templating and mapping to generate the vendor specific consumables for AWS [15], Google Cloud Platform [10], OpenStack [14], Azure [11], and Ansible. The current implementation of the translation layer targets AWS and OpenStack Heat, while also supporting Ansible for pre- and post-deployment orchestration. RAINS provides AWS Cloud Formation Templates and OpenStack Heat Orchestration Templates [9], and allows for automatic procurement, scaling, and other tasks supported by the providers' respective APIs. RAINS also supports running user-provided Ansible scripts against nodes running in an infrastructure build. The user can provide Ansible scripts via the dashboard and then execute them on one or multiple droplets that are active on the canvas.

2.6 Procurement

Once the orchestration scripts are ready, RAINS interfaces with the respected APIs of the cloud provider the user has chosen. The user's account details and subscription plan are taken into account, and an API call is made to procure the machine level nodes, and prepare them for Ansible. Ansible is then utilized to complete the environment creation and software installation tasks. At this point RAINS has created the infrastructure in the cloud.

2.7 Real-Time Feedback Loop

A real-time feedback loop is currently under development for RAINS. The feedback loop will connect to the cloud platforms' metrics and logging services, such as Ceilometer for OpenStack [12]. These metrics will map to layout styling and will be reflected onto the canvas for visualization. The real-time feedback loop will allow RAINS to check the status of the computational nodes and the software stacks running on them in real-time. RAINS will then be in the position to manage the health of nodes, dynamically scale applications up, down, or out depending on load, and even make optimization suggestions to the user on how to best implement an architecture build. Active monitoring will also reduce computational waste and help the user reduce deployment costs.

2.8 Repository

To help foster a collaborative environment and provide for easy templating, a dedicated repository will be used for canonical form tracking and storage.

Researchers will have the option of initiating official builds and custom wikis to highlight the usage and performance of their infrastructure builds. The online repository will serve as a sharing and storage portal through which researchers can pull, push, and share their unique configurations. A voting-based ranking system will be utilized to give credibility to builds.

3 SPEAR

The SPEAR aims to give researchers access to open source big data analytics tools as well as popular machine learning, robotics, and IOT frameworks in a easy to use and integrated environment. SPEAR at it's core is a plug and play platform manager. It manages the integration of various different suites of software into one cohesive clustered application. Furthermore, the usage of the cloud allows for hot swapping of applications in real time, resulting in a robust and resilient system. SPEAR can easily be layered atop RAINS, and employed to connect node endpoints generated by RAINS, and even program and run applications. A typical SPEAR cluster will consist of a data ingestion layer, data processing layer, data storage layer, and a visualization layer, while providing for additional application specific layers such as machine learning and robot operating system (ROS) [16] layers.

Data Ingestion. The data ingestion layer consists of a distributed queuing system along with a message broker. Data ingestion in this manner can scale effectively across cloud nodes to handle multiple concurrent data streams.

Data Processing. The data processing layer consists of big data analytical tools such as distributed stream processing engines and batch processing engines. Distributed stream processing engines are ideal for real time stream based processing. Batch processing can be performed after data has been collected on large aggregates of data.

Data Storage. The data sink layer consists of storage volumes and databases. Users can configure SQL relational databases or NoSQL non-relational databases to serve as data sinks for post processing or direct storage. SPEAR takes into account the effort needed to create database tables and schemas, and uses custom graphical tools to help the user complete this task to reduce the need for a user to learn CLI tools.

Visualization. The visualization layer can consist of various different output forms. HTML5 and Javascript based visualization techniques are very popular, but data can be fed into graphing and mapping programs as well.

3.1 Graphical User Interface

SPEAR's graphical user interface employs graphical wires to define cluster topologies. Users can click and drag wire connections to and from component endpoints to set TCP/IP communication across a cluster. An emphasis on block programming is given to abstract away the necessity for configuring and using each component of the big data architecture. Graphical Programming Interfaces can be utilized to abstract the need to learn specific tools and coding techniques. Instead the end user can click and drag functionality or use graphical wizards to modify, customize, and utilize the underlying software applications.

3.2 Topology Creation

RAINS can create infrastructure and software environments, but it is not designed as a software management platform. SPEAR takes on this responsibility by connecting the individual nodes into the formation of a cluster. SPEAR does this by utilizing various communication tools such as TCP/IP port mappings, communication scripts, and pub/sub message protocols.

3.3 Data Flow

To facilitate the flow of data between ingestion, processing nodes, and visualization SPEAR currently supports Kafka [17], RabbitMQ [18], and Apache Pulsar [19]. Internally publish/subscribe models are utilized for inter-cluster communication. In a typical pub/sub configuration, data packets travel through queues. Each queue consists of a bidirectional channel which supports quality of service level 1. As the load on the system changes, the queues can easily be scaled and parallelized to sustain a high throughput.

3.4 Processing

SPEAR supports popular distributed stream processing engines as well as batch processing engines to satisfy processing needs. Distributed stream processing engines can interface directly with most messaging queues to ingest data and perform permutations and computations. Popular DSPEs such as Apache Storm [20] and Apache Heron [21] utilize controller and worker nodes to create a highly scalable, fault tolerant processing solution. Apache Spark [22] and Hadoop [23] are popular batch processing platforms that can easily be integrated into a SPEAR cluster.

3.5 Data Sink

Data must be persisted in some form for later analysis and reference. SPEAR supports relational databases like MySQL [24], and non relational databases such as MongoDB [25]. Some applications such as Hadoop require custom NoSQL implementations like HBase [26]. SPEAR's graphical interfacing tools will allow users to perform create, update, read, and delete data from the database.

3.6 Visualization

SPEAR supports browser based visualization and allows for easy integration with JavaScript visualization frameworks such as D3 [27]. JavaScript and HTML5 have become popular for visualization because of universal access via browsers, and low maintenance costs. SPEAR offers visualization templates using D3, from which a user can pull data from databases or from data processing engines for real time or aggregate visualization, without having to create the visualization mechanisms themselves. An example framework can be seen in Fig. 2.

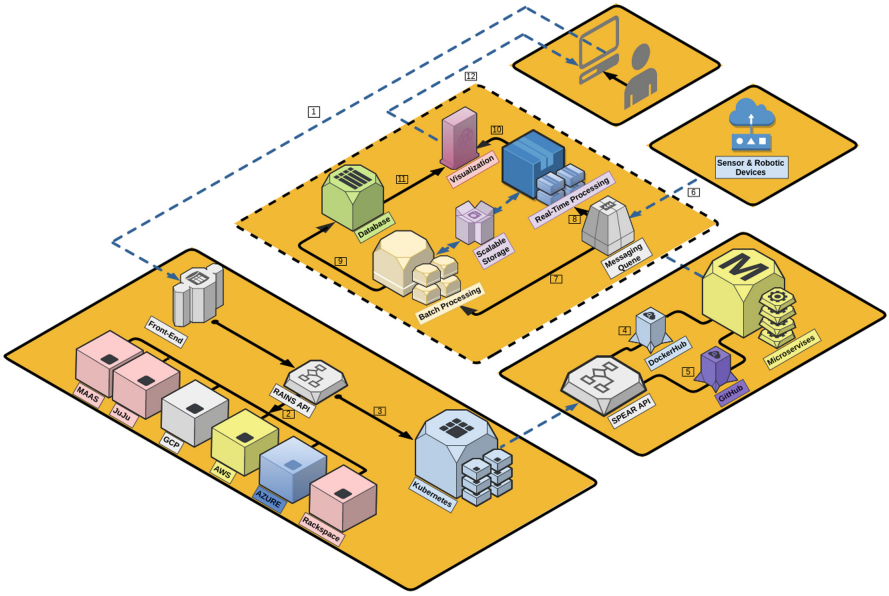


Fig. 2. Overview of the SPEAR & RAINS frameworks

4 Future Work

We plan to utilize RAINS and SPEAR in our research laboratory as a test case. In this manner we can add functionality and manage the development cycle of each software based on the needs of university researchers. RAINS and SPEAR will be utilized in our cyberphysical IOT systems research. We plan to gain performance benchmarks and metrics and further improve our software.

5 Conclusion

The layering of SPEAR and RAINS creates a holistic cloud-based solution for procuring cloud infrastructure and data processing that is customizable and

controllable from end to end, that simultaneously mitigates the need for technical cloud expertise from the researcher. RAINS can serve as the starting base for other platforms which may require a highly customizable cloud-based infrastructure, or can be used standalone to solely generate cloud infrastructure. The SPEAR platform will give researchers quick and seamless access to many current data analytics tool sets without the need to learn complicated coding and command line tools. Together, RAINS and SPEAR will revolutionize research speed and give the researcher full access to the cloud's potential.

References

1. Kondo, D., et al.: Cost-benefit analysis of cloud computing versus desktop grids. In: 2009 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009. IEEE (2009)
2. Wagner, W.S.: Research automation for infrastructure and software: a framework for domain specific research with cloud computing. Dissertation. The University of Texas at San Antonio (2017)
3. Benson, J.O., Prevost, J.J., Rad, P.: Survey of automated software deployment for computational and engineering research. In: 2016 Annual IEEE Systems Conference (SysCon). IEEE (2016)
4. Qian, L., Luo, Z., Du, Y., Guo, L.: Cloud computing: an overview. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 626–631. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10665-1_63
5. Malfara, D.: Platform as a service. Working paper ACC 626, University of Waterloo, Waterloo, Canada, 3 July 2013
6. Gupta, N., Varshapriya, J.: Software as a service. Int. J. Innov. Res. Adv. Eng. (IJIRAE)-2014 **1**(6), 107–112 (2014)
7. Manvi, S., Shyam, G.: Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. J. Netw. Comput. Appl. **41**, 424–440 (2014)
8. Ansible: How Ansible Works (2018). https://www.ansible.com/how-ansible-works
9. OpenStack: Heat-Translator (2018). https://wiki.openstack.org/wiki/Heat-Translator
10. Google Cloud Platform: About the Google Cloud Platform Services (2018). https://cloud.google.com/docs/overview/cloud-platform-services
11. Microsoft Azure: Azure regions, more than any cloud provider (2018). https://azure.microsoft.com/en-us/
12. OpenStack: OpenStack Docs: Welcome to Ceilometer's Documentation (2018). https://docs.openstack.org/ceilometer/latest/
13. Fielding, R.T., Taylor, R.N.: Architectural styles and the design of network-based software architectures, vol. 7. Doctoral dissertation, University of California, Irvine (2000)
14. OpenStack: OpenStack Docs: System Architecture (2018). https://docs.openstack.org/
15. AWS (2018). https://aws.amazon.com/
16. ROS (2018). http://www.ros.org/
17. Kafka (2018). https://kafka.apache.org/
18. RabbitMQ (2018). https://www.rabbitmq.com/
19. Apache Pulsar (2018). https://pulsar.incubator.apache.org/

20. Apache Storm (2018). http://storm.apache.org/
21. Apache Heron (2018). https://twitter.github.io/heron/
22. Apache Spark (2018). https://spark.apache.org/
23. Hadoop (2018). http://hadoop.apache.org/
24. MySQL (2018). https://www.mysql.com/
25. MangoDB (2018). https://www.mongodb.com/
26. HBase (2018). https://hbase.apache.org/
27. D3 (2018). https://d3js.org/

Author Index