Lecture Notes in Computer Science 7150

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

Kurt Vanmechelen Jörn Altmann
Omer F. Rana (Eds.)

Economics of Grids, Clouds, Systems, and Services

8th International Workshop, GECON 2011
Paphos, Cyprus, December 5, 2011
Revised Selected Papers

Springer

Volume Editors

Kurt Vanmechelen
University of Antwerp, Department of Mathematics and Computer Science
Middelheimlaan 1, 2020, Antwerp, Belgium
E-mail: kurt.vanmechelen@ua.ac.be

Jörn Altmann
Seoul National University, College of Engineering
Department of Industrial Engineering
Technology Management, Economics, and Policy Program
599 Gwanak-Ro, Gwanak-Gu, 151-744 Seoul, South-Korea
E-mail: jorn.altmann@acm.org

Omer F. Rana
Cardiff University, School of Computer Science
Queen's Buildings, Newport Road, Cardiff, CF24 3AA, UK
E-mail: o.f.rana@cs.cardiff.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-28674-2 e-ISBN 978-3-642-28675-9
DOI 10.1007/978-3-642-28675-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012932491

CR Subject Classification (1998): C.2.4, K.4.4, H.4, H.3, H.5, J.1

LNCS Sublibrary: SL 5 – Computer Communication Networks and Telecommunications

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

You are holding the proceedings of the 8th International Workshop on the Economics of Grids, Clouds, Systems, and Services. This workshop brings together the research and practitioner communities active in the area of economics and computer science to address the emerging interest in infrastructure, platform, and software services. This includes the operation and structure of the service market, the alignment of cost, revenue, and quality-related objectives, and the creation of innovative business models and value chains.

This year again we received a number of high-quality paper submissions. Each submission was reviewed at least three times by an international Program Committee. Our final program consisted of five highly interactive and thought-provoking sessions (two of which were work-in-progress sessions):

- Session A: Market Mechanisms and Negotiation
- Session B: Cost Models, Charging, and Trading Platforms
- Session C: Resource Allocation, Scheduling, and Admission Control
- Session D: Work in Progress: Risk Assessment and Economics of Cloud Services
- Session E: Work in Progress: Cost-Aware Adoption of Cloud Services

As the five session titles suggest, the workshop brings together contributions from economics, resource allocation, resource management, and risk assessment. In total, there were 14 contributions (consisting of nine full papers and five work-in-progress papers) selected from 27 submitted papers. The acceptance rate of full papers is 33%.

The first paper in Session A by Haque et al. entitled "An Inspiration for Solving Grid Resource Management Problem Using Multiple Economic Models" compares a number of currently used economic models in grid computing, such as commodity markets, continuous double auctions, English auctions, contract-net protocols, and bargaining, in order to identify the settings in which one economic model out performs another. A quantitative experimental evaluation is undertaken to support this comparison—in particular to identify when to switch between such models and when to use a combination of them. The contribution "Concurrent Negotiations in Cloud-Based Systems" by Siebenhaar et al. addresses the lack of quality-of-service guarantees available within existing cloud systems. It proposes an automated negotiation approach that considers both the individual business objectives and strategies of the negotiation partners along with the dependencies between the different services and service tiers within a cloud system. The last contribution in this session from Roovers et al. entitled "A Reverse Auction-Based Market for IaaS Cloud Resources" investigates the creation of an open market for IaaS resources and proposes a continuous reverse auction along with a bidding language. It thereby specifically takes into account the current pricing schemes of real-world cloud resource providers.

Session B starts with a paper by Mohammad Mahdi Kashef and Jörn Altmann entitled "A Cost Model for Running Hybrid Clouds," which identifies a usage-based cost model for running a cloud environment consisting of both public and private (data center based) clouds. The author argues that although cloud computing promises to reduce the cost of IT through lower capital and operational expenses, providing a clear specification of these costs is often lacking in the existing literature. The subsequent contribution by Stefanov et al. entitled "How to Do Successful Chargeback for Cloud Services" utilizes experience of field experts from IBM. It provides factors that identify how to allocate IT service costs to business users based on their service consumption and how to facilitate the transition to a cloud environment. The authors argue that it is often difficult to pinpoint the actual costs incurred through service provisioning and address this limitation in their work. The final contribution in this session from Menychtas et al. entitled "A Marketplace Framework for Trading Cloud-Based Services" proposes a cloud marketplace platform for the development and trading of XaaS products. It provides a single point of access for consumers interested in services and provides specialist support for sellers wishing to make their services available through the platform.

Session C starts with a contribution from Macias and Guitart entitled "Client Classification Policies for SLA Negotiation and Allocation in Shared Cloud Datacenters," focusing on how user types (internal vs. external, preferential vs. standard) could be used by providers during SLA negotiations. Experiments are used to compare two such negotiation strategies: price discrimination and client-aware overselling of resources. In their paper "Budget-Deadline Constrained Workflow Planning for Admission Control in Market-Oriented Environments," Zheng and Sakellariou focus on workflow planning and execution, taking into account the deadline and budget constraints of users submitting workflows. The proposed heuristic also takes account of the existing load on the resources that must enact the workflow. Finally, Li et al. in their paper entitled "Virtual Machine Placement for Predictable and Time-Constrained Peak Loads" discuss how virtual machines should be placed on servers within a data center in order to deal with peaks in workload and make the execution time of tasks more predictable. They discuss the trade-off between computation time and the quality of solutions provided by a binary integer program and three approximations that increase the scale at which this NP complete problem can be solved.

The final two sessions constitute the "Work-In-Progress" papers—primarily focusing on work that is at an early stage of maturity, but likely to make significant contributions to the community. The first of these, Session D, focuses on risk assessment and economic models associated with cloud service provision. Petri et al. in their contribution "Risk Assessment in Service Provider Communities" discuss the notion of financial risk from the perspective of various stakeholders involved in cloud-based service provisioning. Künsemöller and Karl in their paper entitled "A Game-Theoretical Approach to the Benefits of Cloud Computing" identify characteristics of beneficiaries in an infrastructure-as-a-service market and the potential actions they could take to gain financial benefit.

Session E includes three contributions focusing on cost efficiency. The paper by Sengupta and Annervaz entitled "Planning for Optimal Multi-Site Data Placement for Disaster Recovery" discusses strategies for backup of critical business data across (a large number of) multiple data centers (in different geographical locations). The approach takes into account criteria such as cost of storage and network, protection level against site failures, as well as business and operational parameters such as recovery point and time objectives. Their approach uses data-encoding techniques that can facilitate recovery from multiple data center failures. Shi et al. in their contribution "Saga: A Cost-Efficient File System Based on Cloud Storage Service" describe a cost-efficient file system that provides a POSIX interface on top of Amazon S3. Cost reduction is achieved by minimizing the storage space used through "store-one-copy" and "copy-on-write" strategies and by minimizing the number of requests through a distinction of objects loaded by write and read requests in the cache replacement algorithm. The final contribution of this session from Stephen McGough entitled "Developing a Cost-Effective Virtual Cluster on the Cloud" discusses how an entire cluster computing environment could be run on a cloud system, taking into account various usage policies and execution costs.

We would like to thank the reviewers and Program Committee members for completing their reviews on time, and giving useful and valuable feedback to the authors. We would also like to extend our thanks to the organizers of ICSOC for hosting our workshop alongside their conference this year. Furthermore, we would like to express our gratitude toward Alfred Hofmann from Springer for his support in publishing the proceedings of GECON 2011.

December 2011

Kurt Vanmechelen
Jörn Altmann
Omer F. Rana

Organization

GECON 2011 was organized by the Technology Management, Economics, and Policy Program, Seoul National University, the School of Computer Science, Cardiff University, and the University of Antwerp in collaboration with ICSOC 2011.

Executive Committee

Chairs

Jörn Altmann	Seoul National University, South Korea
Omer F. Rana	Cardiff University, UK
Kurt Vanmechelen	University of Antwerp, Belgium

Program Committee

Ashraf Bany Mohammed	University of Hail, Saudi Arabia
Hermant K. Bhargava	UC Davis, USA
Rajkumar Buyya	University of Melbourne, Australia
Jeremy Cohen	Imperial College London, UK
Costas Courcoubetis	Athens University of Economics and Business, Greece
Dang Minh Quan	CREATE-NET, Italy
Karim Djemame	University of Leeds, UK
Torsten Eymann	University of Bayreuth, Germany
Thomas Fahringer	University of Innsbruck, Austria
Wolfgang Gentzsch	DEISA, Germany
Matthias Hovestadt	Technical University of Berlin, Germany
Chun-Hsi Huang	University of Connecticut, USA
Odej Kao	Technical University of Berlin, Germany
Kibae Kim	Technical University of Braunschweig, Germany
Stefan Kirn	University of Hohenheim, Germany
Tobias A. Knoch	Erasmus University, The Netherlands
Bastian Koller	HLRS, University of Stuttgart, Germany
Harald Kornmayer	NEC Laboratories Europe, Germany
Byungtae Lee	KAIST, South Korea
Jysoo Lee	KISTI, South Korea
Dan Ma	Singapore Management University, Singapore
Steven Miller	Singapore Management University, Singapore
Dirk Neumann	University of Freiburg, Germany

Karsten Oberle	Alcatel-Lucent Bell Labs, Germany
Peter Reichl	Telecommunications Research Center Vienna, Austria
Satoshi Sekiguchi	AIST, Japan
Arunabha Sen	Arizona State University, USA
Katarina Stanoevska	University of St.Gallen, Switzerland
Burkhard Stiller	University of Zurich, Switzerland
Bruno Tuffin	IRISA/INRIA, France
Dora Varvarigou	National Technical University of Athens, Greece
Daniel Veit	University of Mannheim, Germany
Gabriele von Voigt	University of Hannover, Germany
Stefan Wesner	HLRS, University of Stuttgart, Germany
Phillip Wieder	University of Dortmund, Germany
Ramin Yahyapour	University of Dortmund, Germany
Wolfgang Ziegler	Fraunhofer Institute SCAI, Germany

Steering Committee

Jörn Altmann	Seoul National University, South Korea
Rajkumar Buyya	University of Melbourne, Australia
Thomas Fahringer	University of Innsbruck, Austria
Junseok Hwang	Seoul National University, South Korea
Hing-Yan Lee	National Grid Office, Singapore
Jysoo Lee	KISTI, South Korea
Steven Miller	Singapore Management University, Singapore
Dirk Neumann	University of Freiburg, Germany
Omer F. Rana	Cardiff University, UK
Daniel Veit	University of Mannheim, Germany

Sponsoring Institutions

Seoul National University, Seoul, South Korea
University of Cardiff, Cardiff, UK
University of Antwerp, Antwerp, Belgium
Springer LNCS, Heidelberg, Germany
ICSOC 2011, Paphos, Cyprus

Table of Contents

Session D: Work in Progress: Risk Assessment and Economics of Cloud Services

Session E: Work in Progress: Cost-Aware Adoption of Cloud Services

Erratum

An Inspiration for Solving Grid Resource Management Problems Using Multiple Economic Models

Aminul Haque[1], Saadat M. Alhashmi[1], and Rajendran Parthiban[2]

[1] School of Information Technology, Monash University, Bandar Sunway, Malaysia
{aminul.haque,alhashmi}@monash.edu
[2] School of Engineering, Monash University, Bandar Sunway, Malaysia
parthiban.rajendran@monash.edu

Abstract. Economic models can motivate resource providers to share resources across multiple administrations in Grid computing. Our survey on existing economic models in Grid computing identified that different economic models are suitable for different scenarios. In this paper, we conduct an experiment to quantify the strengths and weaknesses of widely proposed economic models in the Grid - Commodity Market, Continuous Double Auction, English Auction, Contract-Net-Protocol and Bargaining. Based on this experimental analysis, we identify regions where a particular economic model outperforms others. Then, we indicate that switching between the economic models could be used to maximize benefits in a specific scenario.

Keywords: Domain of strength, economic models, Grid, optimization.

1 Introduction

Grid computing harnesses computational resources across geographical boundaries. The aim of this computing framework is to solve some complex problems, such as drug design in a more cost effective and standard way. Due to multiple boundaries, the problem solving through coordinated distributed environment becomes challenging. Extensive research has been conducted and it has been identified that economic-based approach is more efficient in meeting the challenge compared to traditional non-economic based approach [1,2]. Economic models help to diagnose distributed scheduling problem while ensuring sufficient motivation to the resource providers. Price is a key term of any economic model and can be used to characterize different resources. Price can also be used to maintain equilibrium between supply and demand or sometimes to figure out the true value of resource demands. Different economic models have different pricing strategies and interaction protocols between users and providers. This adds dynamics in the Grid environment.

In spite of the potential that economic models offer, selecting a particular model out of multiple models is challenging; since, 1) the standards of a particular model would be static; however, Grid is dynamic 2) a particular model could only provide limited features to utilize the potential of the Grid, while Grid users could have different aspirations from the Grid and finally, 3) examining the sustainability of a particular model in a large variety of scenarios is harder. We conducted an extensive survey on

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 1–16, 2012.

different economic models in Grid computing [3]. We addressed the suitability of different economic models for different scenarios. This inspired us to conduct an extensive experimental analysis of the different economic models. Therefore, in this paper, we investigate, compare and contrast the performance of five most widely proposed economic models in the Grid. We consider a range of different parameters to evaluate the models and identify the regions/scenarios where a particular model outperforms all others. Our findings would help one to decide which model to use when and for what purpose. In this work, we particularly focus on provider strategy; therefore, our solutions are based on giving flexibility only to Grid providers.

Section 2 provides some related work. Section 3 explains the development of the five economic models. Section 4 talks about the experimental setup and simulative study. Section 5 summarizes the paper. Section 6 gives the conclusion.

2 Related Work

Realizing the distributed resource management problem in Grid, Buyya et al. propose several economic models [4]. Not all the models proposed are suitable for Grid computing all the time [3]. English Auction (EA), Continuous Double Auction (CDA), Commodity Market Model (CMM), Contract-Net-Protocol (CNP) and Bargaining (BAR) are the five most widely proposed models in the Grid we will discuss in this paper. An extensive explanation on these models has already been given in [4]. We describe the core concepts of the five economic models in Section 3.

Despite the significance of economic-based resource collaboration, there are only a few papers that analyze the performances of multiple economic models in the Grid [5,6,2]. Richard et al. propose that CMM would be suitable for maintaining market equilibrium and minimizing communication cost compared to EA [6]. On the other hand, Tan and Gurd criticize CMM due to its system-oriented approach rather than being incentive-oriented [5]. They argue, in CMM, price formation process considering global information on supply and demand does not account for individual's preference optimization; thus become undesirable for the participants [5]. CDA is proposed to be suitable compared to single-sided auctions such as EA in terms of communication and time efficiency [5].

It has been identified that EA is suitable to maximize revenue, economic efficiency (pareto-optimality in resource allocation) and the QoS [1,2]. CNP has been found to be suitable for the utility-based resource allocation and scalability [7]. It is also suitable to solve the distributed cooperation problem and to optimize meta-scheduling process. On the other hand, in a distributed environment such as Grid, BAR is proposed to be suitable; because it supports utility-based negotiation between a user and a service provider [8].

Due to the extensive and arbitrary nature of the Grid, Resinas et al. identity fundamental components for developing automated negotiation systems (ANS) in an open environment [9]. They describe several properties including prerequisites for supporting various negotiation models. On the other hand, a more focussed research on supporting multiple negotiation models in grid environment is studied by Brandic et al. [10]. They propose a meta-negotiation process to deal with the cross-interests of

grid entities. Their generic negotiation architecture would help the grid users to choose their suitable protocols before establishing the Service Level Agreements with the providers. However, neither of the papers extends the protocols to study user or provider benefits. In our research, we investigate the performance of the five most widely proposed models – EA, CDA, CMM, CNP and BAR - and identify their domains of strengths based on different performance metrics including user and provider benefits.

3 System Design and Development

To test the performances of the economic models, we use GridSim simulation toolkit. It is a discrete-event Grid simulation toolkit designed for large scale heterogeneous grid entities' simulation [11]. The toolkit also supports the simulation of economic based resource management across distributed domains. By default, GridSim provides EA and CDA. We extend the existing EA and CDA to support deadline parameter. Additionally, we contribute CMM, CNP and BAR to the current GridSim distribution. The following subsections explain this contribution in detail. Before moving on to the models let us briefly describe about the simulation entities, which are common for all the models.

3.1 User

Grid users can be characterized using their respective applications/Gridlets which need to be executed on the Grid resources. A Gridlet can be defined as a function of several parameters and is denoted by *gl(id, length, dl, budget)*. Where,

id = Gridlet's identity,
length = Gridlet's processing length in MI (Million Instruction),
dl = Deadline to finish processing the Gridlet,
budget = Budget available to process the Gridlet.

A Grid application can again be composed of several tasks. Based on the relationship and dependency among the tasks, Grid applications can be categorized into three types; Bag of Tasks[1], MPI (Message Passing Interface) and Workflow. Currently our work is suitable only for Bag of Tasks applications.

3.2 Broker

In a Grid environment, the broker representing a user plays an important role by discovering and communicating to the resource nodes, submitting Gridlets to a suitable node and finally gets the results back from the execute-node. The crucial task for a broker is to process its Gridlet within the budget and deadline as requested by the user.

[1] This kind of applications consist of multiple independent tasks requiring no communication among the tasks [12].

3.3 Resource-Node

Each resource-node is configured by some resource properties. If a resource-node is denoted by *n*, then it can be characterized as *n(Id, mList, alloc_policy, cost)*, where,

Id	= Node identity,
mList	= Number of machines the node is comprised of. A particular machine further is a function of number of Processing Elements (PE) and Million Instruction per Second (MIPS) rating for each PE,
alloc_policy	= Job scheduling policy by the node,
cost	= Cost per second for using the node.

To obtain the reflection of supply-demand variation, we assume that there is one machine per node and each machine contain one PE with varied MIPS rating or one can assume that all the resource-nodes are part of a big master-node. We employ space-shared allocation policy for the resources in all the models. From henceforth, we use the term "Gridlet" to refer to the *broker*.

3.4 English Auction Interaction Protocol

We consider the auction of ascending-bid type (Forward EA). This version of auction type is mostly proposed for the Grid [3]. The most crucial part of the auction is *Auctioneer*. It conducts the auction process among the interested Gridlets.

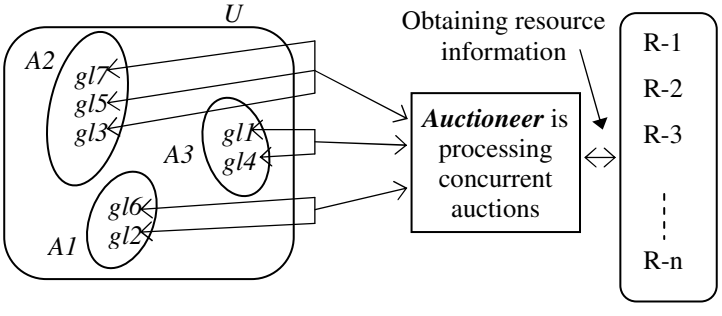


Fig. 1. Forming groups of interested bidders in EA

In our simulation scenario, multiple auctions can process concurrently which is consistent with a distributed market scenario. However, in such a scenario, a single Gridlet cannot participate in multiple auctions at the same time. The *Auctioneer* first obtains the information about resources for which the auctions are about to start. Since different Gridlets could choose different resources to compete, *Auctioneer* then requests for interested Gridlets by broadcasting the resource(s) properties to all the Gridlets. This is illustrated in Fig. 1 which describes how different Gridlets choose different resources to compete. It can be seen from Fig. 1 that Gridlet-1 and Gridlet-4 have selected node-3 for competition; since they form the group as *A3*. If the set of

available Gridlets in the market is U and the set of interested Gridlets for a particular resource n is A_n, then,

$$A_n \subseteq U \quad where, |A_n| \geq 1$$

Let a Gridlet, *gl* be the element of A_n if the respective resource n can meet the Gridlet's deadline; because if the resource is unable meet the Gridlet's deadline, there is no means to compete for the resource. Therefore, before start processing the auction(s), *Auctioneer* groups the Gridlets (bidders) depending on the Gridlets' deadlines and resources properties. The *cpuTime* of a particular Gridlet, *gl* on a resource node n is given by,

$$cpuTime = gl\text{-}length / MIPS\text{-}rating\ of\ n \quad (1)$$

If there are multiple groups competing for multiple nodes, then an auction for each node starts independently with its respective set of Gridlets. Once, the *Auctioneer* gets the A_n ready, it sends the call-for-proposal (*cfp*) to its corresponding Gridlet(s) for a counter-proposal. A *cfp* typically contains total number of rounds θ, current number of round θ_c and a *current-bid* the *Auctioneer* proposes. The bid typically starts with 0 and keeps increasing over rounds. Over each round, Gridlets decide whether to accept or reject the *current-bid*. We use the same strategy to change the *current-bid* over rounds by the *Auctioneer* as defined in the GridSim.

$$current\text{-}bid = current\text{-}bid + \{(max\text{-}bid - min\text{-}bid) / (\theta - 1)\}$$

Detailed simulation parameters are appended in Table-1. The auction continues until the total number of rounds finishes or no Gridlet is willing to accept current *cfp*.

We solve the *Winner Determination Problem* using the following two conditions,

- Gridlet that accepts the latest *cfp* and
- the bid in the *cfp* must satisfy the *cpuCost* of the Gridlet

The second condition acts as a reservation price for the resource. As we are comparing the performances of different economic models, we employ this reservation price for every model. The advantage of this price is that no resource will be provisioned below its original execution cost. In case of auction failure, the failed Gridlets are again prompted to compete for other resources for which no auction has yet been initiated. The *cpuCost* of a Gridlet *gl* on a resource-node, n is given by,

$$cpuCost = gl\text{-}length * (cost\text{-}per\text{-}sec / MIPS\text{-}rating\ of\ n) \quad (2)$$

3.5 Continuous Double Auction

We design the CDA of its most popular form, open cry with order queue [5]. In this form, resource costs (*asks*) are generated continuously until the *Auctioneer* finds a match between a *bid (cfp)* and an *ask*. *Bids* and *asks* can be placed any time during the auction phase. Outstanding bids and asks are maintained in an Order Book while *bid*s are sorted in descending order and *asks* are in ascending order. The most important part of the protocol, *Auctioneer*, is described below (Algorithm-1).

Algorithm 1 is the extended version of the existing CDA in GridSim to support deadline parameter. A similar algorithm is designed to handle new *asks*. Unsuccessful *bids* and *asks* are maintained in their respective Order Books. The *bids* are sorted in the Book in terms of their budgets and *asks* are in terms of their costs.

```
Algorithm 1. On_Receive_Bid
Input: bid, ask-Order-Book, bid-Order-Book
Output: match/mismatch, Order-Books
if (size of asks-Order-Book > 0)
    {Get the first ask from the  asks-Order-
    Book
    Cast job from the received bid
    Cast node properties from the ask
    Get budget and dl from the job
    Compute cpuCost and cpuTime using (2) &
    (1)
    if (budget • cpuCost & dl • cpuTime &
    node-status = free)
    {Inform the Auctioneer about the match
    finalPrice = (cpuCost + budget) / 2
    Update the  asks-Order-Book by removing
    the ask
    Set the node-status = busy}
    }
    }
else add the bid in the  bid-Order-Book
```

3.6 Commodity Market Model

The essence of CMM is to change resource price frequently based on supply and demand function. The price that brings the equilibrium between supply and demand in the market is called *equilibrium/spot price*. We use linear algorithms to determine the *spot price*. According to linear equilibrium theory, the demand and supply functions are given as,

$$Q^D = -aP + b$$

$$Q^S = cP + \alpha$$

Where Q^D refers to the quantity demanded at any specific time and Q^S is for supply; a, b and c are the scalar parts where a, c are the change in demand and supply respectively, b is the *current_demand* which is defined by the number of available Gridlets in the market still looking for resources. The values of a, c are ranges from 0 to 1. The negative sign in demand function presents the relationship between price (P) and demand, which is, an increase in price will induce a decrease in the quantity demanded and vice versa. In supply function, α refers to the shift in supply which can be manipulated as,

$$\alpha = (initial\_supply - current\_supply) / initial\_supply$$

$$\alpha = 1 - (current\_supply / initial\_supply)$$

Again, *a* is determined by calculating *current_demand* over the *total_demand* and *c* is calculated as *current_supply* over the *total_supply*. *Current_supply* is defined by the number of nodes available to serve Gridlets. Now, if we want to know the price at which total supply and demand at any given state diminishes, we need to solve the supply and demand functions for *P* when $Q^D = Q^S$. If P^* be our *spot price*, we get,

$$P^* = (b - \alpha) / (a + c)$$

Fig. 2. The effect of *spot* and *random prices* over supply

At every turn, *cpuCost* of a resource is computed using current *spot-price*. In order to ensure that the defined *spot price* is working for pushing the market into equilibrium, we conduct a sample experiment with 100 Gridlets and 50 nodes. The effect of *random price* and *spot price* in the market is shown in Fig. 2. The effect of *spot prices* is observable here. Because of the effect, the trend for the demand-supply ratio is smothering compared to the trend using *random prices*. On the other hand, because the random prices are not affected by supply and demand, the trend fluctuates randomly.

3.7 Tender/Contract-Net-Protocol

The roles of Gridlet and resource-node are known as manager and contractor respectively in this scenario [4]. However, to maintain the flow, we continue with stating "Gridlet" (manager) and "resource-node" (contractor). As mentioned in Section 2, in this model, Gridlets try to optimize their individual objective functions through selecting one or more resource nodes from available nodes. We have two objective functions, which are *cost* and *time* that the Gridlets are to define in their respective *cfps*.

Preference Optimization Procedure. If the final node selected by a Gridlet is *n*, we can write down,

$$n = (n \in P_n \mid cpuCost\,(n) \leq cpuCost\,(\ \forall n'\,(n' \in (P_n - n)))) \tag{3}$$

Where, P_n be the set of potential nodes those have shown interest to execute the Gridlet. Equation (1) and (2) are used by a node to decide whether to show the interest. Again, $|P_n| \geq 1$ implies that there will be at least one potential node to form P_n and n' is the number of potential nodes those have been rejected finally by the Gridlet. Equation (3) helps the Gridlet to find the resource-node with the minimum cost. A comparator is used to extract the cheapest resource from all the potential resources which is happened in the second part of the equation. In terms of time-optimization, the Gridlet uses the similar process with *cpuTime*.

3.8 Bargaining Protocol

We have briefly described about this model before. In this model, both the Gridlets and resource-nodes try to optimize their individual objective functions through negotiation. Here, a Gridlet might start with very low bid and a resource-node with very high bid and the negotiation process continues until they reach a mutually agreeable condition or any of them does not show any interest to continue [4]. We set the same number of rounds and bid update policy for BAR as we set for EA for consistency. In BAR a Gridlet and a resource-node use the following strategies to update their respective bids over the rounds.

$$cfp\text{-}bid = cfp\text{-}bid + \{(max\text{-}budget - min\text{-}budget) / (\theta - 1)\}$$

$$node\text{-}bid = node\text{-}bid - \{(max\text{-}node\text{-}bid - min\text{-}node\text{-}bid) / (\theta - 1)\}$$

If a *cfp-bid* matches with a *node-bid* over rounds, negotiation succeeds. Otherwise the Gridlet looks for other nodes to negotiate.

4 Simulative Study

Performance of a market mechanism in a Grid environment is greatly influenced by supply and demand; however existing literature only considers a limited number of users and providers and vary them with sufficient gap (e.g. 5,10,15…), which may not give a comprehensive reflection of a model's performance. We consider a parameter space consisting of Gridlets and nodes, which is a 100×100 mesh of (*s, d*), where *s* refers to the number of Gridlets (demand) and *d* is the number of nodes (supply). The inputs used in the simulation are shown in Table 1.

Table 1. Resource configuration

Parameters	Values
Number of rounds (θ)	10
MIPS rating of a node (in MIPS)	(350, 450)
Cost-per-sec for a node (in G$) (*cost*)	(1, 2)
Gridlet length (in MI)	(1000, 10000)
Gridlet deadline (in simulation sec.) (*dl*)	(12, 22)
Gridlet budget (in G$) / *max-budget*	(32, 45)
min-bid, min-budget, min-node-bid	0
max-bid, max-node-bid	45

4.1 Statistical Significance

The behavior of Grid entities is stochastic; hence achieving same performance for a particular model at different times is uncertain. To minimize this uncertainty, we test the models by using 5 different distributions (samples/seeds) and present only their averages.

The impact of economic models on Grid computing is extensive; hence, a complete evaluation of different economic models is almost impossible. There are several metrics to evaluate the strengths of the economic models in Grid computing [3]. Few of them are revenue, communication overhead, average turn-around time, resource utilization, user utility, resource utility and social welfare. We test the five economic models in terms of all of these criteria. We have chosen random values in such a way (in defining the seed value) to produce well-spaced sequences to remove correlation in the samples. Random uniform distribution is used to generate random values between the ranges (Table 1). To inject currency into the system, we use a limit. The rationality behind such currency injection is well explained in [13].

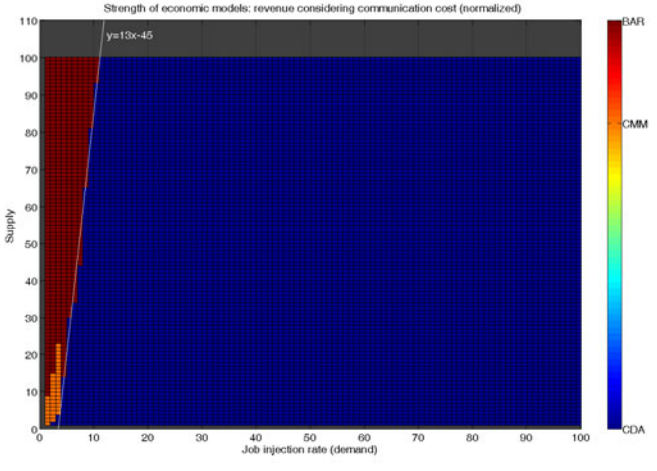


Fig. 3. Revenue comparison (considering communication cost)

4.2 Revenue

To identify which model generates more revenue for resources, we compare the five matrices representing the revenues (after averaging) for five models. We find that the EA always outperforms other four models in this case, which shows the similarity with existing literature (Section 2). While, EA generate the highest revenue, at the same time, it produces huge communication overhead/cost. Thus, we are encouraged to investigate about revenue over communication cost scenario. We normalize the both parameters and manipulate revenue over communication cost matrix. The contour diagram (Fig. 3) shows some promising outcomes. EA is fully absent here and CDA and BAR are the models show their strengths over other models. Due to the lowest communication overhead in most of the scenarios (Fig. 4), CDA performs better. However, when demand is sufficiently lower irrespective of supply, BAR

outperforms all the others. Due to higher supply, chance to start bargaining with appropriate resources is increased. Thus, Gridlets make quick acceptance on the resources, which leads to less communication overhead. Even, in this region, CMM shows less overhead than the BAR (Fig. 4), the ratio of revenue over communication cost is higher for BAR than the CMM; since in CMM, Gridlets are only paying their original job execution costs. In addition, spot prices provided by the CMM in this region are lower due to higher supply and lower demand. Thus, job execution costs are also lower. This prevents the CMM to generate more revenue in this particular region compared to BAR. To facilitate the design of autonomous switching between models, we mathematically model the strengths of CDA and BAR.

Domains of Strength. To define the individual domains, we manually choose the closest trends. To formalize the strengths, we use the terms "*s*" and "*d*" to refer to supply and demand respectively. The domain for BAR is represented by the following boundaries,

1. $d = 0$
2. $s = 13d\text{-}45$

The boundaries to define the domain of CDA are given by,

1. $s = 13d\text{-}45$
2. $s = 0$

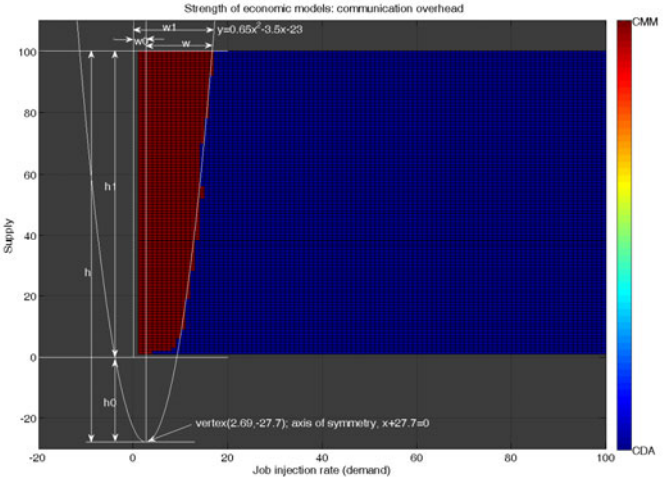


Fig. 4. Communication overhead comparison

4.3 Communication Overhead

We compute the communication overhead by counting the number of messages exchanged among entities during a simulation. Fig. 4 shows the regions where different models show their individual domains of strength. By "strength" of a particular model, we mean lower number of messages has been exchanged compared to other models. CDA and CMM are the two models that show their strengths over EA, CNP and BAR which is also conformant with the current literature (Section 2). Because of having

multiple rounds in BAR, EA, even to provision a single resource, it produces a huge amount of messages compared to other models. In terms of CNP, every potential resource sends its interest on executing a particular Gridlet. Therefore, the average number of messages exchanged to deal with a single Gridlet becomes higher.

In terms of CDA, even though resource costs are generated continuously, *Auctioneer*, in this case, sorts both *bids* and *asks* in an approach (Section 3.5), it becomes much easier and quicker to clear the market. Therefore, at the end, a Gridlet or a node does not require to send too many messages. However, when supply is higher and demand is lower, CMM performs better. Due to the higher supply and demand, spot prices generated by CMM are lower. Thus, Gridlets can quickly find their suitable resources without travelling longer in the market. Eventually, this helps for not generating a lot of messages by CMM.

These results help resource providers to choose which model to use for which scenario if a given network speed is not very high.

Domains of Strength. The domain of strength for CMM is located in the left part of the contour (Fig. 4). The boundaries are represented as,

1. $d = 0$
2. $s = 0.65d^2 - 3.5d - 23$

The boundaries to define the domain of CDA are given as,

1. $s = 0.65d^2 - 3.5d - 23$
2. $s = 0$

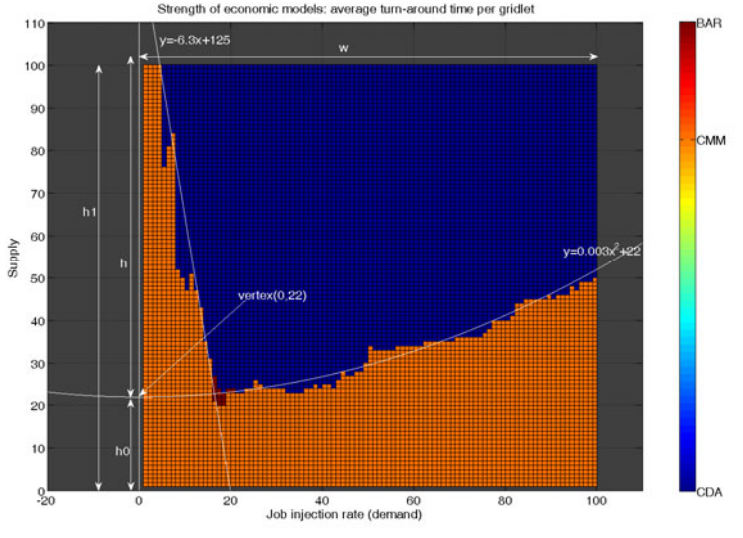


Fig. 5. Average turn-around time per Gridlet

4.4 Average Turn-Around Time per Gridlet

This metric refers to the time required by a particular Gridlet to receive its final acceptance or rejection notification. CDA and CMM are the two models that outperform

all the other models in this case (Fig. 5). In terms of higher supply and lower demand, CMM performs better. Due to lower demand over resources, the spot prices determined by the CMM are lower. This helps the Gridlets to quickly consume the resources without much delay. However, as demand increases and supply is still higher, spot prices start rising up. This forces Gridlets to stay longer in the market due to higher number of rejection by the resources. Again, when supply starts decreasing and demand starts increasing, spot prices also rise up. However, due to higher demand over the resources, most of the resources are quickly consumed, which helps other Gridlets to quickly finish the looking process due to the shortage of resources. The reason for immediate resource allocation by CDA has already been given in an earlier section.

As in previous sections, domains of strength can be identified for CDA and CMM, the details of which are omitted for brevity.

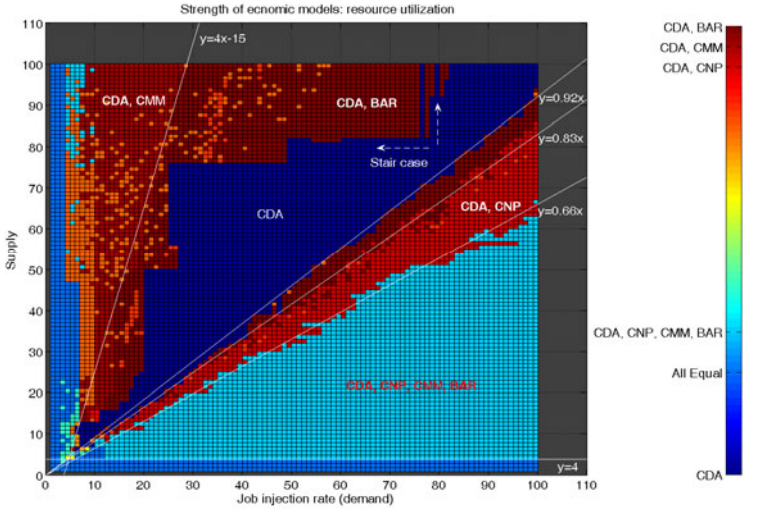


Fig. 6. Resource utilization comparison

4.5 Resource Utilization

Fig. 6 illustrates the regions where different models show their strengths for resource utilization. When the supply is higher and demand is lower, most of the Gridlets are quickly occupied without any hard competition by any of the models. Again, because of more demand and less supply, there is an equal increase in competition for a particular resource for all the models. This helps to yield equal magnitude of utilization by all the models closer to the axes level regions.

As the supply starts moving up all the models tend to perform equally better except EA. To utilize more resources, consistent resource allocation is crucial. CDA's re-arranging process (Algorithm 1) between bids and asks helps to give such consistent allocation. This is why; CDA is present across the whole space. Due to the higher demand compared to supply (until 66%), utilization equally increases for CNP, CMM

and BAR. However, as supply starts decreasing, overall utilization by the models also decreases. When supply further increases (above 66%), the randomness in selecting resources becomes significant by the Gridlets for CMM and BAR. Thus, more resources remain unutilized by the models for this region. When supply is above 83%, Gridlets in the front row by CNP, get chance to optimize their preferences quickly which leaves resources for which getting acceptance becomes harder by the rest of the Gridlets. In the middle of the contour, CDA alone performs better. Due to the higher supply, the drawback of consistent resource allocation by the other models becomes stronger. As demand starts decreasing (below 76%), due to the higher competition, utilization by all the models starts performing equally better again. As before, one can identify the domains of strength and lines where different models perform equally well from Fig. 6. Specific results have been left out for concision.

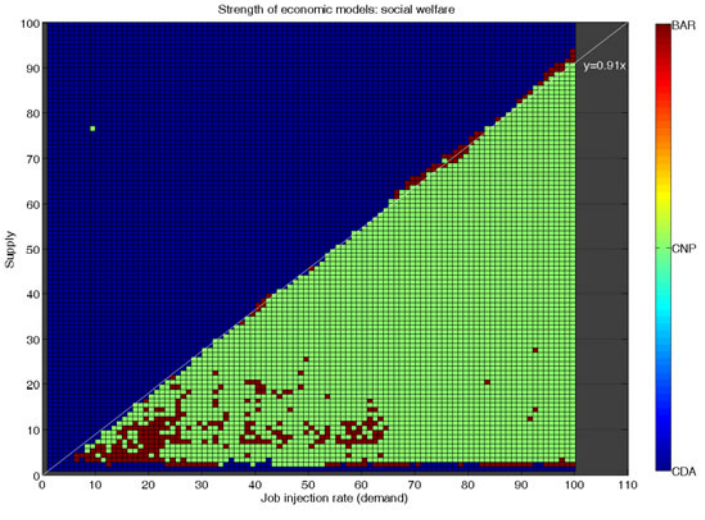


Fig. 7. Social welfare comparison

4.6 Social Welfare

Social welfare is defined in terms of user and resource utility (welfare). As the society is comprised of users and resources, the social welfare is the sum of the utilities of the users and resources. In a nutshell, it is the profit made by the whole society. User utility is defined as the difference between the user's original budget and the paid cost, whereas resource utility is defined as the difference between the agreed price and the resource's original job execution cost [14]. The contour diagram (Fig. 7) demonstrates the regions representing the strengths of different economic models for social welfare. In this case, mainly CDA and CNP are the two models that outperform all the other models in two different regions. When supply is lower irrespective of demand, BAR seems to outperform; however, due to its little contribution, we ignore this. In terms of CNP, the welfare mainly comes from Gridlet (user) utility and for

CDA, it comes from both Gridlet and resource utilities. In terms of CNP, due to its resource selection process (Eq. 3), Gridlets get chance to maximize their utilities. As supply increases, the chance increases proportionally. Due to lower demand and higher supply, Gridlets in CDA, also get chance to maximize their utilities, since, in CDA, resources are sorted according to their costs and the cheapest resource is selected to execute the trade. As demand increases, the aggregated utility for CDA also becomes higher. This is the reason why CDA performs better in this region.

As discussed earlier, domains of strength can easily be identified for CDA and CNP from Fig. 7. One can also come up with equations for the boundaries between these domains.

Our experimental findings agree with our proposed theory, i.e. in a highly competitive and dynamic environment such as Grid, a single model is not suitable to cope with every scenario and for every criterion. This can raise a research question: "how can a resource provider employ these differences in a highly dynamic environment to optimize his/her objective function(s)?" With this in mind, currently we are developing a switching agent that keeps sensing environmental changes and switches between the models dynamically to optimize a particular or a combination of objective function(s).

5 Summary and Future Work

We summarize our findings by using the following proposition which is crucial for the agent's knowledge.

Proposition 1. *There will be metrics for which no economic model performs better than the others, at least not under all circumstances, G. That will depend also on the domain, g.*

$$\forall (G)\ (\exists (g)\ \text{(one eM outperforms other models)})$$

For example, in terms of the communication overhead, we have identified two different regions where two different models (CMM and CDA) perform better compared to each other. Thus, it is possible to design an agent that dynamically switches between the models depending on their domains of strengths if a particular network would like to minimize (optimize) its overall communication cost.

We plan to develop a switching agent and like to investigate its adaptive management capabilities, when it switches from one economic model to another. We further would like to measure the real time adaptability of our system on Grid test-beds. Though we currently consider only computationally intensive applications, we believe our findings would equally be useful for data intensive applications. In data Grids, thousands or millions of datasets are stored and replicated across the Grid network [15]. These datasets are then invoked and processed by the Grid users to generate meaningful results. A range of parameters such as bandwidth for data replication, data storage capacity, computational requirements, and data security can be successfully managed using economic models. Thus, depending on the user's

application and network availability, a specific model can be switched based on changing scenario. Therefore, we also would like to study the different economic models on data-intensive applications.

6 Conclusions

Economic models have the ability to contribute to the Grid from various dimensions. It not only brings sufficient motivation for resource providers to join, but also solves distributed scheduling problems cooperatively and consistently. In this paper, we discussed five most widely proposed economic models for the Grid. Existing literature concentrates on choosing a single model for the Grid. However, due to the limitation of a single model to satisfy a large scale cooperation problem, multiple models can add value. Our experimental findings showed the consistency with existing literature that different economic models are suitable for different scenarios in Grid computing. We identified the scenarios where one economic model shows its strength over other economic models for a range of performance criteria. Finally, we believe, our findings would help the Grid community to decide which model to use for what scenario and ultimately to what reason.

References

1. Beck, R., Schwind, M., Hinz, O.: Grid Economics in Departmentalized Enterprises. J. Grid Computing 6, 277–290 (2008)
2. Buyya, R., Abramson, D., Venugopal, S.: The Grid Economy. Proceedings of the IEEE 93(3), 698–714 (2005)
3. Haque, A., Alhashmi, S.M., Parthiban, R.: A survey of economic models in grid computing. Future Generation Computer Systems 27(8), 1056–1069 (2011)
4. Buyya, R., et al.: Economic models for resource management and scheduling in Grid computing. Concurrency and Computation: Practice and Experience 14(13-15), 1507–1542 (2002)
5. Tan, Z., Gurd, J.R.: Market-based grid resource allocation using a stable continuous double auction. Paper presented at the Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (2007)
6. Richard, W., James, S.P., John, B., Todd, B.: G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid. Paper presented at the Proceedings of the 15th International Parallel and Distributed Processing Symposium (2001)
7. Caramia, M., Giordani, S.: Resource allocation in grid computing: an economic model. J. WSEAS Trans. Comp. 3(1), 19–27 (2008)
8. Dias de Assunção, M., Streitberger, W., Eymann, T., Buyya, R.: Enabling the Simulation of Service-Oriented Computing and Provisioning Policies for Autonomic Utility Grids. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 136–149. Springer, Heidelberg (2007)
9. Resinas, M., Fernandez, P., Corchuelo, R.: A Conceptual Framework for Automated Negotiation Systems. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) IDEAL 2006. LNCS, vol. 4224, pp. 1250–1258. Springer, Heidelberg (2006)

10. Ivona, B., Srikumar, V., Michael, M., Rajkumar, B.: Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services. In: Workshop on Service-Oriented Engineering and Optimizations 2008, in conjunction with International Conference on High Performance Computing 2008 (HiPC 2008), Bangalore, India, December 17-20 (2008)
11. Buyya, R., Murshed, M.: GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. Concurrency and Computation: Practice and Experience 14(13-15), 1175–1220 (2002)
12. Walfredo, C., Francisco, B., Jacques, S., Nazareno, A., Daniel, P., Elizeu, S.-n., Raissa, M., Federal, C.: Grid computing for bag of tasks applications. In: Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, Brazil (2003)
13. Broberg, J., Venugopal, S., Buyya, R.: Market-oriented Grids and Utility Computing: The State-of-the-art and Future Directions. J. Grid Computing (2008)
14. Dumas, M., Aldred, L., Governatori, G., ter Hofstede, A.H.M.: Probabilistic Automated Bidding in Multiple Auctions. Electronic Commerce Research 5(1), 25–49 (2005)
15. Venugopal, S., Buyya, R., Winton, L.: A Grid service broker for scheduling e-Science applications on global data Grids. Concurrency and Computation: Practice and Experience 18(6), 685–699 (2006)

Concurrent Negotiations in Cloud-Based Systems

Melanie Siebenhaar, The An Binh Nguyen, Ulrich Lampe,
Dieter Schuller, and Ralf Steinmetz

Multimedia Communications Lab (KOM)
Technische Universität Darmstadt, Germany
melanie.siebenhaar@KOM.tu-darmstadt.de

Abstract. Utilizing cloud-based services, consumers gain a high level of flexibility, but they cannot obtain individual Quality of Service guarantees or request service compositions according to their specific business needs. Therefore, appropriate mechanisms for an automated negotiation of Quality of Service parameters are required that do not only consider the individual business objectives and strategies of the negotiation partners involved, but do also account for the dependencies between the different services and service tiers in cloud computing. This enables enterprises to increase the quality and flexibility of their business processes and lays the foundation for market-based complex service provisioning. In this paper, we present one such negotiation approach and evaluate the application of different negotiation strategies.

Keywords: Concurrent Negotiation, Cloud Computing, SLA.

1 Introduction

In recent years, enterprises have experienced an increasing need to provide a flexible and competitive business process infrastructure, with IT systems serving as the key enabler. Several computing paradigms have been introduced, which promise to provide more IT flexibility. The latest of these is cloud computing, which enables on-demand access to arbitrary resources as a service (e.g., storage). Although cloud computing is already applied in practice, current vendors have only concentrated their effort on specific issues (e.g., scalability). To date, they offer no or only limited support for dynamic negotiation of individual Quality of Service (QoS) guarantees [3]. Hence, obtaining cloud services according to consumers' specific business constraints remains an open issue. Yet, quality parameters such as reliability and availability are crucial in a business environment. In order to retain control of the service quality, Service Level Agreements (SLAs) can be negotiated to ensure the desired quality is maintained. An SLA is a formal agreement, i.e., a contract between two parties and specifies the consumer's objectives (e.g., QoS parameters) that must be fulfilled by a provider and penalties in case of violations.

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 17–31, 2012.

Negotiating individual SLAs in cloud computing is challenging. Negotiations typically involve consumers and providers with conflicting interests. Consumers usually want to obtain a high-quality service at low costs. Likewise, providers try to achieve the highest possible profit in line with demand, given their currently available QoS levels and capacities. Finally, consumers and providers each have to decide on a promising negotiation strategy. Since both parties try to achieve the highest possible utility and, due to the business context, do not want to disclose too much private information (e.g., business goals, cost factors), a negotiation of QoS parameters is necessary.

In addition, there can be multiple competing providers in the market, which offer the same type of service, but with different properties. Therefore, consumers wish to explore the heterogeneous service properties of different providers in advance in order to determine the most suitable services before establishing an agreement with a specific provider. Furthermore, if consumers want to combine services from different cloud providers, also the composition must satisfy the consumers' QoS requirements. Hence, it is necessary to negotiate concurrently with multiple providers from a consumer's point of view. Likewise, it is also required to conduct concurrent negotiations with multiple consumers from a provider's point of view in order to determine the consumers which generate the highest possible profit. Further challenges arise in relation to the several dependencies between the different service tiers (e.g., software, infrastructure) [2] in cloud computing. While service consumers wish to obtain specific services from different service providers, the service providers in turn must acquire the necessary resources for service execution from infrastructure providers. Hence, besides the issue of resource availability, also the adherence to SLAs across different administrative domains has to be considered.

Due to the large number of cloud providers and cloud consumers, the information exchange between the parties involved is very complex. Thus, a dynamic, scalable, and automated approach is required for negotiating SLAs with multiple providers across heterogeneous domains.

In the past, several approaches have been proposed for SLA negotiation in different fields of research (e.g., [17], [19]). However, very few effective solutions for automated negotiation have been provided so far [18]. In this paper, we present an approach for negotiating SLAs with multiple cloud providers across multiple tiers. We propose a cloud negotiation support system (CNSS) that can be employed on every service tier. Furthermore, we compare different negotiation strategies to be applied in our scenario.

The remainder of the paper is structured as follows. Section 2 discusses related approaches in the field of concurrent negotiations with multiple parties. In Section 3, the requirements for negotiating SLAs with multiple cloud providers across heterogeneous domains are described. Section 4 introduces our negotiation approach and Section 5 presents initial experimental results of our evaluation of different negotiation strategies. The paper closes with a conclusion and future directions in Section 6.

Table 1. Overview of Related Work

Publication	*Mult. Issues*	*Mult. Cust.|Prov.*	*Concurrency*	*Coordination*	*Protocol*	*Strategy*
This approach	×	××	×	×	Mod. Extended CNP	Time-dependent
Aknine et al. [1]	–	××	×	–	Extended CNP	–
Chhetri et al. [4]	×	–×	–	×	Alternate Offer + CNP	–
Di Nitto et al. [7]	×	××	–	×	–	Optimization
Sim and Shi [13]	–	–×	×	×	Alternate Offer	P_{Renege} + Time
Dang and Huhns [6]	×	××	×	–	Alternate Offer	–
Sim [15]	–	××	×	×	Alternate Offer	Market-driven

2 Related Work

Several approaches for concurrent negotiations with multiple providers have been proposed so far in different fields of research (cf. Table 1).

Aknine et al. [1] present an extension of the contract net protocol (CNP) [16] in order to support concurrent many-to-many negotiations. Basically, the contract net protocol is a simple one round-based protocol for task distribution in IT systems. The authors introduce a two-phase negotiation process which enables prospective contractors to overbid other offers. However, strategies and the assignment of multiple tasks are not considered in their approach.

Chhetri et al. [4] also adapt the CNP and propose a coordinated architecture for agent-based SLA negotiations. In their architecture, a global coordinator agent is responsible for determining a service composition according to consumer's QoS requirements. Local negotiation agents in turn conduct negotiations with multiple providers in order to achieve an agreement for a specific service type. A negotiation agent negotiates with multiple providers in an iterative manner over multiple rounds. No bidding strategies are specified.

In [7], Di Nitto et al. suggest a search-based solution for SLA negotiation. Similar to the work in [4], each negotiation participant is represented by a coordinator and several negotiators. In contrast to [4], Di Nitto et al. make use of an intermediate mediator in the form of a marketplace. The marketplace issues proposals to the participants based on an optimization algorithm in order to improve the convergence of the offers. The authors do not explicitly state a protocol for message exchange and private information is disclosed to the marketplace.

Sim and Shi [13] propose an approach for allocating multiple types of resources to perform a particular computation in grid computing. Consumers and providers apply a time-dependent strategy and are allowed to break an intermediate contract by paying a penalty fee. In addition, the strategy of the consumers is based on the calculation of the expected utility of the proposals and the probabilities

that providers will renege from an intermediate contract. This approach requires the management of commitment and decommitment of contracts. Furthermore, a breach of contract may affect reputation.

In [6], Dang and Huhns adapt the alternate offers protocol in order to support concurrent negotiations with multiple issues in case of many-to-many negotiations. Their approach is based on the extended CNP [1] and also introduces two negotiation phases. Since counter-proposals can overbid formal proposals, it is obvious, that the negotiation may result in an infinite loop. The authors argue that this situation can be prevented by enforcing time constraints. However, time-dependent strategies are not considered in their approach.

Sim [15] focuses on market-based SLA negotiations in cloud computing. He considers a three-tier model where negotiation takes place between consumers, brokers and resource providers. A market-driven strategy is applied, where the concession amount depends on time, trading alternatives and competition. In his model, an agent can also renege from an intermediate contract by paying a penalty fee. The main goal for consumers is price minimization. However, for general SLA negotiation support, other QoS parameters must also be considered.

Our approach allows to combine services from multiple providers and to negotiate individual QoS parameters across multiple tiers. We introduce coordinating entities to manage the composition and make use of a two-phase protocol for concurrent negotiations with multiple providers. In the second phase of the protocol, we permit multiple overbidding. Furthermore, we apply time-dependent strategies, which stop the overbidding process if necessary.

3 Negotiation Model and Requirements

The work at hand focuses on service composition in cloud computing, where m services from different providers can be combined to form a complex service. Our approach is based on a market model where consumers submit their requirements to brokers in terms of desired functional and non-functional properties for specific services [3]. The brokers have access to a service registry in the market. By querying the registry, a broker is able to determine the sets of the most suitable cloud providers for the different m services based on the functional properties. A broker acting on behalf of a consumer conducts the negotiation of the non-functional properties, i.e., QoS parameters for the m services resulting in an agreement or in the breakdown in negotiations. For this purpose, a broker has to start m negotiation processes. Each process consists of concurrent one-to-many negotiations for a specific type of service with a set of providers. In addition, it may also be necessary for a service provider to initiate further negotiations with multiple providers on the lower resource level in order to lease infrastructure for the deployment of a specific service instance. To realize our approach, an appropriate negotiation mechanism is required to conduct concurrent negotiations with multiple providers, even across multiple tiers.

3.1 One-to-Many Negotiations

Basically, a negotiation mechanism consists of two components: a negotiation protocol and the negotiation strategies of the negotiating parties [9]. The negotiation protocol specifies the rules for interaction (i.e., message exchange, conditions for agreement) between the negotiating parties and the negotiation strategies must be compatible with the applied protocol. A negotiation strategy defines the sequence of actions planned to make during negotiation by a participant.

In our scenario, each negotiation concerns a specific service and the negotiating parties have multiple conflicting interests (e.g., price). The conflicting interests refer to the negotiable values in the form of non-functional parameters of a service. Similar to Microsoft Office[1] and research conducted in the area of Web services [19], we assume that each service is offered in the form of different priced packages (e.g., Gold, Silver, Bronze).

As input for negotiation, consumers and providers must specify their requirements for each service. We assume that the consumer specifies ranges for the several QoS parameters reflecting the lower and upper bounds he is willing to accept (e.g., response time between 5 ms and 10 ms). Since some parameters may be more important than others, we also assume that the consumers and providers specify weights for each QoS parameter. Furthermore, a goal is required on both, consumer and provider side, in order to make decisions during each round of the negotiation process and, ultimately, to reach an agreement. On both sides, the goal can be expressed based on the expected benefit from a given service offer. From a consumer's perspective, this can be mathematically expressed as follows: Given a set of m attributes $X = \{x_1, ..., x_m\}$ and different weights $W = \{w_1, ..., w_m\}$ for the attributes with $\sum_{i=1}^{m} w_i = 1$ for a desired cloud-based service. Let $U_e^t = f(W, X)$ be the expected utility of the service consumer in round t and let $U_e^{'}$ be the minimum expected utility of the service consumer. Further, let $O^t = \{o_1, ..., o_n\}$ be the set of service offers provided by n cloud providers in round t and $U^t(o_i)$ be the utility of the consumer for the service offer from the i^{th} provider. Given these parameters, the general goal is to choose a service offer o_i during negotiation that results in a minimum distance between the consumer's expected utility U_e^t and the utility $U^t(o_i)$ of the service offer o_i to the consumer (cf. Equation 1).

$$\arg\min_i f(i) = \left|(U_e^t - U^t(o_i))\right| \text{ where } U^t(o_i), U_e^t \geq U_e^{'} \tag{1}$$

3.2 Complex Cloud Service Negotiation Requirements

Several implicit and explicit assumptions are already part of the one-to-many negotiation model mentioned above. However, further requirements must be taken into account for our global cloud service composition scenario.

[1] http://office.microsoft.com/en-us/buy/office-2010-which-suite-is-right-for-you-FX101825640.aspx [last access: 2011-10-01]

- **Common Protocol:** In order to enable interoperability, consumers and providers must agree on a common protocol first before participating in a negotiation. This issue is addressed by conducting so-called meta-negotiations (e.g., [3]), which are performed before the actual negotiation takes place. Meta-negotiations are not part of our work.
- **Imperfect Information:** To allow for optimal negotiation results, the scoring information must be public [11]. However, if the parties have competing interests, the parties do not want to disclose their strategies to other parties. Hence, some information must be private (e.g., decision models) and other information must be public (e.g., expected QoS parameters of a consumer).
- **Time Limits:** We assume a given time limit as a condition for the completion of negotiations, since service brokers do not have an infinite amount of time to reach an agreement [12].
- **Administrative Domains:** In a common three-tier cloud model [2], users need to negotiate with service providers, who must, in turn, negotiate with resource providers to acquire the required resources. The QoS levels provided by service providers depend on the QoS levels provided by the resource providers. Hence, SLAs must be established across tiers while considering the dependencies between the QoS levels on different tiers.
- **Coordination:** Besides the dependencies across tiers, it is also necessary to coordinate the service composition. Coordinators must be established in order to balance the dependencies between the QoS parameters of the different services and to manage the available resources (e.g., [4], [7]).
- **Security:** The communication between the negotiating parties must be performed in a secure manner (e.g., SSL encryption of messages). These issues are not considered in our work.

4 Approach

Based on the assumptions and requirements outlined in the last section, we propose an approach for concurrent negotiations in cloud-based systems in order to combine services from multiple providers. From the discussion of related work in Section 2 it follows that coordinating entities are required when consumers want to combine services from multiple providers. In addition, other entities are necessary, which are controlled by coordinators, to conduct the negotiations.

4.1 Negotiation Architecture

Our negotiation architecture is based on the models proposed by Di Nitto et al. [7] and Chhetri et al. [4]. Each entity participating in the negotiation is represented in the form of a Cloud Negotiation Support System (CNSS) as depicted in Figure 1. After having received a service composition request from a consumer, a broker determines the most suitable cloud providers for each of the different services based on the consumer's functional requirements. Subsequently, the service composition request is passed to the CNSS of the broker together with the provider lists. The

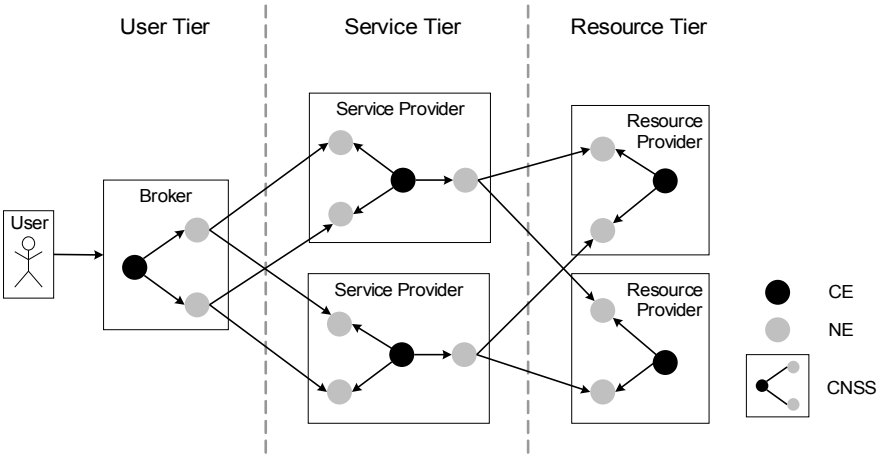


Fig. 1. Negotiation Architecture

CNSS then creates a coordinating entity (CE) responsible for managing the concurrent negotiations with the providers. The CE in turn creates several negotiating entities (NE) according to the number of services in the composition request. Each of them has the task to negotiate concurrently with multiple providers over the QoS parameters of a particular service. It can be observed in Figure 1, that this setup enables negotiations across different tiers. Service providers who want to obtain resources from resource providers initiate concurrent negotiations in a similar way. That is, a CE creates the required number of NEs which start negotiations with the respective resource providers. In order to account for the QoS parameters resulting from the negotiations with the brokers on consumer side, the CE of a service provider's CNSS must also observe these negotiations. Hence, the tasks of a CE can be defined from two perspectives.

- **CE on behalf of consumers:** Before initiating negotiations with providers, a CE has to split the consumer's requirements into global requirements for the composition and local requirements for each service that is part of the composition. The global requirements are observed by the CE and the local requirements are passed to the respective NEs. Since the CE manages the whole negotiation process, additional information such as session deadlines, reserve values or strategies is also passed to the NEs.
- **CE on behalf of providers:** A CE on provider side must be aware of the currently available service levels or resources. Furthermore, CEs on provider level must pass this information to NEs and observe their behaviour during negotiation on a higher level.

4.2 Two-Phase Negotiation Protocol

Communication between negotiating parties is essential in order to reach an agreement. Furthermore, the different parties must be aware of the rules of the

Table 2. List of Message Types

Message ($CE \leftrightarrow NE$)	**Meaning**
inform	CE passes required negotiation information to NE
notify	NE uses notify message to report negotiation status
Message ($NE \leftrightarrow NE$)	**Meaning**
call for proposal	Negotiation is initiated by requesting for proposals
propose	Negotiation is initiated with initiated proposal
pre-proposal	Agent makes pre-proposal in the warum-up phase
pre-accept	Agent temporarily accepts a proposal
pre-reject	Agent temporarily rejects other agent
definitive-accept	Agent is formally bound to an agreement
definitive-proposal	Agent makes final (formal) proposal
definitive-reject	Agent rejects other agent completely
withdraw	Agent leaves the negotiation

negotiation they are participating in. Basically, the rules for interaction can be specified in the form of a negotiation protocol. In our negotiation architecture, a CE creates, informs, and also commands NEs if required. The NEs in turn communicate with multiple other NEs outside their own CNSS by exchanging messages. The message types, which are required in our concurrent negotiation scenario, are depicted in Table 2. The proposed message types are based on the FIPA specification [8] and the work by Aknine et al. [1]. As prerequisite for determining preferences for service offers and for conducting automated negotiations, the issues must be expressed in a common and formal way. Our expression is based on that proposed by Sierra et al. [11]. Each issue is characterized by a constraint interval in terms of a minimum and maximum value. Typically, either the minimum or the maximum value for an issue will be part of the initial proposal. The remaining upper or lower bound of the interval represents the reserve value. In our scenario, a proposal is defined as follows.

Definition 1 (Proposal). *A proposal P is a message sent from a participant p to an opponent o and contains a set S of n issues and their values proposed by p. Each issue x_i with $i = 1...n$ of the proposal corresponds to a specific quality parameter of a cloud service under negotiation. A proposal is valid, if every issue in the proposal is valid. An issue x_i is considered a valid issue, if its proposed value is within the intervals of p and o defined for this issue, i.e., $\forall x_i \in P$, $x_i \in [min_i^p, max_i^p]$ and $x_i \in [min_i^o, max_i^o]$.*

During negotiation, each party aims to maximize its utility. Since the negotiating parties have conflicting interests, concessions have to be made in order to reach a mutual agreement. The amount of concession depends on the applied strategy. In order to reach an agreement, the negotiating parties typically alternate in making proposals. Based on the issues in the proposal, the utility for a proposal is calculated. Intuitively, a participant prefers one proposal over

another, if the utility of the opponent's proposal to the participant is equal or higher than the utility of the participant's own proposal. In this case, a proposal is acceptable. However, since the negotiations in our scenario involve multiple parties, more than a single acceptable proposal may exist. Therefore, a proposal can be (temporarily) accepted, if the following condition holds [5]:

Definition 2 (Acceptance Condition). *Let B_i with $i = 1...n$ be a set of providers a consumer C negotiates with. A Proposal $P_{B_i \to C}$ sent from provider B_i to consumer C can be accepted by C, if $U_C(P_{C \to B_i}) \leq U_C(P_{B_i \to C})$ and $U_C(P_{B_i \to C}) = max(U_C(P_{B_j \to C}))\, \forall P_{B_j \to C}$.*

After having given a formal expression for the negotiation issues and an acceptance condition for proposals, we can now elaborate on the different message types of the negotiation protocol applied in our scenario. The negotiation protocol is based on the protocols proposed by Dang and Huhns [6] and Aknine et al. [1]. As in their approaches, we also introduce two phases into the negotiation process: a *warm-up* and a *countdown* phase. During the warm-up phase, proposals are exchanged to find mutual interests. Afterwards, providers have to compete with other providers for the best proposal during the countdown phase. A two-phase approach has been chosen, since this protocol supports negotiations in a flexible manner. The negotiation participants are not immediately bound to the first accepted proposal. Instead, the negotiation continues, which allows the participants to find more appropriate proposals. The different message types are described in the following.

Message exchange between CE and NE: CEs create NEs to conduct concurrent negotiations with multiple providers for a particular type of service. An *inform* message is sent from a CE to an NE to provide the NE with all the information required for a negotiation (e.g., deadline, reserve value, strategy). In addition, a CE can send an inform message to interfere in the negotiation process (e.g., withdraw an NE from a negotiation). An NE in turn is able to report the negotiation status to a CE by sending a *notify* message.

Message exchange between NE and NE: NEs initiate negotiations with providers upon request of a CE by either sending an initial *proposal* or a *call for proposal* message to their opponents. In the latter case, NEs request initial proposals from providers. The negotiation is carried out in rounds. During the warm-up phase, the participants alternate in sending *pre-proposal* and counter pre-proposal messages until there is at least one acceptable proposal. An NE then determines the best proposal and sends a *pre-accept* message to the owner of this proposal and *pre-reject* messages to all other opponents. Subsequently, the receiver of the pre-accept message formulates a *definitive-proposal* message and sends it back to the NE. Other opponents, who received a pre-reject message, may still participate in the negotiation. They can formulate pre-proposal messages and try to

overbid the currently best proposal. Even a definitive-proposal can be overbid by a pre-proposal sent by one of the participants in the last round. The overbidding process continues until a definitive-proposal is accepted as the best proposal. The owner of this proposal is notified by sending a *definitive-accept* message. All others receive a *definitive-reject* message and the negotiation is over. The last message type to be mentioned is the *withdraw* message. Such a message can be sent in any round during negotiation by a participant, who wants to leave the negotiation.

Simply by using the negotiation protocol, there is no guarantee that an agreement can be reached. Therefore, appropriate negotiation strategies are also necessary in order to increase the probability of reaching an agreement.

4.3 Negotiation Strategies

Negotiation strategies do not only affect the probability of reaching an agreement, but also affect the quality of an agreement (e.g., in terms of the achieved utility). Different strategies have been proposed in related work so far. Since time plays an important role in negotiation, even more so when concurrent negotiations are conducted between multiple competing parties, we apply and assess time-dependent strategies in our scenario.

In Section 4.2 we stated that each participant defines ranges $x_i \in [min_i, max_i]$ for each negotiable issue. The utility values for an issue are typically defined as a value within a range from 0 to 1, with 1 representing the highest utility. Concerning the negotiation strategy, we find it intuitive to make an initial offer with the most preferred value for each issue and make concessions during negotiation to reach an agreement. Therefore, the utility value for the i-th issue x_i of a proposal P can be calculated by a utility function U as follows (e.g., [19], [14]):

$$U_P(x_i) = \begin{cases} \frac{max_i - x_i}{max_i - min_i} & \text{if a decrease of } x_i \text{ leads to a higher utility} \\ \frac{x_i - min_i}{max_i - min_i} & \text{if an increase of } x_i \text{ leads to a higher utility} \end{cases}$$

It can be obtained from the equations that the most preferred value for an issue is the minimum value in the first case and the maximum value in the second case. In order to determine the concession amounts for each issue in round t, we apply the time-dependent tactics as proposed by Sierra et al. [11]. Basically, a negotiation strategy can be defined by using one or more tactics. A pure strategy applies a single tactic to compute the next value for an issue, while a mixed strategy makes use of a weighted combination of tactics [10]. In our scenario, we use pure strategies based on the time-dependent tactic family. Using a time-dependent tactic, a counter proposal for issue i at round $t \leq t_{max}$ with t_{max} representing the deadline is calculated as follows [11]:

$$x_i = \begin{cases} min_i + \alpha_i(t)(max_i - min_i) & \text{if } U_P(x_i) \text{ is decreasing} \\ min_i + (1 - \alpha_i(t))(max_i - min_i) & \text{if } U_P(x_i) \text{ is increasing} \end{cases}$$

Function $\alpha_i(t)$ determines the concession amount made for issue i in round t. Besides the exponential function stated below, also a polynomial function can be used. We decided to use the exponential function, since it concedes slower at the beginning. The function is calculated as follows [11]:

$$\alpha_i(t) = e^{(1-\frac{t}{t_{max}})^{\beta} \ln k_i}$$

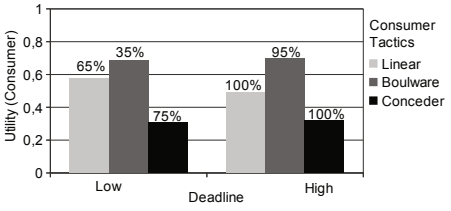
The initial proposal can be obtained from $\alpha_i(t)$ by multiplying k_i with the size of the constraint interval. The constant k_i can be chosen based on experience. The tactics are parameterized by the value β, which influences the concession amount [11]:

- Conceder ($\beta > 1$): the negotiating party makes a great concession already at the beginning and soon reaches the reserve value
- Linear ($\beta = 1$): the negotiating party makes an equal[2] concession in each round during negotiation
- Boulware ($\beta < 1$): the negotiating party maintains its proposed value nearly all the time and only makes a great concession shortly before the deadline

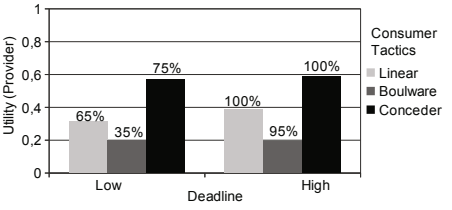
5 Experimental Results

This section presents initial experimental results of our approach. For this purpose, the two-phase negotiation protocol as described in Section 4.2 has been implemented. Furthermore, we perform a comparison of the different time-dependent strategies applied for concurrent one-to-many negotiations in our cloud-based scenario. The evaluation is a proof-of-concept for the proposed approach and at the same time analyzes the influence of different provider and consumer strategies as well as different deadlines on the success rate and on consumer and provider utility. The simulations have been performed on a laptop with a Core 2 Duo 2.40GHz processor with 2 GB RAM and Ubuntu 11.04 as operating system. The different negotiating entities are represented by software agents. Repast Simphony[3] has been used as agent framework. Our setup for the experiments uses the settings applied in [10] as guideline. In each experiment, a consumer negotiates concurrently with 10 cloud providers and a specific type of strategy is applied one each side. All providers use the same type of strategy, but the strategic parameter β (i.e., the concession amount) can be different. The strategies on consumer and provider side do not change during negotiation. We performed separate experiments to analyze the effect of different strategies and deadlines on the utility values and success rate. Each strategy has been evaluated with respect to the other three time-dependent strategies using either low or high deadlines on both sides. Hence, comparing 3 consumer strategies with 3 provider strategies using 2 different deadlines we obtain a total number of 18 experiments. For each experiment, 20 test cases have been generated. The β value of the consumer is 0.5, 1, and 2.5

[2] When using the exponential $\alpha_i(t)$ function concession is made in near constant rate.
[3] http://repast.sourceforge.net/

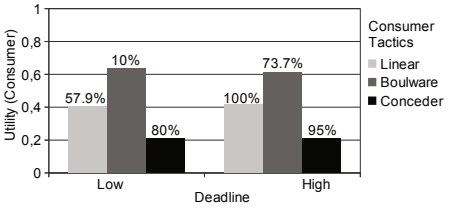


(a) Linear Tactic for Provider

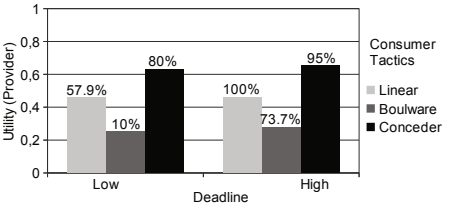


(b) Linear Tactic for Provider

Fig. 2. Average Consumer and Provider Utility and Linear Tactic for Provider



(a) Boulware Tactic for Provider



(b) Boulware Tactic for Provider

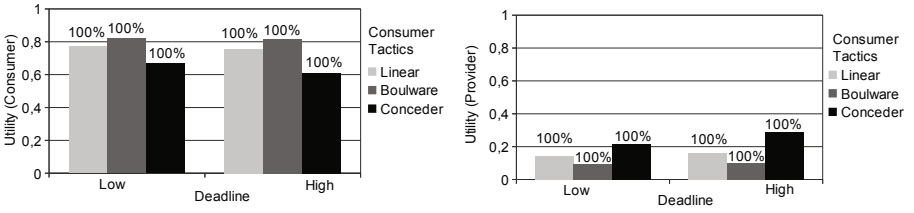
Fig. 3. Average Consumer and Provider Utility and Boulware Tactic for Provider

for strategy boulware, linear, and conceder, respectively. Providers' β values are randomly selected between (0.0,1.0) if boulware strategy is used, are equal to 1.0 if the strategy is linear, and are in range (1.0,5.0] if the conceder strategy is applied. In each experiment, consumer and providers either randomly select their deadline between [10,15] rounds in case of a low deadline experiment or between [25,30] rounds in case of a high deadline experiment. The initial proposals of consumer and providers are also generated at random for each of the 20 scenarios, but the 20 scenarios are fixed for all of the 18 experiments. Proposals in our experiments comprise three different issues: price, response time, and availability. The ranges of the different issues used for proposal generation are listed in Table 3.

The outcomes of the evaluation are depicted in Figures 2 to 4. The Figures display the average utility either to consumer or provider of all successful deals depending on the strategies and deadlines used in each experiment. In addition, each bar indicates the percentage of successful deals (i.e., success rate). For each experiment, average utility and success rate are calculated as follows:

$$avg.\ utility_{cust/prov} = \frac{\sum utility_{cust/prov}}{|Deals_{succ}|} \qquad succ.\ rate = \frac{|Deals_{succ}|}{|Deals_{all}|} \qquad (2)$$

The average run times of a single scenario in case of low and high deadlines are 50 ms and 80 ms, respectively. The simulation of a single experiment (20 scenarios) takes between 0.7 and 2.5 seconds. It can be obtained from the results that the deadline has only a marginal effect on the utility values of the consumer and provider, but even more on the success rate of the negotiations. Therefore, it can



(a) Conceder Tactic for Provider (b) Conceder Tactic for Provider

Fig. 4. Average Consumer and Provider Utility and Conceder Tactic for Provider

be more beneficial to make use of a higher deadline in order to reach an agreement. Nevertheless, in case of low deadlines, the success rates of the different strategies differ considerably. Hence, if there is an urgent need for a consumer to reach an agreement, applying a conceder strategy or, as the second best option, a linear strategy, gives a consumer the biggest chance to succeed. On average, the linear strategy provides a greater utility to the consumer than the conceder strategy. The boulware strategy is the dominant strategy on consumer side concerning the amount of utility, but only with little chance to succeed in case of low deadlines. On provider side, the boulware and the linear strategy provide the highest utility to the provider. They do not differ very much concerning the amount of utility, but their success rates slightly differ depending on the strategy applied by the consumer. The conceder strategy should not be applied from a provider's point of view, since it achieves the worst utility values in all experiments. Finally, the utility values of consumer and provider must be compared to each other. If the consumer applies a boulware strategy, the amount of utility to the consumer is always higher than the amount of utility to the provider. The reverse case occurs, when a consumer applies the conceder strategy, except for this situation when the provider also applies the conceder strategy. Nearly similar utility values are obtained on average, when the consumer applies a linear strategy and the provider applies a boulware strategy.

Summarizing, if we consider the best trade-off between amount of utility and success rate, a consumer should apply a linear strategy in case of low deadlines and a boulware strategy in case of high deadlines. On the provider side, the linear strategy has higher success rates on average, but the boulware strategy provides a slightly higher amount of utility to the provider. If a balance must be achieved between the average utility values of consumer and provider, a consumer should apply a linear strategy while a provider uses a boulware strategy.

Table 3. Ranges for Consumer and Provider Proposals

Issue	**Consumer**		**Provider**	
Price	min[8,12]	max[18,22]	min[10,15]	max[20,25]
Resp. Time	min[1,5]	max[10,15]	min[1,7]	max[13,17]
Availability	min[-99.999,95]	max[-85,-80]	min[-99.9,-90]	max[-80,-70]

6 Conclusion

In this paper, we have presented an approach for concurrent negotiations in cloud-based systems. We have introduced a CNSS-based architecture and a two-phase negotiation protocol to conduct negotiations with multiple providers across multiple tiers. In addition, we have evaluated the applicability of three different time-dependent strategies for negotiation in our scenario. The results reveal that the deadline has a large effect on the success rate, but less on the achieved utility. Furthermore, we have seen in low deadline experiments that the best strategy of a consumer in terms of utility and success rate highly depends on the strategy of the opponent. Therefore, appropriate mechanisms have to be developed to determine a previously unknown strategy or new strategy variants of the opponent. This permits an agent acting on behalf of a consumer to better react to the providers' strategies. For this purpose, also appropriate decision support systems have to be developed, which choose the most promising strategies for consumers. Further enhancements of our approach are the evaluation of other negotiation strategies (e.g., resource dependent tactics) and the assessment of mixed strategies. Finally, a coordinator mechanism must be developed, which specifies the actions of a CE, when and how to interfere in negotiations.

Acknowledgments. The work presented in this paper was performed in the context of the Software-Cluster project SWINNG (www.software-cluster.org). It was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. "01|C10S05". In addition, this work is supported in part by E-Finance Lab Frankfurt am Main e.V. (http://www.efinancelab.com). The authors assume responsibility for the content.

References

1. Aknine, S., Pinson, S., Shakun, M.: An Extended Multi-Agent Negotiation Protocol. Autonomous Agents and Multi-Agent Systems 8(1), 5–45 (2004)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A View of Cloud Computing. Communications of the ACM 53(4), 50–58 (2010)
3. 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 Generation Computer Systems 25(6), 599–616 (2009)
4. Chhetri, M.B., Lin, J., Goh, S., Zhang, J.Y., Kowalczyk, R., Yan, J.: A Coordinated Architecture for the Agent-based Service Level Agreement Negotiation of Web Service Composition. In: Proceedings of the Australian Software Engineering Conference, pp. 90–99 (2006)
5. Dang, J., Huhns, M.: An Extended Protocol for Multiple-Issue Concurrent Negotiation. In: Proceedings of the 20th National Conference on Artificial Intelligence, pp. 65–70 (2005)
6. Dang, J., Huhns, M.: Concurrent Multiple-Issue Negotiation for Internet-based Services. IEEE Internet Computing 10(6), 42–49 (2006)

7. Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.L.: Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 295–306. Springer, Heidelberg (2007)
8. FIPA: Library Specification, `http://www.fipa.org/specs/fipa00037/SC00037J.html`
9. Lomuscio, A.R., Wooldridge, M., Jennings, N.R.: A Classification Scheme for Negotiation in Electronic Commerce. Agent-Mediated Electronic Commerce: A European Agent Link Perspective 12(1), 31–56 (2003)
10. Patel, D., Gupta, A.: An Adaptive Negotiation Strategy for Electronic Transactions. In: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 243–252 (2006)
11. Sierra, C., Faratin, P., Jennings, N.R.: A Service-Oriented Negotiation Model between Autonomous Agents. In: Boman, M., Van de Velde, W. (eds.) MAAMAW 1997. LNCS, vol. 1237, pp. 17–35. Springer, Heidelberg (1997)
12. Cosmin Silaghi, G., Dan Şerban, L., Marius Litan, C.: A Framework for Building Intelligent SLA Negotiation Strategies under Time Constraints. In: Altmann, J., Rana, O.F. (eds.) GECON 2010. LNCS, vol. 6296, pp. 48–61. Springer, Heidelberg (2010)
13. Sim, K., Shi, B.: Concurrent Negotiation and Coordination for Grid Resource Coallocation. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 40(3), 753–766 (2010)
14. Sim, K., Wang, S.: Flexible Negotiation Agent with Relaxed Decision Rules. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(3), 1602–1608 (2004)
15. Sim, K.M.: Towards Complex Negotiation for Cloud Economy. In: Bellavista, P., Chang, R.-S., Chao, H.-C., Lin, S.-F., Sloot, P.M.A. (eds.) GPC 2010. LNCS, vol. 6104, pp. 395–406. Springer, Heidelberg (2010)
16. Smith, R.G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers 29(12), 1104–1113 (1980)
17. Stantchev, V., Schröpfer, C.: Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing. In: Abdennadher, N., Petcu, D. (eds.) GPC 2009. LNCS, vol. 5529, pp. 25–35. Springer, Heidelberg (2009)
18. Wu, L., Buyya, R.: Service Level Agreement (SLA) in Utility Computing Systems. Tech. Rep. CLOUDS-TR-2010-5, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia (September 2010)
19. Zulkernine, F., Martin, P., Craddock, C., Wilson, K.: A Policy-Based Middleware for Web Services SLA Negotiation. In: Proceedings of the 7th International Conference on Web Services, pp. 1043–1050 (2009)

A Reverse Auction Market for Cloud Resources

Joris Roovers, Kurt Vanmechelen, and Jan Broeckhove

University of Antwerp,
Department of Computer Science and Mathematics,
Middelheimlaan 1, 2020 Antwerp, Belgium
kurt.vanmechelen@ua.ac.be

Abstract. The proliferation of the Infrastructure-as-a-Service (IaaS) paradigm has introduced possibilities for trading computational resources on a scale that moves beyond the individual provider level. At present however, the adoption of open markets for trading IaaS resources has been largely unexplored. This paper investigates the design of such an open market. Our focus thereby lies on flexibility and the ability to model and integrate currently deployed pricing schemes of real-world providers instead of imposing new schemes. We discuss the issues encountered by the Continuous Double Auction (CDA) in this regard and introduce a Continuous Reverse Auction (CRA) that is paired with a novel bidding language based on tag and constraint sets.

Keywords: Markets, Bidding Language, Reverse Auction, Double Auction, IaaS, Cloud Computing.

1 Introduction

Infrastructure-as-a-Service (IaaS) providers materialize the cloud computing model by offering consumers access to (virtualized) hardware resources while charging them based on how these resources are used. As the IaaS market continues to grow and the number of IaaS players continues to increase (as predicted by Gartner [6]), possibilities arise for trading IaaS resources on a scale that moves beyond the individual provider level. Such trading is used in other commodities markets such as electricity markets, to efficiently balance supply and demand on relatively short timescales and foster competition.

An important desirable property of such an envisioned market is that it needs to be *open*, accommodating the policies of current real-world providers. Only when a market has a sufficiently low entry barrier that brings together many different consumers and providers, will the price of computing resources be subject to market discipline rather than being dictated by specific providers. Realizing this criterion is non-trivial, and at present, the possibilities for creating such open markets have been largely unexplored. The complexity of an IaaS good is significantly higher than that of many other commodities as IaaS resources are heterogeneous and multi-dimensional. As standardization of resources and APIs among IaaS providers is currently non-existent, defining the exact properties of

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 32–45, 2012.

the goods that are being traded is non-trivial, and imposing a common standard is difficult. Additionally, many providers use different allocation and price models (all based on the pay-what-you-use model, but implemented differently) to charge their customers. Bringing together these different models into a single IaaS market is not addressed in current related work [3,5,11,7,2,1] on market mechanisms for utility and grid computing systems, and forms the focus of this paper.

In this contribution we investigate whether it is feasible to design such an open market for IaaS resources. We spend particular attention to the Continuous Double Auction (CDA) mechanism that is used in many commodity markets, and find it unsuitable for this purpose in the current IaaS setting due to the complexity and diversity of price models used by providers. We therefore introduce a new market mechanism called the *Continuous Reverse Auction* (CRA) that is combined with a flexible bidding language based on *tag* and *constraint* sets to deal with these shortcomings.

2 Tag and Constraint Sets

In order to introduce flexibility into the market with respect to heterogeneity of IaaS goods and pricing models, we rely on the use of *tag* and *constraint* sets. We extend the notion *resource sets* as defined by Dubé within the context of a CDA-based market for Grid resources [4], and accommodate it to the IaaS setting. Dubé introduces *resource sets* that describe resource properties in bids and offers to a CDA. Consider for example the resource set S in 1 which represents a computer with a x86 system architecture, two-core processor clocked at 2.4 GHz, 4GB of RAM and 16GB of hard disk space.

$$S = \{\mathrm{x86}_{\mathrm{proc\_arch}}, 2_{\mathrm{proc\_cores}}, 2.4_{\mathrm{proc\_clock}}, 4_{\mathrm{mem\_size}}, 16_{\mathrm{disk\_size}}\} \quad (1)$$

Obviously, the used unit as well as the type and domain of each of the items in the set should be clearly defined. In [4], this is done by specifying a discrete, finite domain set for each of the resource types, along with the used unit (if applicable). Examples of such domain sets are shown in 2.

$$\begin{aligned} \Phi_{\mathrm{proc\_cores}} &= \{1, 2, 4, 6, 8, 16, 32\} \\ \Phi_{\mathrm{mem\_size}} &= \{1, 2, 4, 6, 8, 16, 32, 64, 128, 256\}\ (\mathrm{GB}) \\ \Phi_{\mathrm{net\_bw}} &= \{10, 100, 1000, 2000\}\ (\mathrm{MB/s}) \end{aligned} \quad (2)$$

Given a *requirement* set R_b accompanying an ask and a *component* set C_o included in an offer:

$$\mathcal{R}_b \text{ matches } \mathcal{C}_o \iff \mathcal{C}_o \text{ matches } \mathcal{R}_b \iff \mathcal{R}_b \cap \mathcal{C}_o = \mathcal{R}_b \quad (3)$$

A shortcoming of this approach is that it lacks the expressiveness and flexibility that is needed to express some of the more complex constraints that are inherently part of IaaS solutions. For example, a constraint on *refund policy*

will typically include very provider specific conditions under which a refund can occur, as well as what this refund actually includes. As such, it cannot be expressed by a match on a single value. Another example is the implementation of a *region* constraint for which geographical knowledge is needed to determine which regions are included or overlap with others. A final issue is that in Dubé's approach a market is needed that has specific knowledge about each of the resource types in order to match sets, i.e. it needs to know the different matching semantics of each of the resources. In a situation where more and more complex constraints are used with complex matching semantics, using such a system is no longer viable. Indeed, given the heterogeneity of IaaS solutions today, using a matching mechanism in which the market must be aware of all possible resources and constraints is not manageable nor future proof.

To address these issues, we introduce *tag sets* and *constraint sets*, which enable the specification of more complex constraints. In particular, the specification of a constraint on a resource that a consumer is requesting or a provider is offering will be separated from the specification of the values that characterize that resource. Splitting the values from the matching semantics enables the specification of complex constraints that span multiple resources. Our approach to matching also removes the need for the market to have knowledge of each type of item in the sets. When a consumer or provider describes a resource, requirement, service or feature it uses tag sets that describe its different characteristics. Formally, a tag $\mathcal{T}$ in a tag set $\mathcal{TS}$ is defined as the tuple in 4. It contains a name and a set of attributes that define the tag.

$$\mathcal{T} = [\mathcal{N}_{\mathcal{T}}, \mathcal{A}_{\mathcal{T}}] \tag{4}$$

The name $\mathcal{N}_{\mathcal{T}}$ of the tag can be any regular string, but has to be unique for each type of tag and within a tag set. The attributes $\mathcal{A}_{\mathcal{T}}$ associated with the tag (see 5) are a set of key-value pairs.

$$\begin{aligned} \mathcal{A}_{\mathcal{T}} &= \{(x, f(x))\} \\ f &: x \in string \mapsto value \end{aligned} \tag{5}$$

These attributes contain the values that further characterize a given tag. For example, in the case of a `Disk` tag ($\mathcal{N}_{\mathcal{T}}$="Disk"), the tag attributes will contain both the unit in which the hard disk size is specified as well as the size of the disk, i.e. `[Disk, {(unit, GB), (value, 160)}]`. Attribute values do not have to be singular, they can also be of a compound nature. That is, it is also possible that these values are lists or nested tags. For example, if multiple operating systems are supported by a provider, they can be listed as follows.

```
[SupportedOperatingSystems, {(list, {
  [OperatingSystem, {(name, Windows), (version, 7)}],
  [OperatingSystem, {(name, Ubuntu), (version, 11.04)}]}
)}]
```

Tag sets can therefore group multiple attributes under a single tag, allowing for a more hierarchical approach compared to resource sets. A less visible difference to [4] is that tag sets themselves only describe the characteristic of a request or offer. The matching semantics are encoded by *constraint sets* that accompany tag sets. These contain constraints that define a certain matching restriction. Formally, a constraint (6) is a function that defines a matching restriction on a request $\mathcal{R}$, by expressing restrictions on the tags of $\mathcal{R}$ and the tags of an offer $\mathcal{O}$ against which $\mathcal{R}$ is matched.

$$\mathcal{C} : (\mathcal{TS}_{\mathcal{R}}, \mathcal{TS}_{\mathcal{O}}) \mapsto \{true, false\} \tag{6}$$

Given the generic definition in 6, it is clear that the outcome of a constraint function can depend on a large number of factors. As an example, consider a memory constraint as given by Algorithm 1.

Algorithm 1. MemoryConstraint($\mathcal{TS}_1$, $\mathcal{TS}_2$)

```
T1 ← TS1["Memory"];
T2 ← TS2["Memory"];
if T1 ≠ null and T2 ≠ null then
  if (A_T1["unit"] = A_T2["unit"]) and (A_T1["value"] ≤ A_T2["value"]) then
    return true;
  end if
end if
return false;
```

A constraint function always takes two arguments: the tag set ($\mathcal{TS}_1$) that belongs to the request to which the constraint belongs and the tag set ($\mathcal{TS}_2$) associated with the offer against which the constraint is verified. The `Memory` constraint works as follows. First, the occurrence of the `Memory` tag is evaluated in both tag sets. If one of both sets does not contain the tag, the result of the constraint is negative. Otherwise, the "unit" attributes are compared first to verify whether both tags are specified in the same unit[1], and then checks whether "value" specified in $\mathcal{T}_1$ is less than or equal to that of $\mathcal{T}_2$. If this is the case, the constraint is fulfilled. In this example, the `Memory` constraint was used to define matching semantics. Besides the `Memory` constraint, many other constraints obviously exist. Some of these operations should be standardized by the market to avoid confusion among the market participants, while others can remain consumer- or provider-specific. An example where a consumer would specify a custom constraint is the previously mentioned case where a trade-off between resource uptime and refund policy is required.

Note that verifying consumer- or provider-specific constraints requires that the algorithm embodying the constraint is made available to the entity performing the matching. The most obvious way to do this, is by allowing consumers and

[1] The operation could be modified to contain logic to do unit conversions.

providers to run custom code that can verify constraints within the market during the matching process. Despite the fact that the action of running custom programming code on a remote location itself is relatively straightforward using a proper platform and software libraries, doing so introduces a set of technical challenges w.r.t. code verification and protecting the market's integrity.

By giving the market a database of standard constraints for which it knows the matching semantics and leaving the matching of non-standardized constraints to the consumer and provider themselves, the technical difficulties that remote code introduces are effectively avoided. In practice, such a system lets consumers and providers specify the name of the constraint as part of the constraint set when a standard constraint is used and specify the endpoint of a web service that can verify custom constraints in the other cases.

To increase the number of constraints that can be checked locally, we introduce a set of generic constraints such as the `LessThan` relational constraint defined in Algorithm 2. In order to express which tag is checked in a generic constraint, the notation `Constraint[Parameters...]` (as in `Resource[Memory]` will be used in the remainder of this contribution.

Algorithm 2. LessThanConstraint($\mathcal{TS}_1$, $\mathcal{TS}_2$, tagName, attributeName)

$\mathcal{T}_1 \leftarrow \mathcal{TS}_1$[tagName];
$\mathcal{T}_2 \leftarrow \mathcal{TS}_2$[tagName];
if $\mathcal{T}_1 \neq$ null **and** $\mathcal{T}_2 \neq$ null **then**
  if $\mathcal{A}_{\mathcal{T}_1}$[attributeName] $<$ $\mathcal{A}_{\mathcal{T}_2}$[attributeName] **then**
    return true;
  end if
end if
return false;

Algorithm 3. ResourceConstraint($\mathcal{TS}_1$, $\mathcal{TS}_2$, tagName)

$\mathcal{T}_1 \leftarrow \mathcal{TS}_1$[tagName];
$\mathcal{T}_2 \leftarrow \mathcal{TS}_2$[tagName];
if $\mathcal{T}_1 \neq$ null **and** $\mathcal{T}_2 \neq$ null **then**
  if ($\mathcal{A}_{\mathcal{T}_1}$["unit"] $=$ $\mathcal{A}_{\mathcal{T}_2}$["unit"]) **and** ($\mathcal{A}_{\mathcal{T}_1}$["value"] $\leq$ $\mathcal{A}_{\mathcal{T}_2}$["value"]) **then**
    return true;
  end if
end if
return false;

To further indicate the usefulness of such constraints, consider the `Resource` constraint in Algorithm 3 which is a generalization of the `Memory` constraint of Algorithm 1 that can also be used to implement many other hardware related constraints (such as `Disk` or `CpuClock`).

3 Combining the CDA with Tag and Constraint Sets

The aforementioned tag and constraint sets could be used in conjunction with a traditional CDA approach. A bid or ask is then represented by a 5-tuple:

```
[Volume, Price, ExpirationDate, TagSet, ConstraintSet]
```

As an example, consider the case where Amazon submits an ask to a CDA, specifying a volume of 5 small instances at the price of $0.10 per hour :

```
Ask = [5, 0.10, 12/07/11 13:00:00 UTC-7, {
        [Memory, {(unit, MB), (value, 1700)}], [EC2_ECU, {(value, 1)}],
        [Disk, {(unit, GB), (value, 160)}], [Platform, {(value, x86)}],
        [EC2_API, {(value, m1.small)}], [SLA_Uptime, {(value, 99.95)}],
        [SupportedOperatingSystems, {(list, ...)}],
        [SupportedEC2_AMI_IDS, {(list, ...)}],
        [Splittable, {(value, 1)}],
        [AllocationType, {(value, OnDemand)}],
        [Time, {(value, 1), (unit, hour)}],
        [Region, {(value, USA/Virginia)}]
        [Provider, {(name, Amazon AWS), (url, http://aws.amazon.com)}] },
      { ConsumerLocationExclusion["Cuba"], ... }
      ]
```

A consumer submits a bid with a price of $0.12 and some hardware constraints.

```
Bid = [2, 0.12, 05/06/11 02:00:00 UTC+1, {
         [Disk, {(unit, GB), (value, 100)}],
         [Memory, {(unit, MB), (value, 1500)}],
         [AllocationType, {(value, OnDemand)}],
         [Time, {(value, 1), (unit, hour)}],
         [Consumer, {(name, "Jan Wolk"),
         (location, "Brussels, Belgium, EU")}] },
       { Resource["Disk"], Resource["Memory"],
         Equal["AllocationType", "value"], EqualAll["Time"] }
       ]
```

Note the flexibility that our tags introduce with respect to describing the resources and conditions under which the good is traded. The `Splittable` tag for example allows the provider to indicate whether its offered volume can be partially matched or not and constrains the granularity of the split. Further note that a constraint is specified on the ask which constrains the geographical location of the consumer with the matching bid (e.g. to conform to trading agreements).

Although the use of tag and constraint sets increases the flexibility of the CDA, a fundamental problem remains that relates to the way real-world IaaS resources are currently priced. The price that consumers pay for a set of resources is not only determined by their defining characteristics, but also by *how* these resources are used. The total usage cost is not determined by a single price (e.g. cost per VM hour) but depends on a variety of different price components (e.g.

cost per GB inbound bandwidth consumed) that have different weights according to the actual resource usage. Because of this, adopting a CDA that only takes a single price component into account to sort the bid and ask queues is not desirable. If such a CDA would be used, it is destined to be gamed by providers that advertise very low prices for that component while charging high prices for the use of other resource components, thus leading to inefficient allocations.

Introducing a multi-price component model is possible if consumers add usage values to their tag sets, and providers provide component-wise prices in their offers. The market could then determine the transfer price for matching bid b with ask a as in 7.

$$P(a,b) = \sum_{\mathcal{T} \in \ (\mathcal{TS}_a \ \cup \ \mathcal{TS}_b)} v(\mathcal{T}) \ p_A(\mathcal{T}) \tag{7}$$

where the price $P(a,b)$ for a certain bid b that is matched with an ask a of provider A is determined by the weighted sum (according to the usage of the consumer) of the different price components specified by the provider's ask. The weights are given by $v(\mathcal{T})$, which is defined to be the value of the usage-tag that is used by the price-component for the resource specified by tag $\mathcal{T}$. The term $p_A(\mathcal{T})$ is then the unit price specified by provider A for usage of the resource specified by tag $\mathcal{T}$. A provider would need to encode the relation between usage tags and price tags (e.g. in a table) as shown in Table 1.

Table 1. Provider specified table that defines the relations between usage and price tags

Resource Tag	**Usage Tag**	**Usage unit**	**Unit price**
Memory CpuClock CpuCores Disk	TimeUsage	hour	0.10
OS (Windows based)	TimeUsage	hour	0.03
LoadBalancing	TimeUsage	hour	0.10
LoadBalancing	OutgoingBandwidthUsage	GB	0.02
LoadBalancing	IncomingBandwidthUsage	GB	0.02
OutgoingBandwidth	OutgoingBandwidthUsage	GB	0.13
Disk I/O	DiskAccessUsage	million accesses	0.05

By grouping the resources, providers can indicate that the consumer should only be charged once for those resources. Similarly, by adding a resource multiple times with different usage tags, the provider can indicate that certain resources incur costs that are determined by multiple factors (in the case of the load balancing service, both per hour and per in- or outgoing GB).

However, this still requires the transfer price to be a linear combination of the component prices, which is not the case in practice. Consider for example

NewServers [8] that provide 3GB of free outgoing bandwidth per hour, charging $0.10 per additional GB, or Amazon's tiered S3 pricing model.

4 Continuous Reverse Auction for IaaS Resources

The Continuous Reverse Auction (CRA) combines properties of the Reverse Auction and the Continuous Double Auction to trade IaaS resources among multiple buyers and sellers to deal with aforementioned issues. A systematic overview of how the CRA works, is provided in Figure 1.

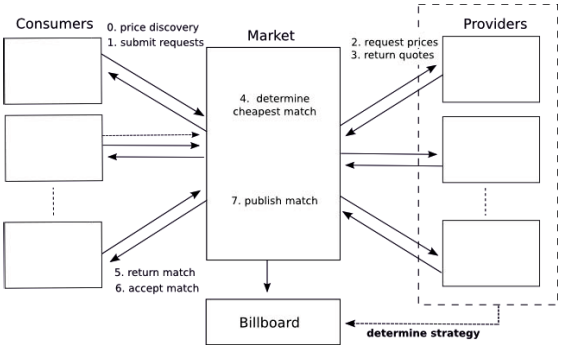


Fig. 1. Schematic overview of the workings of the Continuous Reverse Auction

4.1 Mechanism and Principles

In the Continuous Reverse Auction, consumers direct *requests* to the market, while providers deliver *quotes* in response to these requests. A consumers submits a requests to the market that subsequently returns the cheapest quote by querying a group of previously registered providers. Providers reply by either sending a quote to the market or by indicating that they are unable to match the given request. Upon receiving the quote from the market, the consumer can either accept or decline it.

The bidding process is initiated by a consumer that wants to allocate a set of IaaS resources and submits a request to the market. The bidding language that is used to specify this request, is based on the tag and constraint sets that were introduced in section 2:

```
Request = [Volume, TagSet, ConstraintSet]
```

A consumer can decide to accept or decline an offered quote, it is therefore no longer necessary to specify a price as part of his request. Also, as a request in the CRA is either matched immediately or not at all (if no provider is able to fulfill the consumer specified constraints), there is no need to specify an expiration date as part of the request. The components of a request are a `Volume` indicating

how many units a consumer wishes to allocate, and a `TagSet` and `Constraint` specifying the resource to allocate.

Once the request has arrived at the market, a price request is sent out to all registered providers, to which the providers respond with a quote or with a reply that they cannot match the request.

```
Quote = [Price, Volume, Identifier]
```

Besides the `Price`, the quote also contains a `Volume` that indicates how many units the provider can or is willing to match. The (securely encoded) `Identifier` in the quote allows a consumer to actually allocate resources after a match has been found on the market; it allows the consumer to prove the commitment a provider made when it handed out a certain quote. For further details we refer to [9].

After all registered providers have replied, the market returns the cheapest quote as a match to the consumer. If bids can be split, the market can match different units of a consumer's request with different providers and return multiple matches. Naturally, this matching occurs according to the quote prices specified by the providers (cheaper providers are matched first). Algorithm 4 outlines the matching procedure[2].

Algorithm 4. submitRequest(r: `Request`): Match

```
bestQuotes ← {}
for p in providers do
  bestQuote ← bestQuotes.first();
  currentQuote ← p.getQuote(r);
  if bestQuote = null or currentQuote.getPrice() < bestQuote.getPrice() then
    bestQuotes ← {currentQuote}
  else if bestQuote.getPrice() = currentQuote.getPrice() then
    bestQuotes.add(currentQuote)
  end if
end for
matchingQuote ← selectRandomElement(bestQuotes);
return new Match(matchingQuote);
```

As providers no longer submit asks to the market but are asked to deliver a quote on the basis of an individual request, providers are free to apply their own pricing scheme. Additionally, as providers now determine whether they can match a request with one of their offers or not, the market no longer needs to know how to match constraints, nor does it need to have constraint-specific knowledge. Indeed, when using the CRA mechanism, the market is constraint-agnostic, and acts only as an intermediate. Note that within the CRA, a consumer does not specify a maximum price as part of his request. This frees the

[2] In order to be concise, Algorithm 4 does not incorporate splittable bids.

consumer from the task of performing an accurate price estimation for a request. By allowing the consumer to accept or decline a quote (match) given its price, the consumer can do accurate price discovery and individual rationality is maintained.

4.2 Price Discovery

Price discovery is the process by which buyers and sellers determine or approximate the price of a good in the marketplace. Consumers in particular can make use of price discovery techniques to make an upfront trade-off between the price they will need to pay on the market and the resources and quality of service they will receive for that price. Additionally, price discovery techniques allow consumers and providers to determine bidding strategies by monitoring the price for a specific resource in the market and choosing the time at which they buy or sell certain resources intelligently as to maximize their personal profit.

In the CRA market, price discovery is in fact already built-in for consumers as they have the possibility to request a quote from the market at any given time without any further obligation. As such, both new as existing consumers can always obtain a quote for a request in order to direct their actions. Providers however, have no idea at which price and rate certain resources are being traded except when they are actually matched to a specific request.

In order to solve this lack of price discovery tools for providers, the CRA includes a billboard on which the market publishes anonymized accepted matches. This is done in real-time; after a match is found, it is directly published on the billboard. Because of this real-time behavior, the billboard effectively represents the current market situation. Providers can analyze the anonymized matches and adjust their pricing strategy accordingly. If a provider is able to infer that the price of matches in a specific category of requests is consistently lower than the price it is offering, it can adjust its strategy in order to get more matches. Analogously, it might increase the quote price for a category of requests, in order to increase its profit. As such indirect competition among providers is introduced and competitive prices can be formed. In future work we will focus on the development and experimental analysis of such strategies.

4.3 Sharing Semantics of Tag and Constraint Sets

In order to guarantee correct matching behavior, constraints should be precisely defined so that there is a clear and unique understanding among all market participants of what the constraint exactly specifies. As it is a core characteristic of the CRA that the market itself is unaware of constraint semantics, an additional entity is required to provide these. We therefore introduce a separate (market independent) *constraint catalogue* service that clearly specifies how constraint names map to actual matching semantics. In order to specify constraint semantics, the service can make use of a set of (algebraic) rules, pseudocode or a specific programming language.

As constraints are a core part of the contract between provider and consumer when a match is accepted, it is paramount that catalogue services are officially recognized and accredited by a mutually trusted third party,. This third party (which could potentially be the market itself), can then be trusted to objectively verify whether the contract is met by both parties or not. Such verification entails validating that the consumer has made correct payments, as well as verifying that provided resources actually meet the constraints specified in the consumer's request[3]. Note that the concept of the catalogue does not necessarily need to materialize into a web service; as long as there is a way for providers and consumers to share the semantics of particular constraints. In many cases it suffices that a provider states the meaning of a certain constraint on its website.

In order to lower the entry-barrier for providers, a standard set of common constraints can be provided as a software library. This way, providers will not need to download code from the constraint catalogue or implement the constraints themselves based on the catalogue's definition. If providers internally model their offers using tag and constraint sets, they can even use the software library to determine whether a given request matches one of their offers or not. Alternatively intermediaries such as brokers can perform this function. Additionally, providing such a library will limit the proliferation of constraints that have similar or the same semantics. A consumer specifying a constraint includes the qualified name of the class that implements the desired constraint as part of a `ConstraintDescription` that is attached to its request. A provider then creates an instance of the specified constraint class[4], and calls the `validate` method on the resulting object, passing the consumers tags, the tags used to model the provider's offer and the constraint parameters as specified by the consumer.

Although using constraints with well specified semantics is the main requirement for market participants, it is unlikely that many matches will occur if the tags that consumers and providers use to model resources, requirements, services and features are not standardized. That is, most (if not all) constraints require that there are tags with specific names or attributes in the tag set that accompanies the constraint. For example, when using the `ResourceConstraint`, a tag with a specific name that has `value` and `unit` attributes is required in both the consumer and the provider tag set in order for a match to occur. To address this, catalogues should also define tags (names and associated attributes) besides constraint semantics. As is the case with constraint names, a reverse DNS naming convention can be used to avoid name clashes. The constraint library should include this set of standard tags (and the default catalogue service should consequently also contain them).

Besides standardized constraints, consumers also have the possibility to specify custom constraints that allow them to express specific matching behavior that is not available as part of the standard library or in any of the catalogue services. To do this, the consumer or its broker deploys a web service that is able to check whether

[3] Doing so is certainly non-trivial as it involves thoroughly examining and monitoring the allocated resources.

[4] This can be done generically using *reflection*, e.g. in the Java programming language.

a given constraint is met, given a tag set that was specified by the consumer and a tag set that describes a provider's offer. Figure 2 outlines this approach.

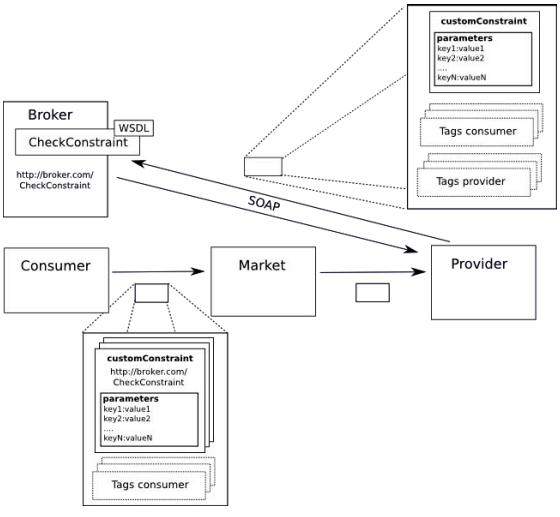


Fig. 2. Implementation of custom constraints using web services

Note that from an efficiency standpoint, it is preferable that a provider or the provider's broker first verifies all standardized constraints in order to decrease the probability that time-consuming web service calls need to be made.

A full implementation of our market based on Web Service technology (Java EE, JAX-WS, Spring, AJAX) is publicly available [10]. The implementation, which has been deployed to Amazon's EC2 platform, includes provider-side models for Amazon, GoGrid, CloudSigma and Rackspace, demonstrating that our bidding language can cope with the large amount of heterogeneity in resource specification and price models of current cloud providers.

4.4 Computational and Communicative Tractability

It is intuitively clear that the algorithmic complexity of the market's operation itself is rather low as the market's only task involves determining which of the quotes for a specific request contains the lowest price. By letting the providers determine whether they match a certain request as well as determine the price for that specific request, the market has effectively distributed the computational complexity that is inherent in matching and pricing heterogeneous IaaS resources. Furthermore, as no bids or asks need to be stored in queues as is the case in the double auction, no state (apart from the billboard) needs to be maintained between subsequent matches. As such, parallelizing and distributing the market's operation over different servers is possible, adding to attractiveness of the CRA mechanism from a computational point of view.

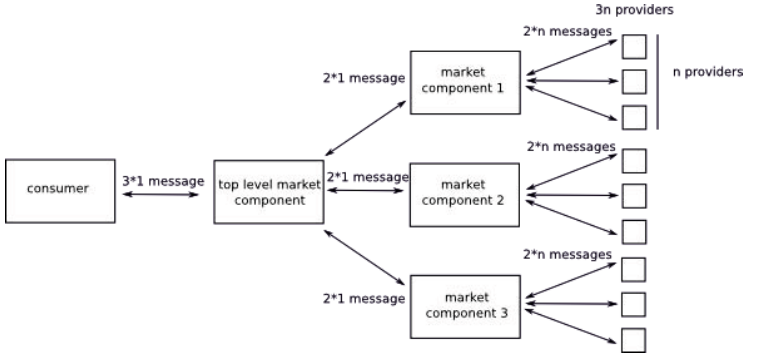


Fig. 3. Hierarchical market setup to reduce the potential for a communication bottleneck in the CRA

A downside of the CRA compared to a CDA is that the amount of communication between all involved market actors will be larger as the market sends out price requests to all providers for each consumer request. If there are n registered providers that can all fulfill a certain consumer request, $2n + 3$ messages will be needed for a match to occur. That is:

- 1 initial request from the consumer to the market
- n price requests from the market to the providers
- n responses from the providers to the market
- 1 match message from the market to the consumer
- 1 match acceptance message from the consumer to the market

As in the CDA no additional messages need to be exchanged with providers for an incoming consumer bid, it is clear that the flexibility the CRA offers comes at a considerable communication cost. In order to keep this cost manageable, providers might use brokers that have very good connectivity to the market. In order to prevent the network connectivity to the market to become a bottleneck, market process can be organized in a hierarchical manner, in which each node determines the cheapest match and subsequently forwards it to its parent component. An overview of this organization is shown in Figure 3.

5 Conclusion

Infrastructure as a Service providers are starting to embrace models that dynamically price their resources. Currently, the application of such models is constrained to individual providers and an open market for IaaS resources is yet to emerge. For such a market to materialize, flexible approaches are required that allow providers within the market to express their pricing schemes and resource models in line with their current modus operandi. The design of a market mechanism and bidding language that is flexible enough to realize this goal is an open research problem. In this contribution we discuss the issues encountered

by CDA-based approaches in this regard and introduce a Continuous Reverse Auction (CRA) that is paired with a novel bidding language based on tag and constraint sets. Our approach can accommodate the heterogeneity present in price and resource models currently used by providers, while allowing consumer bids to be matched with multiple providers. As such, it forms an important step towards a more structured and organized form IaaS resource trading.

References

1. Altmann, J., Courcoubetis, C., Stamoulis, G.D., Dramitinos, M., Rayna, T., Risch, M., Bannink, C.: GridEcon: A Market Place for Computing Resources. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 185–196. Springer, Heidelberg (2008)
2. Bapna, R., Das, S., Garfinkel, R., Stallaert, J.: A market design for grid computing. INFORMS Journal on Computing 20(1), 100–111 (2007)
3. Broberg, J., Venugopal, S., Buyya, R.: Market-oriented grids and utility computing: The state-of-the-art and future directions. Journal of Grid Computing 6(2), 255–276 (2008)
4. Dubé, N.: SuperComputing Futures: the Next Sharing Paradigm for HPC Resources. Ph.D. thesis, Université Laval (2008)
5. Fujiwara, I., Aida, K., Ono, I.: Applying double-sided combinational auctions to resource allocation in cloud computing. In: Proceedings of the 10th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), pp. 7–14 (July 2010)
6. Leong, L.: Adopting cloud infrastructure as a service in the 'real world'. Tech. rep., Gartner (2011)
7. Neumann, D., Stoesser, J., Anandasivam, A., Borissov, N.: SORMA – Building an Open Grid Market for Grid Resource Allocation. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 194–200. Springer, Heidelberg (2007)
8. NewServers: NewServers homepage (2009), `http://www.newservers.com/`
9. Roovers, J.: A Market Design for IaaS Cloud Resources. Master's thesis, University of Antwerp (2011), `http://thesisjorisroovers.blogspot.com`
10. Roovers, J., Vanmechelen, K.: IaaS Reverse Auction Market Prototype Implementation (2011), `http://jorisroovers.com/files/IaaSMarketPrototype.zip`
11. Vanmechelen, K., Depoorter, W., Broeckhove, J.: Combining futures and spot markets: A hybrid market approach to economic grid resource management. Journal of Grid Computing 9(1), 81–94 (2011)

A Cost Model for Hybrid Clouds

Mohammad Mahdi Kashef and Jörn Altmann

Technology Management, Economics, and Policy Program
Department of Industrial Engineering
College of Engineering
Seoul National University
Seoul, South Korea
mmkashef@temep.snu.ac.kr, jorn.altmann@acm.org

Abstract. Cloud computing aims at allowing customers to utilize computational resources and software hosted by service providers. Thus, it shifts the complex and tedious resource and software management tasks typically done by customers to the service providers. Besides promising to eliminate these obstacles of resource management for consumers, Cloud computing also promises to reduce the cost of IT infrastructure. In particular, it promises to reduce the cost of IT through lower capital and operational expenses, stemming from a Cloud's economies of scale and from allowing organizations to purchase just as much computer and storage resources as needed whenever needed. A clear specification of savings however requires a detailed specification of the costs incurred. Although there are some efforts to define cost models for Clouds, the need for a comprehensive cost model, which covers all cost factors, is undeniable. In this paper, we cover this gap by suggesting a cost model for hybrid Clouds (i.e., the combinations of a private data center (private Cloud) and the public Cloud). This model is based on a comprehensive literature research on cost factors and the idea of combining cost of data centers and cost for using Clouds. Finally, we demonstrate the workings of the suggested cost model by applying it to a specific Cloud scenario.

Keywords: Cloud computing, hybrid Clouds, Infrastructure-as-a-Service, Cloud service migration, cost modeling, IT cost factors, Cloud economics, cost model comparison.

1 Introduction

The Cloud computing paradigm has been established as a promising model for using computational resources and software resources on-demand [2, 15, 21]. It shifts the complex and tedious resource and software management tasks typically performed by the customers to the service providers. Therefore, it increases the flexibility to adapt to changes in demand [3] [19]. It also reduces the infrastructure investments and the cost of organizing the IT resources. The capital and operational expenses are lower, stemming from the Cloud's economies of scale. Consequently, Cloud computing has sparked a huge amount of interest in the IT industry. It is not only very attractive to

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 46–60, 2012.

business customers but also to small research groups in the computational science and engineering field [3]. Some statistics state that the market for Cloud computing services was $16billion in 2008 and will rise to $42billion/year by 2012 [17].

Despite these benefits and optimistic outlooks, there are also some researchers opposing those positive claims. A report by McKinsey argued that there would be only a few savings from a migration to the Cloud. They even state that moving to the Cloud would actually cost 144% more than current expenditures [12, 25]. The considerable uncertainty about cost savings and need for more details about actual expenditure were also discussed by Kondo et al. [3]. This still-ongoing discussion highlights the necessity for an overall cost model [11, 13].

Our research objective is to clarify whether and under which conditions Cloud computing can actually save money. In particular, we intend to construct an overall cost model that can be used by Cloud users to decide whether to use the Cloud for certain applications. This overall cost model comprises a comprehensive set of cost factors, necessary for making good cost estimation for running applications in a in-house data center or on a public Cloud.

In particular, we address the following three questions: (1) Which cost factors and what type of cost factors have been considered for Clouds? (2) How is a cost model for hybrid Clouds structured? (3) How can this cost model be applied?

To answer these questions, we conduct the following steps: First, we perform a systematic literature review of papers on cost factors and cost models in Cloud computing. Second, we identify the gaps in the current research on Cloud cost modeling. Based on the result, we perform a cost-benefit analysis to design a comprehensive overall cost model for hybrid Clouds. Finally, we apply this new cost model to demonstrate its workings.

The remainder of the paper is organized as follows. The next chapter gives an overview about cost factors and cost models, which have been proposed already in literature. Chapter 3 introduces our cost model for hybrid Clouds. The application of our cost model in a Cloud computing scenario is shown in chapter 4. The final chapter, chapter 5, concludes the paper.

2 Background

2.1 Cloud Computing

Although there are still many Internet forums and blog discussions on what Cloud computing is and is not, the NIST definition seems to have captured the commonly agreed Cloud computing aspects that are mentioned in most of the academic papers published in this area [19]. The NIST definition states that Cloud computing is *"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* As Cloud computing is still in its infancy, the definition of Cloud computing is likely to be improved later when new ideas, services, and developments in Cloud computing will be explored. Nonetheless, for the purpose of our paper, the definition given by NIST is fully sufficient.

2.2 The Need for Cost Models for Clouds

The majority of participants of a survey (which Rayport and Andrew conducted [18]) indicated that they used already public Clouds or discussed, planned, trialed, or implemented the use of a Cloud infrastructure. Their objective for using Cloud technology has been the potential reduction of data center costs. Even though there is a lot of mentioning about the benefits and even inevitability of migrating to a Cloud [14], the exact costs are still unknown. This lack of a cost model makes any decision on migration uncertain. This has been noticed in the academic and business world manifold [6, 7, 13, 15, 17].

Many statements have also been made about the need for cost models for Clouds [3, 8, 11]. A cost model would allow determining the actual cost of using the Cloud. Using these cost values, one may investigate economic factors like Return-on-Investment (ROI), Net-Present-Value (NPV), Benefit-to-Cost-Ratio (BCR), and Discounted-Payback-Period (DPP). These measures are essential to decide when and under which conditions it is better to use Clouds [6, 7]. In other words, any company using Clouds has to execute a detailed cost-benefit analysis to determine whether running their services on the Cloud is more cost effective than purchasing in-house resources.

However, to perform this cost analysis, intimate knowledge about applications, hardware, and the load levels is required [1]. The difficulty in obtaining this knowledge makes it non-trivial to estimate short-term or long-term costs. As Ali et al. pointed out, it is difficult to know "the actual resources consumed by a system", "the deployment option used by a system, which can affect its costs as resources" and the "probable changes in the Cloud service provider's pricing scheme" [8].

2.3 Existing Research on Cost Factors

In order to find all major papers addressing our research objectives, we searched research databases with a combination of keywords from two groups. The first group comprises the keywords Cloud computing, elastic computing, utility computing, Infrastructure-as-a-Service, IaaS, Platform-as-a-Service, PaaS, Software-as-a-Service, SaaS, Everything-as-a-Service, and XaaS. The second group of keywords includes cost model, business model, cost estimation, cost parameters, and economics. The research databases used are IEEE Explore, ACM Digital Library, ISI Web of Knowledge, SpringerLink, ScienceDirect, and Google scholars.

In total, we have found 10 papers that discuss cost factors and cost models in the context of Clouds.

The first paper evaluates claims about the financial advantages that companies would gain from using a commercial Cloud [1]. The authors have calculated costs in four Cloud scenarios. For that, a comparison of the costs for a company, which purchases resources from Amazon EC2 or from a hardware vendor for its in-house data center, were pointed out.

Helping to decide when and under which situation it is recommended to move to the Cloud, a very good coverage of cost factors has been given by Tak et al. [2]. Moreover, the authors have also classified costs into quantifiable and less quantifiable cost factors. In particular, their classification also includes a grouping into direct or indirect cost.

Even though Kondo et al., the authors of [3], have not proposed any cost model, they conducted an economic comparison. For that, they covered all cost aspects of a Cloud project. In total, they considered 11 cost factors.

Besides identifying 6 cost factors, Armbrust et al. also compared Cloud and in-house cases, using a trade-off formula [4]. The authors did not focus on cost modeling but gave a general understanding of various aspects of Cloud computing. They have pointed to the cost-benefit tradeoff of migrating to the Cloud, and have proposed a simple formula to quantify decision makings [4].

The challenges, which enterprises face when applying hybrid Cloud models, were discussed by Hajjat [5]. Component placement is one of the challenges, addressing the decision of which of the components must be kept local and which components can be migrated. Another challenge is the specification of cost factors with respect to computing components, storage components, and wide-area communication. As the cost savings from migration depend on the placement of compute, storage-intensive, and communication intensive components, graph theory has been used to model the network of components and their data transfer relations.

Truong and Dustbar present a service for estimating and monitoring costs. The service distinguishes three situations: the complete use of on-premise resources, the partial use of Cloud resources, or the complete use of Cloud resources [6]. In particular, they discuss the composability of cost models and the migration of some application components to the Cloud. They have introduced seven cost models. The data used for the comparison of the models came from Cloud providers' pricing specifications [6].

Alford and Morton present an economic analysis to investigate the potential savings from a migration of services to the Cloud. They focus on IT data centers and use a proprietary cost model [7]. The study takes into consideration transition costs and life-cycle operations, as well as migration schedules. Although they did not specify the cost model, they emphasize the existence of the cost model that is owned by the Booz Allen Hamilton Inc. The cost model is used to compare the three scenarios: public Cloud, hybrid Cloud, and private Cloud.

Khajeh-Hosseini introduced the Cloud Adoption Toolkit. The toolkit provides a collection of tools that support decision making with respect to the adoption of Cloud computing in an enterprise. However, the cost model mentioned here has also not been described and presented. Only the cost factors have been described [8].

There is a very detailed analysis of costs in [9]. The authors conducted an analysis of different cost factors that have to be considered by a provider of a computational Grid. This analysis uses realistic values. In addition to this, the authors have shown an estimation of the total costs for resource providers in two real-world examples.

Although the main contribution of [10] is in the area of Software-as-a-Service only, some of the considered cost factors for the software selection process can be used in the overall Cloud environment. The authors have focused on the software license selection support. Their model helps users to select a SaaS Licensing (SaaSL) model or a Perpetual Software Licensing (PSL) model.

Among those 10 papers, which have been covered in this literature review, just three of them have proposed their own sophisticated quantitative model. These are the

works in [4, 5, 6]. Although there are three other papers, which indicated to have their own cost models, they did not provide any details, making it impossible to evaluate or use them.

2.4 Cost Factors

All 20 cost factors, which have been identified within the papers listed in the previous section, are structured and categorized into six groups: electricity, hardware, software, labor, business premises, and service. This classification is shown in Table 1. In the column labeled 'Papers', the paper identifiers are listed. The paper identifiers are the same as the literature references. The meaning of the cost factors is as follows:

Electricity: The power usage for the consumption of in-house electronic devices like servers, gateways, routers, and other network devices is considered in this class [1, 2, 3, 4, 7, 9]. In some papers, the electricity consumed through cooling has been considered separately from other electronic devices [2, 4, 7, 9]. Additionally, for achieving accurate estimates, two kinds of values need to be considered for all devices: the power consumption, when the system is idle, and the power consumption, when the system is heavily used [9].

Hardware: Hardware cost refers to the acquisition of hardware resources. In particular, it distinguishes between the purchasing cost of computing hardware needed in-house [1, 2, 3, 7, 9, 10] and the purchasing cost of network devices (e.g., switches, routers) needed in-house [2, 7, 10]. In order to determine the actual annual costs of the resources, their economic lifetime has to be considered as well [9]. We consider a depreciation time of three years.

Software: The purchasing price of licenses of software, which is used in-house, is considered in this class. There are three types of software, which should be considered separately: basic server software, middleware software, and application software. Basic server software (e.g., operating system software, back-up software) refers to server licensing cost [2, 7, 10]. Middleware software refers to any commercial middleware software that is needed for running applications [7]. Application software refers to customer applications (e.g., Web server, enterprise resource planning software). Inevitably, depending on the application being used, the cost can vary in a very wide range [1, 2, 3, 6, 7, 9, 10].

Labor: This category comprises salaries for technicians, who work on maintaining software [2, 7], hardware [2, 7, 9], or on providing support [2, 3, 7, 9]. This cost factors are impacted by the region or country, in which the data center is located.

Business Premises: This category includes the basic costs, which are essential to establish an in-house data center. These basic costs are the cost for renting or purchasing the data center facility [4], the cost for racks and other non-electronic instruments, which are required for the safety and the reliability of the data center, as well as the cost of cabling the data center [7].

Service: The remaining cost factors, which are categorized as services, are mainly intangible items. Some of the service costs may incur independently to whether the customer uses in-house data center services or Cloud-based services. One cost factor is

the charge for Internet connectivity (i.e., for Internet access for the enterprise) [3, 5]. Another cost factor is the usage cost for servers (CPU hours) [1, 2, 3, 5, 6, 8]. The cost for incoming data transfer (e.g., Cloud data transfer in) is another factor [1, 3, 4, 5, 6, 8, 9]. Similarly, there is the cost for the amount of outgoing data transfer (e.g., Cloud data transfer out) [1, 3, 4, 5, 6, 8, 9]. The Cloud storage factor describes the amount of storage, which is used in the Cloud [1, 3, 4, 5, 6, 8]. Finally, there is the cost for executing input requests (i.e., the cost of a write request to a database) as well as the cost for executing an output requests (i.e., the cost of a read request from a database) [3, 8].

Table 1. Comparison of contributions of papers with respect to cost factors

Cost Type	Cost Factor		Literature									
			[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
		Number of Factors / Number of Papers	7	12	11	6	6	5	11	6	9	4
a) Electricity	a1) Cooling	4		1		1			1		1	
	a2) Electronic devices	6	1	1	1	1			1		1	
b) Hardware	b1) Server	6	1	1	1				1		1	1
	b2) Network device	3		1					1			1
c) Software	c1) Basic server software	3		1					1			1
	c2) Middleware	1									1	
	c3) Application software	7	1	1	1			1	1		1	1
d) Labor	d1) Software maintenance	2		1					1			
	d2) Hardware maintenance	3		1					1		1	
	d3) Other support	4		1	1				1		1	
e) Business Premises	e1) Air conditioner, rack	2		1					1			
	e2) Cabling	2		1					1			
	e3) Facility	1				1						
f) Cloud Service	f1) Internet connectivity	2			1		1					
	f2) Server usage	6	1	1	1		1	1		1		
	f3) Data transfer into Cloud	7	1		1	1	1	1		1	1	
	f4) Data transfer from Cloud	7	1		1	1	1	1		1	1	
	f5) Cloud storage	6	1		1	1	1	1		1		
	f6) Requests to input to Cloud	2			1					1		
	f7) Requests to output from Cloud	2			1					1		

2.5 Shortcomings of Existing Cost Models

Among those ten papers contributing to cost modeling of Cloud computing, just three of them proposed an explicit cost model [4, 5, 6]. However, even these three papers center only on either an analysis of certain scenarios or provide a very generic

economic solution that has not been adapted well to the Cloud application area (e.g., Armbrust et al. has proposed a very broad economic formula [4]). This general formula needs much more efforts to be applicable in practice in the Cloud application area. Although the paper of Hajjat et al. provides details about optimization of application components, no cost formula has been specified [5]. But clearly, any kind of optimization should be based on the total cost. Finally, Truong et al. have suggested seven formulas, each one of them addressing a certain activity in Clouds [6]. But, in case of using hybrid Clouds, those formulas are incomplete. The formulas miss information about data center costs.

For that, none of these three works has given us a comprehensive estimation of costs for hybrid Clouds. Most probably, using those models leads to inaccurate estimates. Consequently, any optimization based on these estimates will be of little help. This uncertainty is an obstacle for applying these cost models in practice in hybrid Clouds.

3 The Proposed Hybrid Cloud Cost Model

3.1 Model Context

As shown in the following figure, the context of our model considers four stakeholders in the Cloud computing market: large enterprises, small and medium-sized enterprises (SMEs), Cloud providers, and Internet data centers (IDCs). Cloud providers offer consumers to buy resources on demand. The IDCs provide solutions for long-term outsourcing of IT infrastructures to consumers. Users in this market can be either SMEs or large enterprises.

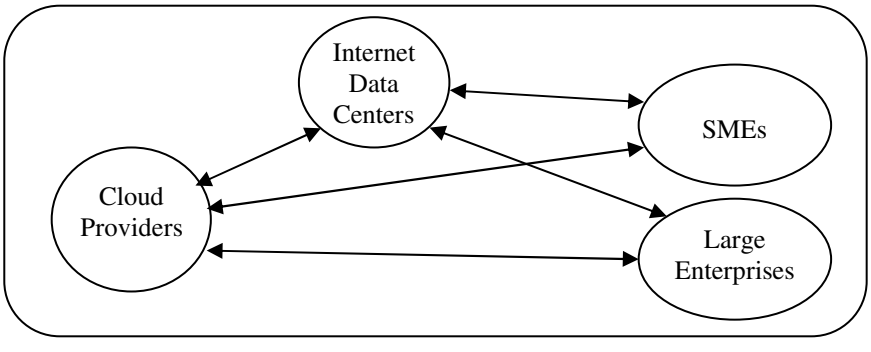


Fig. 1. Stakeholders in the Cloud computing market

The focus of this paper is on large enterprises and their relationship to Cloud providers. The reason for choosing this relationship is that large-sized enterprises have the option of migrating their tasks to the Cloud completely, keeping all tasks in-house (which is equivalent to using an IDC), or run a hybrid solution. This is the most challenging situation with respect to resource allocation decisions. For comparison, SMEs that have a few small IT jobs to complete the most fitting solutions will be to use Cloud providers.

3.2 Conceptual Model

The conceptual model assumes an organization, which comprises the execution of *N* applications and *M* services (across all applications). Applications usually consist of services, which are composed to run a business process. Services are assumed to be basic units, which can perform a set of functions. Users buy those services to construct their applications. Each service can communicate with all other services (if the other services allow) and transfer a specific amount of input data and output data (Figure 2).

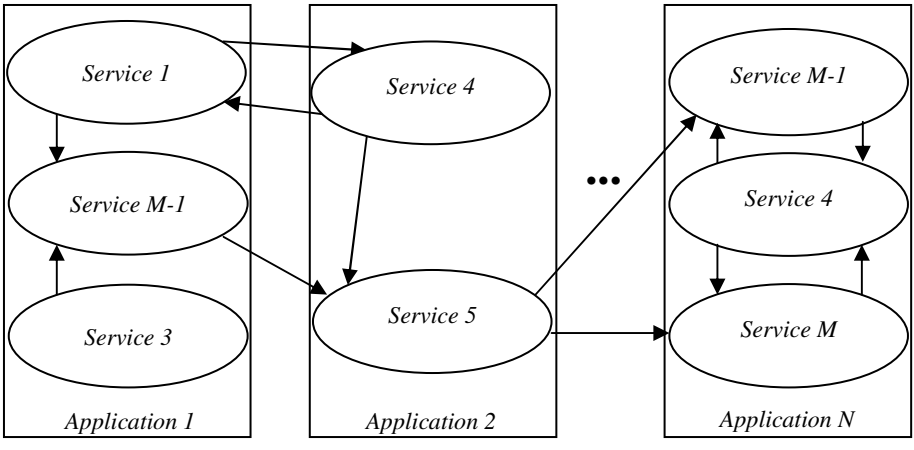


Fig. 2. A schematic view of a set of N applications build on M different services with specific data transfer relations (black arrows) between services

The representation of services and their data transfer relationships using graph theory allows the use of network analysis methods. Obviously, the resulting graph is a directed and weighted graph. Vertices represent services, while edges show data communications. If there is edge from vertex *i* to vertex *j* with a_{ij} as edge label, there is an average monthly data transfer from service *i* to *j*. An example of this representation is shown in Figure 3.

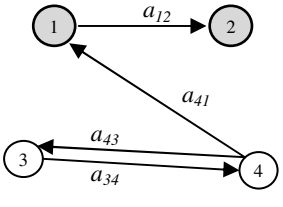


Fig. 3. Graph representation of services and their data transfers

Using this representation of services, the distance matrix *D* can be used to represent the data-transfer related cost factors a_{ij} between service *i* and *j*. The matrix elements a_{ij} of the distance matrix that do not represent an arrow get infinity assigned.

To calculate the total data transfer into the Cloud (i.e., Cloud data transfer inward T^{In}) and the total data transfer out of the Cloud (i.e., Cloud data transfer outward T^{Out}), we propose two formulas:

Cloud data transfer in: $$T_m^{In} = \sum_{i \in I, j \in C} a_{ij} \quad , a_{ij} \in D \tag{1}$$

Cloud data transfer out: $$T_m^{Out} = \sum_{i \in I, j \in C} a_{ji} \quad , a_{ji} \in D \tag{2}$$

where *i* and *j* denote services. The set *I* represents the set of services that run in-house and the set *C* represents the set of services that run in the Cloud.

3.3 Cost Formula

Based on the cost factors listed in section 2.4, we propose a comprehensive formula that covers all aspects of costs. The costs considered are the purchasing cost for all servers needed ($\sum_i^S C_i^{Se}$; b1 of Table 1), the purchasing cost for all network devices needed ($\sum_i^N C_i^{Ne}$; b2 of Table 1), the costs for all basic server software licenses ($\sum_i^B C_i^{BSS}$; c1 of Table 1), the costs for middleware licenses ($\sum_i^M C_i^{MS}$; c2 of Table 1), the costs for application software licenses ($\sum_i^A C_i^{AS}$; c3 of Table 1), the cost of facility space ($C_{Fa}F$; e3 of Table 1) that the company needs for its data center, the cost of the none-electronic equipment (C_{Nee}; e1 of Table 1) that is needed for a data center, and the cost for cabling (C_{Ca}; e2 of Table 1). These costs are fixed costs. The sum of these fixed costs *FC* is shown in equation 3:

$$FC(d) = \sum_i^S C_i^{Se} + \sum_i^N C_i^{Ne} + \sum_i^B C_i^{BSS} + \sum_i^M C_i^{MS} + \sum_i^A C_i^{AS} + C_{Fa}F + C_{Nee} + C_{Ca} \tag{3}$$

The parameters used in equation 3 are as follows: *d* denotes the depreciation period in units of months. S is the number of server devices, which are needed to run all services. C_i^{Se} is the cost of server i. N is the number of network devices that are needed. C_i^{Ne} is the cost of network device i. B denotes the number of basic server software. C_{Bss}^k is the cost of basic server software k. M denotes the number of middleware software which is needed. C_i^{MS} is the cost of middleware software i. A is the number of application software, which is used. C_i^{AS} is the cost of application software i. F denotes the size of the facility in square meters. C_{Fa} is the cost of facility space in square meters. Note, facility cost can be calculated based on rent contract conditions.

The following variable cost factors are calculated over time. They are summed up for each time period (e.g., monthly or annually). They comprise the cost of electricity usage from cooling and from other electric devices ($C_m^{Elec}(E_m^C + E_m^E)$; a1 and a2 of Table 1), the cost for Internet connectivity of the data center ($C_m^{Int} I_m$; f1 of Table 1), the cost of labor for maintaining software ($C_m^{LS} L_m^S$; d1 of Table 1), for hardware ($C_m^{LH} L_m^H$; d2 of Table 1), for other work ($C_m^{LO} L_m^O$; d3 of Table 1), the cost for data transfer that goes into the Cloud ($C_m^{TIn} T_m^{In}$; f3 of Table 1) and out of the Cloud ($C_m^{TOut} T_m^{Out}$; f4 of Table 1), the cost for Cloud storage ($C_m^{Sto} H_m$; f5 of Table 1), the Cloud server usage cost ($\sum_i^{SerType} C_{m,i}^{Se} S_m^i$; f2 of Table 1), and finally the cost of requests for input to the Cloud ($\sum_i^{RTypes} C_{m,i}^{InR} R_{m,i}^{In}$; f6 of Table 1) and the cost of requests for output from the

Cloud ($\sum_i^{RTypes} C_{m,i}^{OutR} R_{m,i}^{Out}$; f7 of Table 1). Consequently, the total variable cost *VC* is calculated as the sum of all these costs as shown in the following equation:

$$VC(m) = C_m^{Elec}(E_m^C + E_m^E) + C_m^{Int} I_m + C_m^{LS} L_m^S + C_m^{LH} L_m^H + C_m^{LO} L_m^O + C_m^{TIn} T_m^{In} + C_m^{TOut} T_m^{Out} + C_m^{Sto} H_m + \sum_i^{SerType} C_{m,i}^{Ser} S_m^i + \sum_i^{RTypes} C_{m,i}^{InR} R_{m,i}^{In} + \sum_i^{RTypes} C_{m,i}^{OutR} R_{m,i}^{Out}. \quad (4)$$

The parameters used in equation 4 are as follows: C_m^{Elec} is the cost of electricity per unit in month *m*. E_m^C specifies the amount of electricity used for cooling in month *m*. Similarly, E_m^E specifies the amount of electricity used for all other devices in month *m*. C_m^{Int} is the cost for Internet usage per unit. I_m is the usage of the Internet in month *m*. (Note: the Internet cost calculation should be adjusted to the Internet contract, i.e., pricing scheme, used). C_m^{LS} is the cost of labor for maintaining software per unit and L_m^S is amount of labor in month *m*. C_m^{LH} is the cost of labor for maintaining the hardware per unit and L_m^H is the amount of labor for maintaining hardware in month *m*. C_m^{LO} is the cost of labor for other tasks per unit and L_m^O is the amount of this labor in month *m*. C_m^{TIn} is the cost of in-bound Cloud data transfer per unit in month *m*. T_m^{TIn} is the amount of data transferred into the Cloud in month *m*. C_m^{TOut} is the cost for out-bound data transfer per unit in month *m*. T_m^{TOut} is the amount of data transferred out of the Cloud in month *m*. C_m^{Sto} is the cost of Cloud storage per unit in month *m*. H_m is the usage of storage in month *m*. (Note: in some cases, different types of storage costs need to be considered). $C_{m,i}^{Ser}$ is the Cloud server usage cost for server type *i* per unit in month *m*. S_m^i is the amount of server usage of type *i* in month *m*. $C_{m,i}^{InR}$ is the cost per input requests to the Cloud for input request type *i* in month *m*. $R_{m,i}^{In}$ denotes the number of input requests to the Cloud for input request type *i* in month *m*. The meaning of $C_{m,i}^{OutR}$ and $R_{m,i}^{Out}$ is similar. $C_{m,i}^{OutR}$ is the cost per output requests to the Cloud for output request type *i* in month *m*. $R_{m,i}^{Out}$ denotes the number of output requests to the Cloud for output request type *i* in month *m*.

Therefore, the total cost of running a hybrid Cloud over *d* months (i.e., a depreciation period of *d* months) is the sum of the fixed cost and the variable cost over the period of d months:

$$TC = FC(d) + \sum_m^d VC(m). \quad (5)$$

3.4 Discussion and Open Issues

The first eight terms of the formula 3 (i.e., the fixed cost formula) can easily be populated, since it requires only capital expenditure (CAPEX) information. This information can be obtained from vendors or IT consultancy companies that design data centers for enterprises.

In making the decision about whether to move an existing service to the Cloud, one must additionally estimate the expected average and peak resource utilization. This is very difficult, since the applications deployed may have highly variable spikes in resource demand, the practical limits on real-world utilization of purchased equipment are not widely known, and the operational costs differ widely, depending on the type of Cloud environment considered [8]. In addition to this, finding the estimates for

variable cost factors (e.g., cost factors on data transfer, Cloud storage usage, and server usage) requires further effort. In particular, a good estimate of the distance matrix for the in-bound and out-bound data transfer is difficult to get. For that, econometric methods need to be applied on past trends of data transfers between services.

Besides, the importance of each cost factor should be considered when estimating the overall cost. To be more precise, accurately focusing on major cost factors is suggested, since any error in their estimates has a large impact on the accuracy of the overall cost estimation. For instance, in many cases, 31% of cost of data centers comes from labor, 30% from servers, and 25% from cooling [20]. A low-quality estimate on those cost factors has a large impact on the result.

4 Analysis Example

For demonstrating the workings of the cost model, we assume a small scenario. The scenario comprises a large enterprise with a few software applications, which require 10 different services. Some services run on an in-house data center, which is owned by the enterprise, and some services run in the Cloud. All 10 services require the same amount of computational resources. As the enterprise has its own data center, the information about the fixed cost factors (equation 3) can easily be obtained from the accounting department of the enterprise. The fixed cost information includes the purchasing price of servers and network devices, the license costs for basic server software, middleware software, and application software, as well as the business premises costs (Table 2).

Table 2. Cost factor values for running an in-house data center, based on market research [16]

Cost Category	Cost Factor	Usage	Cost (€/year) Assuming a Three Year Depreciation Period
Electricity	Cooling		6000
	Elect. devices		
Hardware	Server		6000 (for 6 servers)
	Network device		15000
Software	Basic server software		10000
	Middleware software		5000
	Application software		40000
Labor	Software maintenance		90000
	Hardware maintenance		
	Other support		
Business Premises	Air conditioner, Rack		25000
	Cabling		5000
	Facility		70000 (in case of purchasing).
Cloud Service	Internet usage		1920
	Server usage	12*2.5 KH	21000 (0.07 €/CPU-hr[1])
	Storage usage	12*200 GB	192 (0.08 €/GB-month[2])
	Requests to input	12*18 M I/O	17,28 (0.08 €/million I/O requests)
	Requests to output	12*120 M I/O	115,2 (0.08 €/million I/O requests)

[1] Based on Amazon EC2.
[2] Based on Amazon S3.

The variable cost, which comprises the usage-based cost for electricity, Internet, and storage, the labor costs in terms of maintaining the software, hardware, and other tasks, need to be estimated using the past consumption and the price development in the market. For our scenario, we assume the following cost factor values, which are based on market research for data center (Table 2).

The cost for the in-house data center covers sufficient computational resources to run 6 services, of which are five security critical services (i.e., they have to run on the in-house data center). Services numbered 1-4 run currently in the Cloud (e.g., Amazon EC2). Since all services require the same computational resources, the cost of running in the Cloud would incur the same cost for all 10 services (i.e., the cost for Cloud server usage would be the same). The only missing values are the data transfer into the Cloud and out of the Cloud.

For calculating the data transfer cost into and out of the Cloud, we analyze the data transfer. Figure 4 shows the ten services and their data transfers. The average monthly data traffic between these services has been illustrated as weights of the edges (in units of 100 GB). Services labeled 5 to 10 run on the in-house data center. Services 6 to 10 are the security critical services.

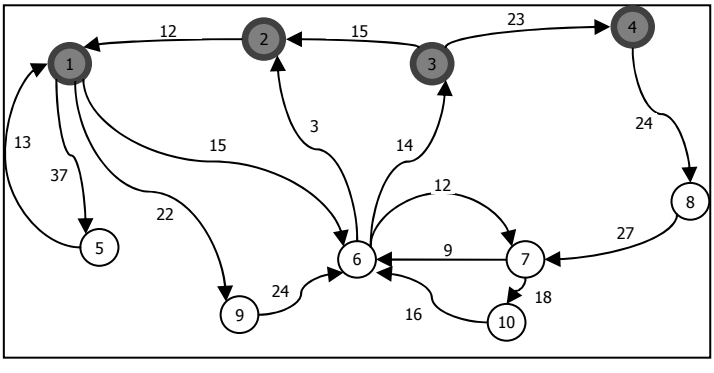


Fig. 4. Scenario services that run in the Cloud (dark vertices) and on the local data center (white vertices) and the data transfer between these services

Based on the data shown in Figure 4, the distance matrix *D* can be constructed, as described in section 3.2:

$$D = \begin{bmatrix} 0 & \infty & \infty & \infty & 37 & 15 & \infty & \infty & 22 & \infty \\ 12 & 0 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & 15 & 0 & 23 & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty & 24 & \infty \\ 13 & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & 14 & \infty & \infty & 0 & 12 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 & \infty & \infty & 18 \\ \infty & \infty & \infty & \infty & \infty & \infty & 27 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 24 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 16 & \infty & \infty & \infty & 0 \end{bmatrix} \quad (6)$$

Applying equation 1 and equation 2 on the distance matrix *D* (equation 6), the in-bound and out-bound traffic for the current situation (i.e., service 1 to 4 run in the Cloud) can be calculated. In order to figure out which of the 5 services (service 1 to 5) should run in-house, we calculate equation 1 and equation 2 also for the other 4 cases

(e.g., services 1, 2, 3, 5 run in the Cloud and service 4 in-house). The results for these five cases are shown in Table 3.

Table 3. Calculated monthly in-bound and out-bound data traffic (in GB) for the five different cases of the scenario

		Services that Run in the Cloud				
		Case 1	Case 2	Case 3	Case 4	Case 5
		1, 2, 3, 4	1, 2, 3, 5	1, 2, 4, 5	1, 3, 4, 5	2, 3, 4, 5
Data Traffic	T_m^{In}	3000	1700	4100	2600	5400
	T_m^{Out}	9800	3700	6100	7600	4900

The data transfer cost is set to 0.14 €/GB. By using these values, we can apply the fixed cost function (equation 3) and the variable cost function. The result shows that the fixed cost equals to €388,000. The monthly variable cost for each of the five cases is depicted in Table 4.

Table 4. Total variable cost per month (€) for each case

	Services that run in the Cloud				
	Case 1	Case 2	Case 3	Case 4	Case 5
	1, 2, 3, 4	1, 2, 3, 5	1, 2, 4, 5	1, 3, 4, 5	2, 3, 4, 5
Total Variable Cost	21012	13612	18412	18412	18512

Table 4 shows that case 1 (i.e., service 1 to 4 run in the Cloud and service 5 runs in in-house data center) incurs the highest cost. Therefore, the current allocation needs and can be improved. Case 2 (i.e., services 1, 2, 3, 5 run in the Cloud and service 4 runs in the in-house data center) incurs the lowest variable cost. Therefore, case 2 is the best option to lower the cost. This result is also depicted in Figure 5.

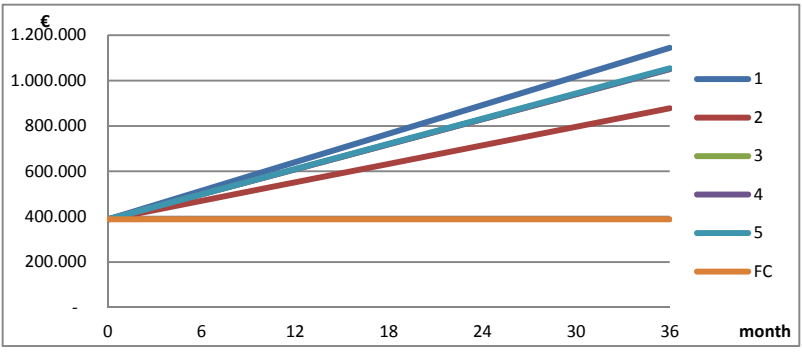


Fig. 5. Comparison between the overall costs for five scenarios and the fixed cost (FC)

Figure 5 also shows that there is a large cost difference between the current alloation (i.e., case 1) and the low cost option (i.e., case 2). The savings over a period

of 3 years (i.e., the depriciation period for fixed costs) are significant. Case 1 incurs cost of 1,144,429 Euro while case 2 incurs cost of 878,029 Euro only.

5 Conclusion and Future Work

In this paper, we have reviewed cost factors of Cloud computing, which have been considered in the research community, and have represented them in a categorized form. Our contribution is the presentation of these cost factors in a comprehensive cost model for hybrid Clouds (i.e., some of the enterprise services run on an in-house data center and some run in the Cloud). This cost model is useful for enterprises that own a data center.

For the calculation of the data transfer cost factor within the cost model, we suggest to use a graph representation for the data transfer between services. This allows performing service placement optimization, considering data transfers in and out of the Cloud. In order to show the workings of the cost model, we apply it to a specific scenario. The analysis example focuses on the application of the cost model and demonstrates the service placement optimization.

The main objective of our present work has been to cover all costs for hybrid Clouds. However, to be more useful in practice, enterprises need decision making support for running hybrid Clouds. Therefore, a future step of our research work is to propose a resource allocation model that considers the cost model presented in this paper. The resource allocation model will help finding the optimum set of services to run in the Cloud. The resource allocation model can also be extended to consider hybrid Clouds with more than one Cloud provider.

Acknowledgement. This work has been funded by the Korea Institute for Advancement of Technology (KIAT) within the ITEA 2 project 10014 EASI-CLOUDS.

References

1. Risch, M., Altmann, J.: Cost Analysis of Current Grids and Its Implications for Future Grid Markets. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 13–27. Springer, Heidelberg (2008)
2. Tak, B.C., Urgaonkar, B., Sivasubramaniam, A.: To Move or Not to Move: The Economics of Cloud Computing. In: Third USENIX Workshop on Hot Topics in Cloud Computing (HOTCLOUD 2011), Portland, Oregon (2011)
3. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.P.: Cost-Benefit Analysis of Cloud Computing versus Desktop Grids. In: IEEE International Symposium on Parallel & Distributed Processing, Rome, pp. 1–12 (2009)
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, U.C. Berkeley (2009)
5. Hajjat, M., Sun, X., Sung, Y.W.E., Maltz, D., Rao, S., Sripanidkulchai, K., Tawarmalani, M.: Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In: ACM SIGCOMM 2010, New Delhi, India (2010)

6. Truong, H.L., Dustdar, S.: Composable Cost Estimation and Monitoring for Computational Applications in Cloud Computing Environments. In: Intl. Conference on Computational Science (ICCS 2010), pp. 2175–2184. Elsevier, Amsterdam (2010)
7. Alford, T., Morton, G.: The Economics of Cloud Computing, Addressing the Benefits of Infrastructure in the Cloud. Booz Allen Hamilton (2009)
8. Khajeh-Hosseini, A., Greenwood, D., Smith, J.W., Sommerville, I.: The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise. In: Software: Practice and Experience (2011)
9. Opitz, A., König, H., Szamlewskax, S.: What does Grid Computing Cost? Journal of Grid Computing 6(4), 385–397 (2008)
10. Altmann, J., Rohitratana, J.: Software Resource Management Considering the Interrelation between Explicit Cost, Energy Consumption, and Implicit Cost: A Decision Support Model for IT Managers. In: Multikonferenz Wirtschaftsinformatik (MKWI 2010), IT Resource Management, Göttingen, Germany (2010)
11. Khajeh-Hosseini, A., Greenwood, D., Sommerville, I.: Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS. In: IEEE Third International Conference on Cloud Computing, Miami, U.S.A, pp. 450–457 (2010)
12. West, D.M.: Saving Money Through Cloud Computing, Governance Studies at Brookings (2010)
13. Khajeh-Hosseini, A., Sommerville, I., Sriram, I.: Research Challenges for Enterprise Cloud Computing. Technical Report, arXiv:1001.3257v1 (2010)
14. Weinman, J.: Mathematical Proof of the Inevitability of Cloud Computing (2009), http://www.joeweinman.com/Resources/Joe_Weinman_Inevitability_Of_Cloud.pdf (accessed on December 2011)
15. Klems, M., Nimis, J., Tai, S.: Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. In: Weinhardt, C., Luckner, S., Stößer, J. (eds.) WEB 2008. LNBIP, vol. 22, pp. 110–123. Springer, Heidelberg (2009)
16. Koomey, J., Brill, K., Turner, P., Stanley, J., Taylor, B.: A Simple Model for Determining True Total Cost of Ownership for Data Centers. The Uptime Institute, Santa Fe (2007)
17. Whiteside, R.: Taylor Woodrow Migrates 1,800 Users to Google Apps (2008), http://googleenterprise.blogspot.com/2008/07/taylor-woodrow-migrates-1800-users-to.html (accessed December 2011)
18. Rayport, J., Andrew, H.: Envisioning the Cloud: The Next Computing Paradigm. Marketspace Point of View (2009)
19. Mell, P., Grance, T.: Draft NIST Working Definition of Cloud Computing (2009), http://www.scribd.com/doc/19002506/Draft-NIST-Working-Definition-of-Cloud-Computing-v15 (accessed December 2011)
20. Gleeson, E.: Computing Industry Set for a Shocking Change (2009), http://www.moneyweek.com/investment-advice/computing-industry-set-for-a-shocking-change-43226.aspx (accessed December 2011)
21. Bany Mohammed, A., Altmann, J., Hwang, J.: Cloud Computing Value Chains: Understanding Business and Value Creation in the Cloud. In: Economic Models and Algorithms for Distributed Systems. Autonomic Systems book series. Birkhäuser, Springer, Heidelberg (2009)

How to Do Successful Chargeback for Cloud Services

Hristo Stefanov[1,2], Slinger Jansen[1], Ronald Batenburg[1],
Eugene van Heusden[2], and Ravi Khadka[1]

[1] Utrecht University, Utrecht, The Netherlands
{h.stefanov,slinger.jansen,r.s.batenburg,r.khadka}@uu.nl
[2] IBM, Amsterdam, The Netherlands
{hristo.stefanov,heusden}@nl.ibm.com

Abstract. With pay-per-use pricing models, elastic scaling of resources, and the usage of shared virtualized infrastructure, 'the Cloud' offers more efficient use of capital, great cost reductions, and breakthrough agility. Yet, it turns out that to leverage the cloud advantages, organizations have to introduce cloud-specific chargeback practices. That is, they have to allocate IT service costs to business users in a way that reflects service consumption. To help organizations transition to a cloud environment, this work provides an overview of the factors that impact the design of successful cloud-specific chargeback models. The findings can assist organizations in the design of chargeback models that allow the business flexibility and cost reductions associated with the Cloud to be fully leveraged. The results are based on an empirical study involving twenty-five field experts from IBM and its client and partner network.

Keywords: IT chargeback, IT cost allocation, billing, IT chargeback models, economics of cloud computing, factors impacting chargeback success.

1 Introduction

In Information Technology (IT) management, chargeback or charging, refers to the practice of charging the costs of IT back to the different departments and business units that use IT [3, 16, 31]. Chargeback makes service consumers aware of the costs of IT and it is generally used to control escalating IT costs, to improve decision making, to align behavior with organizational goals, and to lead to a more effective use of IT [13]. However, in comparison to charging back for physical products, chargeback for intangible products such as IT is still poorly understood by many organizations [16], and is rarely applied to their advantage because of the lack of successful chargeback models that are well aligned with organizational objectives and are clear and acceptable to all the involved stakeholders [26].

To make the situation more complicated, organizations are adopting cloud computing (CC), an environment with untraditional and non-fitting characteristics from a chargeback perspective. With pay-per-use pricing models, elastic scaling of resources, and the usage of shared virtualized infrastructure, the Cloud fundamentally changes the economics of IT [18]. It enables more efficient use of capital, cost

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 61–75, 2012.

reductions, and business flexibility. However, to fully leverage those benefits organizations also need to employ some form of pay-per-use based chargeback, something that is seldom done in the current chargeback practices [29]. Failure to allocate cloud-based costs in a per-use manner can lead to an explosion of unnecessary consumption that can offset the cost reductions and to an inability to leverage the business advantage offered by flexible pricing [29]. Yet, the current chargeback models are oriented towards more traditional IT environments in which costs do no vary with usage. This raises the question how to develop chargeback models that are suitable for cloud services.

The design of better chargeback models for cloud services requires understanding of the factors that make a chargeback model successful in a CC environment. This need was expressed in six interviews by three service management specialists from IBM who were increasingly faced with the problem of helping their clients to adjust their chargeback practices for the dynamics of the Cloud. The scientific literature offers hardly any answers. This leads to the formulation of the research question:

What factors that can be influenced through the design of a chargeback model impact the success of a chargeback model for cloud services?

Answering the above research question adds both scientific and societal value. From an academic perspective, to the best of own knowledge, this is the first work to offer a systematic list of factors influencing the success of chargeback for cloud services. As we validate the list with a diverse group of field experts, the findings can help organizations realize the implications of the Cloud for their chargeback models, and how to improve the latter.

The research question is answered by reviewing the relevant literature on the topic of chargeback and Cloud. In addition, twenty-five semi-structured interviews are conducted with field specialists from IBM and its client and partner network.

The rest of this paper is structured as follows. Section 2 provides an overview of the relevant literature. This is followed by descriptions of the research method in Section 3, and of the discovered and validated factors in Section 4. The discussion and conclusion are in Section 5 and Section 6, respectively.

2 Theoretical Background

According to the notion used behind the term chargeback in the scientific and management literature [15, 16, 21, 23, 28, 31], chargeback for cloud services can be defined as the process of allocating the costs of the cloud services that an organization provides to its employees to the organizational units that use those services. In contrast, a chargeback model is just a conceptual representation of how the costs are allocated to the organizational units using those services. Thus, the chargeback process could be viewed as an implementation of the model.

Regardless of the terminology, chargeback is mainly recognized as a means of IT governance [8, 16, 28] that enables IT cost reductions. On the consumption side, it makes users cost-aware and results in a more cost-efficient choice of services [12,

16]. On the delivery side, increased understanding of service costs facilitates more effective IT investment and provisioning decisions [11, 13].

2.1 Cloud Computing

Vaquero et al. [32] provide a comprehensive definition of CC that is adopted here. According to them cloud services are provisioned from clouds and "clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the [provider] by means of customized [Service Level Agreements] (SLAs)." Furthermore, it should be added that cloud services, are delivered over a computer network or the Internet [1].

Especially relevant for the topic at hand are the characteristics that differentiate CC from a traditional IT environment. Thus, Table 1 describes a list of five fundamental characteristics of CC that describe it well and differentiate it from traditional IT [33].

Table 1. Five characteristics of the Cloud that set it apart from traditional IT (based on [33])

Cloud characteristic	Description
Flexible pricing / Pay-per-use	Computing capabilities are exploited in pay-per-use models. Users have to pay for the actual consumption of services and/or resources.
Elastic scaling	Resource availability can scale up or down by large factors as demand increases/decreases
Rapid provisioning	Computing capabilities and services are immediately provisioned without physical delivery and transferring ownership or resources.
Standardized offerings (through self-service)	Cloud services are highly standardized and offer limited customization. They are usually available through self-service interfaces and the request and provisioning processes are automated.
Virtualization	Computing resources are virtualized to provide independence of the underlying infrastructure and optimal utilization of resources.

Two high level business advantages of the Cloud are also relevant for understanding the discussion in the following sub-section on the impact of CC on chargeback. Firstly, the Cloud can offer substantial cost reductions that are enabled by high utilization of resources due to virtualization, efficient use of capital due to pay-per-use, automation, standardization, and self-service [18, 22]. Secondly, the Cloud also offers great business flexibility, because of the flexible pricing, rapid provisioning, and elastic scaling [1, 22].

2.2 Impact of Cloud Computing on Chargeback

Recent research on the impact of CC on chargeback shows that organizations that want to fully leverage the benefits of the Cloud should apply pay-per-use chargeback practices [29]. No per-use charging for cloud services leads to an explosion in

consumption and to a subsequent increase in costs that can offset the cost reduction benefits of the Cloud. Furthermore, for internal users, no pay-per-use chargeback for cloud services diminishes the business value of CC. From a business perspective pay-per-use is an advantage because costs follow value generation and demand, subsequently allowing efficient use of capital and flexibility in consumption behavior.

To sum up, the need to charge for Cloud is compelling. However, there are numerous challenges to that [29]. For example, how to match between individual consumers and the costs of the shared virtualized infrastructure behind the Cloud? What pricing models (e.g. subscriptions or some forms of metered service consumption) are most suitable for the different types of cloud services? How to charge the costs of the overprovisioned resources that are necessary to enable elastic scaling? To provide practitioners with a helpful tool with which to address those problems the following section lists some factors that influence chargeback success.

2.3 Factors That Influence Chargeback Success

A review of the literature reveals five factors that are explicitly mentioned by some authors. Furthermore, a few more factors can be inferred through detailed analysis.

Explicitly Mentioned Factors. The five concepts that are explicitly recognized in the scientific literature to influence chargeback success are accuracy, cost of costing, transparency and understandability, controllability, and fairness.

Accuracy. Accuracy is a factor influencing chargeback success that is universally recognized by scholars [3, 7, 11, 16, 20, 28]. This is a property of a chargeback model that describes to what extent the charges allocated to an organizational unit for each service accurately approximate the actual costs incurred by the organization for delivering the service to the unit. The better the costs are approximated with the charges, the higher the accuracy of the chargeback model. High accuracy has a profound two-fold effect on the success of chargeback. Firstly, it is conducive to realizing cost reductions, because it bases both provisioning and consumption decisions on actual costs [7, 11]. Secondly, it makes chargeback more acceptable to the involved stakeholders, because it motivates the correctness of the charges and prevents attempts for overthrowing the model based on low accuracy arguments [28].

Cost of costing. While high accuracy has positive impact on the successfulness of a chargeback model, it might be expensive to achieve. The term *cost of costing* is used to quantify the costs of the application of chargeback and accounting models and to compare them against the potential benefits [2, 9, 19]. The employment of chargeback models is associated with significant design, implementation, labor, and IT systems costs. Therefore, chargeback designers should take the costing factor into account in order to develop models, the benefits of which outweigh the related expenses.

The direct impact of the cost of costing on success is negative, because those costs offset the cost savings realized through chargeback. However, by investing in improving accuracy, for example, the success of the chargeback model could be

indirectly enhanced. Therefore the design of a chargeback model should be optimized to balance between costs and accuracy, as well as other factors that positively affect success at the expense of higher costs [9, 19].

Transparency and understandability. Researchers who investigate organizational behavior in relation to chargeback note that a chargeback model should be understandable and transparent to the involved stakeholders [11, 30].

Understandability describes whether the charges recipients understand what they are charged for. That is, do they understand the units of service and their prices? For example, charges for cloud services based on a user subscription are more understandable than charges based on utilized central processing unit (CPU) cycles.

Transparency, on the other hand, describes whether the involved stakeholders understand how the charges are formed. Therefore, transparency characterizes the capability of stakeholders to comprehend the chargeback model and the opportunity to enjoy non-obscure chargeback processes, while understandability deals with the capability to understand the charges, the end product of applying the chargeback model.

The two concepts are closely related and discussed together in the literature [11, 30]. Understandability is necessary in order to have transparency, because lack of understanding of the charges themselves (low understandability) leads to inability to comprehend the process of forming the charges (low transparency).

The major effect of transparency and understandability on chargeback success is related to obtaining stakeholders' buy-in. Low transparency and understandability lead to resentment of the chargeback model [11, 30].

Controllability. Nolan [25] uses the controllability concept to denote to what extent consumers are in control of their IT costs. Chargeback models that enable users to have impact on IT bills by changing consumption behavior have high controllability. On the contrary, if users cannot influence IT bills, then controllability is low.

Controllability has profound impact on chargeback success. On the one hand, it is essential to enable cost reduction opportunities on the consumption side, because it allows managers to reduce their IT bills by changing consumption behavior. On the other hand, controllability influences the users' acceptability for the chargeback model [25]. Low controllability leads to resentment, because chargeback is perceived as unnecessary overhead that does not benefit managers, while high control allows them to realize cost reductions and accept the chargeback model.

Fairness. Fairness is another concept investigated by chargeback researchers [20, 21]. Those authors dub *perceived fairness* (PS) "*the key to chargeback systems effectiveness*." In their works, "*allocative*" *fairness* is used as a synonym to accuracy, while *perceived fairness* refers to user's perception of how fair the method is. It is unclear, however, whether in this case the term *fair* can be used as a synonym to the words "just" or "unprejudiced." Managers' self-interest and opportunism lead to perception of high fairness only when the chargeback model is consistent with their goals (e.g. allows them to get a higher bonus because their profit increases due to

lower IT costs). On the other hand, if a chargeback model is highly accurate, or "allocatively" fair, but leads to higher IT costs for a manager, he/she might be likely to perceive the model as unfair.

Therefore, the term PS could be considered a misnomer, because of its slightly contradictory meaning to the word *fair* (just, unprejudiced). Nevertheless, it is a useful concept to describe stakeholders' attitude towards the chargeback model and whether they are likely to accept (high PS) or reject it [20, 21].

Inferred Factors. A close inspection of the above factors influencing chargeback success reveals that their impact is realized in two separate ways. The examined forces either affect the effectiveness of the model [3, 11, 12, 25, 27], or the acceptability to the stakeholders [11, 20, 21, 25, 30]. These two aspects of chargeback success are used in the authors cited above to describe how accuracy, cost of costing, transparency and understandability, and controllability affect success. According to the above-referenced works, Fig. 1 visualizes how acceptability and effectiveness can be recognized as two high level dimensions that explain how the other forces influence success (Fig. 1).

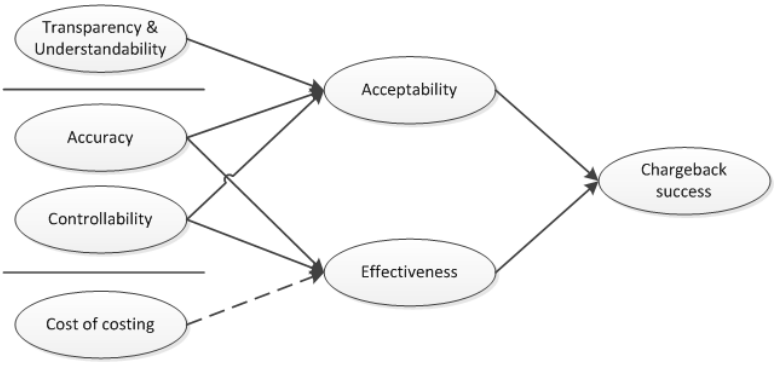


Fig. 1. Factors that impact success according to the scientific literature [3, 11, 12, 20, 21, 25, 27, 30]. Solid arrows visualize a positive relationship, while the dashed ones depict a negative relationship. Fairness is merged into acceptability and discussed further below.

Effectiveness determines whether the chargeback model stimulates the desired chargeback results, such as cost awareness and control, behavior steering, minimization of internal conflicts, more competitive costing and pricing.

Acceptability describes whether all the involved stakeholders find the model acceptable. High acceptability means that stakeholders find the model agreeable and are supportive of the chargeback process, while low acceptability stands for the lack of support for the model and results in resentment for the model.

In Fig. 1, fairness has been merged into acceptability because the latter excellently captures the meaning implied by Hufnagel and Birnberg [20, 21]. The substitution is appropriate, because those authors equate low PF to the resentment of the model, while high PF is considered equivalent to the acceptance of the model.

3 Research Method

The above factors resulting from the literature review tend to apply for chargeback in general and do not to take into account the characteristics of the Cloud. In the following research step, semi-structured interviews with Cloud and chargeback professionals were conducted to verify whether those factors indeed applied in a cloud environment and what other dimensions had to be also taken into account.

3.1 Interviewee Selection Process

To identify an extensive list of factors that influence chargeback success, twenty-five semi-structured interviews were conducted with field experts from IBM and IBM's client and partner network. Respondents from different backgrounds and positions were selected to ensure the comprehensiveness of the findings. Five types of stakeholders were identified in total: executives and board members, chargeback specialists, charges recipients, IT specialists, and consultants.

Despite the differences in background, all interviewees also shared some common characteristics. They were from Dutch origin and worked for seven large organizations in The Netherlands. In Table 2 below, each organization is briefly described in terms of industry and size.

Table 2. Organizations participating in the research by industry, size and number of respondents

Alias[1]	Industry	Size (№ employees)	№ of respondents
IBM	Consulting, technology	> 10,000	16
INS1	Insurance	1,000 – 10,000	1
INS2	Insurance	1,000 – 10,000	2
MUN	Government/municipality	1,000 – 10,000	1
GOV	Government	> 10,000	3
TRANS	Transportation	> 10,000	1
HOUS	Housing/construction	1,000 – 10,000	1

3.2 Interview Structure and Research Approach

The objective of the interviews was to ask the participants to identify what characteristics of a chargeback model affected its success and how. This required respondents to be given the opportunity to freely discuss the topic and the researcher to ask clarifying questions. The need for free bi-directional communication precluded the usage of questionnaires or formally structured interviews [10, 24]. Thus, semi-structured interviews were preferred over unstructured interviews to allow comparison between the responses of the different participants and to keep the interviews focused on the topic [24].

A two-page interview guide was developed to impose a common structure on all interviews. On top of an introduction and wrap-up, it contained sections with

[1] For confidentiality reasons aliases are used for the names of partners and clients of IBM.

interviewee background and experience, unguided questions, guided questions, and cross-validation. In the introduction the researcher and the research topic were introduced, while the background check was used to determine the stakeholder category of the interviewee.

In the guided and unguided sections the respondents were asked questions such as: "What characteristics of a chargeback model do you find relevant for the success of a chargeback model for cloud services?" and "How do those characteristics influence success?" No directions were provided by the interviewer, and subsequent questions were asked mainly to clarify what the interviewee meant by a certain concept. In the guided part of the interview, the concepts identified from literature were introduced and the respondents were asked to comment on them. It was also queried if they brought to mind additional factors. Finally, the cross-validation phase was used to validate the findings. This is discussed in the following sub-section. The wrap-up was used to verify whether the researcher had properly captured all the points made by the interviewee.

3.3 Cross Validation

The twenty-five interviews were used to empirically cross-validate the findings for correctness and completeness. Correctness was evaluated by asking the respondents whether all the discovered factors were relevant for the success of a chargeback model. Completeness was determined by asking the participants to comment on the comprehensiveness of the compiled list. Since the latter was always larger than the list identified by each individual interviewee, the participants mostly made positive comments on comprehensiveness. Therefore, the method described below was also applied to more reliably evaluate completeness.

Since it was impossible to guess how many interviews would be necessary to reach an "extensive" list a priori, a theoretical data saturation principle was applied to empirically determine the required number [14, 17]. Interviews were conducted until the moment a data saturation point was reached, i.e. the list of discovered forces started to converge and no additional factors were brought up in subsequent interviews. At this point it was deemed that the probability for additional findings from questioning more respondents was too low to justify the required research efforts and the list was considered as sufficiently complete. It was experienced that the saturation point had been reached by the twenty-fifth interview, because after interview number twenty-one each discovered factor had been mentioned at least four times, and no new factor had been identified since the twelfth interview.

4 Results

To start with, the results confirmed the relevance of the dimensions listed in Fig. 1. Furthermore, four additional factors were found to be conducive to chargeback success, and to be especially relevant in a cloud environment. These are measurability, predictability, accountability, and comparability. Table 3 displays how

many respondents mentioned each factor during the interviews. Fig. 2 visualizes how all the identified factors impact success. Subsequently, the newly discovered factors are explained.

Table 3. Number of times each factor was mentioned during the interviews by background of the twenty-five respondents

	Total[2]	Executives	Chargeback recipients	IT specialists	Chargeback specialists	Consultants
Effectiveness	13	2	0	4	6	5
Acceptability	12	1	1	5	4	6
Measurability	6	0	0	6	0	3
Accuracy	21	2	4	9	6	9
Transparency & understandability	12	1	3	4	5	4
Controllability	8	0	3	4	5	1
Predictability	14	0	4	11	1	7
Accountability	7	0	1	4	3	1
Cost efficiency	8	2	0	3	1	5
Comparability	4	0	0	2	2	2

Measurability was a concept that was mentioned mostly by the interviewed IT specialists who had in depth experience with Cloud. The term was used to refer to the degree of ease with which the chargeback model allowed measuring how many service consumption units had been used and to determine who had used those. For example, if a certain service is charged based on completed transactions, it must be possible to count how many transactions have been completed over a charging period and by whom. High measurability implied that it was possible to measure usage without highly specialized or custom made metering systems, while low measurability required such technology.

The interviewees motivated the importance of measurability, by explaining that it was difficult to measure usage in the shared virtualized infrastructure behind the Cloud and that there were still a number of technical limitations of metering systems, especially when it came to charging units such as CPU cycles or memory. Yet, as one respondent remarked, *"(…) usage based charging [was] possible only through measuring."*

Regarding the impact of measurability on success, the interviewees suggested that it positively influenced both acceptability and effectiveness. From an effectiveness perspective the capability to gather detailed usage data allowed for effective decision making and more accurate charges. On the other hand, high measurability reassured the stakeholders in the accuracy of the model, consequently improving acceptability.

Predictability describes to what extent chargeback recipients are able to predict future bills. Predictability becomes far more significant in a cloud world, because pay-per-use and elastic scaling of resources could lead to fluctuating bills that tend to bother budget minded managers. The latter do not mind bills that are smaller than expected, but it turns out that they are afraid of bills that greatly exceed expectations.

[2] Since a few respondents fell into two stakeholder categories due to their diverse job responsibilities, the sum of the five rightmost columns is greater than the total count.

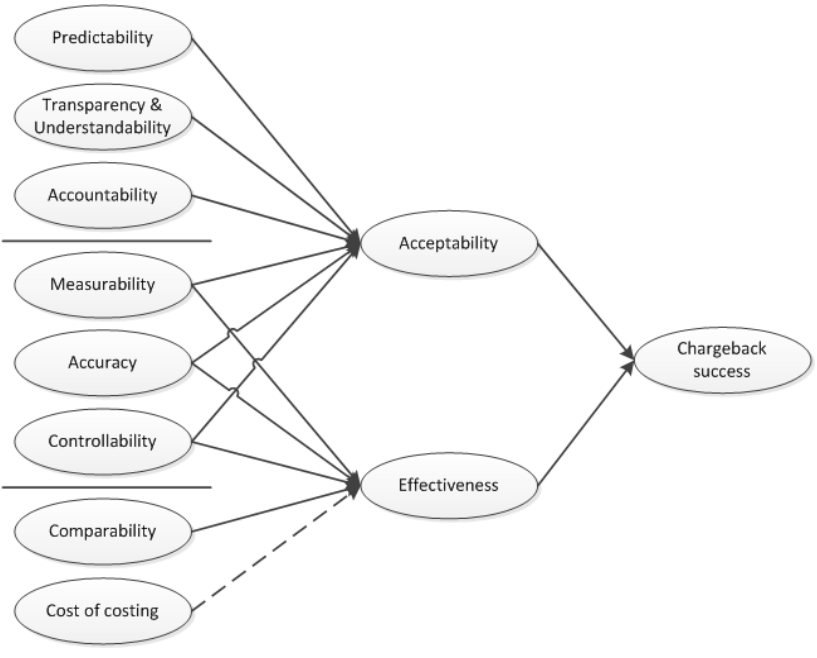


Fig. 2. Factors impacting the success of a chargeback model

Despite these concerns, the interviewees commented that predictability did not impact the effectiveness of the chargeback model. It affected only the acceptability, as unpredictability led to resentment towards the chargeback model.

Accountability is the extent to which charges recipients are able to verify the correctness of the bill. The easier it is for them to do it, the higher the accountability, and consequently, the higher the acceptability. Fixed subscription fees on a per user basis offer the highest accountability, because managers are just able to multiply the number of people in their team by the amount of the subscription fee and verify the correctness the bill. However, accountability decreases when metering of IT resource consumption is involved, such as transactions made and bandwidth consumed, because that requires from managers higher technical competence and access to specialized IT systems.

The respondents motivated the relevance of accountability by explaining that they often witnessed disputes, sometimes well-grounded, over the correctness of the bill. One interviewee summarized the general opinion as follows: *"The lower the accountability, the more often you see disputes over the bill."*

Comparability is the degree to which consumers are able to compare the prices of internally provisioned services to similar or equivalent services offered on the market. The standardization characteristic of the Cloud leads to the availability of highly similar standardized services available on the market. Therefore, in a cloud environment it becomes much easier to benchmark IT costs on a per-service basis. Yet, this still depends on how the chargeback model is designed, and whether internal service pricing follows the pricing patterns of public providers. If the chargeback

model is designed in a way that allows easy price comparison, then comparability is high. This results in cost reductions on the provisioning side, because internal users who are able to compare prices start to exercise pressure for more efficient IT delivery on the IT department.

5 Discussion

A closer analysis of the results also provides additional insights about the chargeback landscape. To start with, the interview data show that the current chargeback literature is almost oblivious in regards to the Cloud and that further research in the field is necessary. This is suggested by the fact that all the factors identified through literature research hold both for traditional and cloud environments. With CC, however, additional factors such as measurability, predictability, and comparability have been put forward by the interviews.

Another intriguing conclusion that can be drawn from Table 3 is that different types of stakeholders tend to recognize different forces. This can be used to better explain the chargeback landscape, the rationale behind the behavior of individual stakeholders, and thus to remind chargeback designers to address the unique concerns of all stakeholders. Some notable patterns that can be identified from the results are the following: chargeback specialists do not seem to be concerned with the cost of costing; measurability is a force recognized mainly by IT specialists; predictability tends to be mainly a concern of chargeback recipients and IT specialists, but not of chargeback specialists. Below these trends and their implications are discussed.

The fact that only one out of six chargeback specialists mentioned the cost of costing is at first sight perplexing, because these professionals usually have strong accounting backgrounds and their task is to keep an overview of the involved financials. One possible explanation supported during the interviews is that the negative impact of the cost of costing is offset by the benefits arising from investing in improving other factors. Yet, this view seems to contradict previous research [9].

Opportunism might provide an explanation why chargeback specialists tend to avoid recognizing the importance of cost of costing. Firstly, higher cost of costing could be expected to provide more work for the chargeback specialists and to lead to increased job security. Secondly, the application of more complex and expensive chargeback practices, but more challenging and interesting from a practitioner's point of view, could be regarded as a source of professional satisfaction.

The fact that only one out of six chargeback specialists mentioned predictability and none of them recognized measurability suggests that chargeback for Cloud is poorly understood, and that the current practices fail to leverage pay-per-use. If it were otherwise, then the concerns of charges recipient about predictability and of IT specialists about both predictability and measurability would have had proliferated also to chargeback specialists.

On top of information about the current chargeback practices, the discussion also confirms that chargeback involves a lot of political play and opportunistic behavior. For example, chargeback specialists might be inclined to adopt heavy chargeback

processes both for job security and out of professional interest. Chargeback recipients might show resentment for the charging practices if the latter put them at a disadvantage, and external consultants and solution providers might have incentives to push through expensive technology and services. Therefore, chargeback designers should be well aware of possible opportunistic behavior and should take into account the positions of all stakeholders, as well as all the factors impacting success.

5.1 Limitations

The field specialists who participated in the semi-structured interviews, the performed cross-validation of the findings, and the adherence to a data saturation principle support the validity of the findings. Nevertheless, this research has some limitations related mainly to the choice of interview participants.

To start with, the Netherlands based chargeback experience of the interviewees brings forward the question whether the findings apply globally. European and North American organizations, for example, have a history of using different accounting approaches [6]. Nevertheless, there are a number of reasons that allow for a relatively reliable generalization of the results.

Firstly, no evidence for significant differences between chargeback practices in European and non-European environments was found in the literature. Moreover, the results confirm the findings from the predominantly U.S. based literature used for this research. Thirdly, the majority of the newly discovered criteria were mainly related to the technology characteristics of the Cloud and those are globally valid. Finally, the findings are motivated by location independent arguments why certain forces impact success. Due to all those reasons, it can be considered relatively safe to assume that the findings are valid at a global level.

Another potential limitation is that the results rely mostly on IBM's chargeback expertise, because more than half the respondents were affiliated with IBM. However, the aim of this research was to provide an extensive set of factors influencing chargeback success, rather than to rank the different factors in importance or to describe how chargeback concerns differed between organizations. Therefore, expanding the list of interviewees with additional respondents with broad chargeback and cloud experience from IBM can be viewed as an appropriate decision.

5.2 Future Research

To start with, the limitations presented above already offer some opportunities for future research. For example, this study could be replicated with different companies and in different geographic areas to explore whether there are additional factors that impact success in those environments. However, the findings presented here enable further research with far more significant contributions.

This work contributes to the understanding of the impact of Cloud on chargeback and the knowledge which dimensions of a chargeback model contribute to its success. These insights can be leveraged by researchers to develop cloud-friendly chargeback models and methods that assist chargeback designers to create such models.

Furthermore, a system of metrics that allows the operationalization of the proposed above model of success factors can be developed. Such a system would not only allow for the impact of the different factors to be quantitatively verified, but would also enable research that measures the impact of different chargeback design decisions on success, and that fine-tunes chargeback models accordingly.

5.3 Contribution

To the best of own knowledge, this research provides the most comprehensive overview of the factors that impact chargeback success. Moreover, it explores the topic from the perspective of CC. Not only does this work empirically confirm the findings reported in the literature [3, 7, 9, 11, 16, 19–21, 25, 28, 30], but it also adds new factors that have not been known to researchers so far. Finally, it enables further research on the development of more cloud-specific chargeback models that can further improve how organizations benefit from chargeback and CC.

This work also has a significant societal contribution. In times of common views that IT investments do not pay back [4, 5], leveraging the findings could help improve the effectiveness of delivering and using IT in contemporary organizations. To start with, cloud adopters can benefit from this work by implementing chargeback models that can help them leverage the cost advantages and business flexibility of the Cloud and further reduce IT costs. Moreover, the results can be useful to cloud providers and providers of chargeback systems and services. Cloud providers can benefit by acquiring more knowledge about the chargeback requirements of their clients in order to offer more competitive pricing schemes. Vendors of chargeback software can update their products to better support cloud-specific chargeback models. Finally, providers of chargeback services can also use it to convince their clients of the value of external help in chargeback design and implementation.

6 Conclusion

This paper provides an overview of the factors that should be taken into account in the process of chargeback model design. Eight factors are empirically validated to affect the success of chargeback in a cloud environment – accuracy, cost of costing, transparency and understandability, controllability, measurability, predictability, accountability, and comparability – and their impact on success is explained through two higher level variables, acceptability and effectiveness. The findings suggest that to develop successful chargeback models, chargeback designers should try to optimally balance between the presented dimensions and should address the concerns of the different stakeholders. Furthermore, while measurability, predictability, and comparability tend to be irrelevant or neglected in a traditional IT environment, the results evidence that special attention should be paid to these factors in a CC context.

Finally, a few concluding remarks are made on the timeliness of this work. The results emerge at a moment in which the adoption of cloud services is at its beginning. Therefore, the findings can promptly assist organizations that need to implement

cloud-specific chargeback models as part of the transition to the Cloud. Rather than summarizing the accumulated chargeback experience once the transition is in a more developed stage, this research provides timely empirical findings that can enable immediate actions and future research that can further facilitate the move to the Cloud.

References

1. Armbrust, M., et al.: A view of cloud computing. Commun. ACM 53(4), 50 (2010)
2. Atkinson, A.A., et al.: Management Accounting. Prentice Hall, New Jersey (2003)
3. Bergeron, F.: Factors Influencing the Use of DP Chargeback Information. MIS Quarterly 10(3), 225–237 (1986)
4. Carr, N.G.: Does IT Matter? Information Technology and the Corrosion of Competitive Advantage. Harvard Business Press, Boston (2004)
5. Carr, N.G.: IT Doesn't Matter. Harvard Business Review 81(5), 41–49 (2003)
6. Chandler, A.D., Daems, H.: Administrative coordination, allocation and monitoring: A comparative analysis of the emergence of accounting and organization in the U.S.A. and Europe. Accounting, Organizations and Society 4(1-2), 3–20 (1979)
7. Cisco: Managing the Real Cost of On-Demand Enterprise Cloud Services with Chargeback Models (2010), http://www.cisco.com/en/US/services/ps2961/ps10364/ps10370/ps11104/Cloud_Services_Chargeback_Models_White_Paper.pdf
8. Computer economics: IT Budget Chargebacks: Making Users Pay, http://www.computereconomics.com/article.cfm?id=1293
9. Cooper, R.: Cost management concepts and principles. Journal of Cost Management, 45–49 (Spring 1987)
10. Corbetta, P.: Social Research: Theory, Methods and Techniques. Sage Publications, London (2003)
11. Drury, D.H.: Assessment of Chargeback Systems in IT Management. Infor. 38(3), 293–313 (2000)
12. Drury, D.H.: Conditions affecting chargeback effectiveness. Information & Management 5(1), 31–36 (1982)
13. Drury, D.H.: The dialectic of IT chargeback systems. International Journal of Technology Management 14(5), 496–512 (1997)
14. Eisenhardt, K.M.: Building Theories from Case Study Research. The Academy of Management Review 14(4), 532–550 (1989)
15. Finden-Brown, C., Long, C.: Introducing the IBM Process Reference Model for IT: PRM-IT Sequencing the DNA of IT Management, vol. 3. IBM Global Services, New York (2008)
16. Gerlach, J., et al.: Determining the cost of IT services. Commun. ACM 45(9), 61–67 (2002)
17. Glaser, B., Strauss, A.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Transaction, Piscataway, New Jersey (1967)
18. Harms, R., Yamaritino, M.: The Economics of the Cloud (2010)
19. Holzer, P., Norreklit, H.: Some thoughts on cost accounting developments in the United States. Management Accounting Research 2(1), 3–13 (1991)
20. Hufnagel, E.M., Birnberg, J.G.: Perceived Chargeback System Fairness in Decentralized Organizations: An Examination of the Issues. MIS Quarterly 13(4), 415–430 (1989)

21. Hufnagel, E.M., Birnberg, J.G.: Perceived chargeback system fairness: A laboratory experiment. Accounting, Management and Information Technologies 4(1), 1–22 (1994)
22. IBM: Smarter Planet: Using cloud computing to deliver innovation and efficiency. Smarter Planet Client Events, New York (2010)
23. Iqbal, M., Nieves, M.: Service Strategy Book. The Stationery Office, Norwich (2007)
24. Kajornboon, A.B.: Using interviews as research instruments. E-Journal for Research Teachers 2(1) (2005)
25. Nolan, R.L.: Effects of chargeout on user/manager attitudes. Communications of the ACM 20(3), 177–185 (1977)
26. Oleson, T.D.: Price of precision. CIO Magazine, 13 (1998)
27. Raghunathan, B., Raghunathan, T.: A discriminant analysis of the relationship between IS charging systems and organizational variables. Omega 22(4), 321–330 (1994)
28. Ross, J.W., et al.: The Untapped Potential of IT Chargeback. MIS Quarterly 23(2), 215–237 (1999)
29. Stefanov, H.: How to Charge for the Cloud? Towards a structured view of successful IT charge back models for Cloud services. Utrecht University (2011)
30. Stiller, B., et al.: Charging and accounting for integrated internet services - state of the art, problems, and trends. Presented at the INET 1998: The Internet Summit, Geneva, Switzerland (1998)
31. VanLengen, C.A., Morgan, J.N.: Chargeback and maturity of IS use. Information & Management 25(3), 155–163 (1993)
32. Vaquero, L.M., et al.: A break in the clouds: Towards a Cloud Definition. ACM SIGCOMM Computer Communication Review 39(1), 50 (2008)
33. de Vos, J., van Heusden, E.: Implications of Cloud Computing on IT Service Management (2010)

A Marketplace Framework for Trading Cloud-Based Services

Andreas Menychtas[1], Sergio Garcia Gomez[2], Andrea Giessmann[3], Anna Gatzioura[1], Katarina Stanoevska[4], Jürgen Vogel[3], and Vrettos Moulos[1]

[1] National Technical University of Athens, Greece
{ameny,agatz,vrettos}@mail.ntua.gr
[2] Telefónica Investigación y Desarrollo, Spain
sergg@tid.es
[3] SAP Research Center St. Gallen, Switzerland
{andrea.giessmann,juergen.vogel}@sap.com
[4] University of St. Gallen, Switzerland
katarina.stanoevska@unisg.ch

Abstract. The importance of marketplace frameworks, where demand and supply for electronic services meet, has gained momentum with the recent technological innovations of cloud computing. In particular the emerging market for cloud and XaaS offerings is, in the current early stage of development, scattered and represented by many single offerings. New intermediaries are required for the consolidation of the available service offerings and for providing a one-stop-shopping opportunity for customers. This paper proposes a new cloud marketplace solution that enables on the one hand an integrated platform for the development and selling of XaaS products and on the other hand a one-stop-shopping for customers interested in services. Service providers can merchandise and sell their products through the marketplace supporting the whole lifecycle of these products. Service consumers are provided with a unique personalized service search and resolution engine, helping them to find and customize the products they need.

Keywords: Electronic marketplace, cloud computing, trading services, business resolution, analytics.

1 Introduction

Clouds can be considered nowadays as a common solution for trading and provisioning any type of ICT assets as products, which in the cloud terminology is denoted as XaaS (Everything as a Service) [2, 19]. The term refers to an increased number of cloud-based resources and services provided over the Internet, with the most common examples, following the SPI model [18], Software (SaaS), Platform (PaaS) and Infrastructure (IaaS) as a service. Other examples of XaaS may be storage, communications, network and monitoring as a service. Besides the technical

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 76–89, 2012.

advancements in the area of cloud computing, there are still several limitations [3], especially from the business perspective that clouds need to overcome in order to allow the wide adoption of clouds as true business ecosystems. In this paper, we propose a marketplace for cloud-based services that provides simplified, effective and agile processes to all stakeholders involved in the value chain of a product, for consuming or providing services and resources. The purpose of this marketplace is not only to offer a single, well-known meeting point for the different stakeholders but also to support the various technical and business requirements in all phases of the service lifecycle (planning, analysis and design, development and testing, provisioning, deployment, discovery, composition, execution, and monitoring, - see [14]). The marketplace, as part of an integrated cloud platform [1], will allow XaaS providers to publish their products in a managed environment, which controls the business terms and conditions (price, revenue sharing, promotion, etc.), including advanced pricing and billing capabilities [12].

In that sense, a cloud marketplace should address several challenges in order to provide efficient communication between the various stakeholders involved. In the service discovery phase, marketplaces should be capable to interpret the high level business and technical requirements from customers and select products or recommend product compositions. While various approaches exist on performance prediction and according to that, proceed with the aforementioned discovery phase (e.g. [8]), marketplace operations mainly depend on cost effective pricing models and Service Level Agreements (SLAs) which capture the technical and business requirements of customers and providers [6]. Based on the selected pricing models and SLAs, the required services and resources are identified, resolved (analysis of the dependencies) and deployed while respective monitoring and management policies are negotiated with the underlying cloud layers.

The proposed framework aims at providing advanced functionality for trading of services in the cloud that fosters the development of a dynamic and fair ecosystem for services and service-based applications. To this direction, a provider that will use the marketplace will be able to define new product offerings, including all business information (price, application level SLAs, etc.), while the customer will be able to search for services, customize and contract them.

Even though the marketplace is currently an integral part of the overall cloud platform developed in the 4CaaSt project [1], we envision that it could be integrated to any cloud environment since it follows a service oriented architectural design and state of the art technologies and standards. The marketplace represents the customer interface of the platform and therefore will simplify the interactions between the customer and the cloud and cover many aspects of the overall lifecycle. In addition, its functionality will be enhanced with processes for collecting valuable information for rating, billing and settling the incomes as well as statistics analytics in favour of providers and the marketplace itself. By analysing past interactions between providers and consumers, the marketplace will be able to offer best practices as an added value.

Through the marketplace, service providers are allowed to define and customize a commercial offer that will be resolved, deployed and fulfilled in a personalized manner for each customer by selecting the most appropriate services from the available ones. To this direction, the customers will be able to receive a customized product, the price of which may be determined dynamically based on the configuration options defined by the provider and the deployment decisions taken by the cloud platform. In addition, extended functionality for simulating the business aspects of the offered products, for defining effectively their business terms and conditions, is provided.

The rest of the paper is structured as follows. Chapter 2 presents the related work on electronic marketplaces and the outcomes of a survey on the capabilities of existing electronic marketplace environments. Chapter 3 highlights the functionality of the proposed solution as part of a cloud ecosystem. The architecture and implementation details of the framework are presented in chapters 4 and 5 respectively. Chapter 6 analyses the innovations of the marketplace based on the evaluation of the first implemented prototype and finally chapter 7 concludes our work.

2 Related Work

An electronic marketplace is a platform where demand and supply for certain goods meet in order to: a) offer products and services in an structured manner as well as to select and find required products and services, b) to negotiate the price and conditions, c) to set up a contract, and d) to pay and deliver the offered products and services [7, 10]. A typical market transaction within an electronic marketplace therefore contains four specific phases: a) information, b) negotiation and price setting, c) contracting, and d) settlement. This already indicates the three major roles and players that are active on a marketplace. First, the providers of products and services, second the consumers of products and services, and finally, the marketplace provider himself who is basically providing the market infrastructure and may support and participate actively, also taking a share of the revenues generated on the marketplace. In contrast to traditional markets, electronic markets are placeless resp. ubiquitous, just like cloud-based services. In this sense, an electronic marketplace corresponds to the concept of cloud-based services, which also have a global and instantaneous reach [20].

According to [4] "...the interesting question is whether and how services will be traded in the future". Further prior studies that have noted the importance of the role of intermediaries in electronic markets are among others [5, 9]. Within Table 1 six of the most sophisticated, existing electronic marketplaces have been evaluated based on their support of a typically market transaction. The four main phases of market operation plus the analytics process (which is a cross-phase process) have been broken down into a list of features, which serve as evaluation criteria within the paper at hand.

Table 1. Evaluation of existing marketplaces for electronic services

Functionality \ Marketplace		Windows Azure Marketplace	Amazon WS	Google Apps Marketplace	App Store	AppExchange on Force.com	Android Market	SuiteApp.com	Zoho
1.	**Information phase**								
1.1	Product Definition	+	+	+	+	+	+	+	+
1.2	Product Description	-	+	-	+	+	+	-	-
1.3	Product Search	+	+	+	+	+	+	+	+
1.4	Bid Search	-	-	+	-	+	-	-	+
1.5	Related Products	+	+	+	+	+	+	-	+
1.6	Recommendation	-	-	-	+	+	+	-	-
1.7	Competition Analysis	-	-	-	+	+	+	-	-
1.8	Business Analytics	-	+	+	+	+	+	-	-
1.9	Community rating & comments	+	+	+	+	+	+	-	-
1.10	Social Graph Analysis	-	-	-	-	-	-	-	-
2.	**Negotiation phase**								
2.1	Bid to Product	-	-	-	-	+	-	-	+
2.2	Product Resolution	-	-	-	-	+	-	-	-
2.3	Product Customizing	-	+	+	-	+	+	-	-
2.4	Product Specification	-	-	+	+	+	+	-	-
2.5	Composite Resolution	-	-	+	-	+	-	-	-
2.6	Real-time Resolution	-	-	-	+	+	-	-	+
2.7	Profile based Resolution	-	-	-	+	-	+	-	-
2.8	Basket Management	-	-	+	+	+	+	-	-
3.	**Contracting phase**								
3.1	Contract Management	-	+	+	-	+	+	-	-
4.	**Settlement phase**								
4.1	Delivery Support	-	+	+	+	+	+	+	+
4.2	Payment Support	-	+	+	+	-	+	-	-
4.3	Rating & Charring	-	+	-	+	-	-	-	-
5.	**Analytics**								
5.1	Show Competitive Products	-	-	-	+	-	-	+	+
5.2	Reporting on Products, Incomes	-	+	-	-	-	+	+	+

The Windows Azure Marketplace, Amazon Web Services, SuiteApp.com and Zoho provide advanced service and application directories, where service providers offer their products and services in a structured manner. However, they mainly focus on the first phase of a market transaction. The negotiation, contracting and settlement phase are not covered. These activities are usually handled by the service providers themselves rather than being supported by the marketplace.

The focus of the Android Market, as well as Apples' App Store, is slightly different, since most of the traded products are games and entertainment applications. Furthermore, these products are no cloud-based services as such, since they are installed locally. However, both marketplaces provide sophisticated features for service/application providers, as for instance service definition and description, business model specification and revenue analysis.

Google Apps Marketplace takes advantage of the broad palette of Google tools to cover the different phases of a market transaction. Marketplace users can utilize other services provided by Google like Checkout, Analytics or Product Ideas. The Google Apps Marketplace itself puts emphasis on refined search functions for consumers and delegates other phases of the market transaction to complementary Google products.

AppExchange from Salesforce.com is the most advanced marketplace within our evaluation, since it covers most of the required features for trading cloud-based services. However AppExchange, just like the other marketplaces within our investigation, does not support dynamic price and revenue networks.

Composite services and resolutions are supported by the Google Apps Marketplace and AppExchange from Salesforce.com. According to the imposed restrictions and the current demand, in AppExchange from Salesforce.com and Zoho real-time service resolution takes place. Most of the examined marketplaces keep some information about users which help them personalize and ease service recommendations but only Android Market performs a customer profile based resolution.

3 Marketplace Functionality in the Cloud Ecosystem

Following the analysis of the electronic marketplaces in the previous section, we have identified the most relevant user roles for a cloud-based marketplace. On the consumer side, we can distinguish between a) end-consumers, who just consume a certain service, b) business customers, in particular SMEs, who take advantage of the cloud services available on the marketplace and c) enterprises that exploit the platform resources as a special case of cloud bursting. On the provider side, we can distinguish between Network as a Service (NaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) providers [18]. Another important entity in service oriented infrastructures and Clouds is the service aggregator with the role of both consumer and provider, who combines offerings of different types into new composite products that will be merchandised afterwards in the marketplace.

The main marketplaces processes are separated in four different phases (Fig. 1). In the first phase, so called information phase, the market players need to exchange information. A service provider describes his products technically and economically, while a consumer is looking and searching for the specific service that might fit his needs.

In the second phase, after the consumer has chosen certain services, resolution and price building takes place. The consumer is allowed to define specific constraints for his/her request or select from a set of customization options, predefined by the service provider, for a particular product. The marketplace will identify any business dependencies and propose a selection of services and resources that satisfy the

constraints that are defined by the consumer. Part of this process are the price aggregations and the selection of appropriate pricing models based on a) the request parameters, b) the availability of services and resources from the Cloud, c) the user profile and finally d) the experience from previous selections exploiting the analytics functionality of the marketplace. Following the proposal(s) from the marketplace, customers are able to negotiate (if allowed by the product definition) the explicit terms and conditions under which the product and its dependencies will be provisioned. In that sense, if a customer is not satisfied with the returned results, he/she is able to change his/her constrains and customization options and repeat the resolution process.

In the third phase, when provider and consumer have agreed on a specific product and price, a legally-binding contract has to be defined. The contract, which is created automatically from the information obtained in the previous phases includes also a set of appropriate SLAs with contains, compensations and provisioning policies for all involved entities. The outcome of this phase is a signed contract which becomes a legally binding document for both sides.

The final phase of the marketplace is the so-called settlement. The settlement has two major activities: a) the delivery, which means that the consumer gets access to and can use the contracted service or application and b) the payment. Part of the settlement phase is the actual deployment and instantiation of the required services on the Cloud, a process that is carried out from the underlying layers of the platform. Payment, on the other hand, involves the transfer of money from the consumer to the provider through the marketplace. This activity also includes the automatic distribution of the income among all the involved parties in case a product is an aggregation of offerings from different providers.

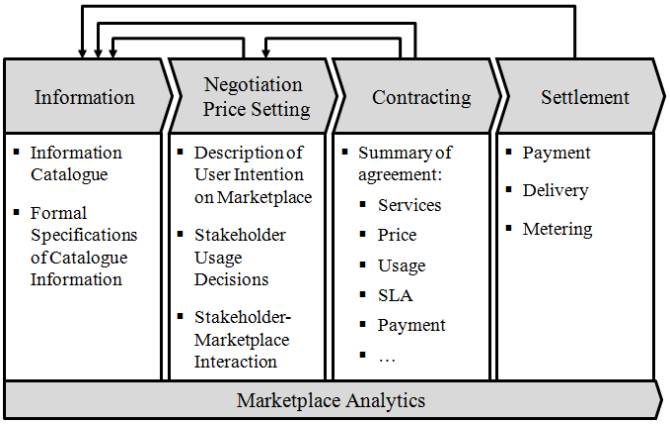


Fig. 1. Marketplace processes

The four main phases of the marketplace are also enhanced with the marketplace analytics functionality. The marketplace analytics is a horizontal process that aggregates information from various sources (operational, business, social, user feedback) in order to gain knowledge for improving the business terms and conditions for new products and business models. Based on the knowledge on the performance of previous business resolutions, the marketplace can perform more efficient selections of products, identify

optimal pricing models and recommend product compositions, which will lead to increased revenues for all involved stakeholders that are involved in the product lifecycle.

4 Architecture

The detailed architectural design of the marketplace including the main components, repositories and interfaces is depicted in Fig. 2. On the top layer, the marketplace includes a frontend for communication with the end-users which is composed of four elements: a) *marketplace management*, where the marketplace manager can define price models, settlement rules and policies, the service provider defines the business terms and conditions for a product based on its functional and non-functional specification and also customers that contract applications and services; b) *composition support*, where a consumer defines the constraints for his request and customizes the product; c) *payment*, where providers and consumers get information about payments, revenues and the revenue sharing; and finally d) *reporting*, where providers and customers can access reports and information about services consumption and billing, statistics and results of simulations.

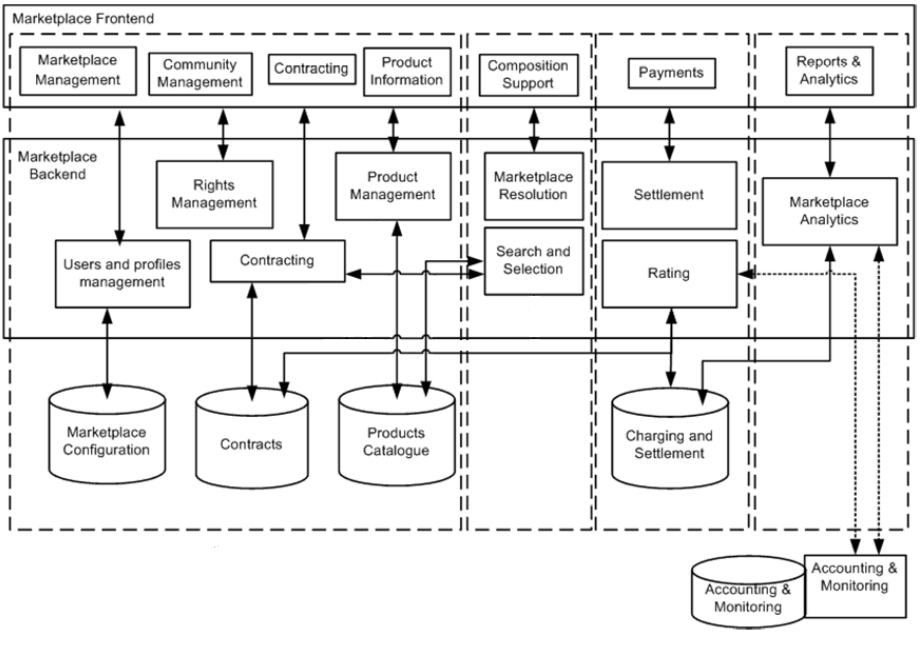


Fig. 2. Marketplace architecture

The marketplace architecture defines four main repositories (although other components may have their own persistence mechanisms). The first repository is the

Product Catalogue, where the product specification is stored and linked to the corresponding services and applications in the service repository. This repository is always up-to-date, with the providers allowed to update or delete dynamically the created products. Off course these changes only affect the future selections and not the contracted or past products. A second repository is the *Contracts Repository*, where the agreements between providers and consumers for each marketplace transaction are stored. The contracts stored in this repository also include the detailed specification of the contracted product (customization parameters, SLAs for the QoS parameters, business and operational management policies, etc.). The *Marketplace Configuration repository* is used for the internal operation of the marketplace environment storing information for user profiles, user rights etc. Finally, the *Charging and Settlement repository* keeps the accounting information for each product that is instantiated and provisioned on the Cloud. This information is used afterwards for the payment and revenue share between the various stakeholders.

In the business logic layer we have identified and implemented the following components:

- *Resolution Engine*, where the business constraints defined by both the service provider and the customer are used to select the best platform, infrastructure and services to resolve the service description.
- *Contracting* that handles negotiation, configuration and price building of services.
- *Search Engine* that helps in selecting the best product according to a number of criteria, including past experience and user profiles.
- *Rating and Settlement* components are in charge of rating the consumed services and settling the incomes according to the participation in the services and revenue sharing policies.
- *Business Analytics* that aggregates the functionality of generating reports and statistics, business intelligence and simulations of the marketplace under different conditions.
- *Marketplace Management* components are responsible for the basic operational aspects of the marketplace (users, profiles, rights, products).

5 Implementation

5.1 Product Definition

One of the main objectives of the proposed marketplace is to be able to trade any kind of cloud-based services following the notion of XaaS. In that sense, the product model that is used in all marketplace processes should be carefully designed in order to cover the technical and business aspects of any XaaS offering. The product model is depicted in Fig. 3. It is described as an XML Schema, in order to facilitate the exchange of product instances between the different components of the marketplace.

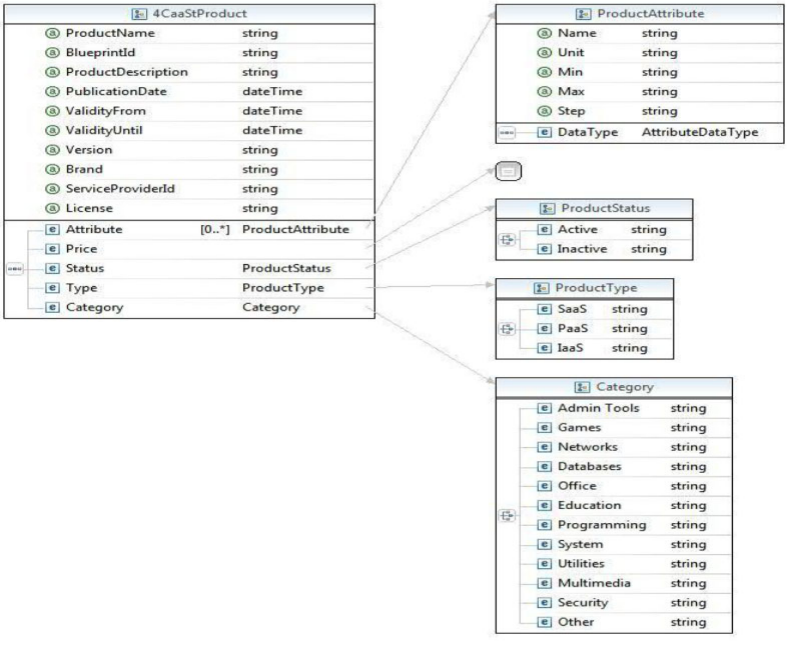


Fig. 3. Product XML schema

The product description includes the following information:

- *ProductName*: The name of the product to be shown to the customer.
- *BlueprintId*: The Id of the service on which the product is based.
- *ProductDescription*: A customer friendly description of the product.
- *PublicationDate*: Date of release of the product in the marketplace.
- *ValidityFrom*: Date from which the product can be contracted in the marketplace.
- *ValidityUntil*: Date until which the product can be contracted in the marketplace.
- *Version*: Release version of the product.
- *Brand*: Commercial Brand of the product/service provider.
- *ServiceProviderId*: Id of the service provider in the marketplace.
- *License*: Textual name of the (usually software) license that applies to the product.
- *Attribute*: A set of configuration attributes, fixed in design/definition type that allows the customer to customize the application or service. This attributes are specified with a name, unit, range of values and step.
- *Price*: Defines the cost of the product based on other technical or business parameters and is described by the unified service description language (USDL).
- *Status*: If the service is already active or not in the marketplace.
- *ProductType*: Description of the type of product/service (SaaS, PaaS, IaaS).
- *Category*: Application category (games, databases, etc.).

5.2 Product Search and Selection Lifecycle

The Product Search and Selection functionality is considered as fundamental in any electronic marketplace, and therefore the implementation and evaluation of the initial marketplace prototype focused on this aspect. The various interactions among the end user and the different components of the marketplace are presented in Fig. 4 and described in detail below.

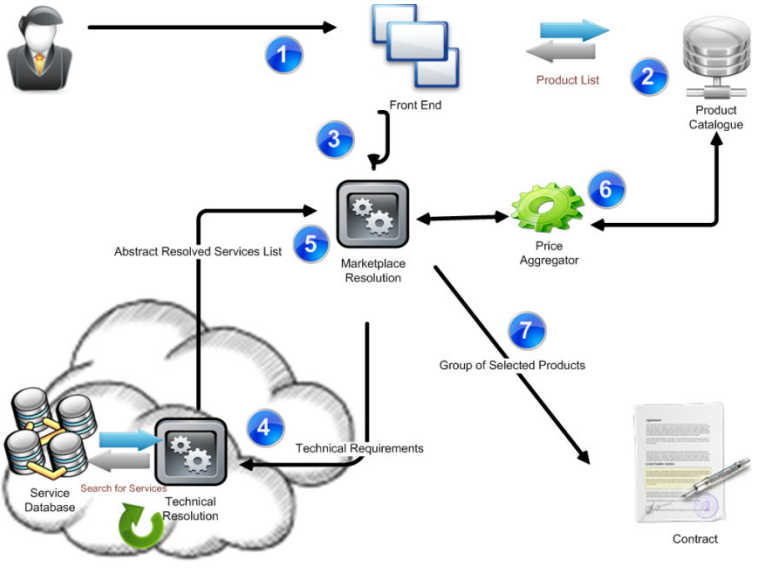


Fig. 4. Product search and selection lifecycle

1) The search and selection process is triggered when a user accesses the marketplace through its frontend in order to find possible products addressing effectively his needs.
2) The user browses the product catalogue and searches among the already existing products. In order to make his search more effective and less time consuming, the queries may be categorized into groups, according to the product type etc. The result of this query is the set of candidate products that are compliant with the initial request.
3) Based on the results of the query, the end-user is asked to impose, through the portal, QoS and/or business constraints, as well as to select from a set of customization options for the available products. These constraints and/or customization options, which may be of different types, are categorized into business and technical. The technical are provided as input to the technical resolver, while the business ones are treated in the marketplace resolution.
4) The technical resolution (part of the underlying cloud layers as tightly coupled to the technical and deployment aspects of the service) performs a functional analysis and resolution, to an abstract list of services (and service dependencies) based on which the final set of products will be selected. The technical

resolution process uses as main input "blueprint" documents that are stored in the service database. These documents, provided by the service developers, describe the requirements and dependencies of each service from the undelaying layers or from other services to be functional.

5) From a business perspective, the resolution refers to identifying those products that fulfil some business constraints, especially related to pricing and SLAs. To this direction, the products that are loaded and then evaluated according to the user requests, correspondent to the abstract services list from the previous step. The marketplace resolution engine in order to perform an effective business resolution and product selection must take into consideration also the pricing aspects and therefore the Price Aggregator (step 6) is "called". The final product selection will be made based on the predefined, by the user, desirable performance and characteristics, the weighted parameters showing their importance to the user and the predicted usage (ex: per use or month). Each selection comes with detailed policies for provisioning the product, from both business and technical perspectives, considering a) the multi-tenancy and elasticity capabilities of each product, b) the capabilities of the platform and infrastructure layers and c) the customer requirements and constraints.
6) The Price Aggregator calculates the final price for each product, based on the different pricing functions and business models suitable for each one. In addition, it analyses the pricing and business functions and describes the corresponding revenue sharing schemas.
7) When the final selection is made, a contract is generated and signed electronically by all involved stakeholders. This contract contains the description of the terms for the provisioning of the product, the pricing model and the SLA terms.

6 Marketplace Innovations

6.1 Integrated Marketplace

The proposed marketplace fosters the creation of dynamic business ecosystems in which multiple service providers may find a fair environment to develop and provide new services [13]. The support of flexible revenue models will help cloud providers to monetize the effort of creating and maintaining the platform, while service providers can make business with a zero capital expenditure (CAPEX) entrance barrier [11].

The marketplace supports the trading of different types of services in a unified way, a single access point to all type of service offerings: SaaS, PaaS, IaaS, NaaS or generally XaaS. A tight relationship with the service engineering layer supports the specification of commercial offers for any type of service, using the most appropriate revenue models for each service, simplifying the process of building applications. The marketplace can also apply specific policies and constraints for deciding which platforms and services will be used in the final service deployment, based on the business requirements of both the service provider and the customer.

At the settlement phase, the services running in Cloud are provisioned, monitored and accounted according to the signed contracts. Since the services may be the result of a resolution and composition process, the marketplace revenue sharing mechanism simplifies the accounting process allowing the service providers to foster the usage of their offer in the final applications and services.

6.2 Service Search, Selection, and Resolution

One of the most important functions of the marketplace is the resolution of the customers' requirements to appropriate products or product compositions that will be contracted and provisioned. The requirements are typically associated with high level, application-specific aspects of the product and in that sense, a great advancement of the marketplace will be the capability to analyse and translate customer requirements to a set of technical and business parameters for services, technologies and resources. During this resolution (technical and business), appropriate SLAs that include obligations and policies for the provisioning of a product on the required Quality of Service (QoS) level, are identified and selected.

Part of this process is also the selection of appropriate pricing model(s) capable to support effectively the business aspects of a product, for both the provider and the customer, as well as to identify appropriate SLAs for the products of the resolution tree, based on the customers QoS and/or business constraints. The resolution and selection of products and pricing models is based on a dynamic set of parameters, taking into consideration not only the customer's request but also information from the cloud such as the user profile (contextual information related with a user), the market and infrastructure status and the experience about the effectiveness of previous resolutions. In order to support this, the business resolution engine implemented following a pluggable architectural design, will allow developers, providers and marketplace operators to produce advanced and sophisticated algorithms for the resolution of specific types of requests or products. The dynamicity of the resolution process enables further optimization of native and immigrant products and platform resources, in both technical and business level and also the development of new, personalized product offerings, product compositions, and pricing models.

6.3 Generic Handling of Price and Revenue Models

Depending on his business model, a service provider may choose one or more price models for each of his service offerings. In the general case, the price is a function that depends on one or more parameters. For instance, a price may be the combination of a fixed subscription rate s plus a transaction fee tf, say price $p(x) = s + x\ tf$, where x is number of service calls. A provider may also choose to define several different price model alternatives for the same service in order to meet different customer demands. Also, a price model may be adapted during the negotiation phase between service provider and service consumer. A key functionality of our marketplace is that it allows the service provider to define appropriate price models and supports him on this task, e.g., by preconfigured, best practice price models or by simulation and analysis.

The proposed framework also envisions that several existing service offerings can be combined into new, composite ones, either by one of the original service providers or by another one (who would then either be a mere reseller or provide a value-added service). In the general case, each service may be based on other services forming a service network. At the same time, each service is offered by a certain provider who may then be the consumer of another provider. The marketplace processes allow service providers (and consumers) to analyse this business network (e.g. by simulating alternatives when selecting services and price models) and calculate the overall price model for the consumer of a service network. Vice versa, when distributing the generated income for a certain market transaction from the final consumer to all involved providers according to the business network, the resulting complex revenue sharing model is also managed by the marketplace.

7 Conclusions and Outlook

At present, the cloud market is scattered and consists of providers specializing in IaaS, PaaS or on specific SaaS offerings. However, the potential customers of cloud services increasingly require integrated solutions for both developers and providers of services. This paper presents a new cloud marketplace as a solution for increasing the transparency and efficiency of the cloud market.

The proposed marketplace offers a one-stop support for providers and customers. Providers can draw upon an integrated offering of PaaS and support for the full development life cycle of their services on the one hand and for the selling phase of the software on the other hand. In particular, in the selling phase of the life cycle of a service, the developers can rely on the marketplace for sophisticated price resolution and revenue sharing support. Potential customers of services are supported by functionality for a broad selection of XaaS, sophisticated service search, selection and resolution as well as one-stop pricing of bundles of services. In our future work, the search and resolution facility will be enhanced with social recommendations including information for relationships among suppliers, customers and services, allowing more effective and personalized use of all cloud assets.

Acknowledgement. This work has been supported by the 4CaaSt Project (http://www.4caast.eu) and has been partly funded by the European Commission's IST priority of the 7th Framework Programme under contract number 258862. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. 4CaaSt EU Project, `http://www.4caast.eu`
2. Lenk, A., Klems, M., Nimis, J., Tai, S., Sandholm, T.: What's inside the Cloud? An architectural map of the Cloud landscape. In: ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD 2009), pp. 23–31. IEEE Computer Society (2009)

3. Menychtas, A., Kousiouris, G., Kyriazis, D., Varvarigou, T.: Minimizing technical complexities in emerging cloud computing platforms. In: Proceedings (LNCS) of the Cloud Computing: Project and Initiatives, Ischia, Italy, August 31-September 3 (2010)
4. Legner, C.: Is There a Market for Web Services? In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 29–42. Springer, Heidelberg (2009)
5. Weinhardt, C., Anandasivam, A., Blau, B., Stößer, J.: Business Models in the Service World. IT Professional 11(2), 28–33 (2009)
6. Kyriazis, D., Menychtas, A., Varvarigou, T.: Grid Workflows with encompassed Business Relationships: An approach establishing Quality of Service Guarantees. Quantitative Quality of Service for Grid Computing. IGI Global Publishers (2009)
7. Gillett, E., Brown, E.G., Staten, J., Lee, C.: Future View: The New Tech Ecosystems Of Cloud, Cloud Services, And Cloud Computing, Forrester, for Vendor Strategy Professionals, August 28 (2008)
8. Kousiouris, G., Kyriazis, D., Konstanteli, K., Gogouvitis, S., Katsaros, G., Varvarigou, T.: A Service-Oriented Framework for GNU Octave-Based Performance Prediction. In: IEEE SCC 2010, pp. 114–121 (2010)
9. Giaglis, G.M., Klein, S., O'Keefe, R.M.: The role of intermediaries in electronic marketplaces: developing a contingency model. Information Systems Journal 12, 231–246 (2002)
10. Li, H., Jeng, J.: CCMarketplace: a marketplace model for a hybrid cloud. In: CASCON 2010 Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (2010)
11. Armbrust, M., Fox, A., Griffith, R., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. Communications of the ACM 53, 50–58 (2010)
12. Eurich, M., Giessmann, A., Mettler, T., Stanoevska-Slabeva, K.: Revenue Streams of Cloud-based Platforms: Current State and Future Directions. In: Proceedings of the Seventeenth Americas Conference on Information Systems (AMCIS), p. 10. AISel, Detroit (2011)
13. Morris, M., Schindehutte, M., Allen, J.: The entrepreneur's business model: toward a unified perspective. Journal of Business Research 58(6), 726–735 (2005)
14. Papazoglou, M.P., Heuvel, W.: Business Process Development - Life Cycle Methodology. Communications of the ACM 50, 79–86 (2007)
15. Sarkar, M., Butler, B., Steinfield, C.: Cybermediaries in Electronic Marketspace: Toward Theory Building. Journal of Business Research 41, 215–221 (1998)
16. Johnson, M.W., Christensen, C.M., Kagermann, H.: Reinventing your business model. Harvard Business Review, 51–59 (2008)
17. Drucker, P.F.: The Discipline of Innovation. Harvard Business Review, 95–103 (August 2002)
18. Mell, P., Grance, T.: NIST Definition of Cloud Computing, Version 15 (2009) http://csrc.nist.gov/groups/SNS/cloud-computing
19. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Br, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility
20. Bakos, Y.: The Emerging Role of Electronic Marketplaces on the Internet. Communications of the ACM 41, 35–42 (1998)

Client Classification Policies for SLA Negotiation and Allocation in Shared Cloud Datacenters

Mario Macías and Jordi Guitart

Barcelona Supercomputing Center/Universitat Politecnica de Catalunya
Jordi Girona 29, 08034 Barcelona, Spain
{mario.macias,jordi.guitart}@bsc.es

Abstract. In Utility Computing business model, the owners of the computing resources negotiate with their potential clients to sell computing power. The terms of the Quality of Service (QoS) to be provided as well as the economic conditions are established in a Service-Level Agreement (SLA). There are situations in which providers must differentiate the SLAs in function of the type of Client that is willing to access the resources or the agreed QoS e.g. when the hardware resources are shared between users of the company that own the resources and external users.

This paper proposes to consider the information of potential users when the SLA is under negotiation to allow providers to prioritize users (e.g. internal users over external users, or preferential users over common users). Two policies for negotiation are introduced: price discrimination and client-aware overselling of resources. The validity of the policies is demonstrated through exhaustive experiments.

Keywords: Cloud Computing, Client Classification, SLA Negotiation, SLA Allocation, Business Modeling.

1 Introduction

In recent years, the Utility Computing business model is increasing its acceptance in the Information Technology sector [19] thanks to the burst of Cloud Computing paradigm [8]. In Utility Computing, the users of the resources are not necessarily their owners: users run their applications or services in remote data centers and pay in function of the usage, as with other utilities such as water provision or the electric grid. The terms of the Quality of Service (QoS) to be provided and the economic conditions are established in a Service-Level Agreement (SLA). Utility Computing allows the users to economically benefit from economies of scale, because it minimizes the space and maintenance costs. However, despite of the economic benefits of using computing as a utility, there are still open security reasons to not submit the critical or confidential data to resources that are located in third parties [13].

Companies may decide to hire out the spare resources of their data centers to external users that do not have such security or confidentiality restrictions [11]. The price that external users pay to use the resources contributes to amortize the cost of the data centers. However, a binary classification of the users as

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 90–104, 2012.

internal/external is not accurate enough in many situations. For example, headquarters of a big company may classify the users of its data centers according to different levels: users from the headquarters that owns the resources are completely internal, users from other companies are completely external, and users from other headquarters of the same company have an intermediate range. Even multinationals could define more degrees of proximity for headquarters in the same country and headquarters in other countries. Whilst completely external users pay a fee and completely internal users use the resources for free, the users in between would pay a reduced fee that does not report profit, but encourages each location to only use resources from external locations when strictly necessary.

Another example of intermediate users are users from trusted entities that decide to share their computing resources for sharing risks and dealing with peaks of workload without the need of overprovision resources. Examples of trusted entities are different companies from the same business cluster [17].

Clients may be classified according to other criteria. Many service providers classify their clients according to the QoS that they have purchased. For example, Spotify [5] is an online music provider that classifies its clients in three categories (*free*, *unlimited* and *premium*) according to their monthly fee. The higher the fee the more services and QoS: unlimited streaming hours, highest quality of sound, available downloads, etc. The provider must consider the purchased QoS when allocating the resources.

The usage of the resources by external users can affect the QoS of internal users if the SLAs do not reflect priorities between clients in terms of pricing or allocation of resources. This paper suggests applying Client Classification to keep high QoS to internal users or users with high QoS requirements. Client Classification considers the information about the users when giving them access to the resources and prioritizes some SLAs according to two criteria:

QoS that the users are willing to acquire: the higher the QoS the higher the price. This is the traditional classification of services in Utility Computing.

Affinity between the client and the provider: clients from the same company as the provider or from entities that have a privileged relationship with the provider can hire the services at better prices, better QoS, or any other privilege. This novel approach was devised with the success of Cluster and Grid Computing, in which organizations share part of their resources with users from other organizations. By prioritizing users to which there is high affinity, organizations can ensure that their internal users will have enough resources or QoS when there is a peak of external demand.

According to previous considerations, our contributions are:

1. Proposal of new approaches to perform Client Classification in pricing and SLA allocation policies.
2. Demonstration of the validity of the model through fine-grained experiments that demonstrate how a provider can reach its Business-Level Objectives

(BLO) without penalizing its internal users or external users with priority SLAs. The results are evaluated in terms of revenue and proportion of priority users in the system.

We propose policies to allocate SLAs by pursuing a main BLO: users differentiation according of their Affinity/QoS relationship with the provider. In addition, our model also considers the economic profit as secondary BLO when negotiating the SLAs: prices at peak hours are higher than prices at off-peak hours for all the users. In that way, costs are amortized faster and companies are encouraged to move part of their tasks (such as resource-intensive unattended batch executions) to hours with low demand, such as the late night.

The experiments have been performed with the Economically Enhanced Resource Manager (EERM) simulator [4]: a customizable Cloud market simulator that applies several Business policies and allows users to define new policies as JBoss Drools rules [3].

The remainder of this paper is structured as follows. After the discussion of the related work, Section 3 describes the scenario in which Client Classification is applied: its participants and some preliminary definitions. Section 4 introduces the proposed rules for Client Classification: their motivation and their concrete implementation. Next, Section 5 describes the simulation environment and shows the experimental results that demonstrate the validity of the rules. At the end, Section 6 describes the conclusions of this paper and states the future research.

2 Related Work

In this paper, we extend part our previous work in Negotiation Models [16] and Rule-Based SLA Management for maximizing BLOs [15]. Previous work introduced several policies for maximizing the revenue of providers in Cloud Computing Markets [9,14]: dynamic pricing, overselling of resources, dynamic scaling of resources, migration of Virtual Machines (VMs), etc. This paper introduces rules that are essentially similar, but focused in Client Classification from the provider side.

Many previous works classify SLAs by considering the client information. The innovation of this paper relies on the proposal of new rules for price discrimination and client-aware overselling of resources, and their exhaustive evaluation in terms of revenue, client affinity, QoS, and SLA fulfillment. In addition, whilst related works tend to classify users in function of their internal/external condition, this paper defines them in a continuous range between 0 (lowest preference) to 1 (highest preference).

Client Classification is a usual practice in many businesses, such as banking services [2]. These businesses categorize clients in function of their size, budget, etc. and establish policies that define clearly the priorities of the clients, their protection level, their assigned resources, Quality of Service, etc. In Cloud Computing, Amazon Elastic Computing Cloud (EC2) provides a set of predefined VM instances [1], each one with different performance profiles (CPU load, Memory, etc.), but a fixed Quality of Service: they promise that their machines have

an annual availability of 99.5%. This approach may be economically suitable for huge resource providers, but not for smaller providers. With this paradigm, small providers should overprovision resources for minimizing risks and provide high availability. We try to channel the risk to the SLAs with the lowest priority according to the defined BLOs. In case of SLA violation, the Clients will receive an economic compensation proportional to the seriousness of the violation.

Previous papers introduced some policies similar to those introduced in this paper. Sulistio et al. [22] propose overbooking strategies for mitigating the effects of cancellations and no-shows for increasing the revenue. The overbooking policies used in this paper consider in addition the possibility of under-usage of the reserved resources of the client. Dube et al. [10] establish different ranges of prices for the same resource and analyze an optimization model for a small number of price classes. Their proposal is similar to our proposal about establishing Gold, Silver and Bronze ranges and optimizing their QoS performance giving priority to the contracts that report the highest economic profit. We extend this work by combining the QoS ranges with several other policies, such as Price Discrimination. Another main difference between this paper and the work from Sulistio et al. [22] and Dube et al. [10] is that the main BLO of our work is the Client Classification instead of the Maximization of the Revenue.

Püschel et al. [18] propose a scheme for Client Classification by means of price discrimination, different priorities in job acceptance and differentiation in Quality of Service. They adopt the architecture of an EERM. The EERM supports the optimization of SLA Negotiation and Management by dealing with both economic and technical information of Cloud Computing Markets. In addition, this paper extends the research of Püschel et al. [18] in Client Classification with the extension and detail of the policies, and deeper validation of them by means of a tailored simulation of Clients, Cloud Market, EERM, and Resource Fabrics.

3 Preliminary Definitions

A Cloud Market has two main actors: Clients and Providers. Clients try to buy resources in the Market to host their services, by sending offers to providers to start a negotiation. Each provider owns a set of N physical machines. Each physical machine can host several VMs that execute single tasks, such as Web Services or Batch Jobs. The QoS terms of a task are described in $SLA = \{Rev(vt), C, \overrightarrow{S}, \Delta t\}$:

- $Rev(vt)$ is a revenue function that describes how much money the provider earns after finishing correctly or incorrectly a task. vt is the amount of time in which the provider has not provided the agreed QoS to the client. Let MP be the Maximum Penalty (can be seen as negative revenue: lower MP implies higher penalties), MR the Maximum Revenue, MPT the Maximum Penalty Threshold, and MRT the Maximum Revenue Threshold, Equation 1 describes the revenue function. If $vt < MRT$ the SLA is not violated

(0 violations); if $vt > MPT$, the SLA is completely violated (1 violations). $MPT > vt > MRT$ implies a partial violation ($\frac{vt-MRT}{MPT-MRT}$ violations).

$$Rev(vt) = \frac{MP - MR}{MPT - MRT}(vt - MRT) + MR \quad (1)$$

This equation allows a grace period where the provider can violate the SLA without being penalized. When vt surpasses the MRT threshold, the revenue linearly decreases (see Figure 1) in function of vt. The Maximum Penalty MP is defined for avoiding infinite penalties. Client and provider can negotiate the values of MRT, MR, MPT, MP for establishing different QoS ranges for the clients, which report different revenues and penalties for the providers [16].

- C is the client information. Let id be the client identifier and $\overrightarrow{CD}$ a vector that handles the description of the client, then $C = \{id, \overrightarrow{CD}\}$. The information contained in $\overrightarrow{CD}$ must be decided by the System Administrator and applied consequently in the policies.
- $\overrightarrow{S}$ describes the QoS of the purchased service: throughput, response time, and so on.
- Δt is the time period requested to allocate the task.

The revenue function $Rev(vt)$ (as well as all the revenue figures in the evaluation) subtracts the penalties from the incomes, so it indicates how profitable is the allocation and execution of a SLA with a given set of policies. However, it does not indicate the provider's net benefit because it does not consider other costs, such as infrastructure maintenance.

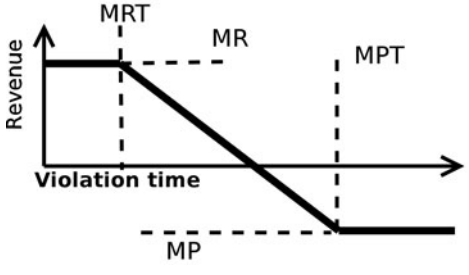


Fig. 1. Revenue of a SLA in function of the violation time (Equation 1)

3.1 Client Classification Criteria

We propose the classification of clients according to the **priority** that the provider assigns to them. This priority can be described using two different criteria:

Client Affinity: The affinity (aff $\subseteq [0, 1]$) measures how the client is related to the provider. For example, aff $= 1$ for a completely internal user; aff $= 0.25 \sim 0.75$ for a client from a company with privileged relationship with the provider

(e.g. in the same business cluster); aff = 0 for a completely external client. The calculation of the affinity may be different among different providers, depending on their business goals. How affinity is calculated is not important in this paper: the main topic is how to discriminate clients in function of their affinity.

Quality of Service: The same Cloud provider could host critical tasks and tasks that can tolerate lower QoS. For example, e-commerce applications may need extra QoS guarantees to avoid losing money on service unavailability. It is reasonable to allow critical clients to buy extra QoS guarantees at higher prices, and keep cheap prices (but fewer QoS guarantees) for non-critical tasks. The different ranges of QoS are defined by establishing different values for MRT, MR, MPT and MP in $Rev(vt)$ (Equation 1). We define three ranges of QoS, in descending order: Gold, Silver, and Bronze. The higher the QoS range, the higher MR and the lower MP, MRT and MPT (lower values of these three values imply higher penalties).

The policies for Client Classification are applied when the SLAs are negotiated between client and provider and allocated by the provider: the EERM gives priority to users to which the provider has high affinity when providing access to the resources by applying policies for Price Discrimination and Overselling of Resources.

4 Applying Client Classification in Negotiation Time

To facilitate the reading of this paper, the names of the policies have been abbreviated according to the next notation: $PolicyName^{PriorityType}$. $PolicyName$ is an abbreviation of the policy name. The abbreviations of all the policies are shown below, enclosed in parentheses next to their names. $PriorityType$ is an abbreviation of the magnitude that is used for calculating the priority of the client: the affinity (Aff) or the Quality of Service (QoS). When the policies for Client Classification are compared with policies that prioritize the maximization of the revenue, the abbreviation for this last priority is RM (Revenue Maximization). As example, Price Discrimination policies that apply discount to clients according to their affinity are notated as $PrDsc^{Aff}$.

The proposed policies are:

Price discrimination (PrDsc): The price of a task varies in function of the time slot, the workload of the resources of the provider, and the amount of resources required for providing the agreed QoS [16]. In addition, we propose to apply discounts to clients proportionally to their affinity.

Overselling of Resources (Ovrs): Clients do not always use all the resources that they buy. In consequence, the spare resources are resold to other clients according to their priority. This policy will increase both the revenue of the provider and the average priority of the clients in the system.

4.1 Price Discrimination (PrDsc)

In our previous works, providers dynamically establish the prices for maximizing their revenue. They ask for high prices when the workload is high (peak hours)

and low prices when it is low (off-peak hours) [16,15]. This maximizes the profit by attracting clients when the system is idle and maximizing prices when the demand is high.

$PrDsc^{Aff}$ policy is built on top of the $PrDsc^{RM}$ policy: after calculating the best resource allocation for maximizing the economic profit according to the Dynamic Pricing policies introduced in our previous works ($PrDsc^{RM}$) [16,15], the calculated revenue is multiplied by $(1 - affinity)$. This allows users with some affinity to receive a discount that is proportional to their affinity. This policy combines Client Classification with Revenue Maximization as a secondary BLO and always considers affinity as the main priority. Multiplying price by $(1-affinity)$ will linearly prioritize users (a user whose affinity is 1 will have the double of priority than a user whose affinity is 0.5). However, other distributions such as $(1 - affinity^2)$ could be considered in function of the provider policies.

The $PrDsc^{QoS}$ policy is not considered because it would not have sense: Gold tasks must not be cheaper than Silver tasks, and Silver tasks must not be cheaper than Bronze tasks.

4.2 Overselling of Resources (Ovrs)

Sometimes the clients do not use all the capacity that they have reserved because they tend to slightly overprovision the required computing resources that they finally use. Thanks to Cloud Computing elasticity mechanisms, the overprovisioning required by clients is very low [12]. However, the summation of the spare resources of all the clients may be sold to other clients to increase the resources usage.

We propose the sale of capacity that has been sold previously but the client is not using: when a client negotiates a SLA and there are not enough resources to allocate it, the scoring function in Equation 2 is calculated over the set $j = \{1 \dots N\}$ of N physical machines. The physical resource j with the highest positive score is selected as candidate for executing the task and the $PrDsc$ policy is triggered for establishing a price. If there are not physical resources whose score is positive, the job is rejected.

$$score_j = 1 - \frac{\int_{t_i}^{t_f} R'_{used}(t) + R_{req}(t)\, dt}{\int_{t_i}^{t_f} R_j(t)\, dt} \tag{2}$$

The terms of Equation 2 are described herewith:

- $R_{req}(t)$ is a constant function that represents the amount of bottleneck resources requested in the SLA under negotiation.
- $R_j(t)$ is a constant function that represents the amount of bottleneck resources in the physical resource j.
- Let $R_{used}(t)$ be a prediction of the bottleneck resources that the SLA under negotiation will use; let $\delta \subset [0, 1]$ be the maximum percentage to penalize or unpenalize the predicted workload in function of the client priority P. The priority-corrected prediction is defined as $R'_{used}(t) = (1 + \delta - 2\delta P)R_{used}(t)$.

The prediction of the used resources is artificially increased when the priority of the client that negotiates the SLA is low and artificially decreased when the priority is high.

The amount of resources used by services at a given time can be obtained from monitoring information. The prediction of resource usage for a given service ($R_{used}(t)$) can be calculated statistically or by several Machine Learning algorithms [20] from the monitoring information. In this paper, we use the CPU usage from Resource Monitoring because it is the bottleneck resource for the majority of services to be executed in the Cloud Provider. Equation 2 is abstract enough to allow using any other type of resource, such as memory or network bandwidth.

The corrections made in $R'_{used}(t)$ will motivate a higher acceptance of clients to which the provider has high affinity. As example, $\delta = 0.4$ in the experiments. This value is only chosen for showing the tendency of the graphs. Higher values of δ would decrease the revenue and increase both the average affinity and the number of SLA violations. Lower values of δ would have the opposite effect.

5 Experimental Results

This section describes the experimental environment and its configuration values. We have used the EERM Simulator [4] to execute and evaluate the policies that are introduced in this paper. The EERM Simulator is a fine-grained Cloud Market simulator, which simulates the complete cycle of a Cloud Resource sale and execution: services discovery, SLA negotiation process between provider and client, execution of web services or batch jobs and monitoring of the resources. It supports many features of Cloud Computing, such as elasticity of resources or migration of VMs. In addition, it integrates the Drools [3] Rule Engine to configure the SLA allocation and management policies in function of the BLOs (e.g. the Client Classification policies described in this paper).

The advantages of using a simulation environment instead of real machines is the possibility of generate more data with fewer resources in less time, so the evaluation is more accurate. For every experiment, a total of 64 CPUs working during a week have been simulated using real web workloads to acquire statistically representative data.

5.1 Simulation Environment

The constant values and the parameters of the simulation described are arbitrary because there are no real market traces to extract data from. Different real market scenarios could require different values, but the contribution of this paper is to show how Client Classification reports benefit **qualitatively** but not quantitatively. In other words, the paper shows how a given policy can improve the average affinity of the clients that use the system but not whether its values are optimum, because they would vary in function of the market status.

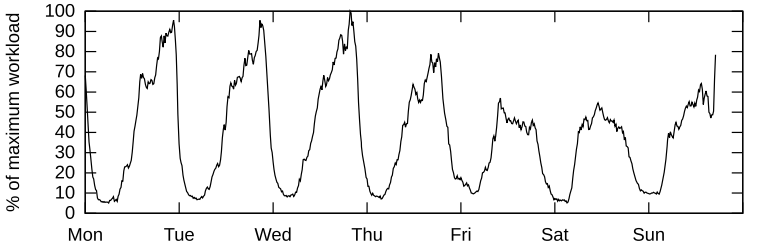


Fig. 2. Sample pattern of web workload

In our future work, the provider will automatically adjust its parameters for self-adapting to changing market environments.

A Cloud Market has two main actors: Clients and Providers. Providers offer VMs of variable sizes. Clients try to buy resources in the Market to host their services, by sending offers that contain $\{QoS, C, \overrightarrow{S}, \Delta t\}$, in which $QoS = \{Gold, Silver, Bronze\}$. For the same task in equal time and load conditions, the maximum price that the client is willing to pay for Gold QoS is 50% higher than the one for Silver QoS, and the maximum price that the client is willing to pay for Silver QoS is 20% higher than the one for Bronze QoS.

The Web workload is acquired from a real anonymous ISP (see figure 2), and varies in function of the hour of the day and the day of the week [7].

When the offer is in the market, the providers that accept it return a revenue function $Rev(vt)$, which specifies the prices and penalties to pay for the execution of that service. Finally, the client chooses the provider with a best price and time schedule for its interests and sends him a confirmation.

When a provider checks the offer from the client, it applies Machine Learning techniques to predict future workloads and verify whether the offered job can be executed correctly [20,21]. A bad prediction might entail a violation of the SLA.

In all the simulations, four different Cloud providers sell their services in a market during a week. For each experiment there is:

1. A provider that executes all the introduced policies until that subsection. It prioritizes users to which there is high affinity.
2. Same as Provider 1, but prioritizing tasks with high QoS.
3. A provider that executes all the introduced policies until the previous subsection. It prioritizes users to which the provider has high affinity in experiments that compare it with Provider 1 or tasks with high QoS in experiments that compare it with Provider 2. In the first section, it does not execute any policy and uses a fixed pricing schema as current Cloud providers [1,6].
4. A provider that executes the same policies as Providers 1 and 2 but without client classification as a main BLO. Its priority is the maximization of the economic profit [15].

Every provider belongs to a different organization. All of them have the same number of resources: two 8-CPU physical machines. Every provider has an affin-

ity higher than 0 to the 25% of the clients in the market, and equal to 0 to the other 75% of clients. The affinity of the clients of the same organization than the provider ranges from 0 (non-inclusive) to 1 (inclusive) following a uniform distribution. Summarizing, the average affinity of all the clients is ~ 0.21 for every provider.

Each client asks for Gold, Silver or Bronze QoS, independently of their organization. 1/6 of the clients ask for Gold QoS, 2/6 ask for Silver QoS, and 3/6 ask for Bronze QoS.

It is important to evaluate how the providers behave and how effective the policies are in different scenarios. For example, if there are many providers and few clients, the prices and the load of the system will be low; if there are too many clients and the providers cannot host all of them, prices and the system workload will be high. To evaluate the policies in all the scenarios, the experiments are repeated with different offer/demand ratios for each policy, gradually from low to high demand.

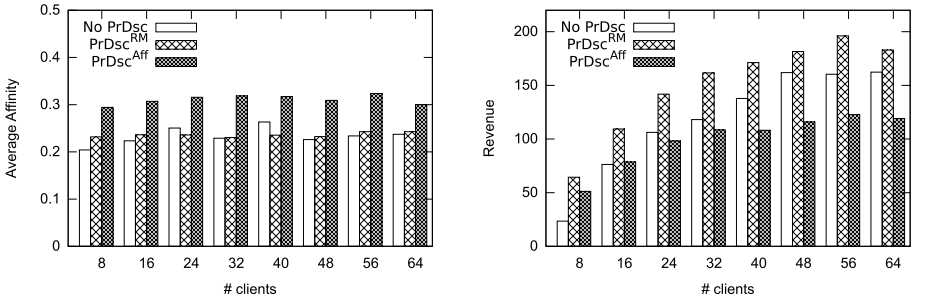
5.2 Price Discrimination (PrDsc)

Figure 3a compares the average affinity of the clients that buy services in providers that are competing in the market (see Section 5). Every provider has different policies for Price Discrimination: $NoPrDsc$ policy, $PrDsc^{RM}$, and $PrDsc^{Aff}$. The x axis shows the number of clients in each experiment, and the y axis represents the average affinity of the clients that used each resource. Each column group represents the obtained results of the providers in different experiments. The figure shows that the provider that implements $PrDsc^{Aff}$ increments the average affinity of its clients by 50%. The average affinity of clients in providers without $PrDsc^{Aff}$ is almost the same as the average affinity of all the clients in the market (~ 0.21).

Figure 3b is structured similarly to Figure 3a, but instead of showing the average affinity for each provider in each experiment, it shows the revenue of the providers (y axis). It shows that revenue is noticeably decreased if compared with fixed-pricing and revenue maximization providers. It is demonstrated that the increment of the average affinity of the clients of $PrDsc^{Aff}$ penalizes the revenue. To compensate the impact in revenue of $PrDsc^{Aff}$, hardware resources may be oversold as explained in next section.

5.3 Resources Overselling (Ovrs)

Figure 4a has a similar structure to Figure 3a: it shows the average affinity of the clients according to the policy combination in the provider. The three providers apply $PrDsc^{Aff}$ but they differ in how they implement Overselling. The provider of the previous section is labeled $NoOvrs$, because it does not apply Overselling. To compare the usefulness of overselling based on affinity discrimination ($Ovrs^{Aff}$), the figure also includes the results of a provider that performs $PrDsc^{Aff}$, but its overselling policy is driven by revenue instead of affinity (labeled as $Ovrs^{RM}$). As the intention of $Ovrs^{QoS}$ is not to attract

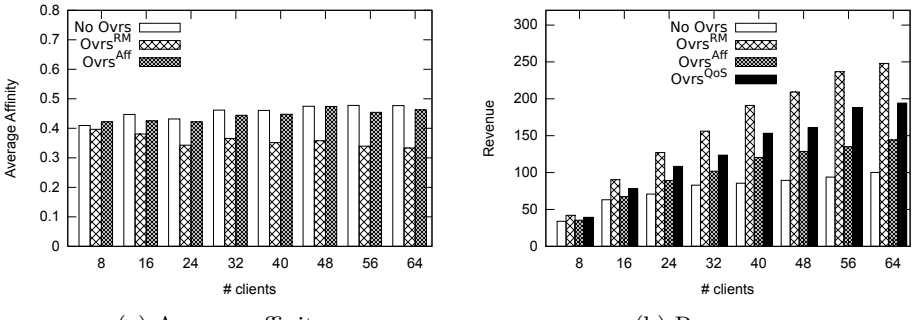


(a) Average affinity (b) Revenue

Fig. 3. Average affinity and revenue when using different $PrDsc$ policies

clients to which the provider has high affinity, this policy is not included in the figure. Figure 4a shows that not considering the client affinity in the overselling policy decreases average affinity of the clients. It is not caused by any type of penalization, but it is a statistical fact: more clients enter the system, regardless their affinity. $Ovrs^{Aff}$ maintains similar affinity levels to those of $NoOvrs$ but increasing the revenue of the provider, as in Figure 4b.

Figure 4b compares the revenue of four providers. All four implement $PrDsc^{Aff}$, but different overselling policies. The provider labeled as $NoOvrs$ does not apply any overselling policy, as in previous section. The other providers apply overselling policies based on Revenue Maximization ($Ovrs^{RM}$), affinity discrimination ($Ovrs^{Aff}$), and QoS range ($Ovrs^{QoS}$). Figure 4b shows that all the overselling policies have a positive impact on earnings. The provider labeled as $NoOvrs$ is the lower bound and the provider labeled as $Ovrs^{RM}$ is the higher bound. $Ovrs^{Aff}$ and $Ovrs^{QoS}$ stay in the middle of both: the clients are classified without renouncing the revenue completely. The revenue with $Ovrs^{Aff}$ is lower than the revenue with $Ovrs^{QoS}$ because $Ovrs^{QoS}$ prioritizes Gold and Silver contracts, which report more revenue than Bronze ones.



(a) Average affinity (b) Revenue

Fig. 4. Average affinity and revenue when using different $Ovrs$ policies

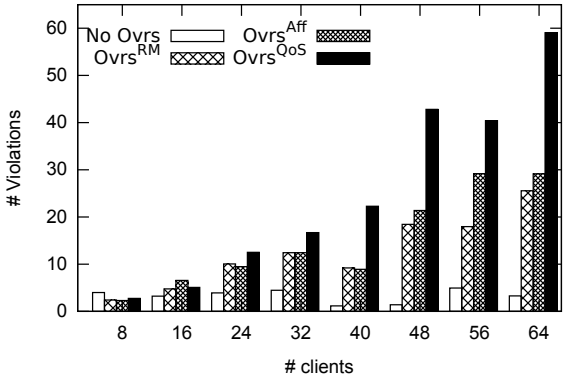


Fig. 5. Number of violations when using $Ovrs$ policies

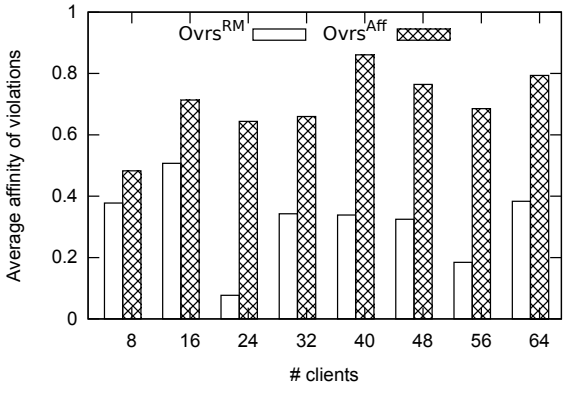


Fig. 6. Average affinity of the violations when using $Ovrs^{Aff}$

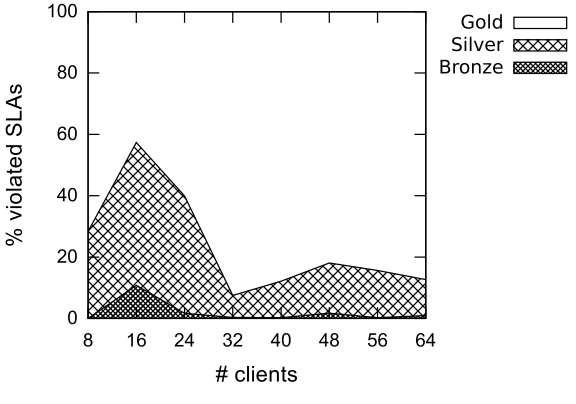


Fig. 7. Proportion of violations by QoS range when using $Ovrs^{QoS}$

However, overselling considerably increases the number of SLA violations (Figure 5) because of two reasons: the associated error to the predictor component, and the permissiveness with clients to which the provider has high affinity in terms of workload that makes the provider to violate a highest proportion of SLAs of this kind of clients (Figure 6). The provider that applies $Ovrs^{QoS}$ reports the highest number of violations because Gold and Silver SLAs have stricter requirements that are more difficult to accomplish, as shown in Figure 7. Despite the increase in the number of violations, the proportion of violated SLAs over the total of allocated SLAs remains below 2% in the worst case. Figure 7 is a stacked chart that shows the percentage of each QoS range from the total of violations for $Ovrs^{QoS}$ provider in several market simulations with different number of clients. It shows that the higher the QoS rank, the higher the percentage of violated SLAs. More violations of high-QoS SLAs do not mean that the QoS for Gold SLAs is lower than the QoS for Silver SLAs: it is more difficult to achieve the QoS requirements of Gold SLAs because the QoS requirements are higher, but an achievement of 90% of the QoS for Gold is still higher than the 100% of the QoS for Silver or Bronze.

The negative effects of $Ovrs^{Aff}$ and $Ovrs^{QoS}$ can be minimized by applying policies for runtime management of resources from our previous work [15]: usage of VMs elasticity, selective SLA violation and live migration of VMs for dynamically reconfiguring the Cloud data centers and minimize the number of violations.

6 Conclusions and Future Work

In this paper, we have introduced a set of policies for managing SLAs in a Cloud provider considering the classification of clients. Two facets can be used to classify the clients: client affinity and QoS. These policies have been evaluated through experiments that show the improvement of adding each policy to the set of previously introduced policies. We have introduced $PrDsc$ and $Ovrs$ for increasing the proportion of high-priority SLAs in the provider. After the application of all the policies, the EERM increases the number of priority clients (high-affinity or Gold and Silver, depending on the chosen type of priority), keeping a reasonable compromise between giving access preference to priority users and keep a high revenue from non-priority users.

We conclude that Client Classification policies in SLA negotiation achieve their objectives: the percentage of priority users is increased when applying them in negotiation time. Classification by QoS is suitable for a pure Cloud provider whose business is only based on selling its resources (it does not use them for its internal applications). Classification by affinity is more suitable for organizations that mix internal and external applications on their resources.

The policies presented in this paper rely on some constant values that may not lead to the optimum achievement of the BLOs. However, getting the optimum results is not the main objective in this paper. The key value of this work is to show the tendencies of applying the explained policies in terms of increment of

high-priority clients that use the system. The aim to further improve brings a research opportunity for future work: adding dynamism to rules to allow them self-adapting at runtime. Dynamic Rules will allow providers to autonomously adapt to the changes in the environment and achieve the optimum results according to their own BLOs.

Applying the policies enhances client classification but also increases the number of SLA violations and the proportion of violations of high-priority SLAs. Since the violations percentage is acceptable (below 2% of the allocated SLAs), we can state that the advantages of applying $PrDsc$ and $Ovrs$ outweight the disadvantages for achieving the BLO for which both policies have been designed.

Our previous work in Rule-based SLA Management [15] introduced several policies for mitigating the negative effects of $Ovrs$ policies. Our future work will enhance existent policies for resources elasticity, selective SLA violation and live migration of VMs, by adding them client awareness to allow highest achievement of Client Classification BLOs.

Acknowledgements. This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625, by the Generalitat de Catalunya under contract 2009-SGR-980, and by the European Commission under FP7-ICT-2009-5 contract 257115 (OPTIMIS).

References

1. Amazon EC2 instances, `http://aws.amazon.com/ec2/instance-types/` (last visit: February 2011)
2. Client classification and reclassification policy of rabobank polska sa, `http://goo.gl/AKu86` (last visit: February 2011)
3. Drools rule engine, `http://www.jboss.org/drools`
4. Economically Enhanced Resource Manager, `http://www.sf.net/projects/eerm` (last visit: August 2011)
5. Spotify, `http://www.spotify.com`
6. Windows azure, `http://www.microsoft.com/windowsazure/` (last visit: February 2011)
7. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 1998/PERFORMANCE 1998), vol. 26, pp. 151–160. ACM Press, Madison (1998)
8. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: 10th IEEE Intl. Conf. on High Performance Computing and Communications (HPCC 2008), pp. 5–13. IEEE Computer Society, Dalian (2008)
9. Choong, H.L., Jinwoo, S., Kyoungmin, P.: Grid and p2p economics and market models. In: I.C. Society (ed.) 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004), Seoul, South Korea, pp. 3–18 (April 2004)
10. Dube, P., Hayel, Y., Wynter, L.: Yield management for IT resources on demand: analysis and validation of a new paradigm for managing computing centres. Journal of Revenue and Pricing Management 4(1), 24–38 (2005)

11. Foster, I.: The anatomy of the grid: Enabling scalable virtual organizations. In: IEEE International Symposium on Cluster Computing and the Grid, p. 6 (2001)
12. Goiri, Í., Julià, F., Fitó, J.O., Macías, M., Guitart, J.: Resource-Level QoS Metric for CPU-Based Guarantees in Cloud Providers. In: Altmann, J., Rana, O.F. (eds.) GECON 2010. LNCS, vol. 6296, pp. 34–47. Springer, Heidelberg (2010)
13. Kaufman, L.M.: Data security in the world of cloud computing. IEEE Security and Privacy 7, 61–64 (2009)
14. Lee, H.Y., Choo, T.T., Khee-Erng, J.L., Wong, W.: A grid market framework. In: 3rd International Workshop on Grid Economics and Business Models (GECON 2006), Singapore, pp. 70–79 (May 2006)
15. Macias, M., Fito, O., Guitart, J.: Rule-based sla management for revenue maximisation in cloud computing markets. In: 2010 Intl. Conf. of Network and Service Management (CNSM 2010), Niagara Falls, Canada, pp. 354–357 (October 2010)
16. Macias, M., Guitart, J.: Using resource-level information into nonadditive negotiation models for cloud market environments. In: 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010), Osaka, Japan, pp. 325–332 (April 2010)
17. Porter, M.E.: Clusters and the new economics of competition. Harvard Business Review 76(6), 77–90 (1998)
18. Püschel, T., Borissov, N., Macías, M., Neumann, D., Guitart, J., Torres, J.: Economically Enhanced Resource Management for Internet Service Utilities. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 335–348. Springer, Heidelberg (2007)
19. Rappa, M.A.: The utility business model and the future of computing services. IBM Syst. J. 43(1), 32–42 (2004)
20. Reig, G., Alonso, J., Guitart, J.: Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In: 9th IEEE Intl. Symp. on Network Computing and Applications, Cambridge, MA, USA, pp. 162–167 (July 2010)
21. Sandholm, T., Lai, K.: Evaluating demand prediction techniques for computational markets. In: 3rd International Workshop on Grid Economics and Business Models (GECON 2006), Singapore, pp. 3–13 (May 2006)
22. Sulistio, A., Kim, K.H., Buyya, R.: Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. In: Intl. Symp. on Cluster Computing and the Grid (CCGRID 2008), pp. 267–276. IEEE Computer Society, Lyon (2008)

Budget-Deadline Constrained Workflow Planning for Admission Control in Market-Oriented Environments

Wei Zheng[1] and Rizos Sakellariou[2]

[1] School of Information Science and Technology, Xiamen University, China
[2] School of Computer Science, University of Manchester, UK

Abstract. There is an increasing interest for distributed computing technologies to be delivered through a market-based paradigm, which allows consumers to make use of and pay for services that meet certain Quality of Service requirements. In turn, providers receive income for successful provision of these services. In this paper, we assume an environment with multiple, heterogeneous resources, which provide services of different capabilities and of a different cost. Users want to make use of these services to execute a workflow application, within a certain deadline and budget. The problem considered in this paper is to find a plan for admission control. This allows providers to agree on constraints set by the user and allocate services for the execution of a workflow so that both deadline and budget constraints are met while account is also taken of the existing load (confirmed reservations) in the environment and the planning costs. A novel heuristic is proposed and evaluated using simulation with four different real-world workflow applications.

Keywords: Market-Oriented Computing, Workflow Execution, Workflow Planning, SLA.

1 Introduction

In market-oriented environments, such as grid or cloud platforms where resource owners provide services of different capacities and of different prices [15], users may want to use these services to execute complex applications, such as workflows [3]. Typically, a user may require his/her workflow application to complete within a certain deadline and budget; such requirements are generally recognised as Quality of Service (QoS) requirements. In analogy to markets in the real world, a Service Level Agreement (SLA) [10], which acts as a bilateral contract between a user and a service provider, is usually specified to capture the user's QoS requirements and act as a guarantee of the expected QoS. However, to establish an SLA, the service provider must have a way of determining in advance if it is feasible to fulfil a user's request. From the service provider's point of view, this implies that there is a need to find a plan for the execution of every new workflow to see if both the budget and deadline constraints requested by

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 105–119, 2012.

the user can be met according to the current load of the service provider's resources. Such a plan is called a Budget-Deadline Constrained plan, or, in short, '*BDC-plan*'. The planning procedure is called '*BDC-planning*'. BDC-planning should be part of the *admission control* of a workflow request: if a BDC-plan is found, a user's request can be accepted and a relevant SLA can be agreed; otherwise, the user's request should be rejected.

BDC-planning is an important but also remarkably challenging problem for market-oriented environments. First, such a planning problem is NP-complete [12]. Second, the non-dedicated nature of resources imposes more difficulties as the contention for shared resources (due to other, already agreed workloads) needs to be considered during planning. This suggests that the planner may have to somehow query resources for their runtime information (e.g., the existing load) to make informed decisions. Moreover, at the same time, BDC-planning should be performed in *short time*, because: (i) users may require a real-time response, and (ii) the (runtime) information, on which a planning decision has been made, varies over time and, thus, a planning decision made using out-of-date information may not be valid any more.

Essentially, the problem considered in this paper boils down to bi-criteria DAG planning, as we assume that every workflow application is represented by a Directed Acyclic Graph (DAG). This problem involves the planning process to optimize two metrics at the same time to meet the specified constraints (budget and deadline). There have been quite a few bi-criteria DAG planning heuristics in the literature [12,25,26,16,5,20,19]. However, some of them do not take the existing load of resources into account (or adopting them to do so could be too costly). Moreover, most of these heuristics have sophisticated designs, such as guided random research or local search, which usually require considerably high planning costs. Such features do not make existing heuristics particularly suitable for the BDC-planning problem discussed above (as opposed to the problem of scheduling a workflow already admitted, in which case high-cost approaches could be justified). This motivates the work presented in this paper.

In this paper, a new BDC-planning heuristic is proposed with the objective to simultaneously provide effective BDC-planning and fast planning time. The proposed heuristic is based on the Heterogeneous Earliest Finish Time (HEFT) algorithm [21], which is a well-known list scheduling heuristic aiming at minimizing the overall execution time of a DAG application in a heterogeneous environment. While being powerful at optimizing makespan, the HEFT algorithm does not consider the monetary cost and budget constraint when making scheduling decisions. In this paper, the HEFT algorithm is extended in order to resolve the BDC-planning problem and the new algorithm is named the *Budget-constrained Heterogeneous Earliest Finish Time (BHEFT)*. In the experimental section of the paper, it is demonstrated that, for the BDC-planning problem, the proposed heuristic addresses well the aforementioned challenges. In addition, it performs at least as effectively as sophisticated heuristics, but costs much less in terms of computation and communication overheads.

In the rest of this paper, related work is reviewed in Section 2. The model assumed and a problem definition are presented in Section 3. A novel BDC-planning heuristic (BHEFT) is described in Section 4. Experimental details and simulation results are discussed in Section 5. The paper is concluded in Section 6.

2 Related Work

Admission control problems have been studied in various computing platforms where QoS is considered. Yeo and Buyya [23] investigated the advance impact of inaccurate runtime estimates for deadline constrained job admission control in clusters. Yin *et al.* [24] proposed a predictive admission control algorithm to support advance reservation in equipment grids. Admission control issues were also studied as a subproblem of resource management in grids which support SLAs [9,1]. Nevertheless, none of these works takes budget requirements from users into account; moreover, their targeted applications are not workflows.

Admission control for workflows in market-oriented grids requires bi-criteria DAG planning techniques. A grid capacity planning approach is presented in [17], which aims at producing a plan for a workflow without reservation conflicts to optimize resource utilization and multiple QoS constraints. However, this paper mainly focused on a 3-layer negotiation mechanism rather than a planning heuristic itself. The studies in [14,13] proposed mapping heuristics to meet deadline constraints, at the same time minimizing the reservation cost of workflows, but they regarded workflow tasks as being multiprogramming, something not commonly adopted in workflow scheduling studies [22]. Based on the model of Utility Grids, the time-cost constrained optimization has been studied for meta-scheduling [8,6,7] in which planning is considered at application-level, but applications are assumed to be independent rather than task-based and bounded by dependencies as is the case in workflow DAGs. Therefore, although they consider both time and cost constraints in planning, these techniques are not really applicable for admission control for workflows.

To resolve the multi-objective (time and cost, commonly) DAG planning problem, evolutionary techniques (e.g., genetic algorithms) have been widely used. Examples can be found in [25,26,20,19]. Although algorithms based on evolutionary techniques normally perform well on optimization, they also require significantly high planning costs and thus are naturally too time-consuming for BDC-planning.

There are also bi-criteria scheduling heuristics for workflow applications derived from local search and list scheduling techniques. Wieczorek *et al.* [12] propose a two-phase algorithm (DCA) to address the optimization problem with two independent generic criteria for workflows in Grid environments. The algorithm optimizes the primary criterion in the first phase, then optimizes the secondary criterion while keeping the primary one within the defined sliding constraint. In [16], two scheduling heuristics based on guided local optimization, LOSS and GAIN, were developed to adjust a schedule, and these may be generated by a time-optimized heuristic or a cost-optimized heuristic, to meet

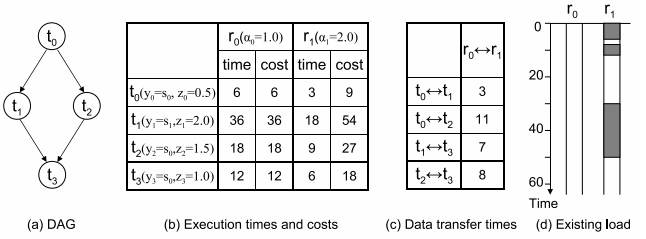
users' budget constraints. As an extension to the DLS algorithm [18], BDLS is presented in [5] which focuses on developing bi-criteria scheduling algorithms to achieve a trade-off between execution time and reliability. Based on local search, DCA and LOSS require a considerable number of repetitions to obtain a final result. As a list scheduling heuristic, BDLS may have low complexity. The main planning costs of BDLS arise from the computation of dynamic priorities when making scheduling decisions. The main issue with these heuristics is that they do not consider the existing load of resources in their assumptions, and thus tend to produce plans which may lead to reservation conflicts, i.e., given that one resource can only execute one task at a time, the planned task may overlap with the tasks of other workflows which have already been reserved. However, with an added communication phase between the planner and service providers (as mentioned in Section 3) and a slight change of algorithm design, these heuristics may be able to produce BDC-plans without reservation conflicts. In Section 5, these slightly-changed heuristics will be compared with BHEFT in terms of planning performance and overheads.

To the best of our knowledge, there is no previous study which attempts to address equally all four key elements of the problem at the same time, that is: (i) workflow planning for (ii) admission control of (iii) market-oriented environments while (iv) considering dynamically existing loads in non-dedicated resources. Unlike the aforementioned works which exhibit drawbacks according to the BDC-planning challenges mentioned in Section 1, BHEFT is a novel bi-criteria DAG planning heuristic proposed to address these challenges. By applying BHEFT, the planner of a market-oriented environment is enabled to effectively determine whether a workflow request should be accepted or not in a real-time manner so that the establishment of an SLA can be facilitated.

3 Problem Description

A workflow is modelled as a DAG consisting of a group of nodes and a set of directed edges. A node denotes a task $t_i, (1 \leq i \leq n)$, where n is the number of tasks. An edge represents a task dependency $t_i \rightarrow t_j$, where t_i is called a parent task of t_j and t_j a child task of t_i. A child task cannot be executed until all the input data depending on parent tasks have been received. Information associated with each task (t_i) are: the service type the task wants to use (y_i) and the task size (z_i). A workflow request is submitted with a budget B and a deadline D.

There is a group of service types $\mathcal{Y} = \{s_0, s_1, \cdots\}$, and a set of heterogeneous resources which are fully interconnected. A resource r_p may provide a set of service types $\mathcal{Y}_p \subseteq \mathcal{Y}$. Service instance $s_{x,p}$ exists if $s_x \in \mathcal{Y}_p$. Mapping a task to $s_{x,p}$ means allocating the task to resource r_p. Task t_i can be mapped to r_p for execution if and only if $y_i \in \mathcal{Y}_p$. Different service types may have different capacities with a different executing cost. For each service type s_x, a parameter β_x is given to depict its standard execution time, which is one of the factors to estimate the execution time of a task which uses this service type. Similarly,



	$r_0(\alpha_0=1.0)$		$r_1(\alpha_1=2.0)$	
	time	cost	time	cost
$t_0(y_0=s_0, z_0=0.5)$	6	6	3	9
$t_1(y_1=s_1, z_1=2.0)$	36	36	18	54
$t_2(y_2=s_0, z_2=1.5)$	18	18	9	27
$t_3(y_3=s_0, z_3=1.0)$	12	12	6	18

	$r_0 \leftrightarrow r_1$
$t_0 \leftrightarrow t_1$	3
$t_0 \leftrightarrow t_2$	11
$t_1 \leftrightarrow t_3$	7
$t_2 \leftrightarrow t_3$	8

Fig. 1. An example of program input

different resources may have different powers with different prices. For each resource r_p, a power ratio α_p is given to depict its power. The larger α_p is, the more powerful the resource will be. Thereby, for t_i, the execution time on r_p (i.e., $et_{i,p}$) is defined as $et_{i,p} = z_i \times \beta_x/\alpha_p$, and the execution cost $ec_{i,p} = \mu_p \times et_{i,p}$, where μ_p is the price unit for resource r_p; it is assumed $\mu_p = \alpha_p(1+\alpha_p)/2$. Also, the time to transmit data between two dependent tasks which are respectively mapped to specific (different) resources is given. Moreover, in a resource, confirmed reservations may exist. This is regarded as existing load denoted by the set of pairs $\mathcal{L} = \{(st_0, ft_0), \cdots (st_k, ft_k), \cdots\}$, where st means the start time of a reservation and ft the finish time. Here, it is assumed that only one service can run at a time on a resource. Thus, each reservation reserves a certain period of a whole resource for a task which wants to use a service instance provided by the resource.

All the above-mentioned types of input are illustrated with an example shown in Fig. 1, which includes a 4-node DAG and two resources. Every resource implements two service types s_0 and s_1, of which the standard execution time is given by $\beta_0 = 12$ and $\beta_1 = 18$. In Fig. 1(b), the parameters associated with each task and each resource are presented and used to compute execution time and cost on different resources. The data transfer times and existing load are respecitively depicted in Fig. 1(c) and Fig. 1(d).

The BDC-planning problem is to map every task onto a suitable service instance (i.e., a resource) and specify an appropriate start time for each mapped task so that the execution time and overall cost of the workflow is within D and B, respectively, and the produced plan does not overlap with existing reservations. With the same input, different heuristics may differ at deciding whether a BDC-plan can be obtained. The objective of a BDC-planning heuristic is to maximize the likelihood that a BDC-plan can be successfully found for a given workflow request.

It is worth mentioning that the planner has to communicate with resource owners to produce a plan without reservation conflict. We assume that the planner has to send a *Time Slot Query* (TSQ), i.e., ask for a certain length of time slot on a specific resource, and then the resource owner responds with the earliest availability. Here, the alternative of allowing the planner to retrieve all free time slots of resources is not considered, since the service providers may not want their workload, which may be commercially sensitive, to be exposed.

Input: DAG G with Budget B and Deadline D;
Output: A BDC-plan

1. Compute *rank* (as defined in Eq.(1)) for all tasks.
2. Sort all tasks in a planning list in the non-ascending order of *rank*.
3. **for** $k := 0$ to n **do** (where n is the number of tasks)
4. Select the kth task from the planning list.
5. Compute the Spare Application Budget for task k (as defined in Eq.(2)).
6. Compute the Current Task Budget for task k (as defined in Eq.(3)).
7. Construct the set of Affordable Services (as defined in Eq.(4)) for task k.
8. **for** each service which can be used by task k **do**
9. Compute the earliest finish time of mapping task k to the service using TSQ as described in Section 3.
10. **endfor**
11. Select a service for task k according to the defined selection rules.
12. **endfor**

Fig. 2. The BHEFT Heuristic

Let $\mathcal{L}_p$ be the existing load of resource r_p, we define TSQ in the form of $f_Q(t_i, r_p, dat_{i,p}, dur) = \min\{(a,b)|(a,b) \cap \mathcal{L}_p = \emptyset, a \geq dat_{i,p}, b = a + dur\}$, where $dat_{i,p}$ means the time all required data is available for task t_i on resource r_p, and dur denotes the required duration which is considered to be equal to the estimated execution time $et_{i,p}$. According to Fig. 1, $\mathcal{L}_1 = \{(0,6),(8,12),(30,50)\}$ and for task 0, $dat_{0,1} = 0$ and $et_{0,1} = 3$, then it holds that $f_Q(0,1,0,3) = (12,16)$.

With TSQ, there are two ways for a planning heuristic to avoid reservation conflicts. One is invoking TSQ every time when computing the estimated earliest finish time for a task on a resource. A planning heuristic, such as DCA [12] or LOSS [16], normally involves lots of such estimates, and thus may introduce heavy communication costs if using this approach. The other way is producing an initial plan without considering the existing reservations and then using TSQ to reallocate the time slot for each mapped tasks in the order that tasks are initially scheduled. In this case, the communication costs may be small but the performance of the heuristic may degrade.

4 The Proposed Heuristic

This section describes the details of BHEFT, of which the outline is shown in Figure 2. Similar to the original HEFT algorithm, the BHEFT also has two major phases: *task prioritizing* and *service selection*.

In the task prioritizing phase, the priorities of all tasks are computed using upward ranking which is the same as defined in HEFT. The rank of a task i is recursively defined by

$$rank_i = \overline{et}_i + \max_{j \in Succ(i)} \{\overline{dt}_{i,j} + rank_j\} \quad (1)$$

where $Succ(i)$ is the set of the child tasks of task i, $\overline{et}_i$ is the average execution time of task t_i, $\overline{dt}_{i,j}$ is the average data transfer time of edge $t_i \rightarrow t_j$. In the case of childless nodes, the rank equals to the average execution time.

In the service selection phase, the tasks are selected in order of priority. Each selected task is allocated to its "best possible" service, of which the metric may change according to an assessment of the spare budget which varies as planning proceeds. For this assessment, two variables are used: *Spare Application Budget* (SAB) and *Current Task Budget* (CTB). Suppose that the kth task is being allocated, SAB_k and CTB_k are respectively computed by

$$SAB_k = B - \sum_{i=0}^{k-1} c_i - \sum_{j=k}^{n-1} \overline{c}_j \tag{2}$$

$$CTB_k = \begin{cases} \overline{c}_k + SAB_k \times \overline{c}_k / \sum_{i=k}^{n-1} \overline{c}_i & : \quad SAB_k \geq 0 \\ \overline{c}_k & : \quad SAB_k < 0 \end{cases} \tag{3}$$

where B is the given budget, c_i is the reservation cost of the allocated task i, $\overline{c}_j$ is the average reservation cost of the unallocated task j over different resource mappings, n is the number of tasks. Provided that task t_k uses service type s_x, a set $\mathcal{S}_k^*$ is constructed consisting of an *affordable service* for task k, i.e.,

$$\mathcal{S}_k^* = \{s_{x,p} | \exists s_{x,p}, c_{k,p} \leq CTB_k\} \tag{4}$$

Then the "best possible" service is selected by the selection rules as follows:

1. If $\mathcal{S}_k^* \neq \emptyset$, the affordable service with the earliest finish time is selected;
2. If $\mathcal{S}_k^* = \emptyset$ and $SBA \geq 0$, the service with the earliest finish time selected;
3. If $\mathcal{S}_k^* = \emptyset$ and $SBA < 0$, the cheapest service is selected;

Using the example presented in Fig. 1 and assuming a budget $B = 89$ and a deadline $D = 60$, the planning results derived by using the above-described heuristic are shown in Fig. 3, where tasks are planned in the order t_0, t_1, t_2, t_3 and a BDC-plan is successfully obtained.

5 Performance Evaluation

5.1 Experimental Setting

To run the experiments, a job planner (broker) and a set of resources were simulated by java programs distributed on computing nodes with 3.0 Ghz CPU, 1 GB memory and connection through Gigabit Ethernet. The communication between the broker and the service providers was implemented by socket programming. The existing load of resources was also randomly generated for simulation. Given a specific period between time a and b, the existing load of each resource p (i.e., $\mathcal{L}_p$) is parameterized by two pre-specified values: Utilization Rate (UR) and Average Task Load (ATL). The former is the ratio of the total reserved time to the whole period, and the latter is the ratio of the number of tasks appearing during a certain period to the length of this period. Then, the average duration

	Avg. Time $\overline{et_i}$	Avg. Cost $\overline{c_i}$	Rank	SAB	CTB	S*	Selected Service	Start Time	Finish Time	Actual Cost c_i
t_0	4.5	7.5	50.5	-1	7.5	ϕ	$\{s_{0,0}\}$	0	6	6
t_1	27	45	43	0.5	45.27	$\{s_{1,0}\}$	$\{s_{1,0}\}$	6	42	36
t_2	13.5	22.5	30.5	9.5	28.2	$\{s_{0,0}, s_{0,1}\}$	$\{s_{0,1}\}$	17	26	27
t_3	9	15	9	5	20	$\{s_{1,0}, s_{1,1}\}$	$\{s_{1,0}\}$	42	54	12
The given budget is 89, and Sum =										81

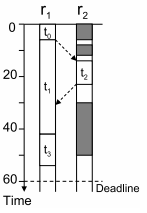


Fig. 3. Planning derived using BHEFT (Avg.=Average)

of a reservation slot is $\overline{RD} = UR/ATL$ and the average duration of an idle slot is $\overline{ID} = (1 - UR)/ATL$.

The following procedure describes how the existing load of resource p ($\mathcal{L}_p$) was constructed: (1) Set $\mathcal{L}_p = \emptyset$ and current time $CT = a$. (2) Randomly determine current state among RESERVED and IDLE. (3) If RESERVED: (3.a) randomly generate reserved duration RD by normal distribution with mean $\overline{RD}$ and standard deviation $\overline{RD}/2$ whereas only $RD > 0$ is adopted; (3.b) set $\mathcal{L}_p = \mathcal{L}_p \cup (CT, CT + RD)$; (3.c) set $CT = CT + RD$; (4.d) switch current state to IDLE. (4) If IDLE: (4.a) randomly generate idle duration ID by normal distribution with mean $\overline{ID}$ and standard deviation $\overline{ID}/2$ whereas only $ID > 0$ is adopted; (4.b) set $CT = CT + ID$; (4.c) switch current state to RESERVED. (5) Repeat Steps 3 and 4 till CT reaches b.

There were 2 service providers in the evaluation, each of which managed 3 resources, hence, there were 6 resources in total. There were 4 service types having a standard execution time of 10, 15, 25 and 30 respectively. For each resource p, the capability ratio α_p was randomly generated from the interval [0.5, 2.0]. The period considered for existing load modelling was [0, 5000].

Four types of DAGs, corresponding to real-world workflow applications, were considered in the experiments; these are: fMRI [27] with 17 nodes, Montage [2] with 34 nodes, AIRSN [11] with 53 nodes and LIGO [4] with 77 nodes. For each task i, y_i was randomly selected from the provided service types, and z_i was randomly generated from [0.5, 2.0]. The communication computation ratio (CCR) was randomly selected from [0.1, 1.0].

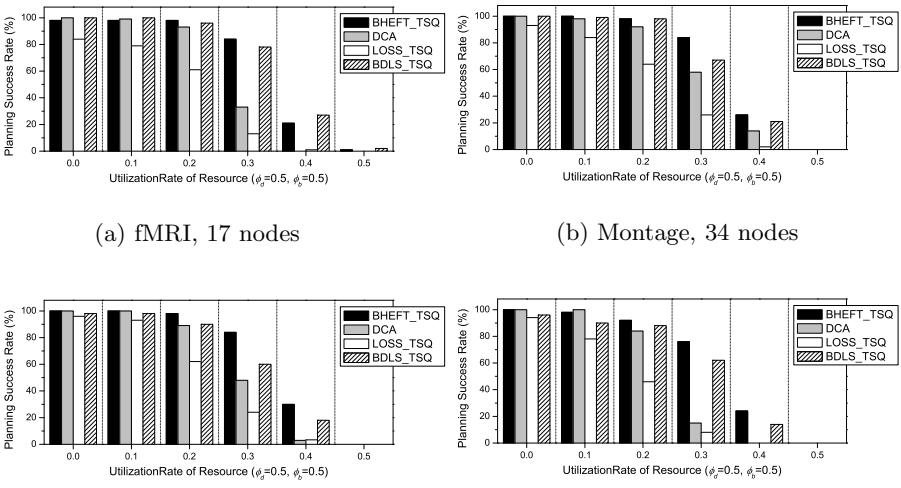
Given a DAG, constraints for reasonable values for deadline and budget were generated as follows. For simplicity, a job was always assumed to start at time 0. The makespan M_{HEFT} was computed by applying the HEFT algorithm [21] to the DAG without considering the existing load of resources. The deadline constraint DC was considered to be located between the lower bound $LB_{dc} = M_{HEFT}$ and the upper bound $UB_{dc} = 5 \times M_{HEFT}$. A deadline ratio ϕ_d was used to depict the position of DC by $DC = LB_{dc} + \phi_d \times (UB_{dc} - LB_{dc})$, where $0 \leq \phi_d \leq 1.0$. For budget constraint, LB_{bc} was the lowest total cost obtained by mapping each task to the cheapest service, and UP_{bc}, the highest total cost obtained conversely. Similarly, a budget ratio ϕ_b was used to specify the possible budget constraint $BC = LB_{bc} + \phi_b \times (UB_{bc} - LB_{bc})$, where $0 \leq \phi_b \leq 1.0$.

BHEFT was compared with DCA [12], LOSS [16] and BDLS [5] in the experiments. As mentioned in Section 3, some modification is needed to adapt these heuristics, which do not consider the existing load of resources, to produce a contention-free plan. According to the evaluation in [12], where existing loads on resources (and hence TSQ) are not considered, DCA, which is based on extensive local search, has the best optimization performance but the highest time overhead, as opposed to BDLS which is a static list scheduling heuristic using a dynamic priority. Therefore, TSQ was introduced into LOSS and BDLS only, while DCA was modified in the other way mentioned in Section 3 (that is, a plan is first generated without considering existing loads and, then, TSQ is used to reallocate the time slot allocated to each task to resolve reservation conflicts). When showing the experimental results in figures, the suffix '_TSQ' was added to the names of the algorithms which used TSQ, to distinguish them from DCA which does not consider TSQ, while the original names are used for short in the discussion. In terms of the configuration of DCA and BDLS, the same settings as used in [12] are adopted, i.e., LOSS3 in [16] is adopted to represent LOSS, a memorization table consisting of 100 cells with up to 10 intermediate solutions stored in each cell was used by DCA, and the parameter δ for BDLS was determined by a binary search with a maximum of 15 loop iterations. Moreover, all heuristics terminate immediately when a BDC-plan is found.

For each experiment, all of the parameters except for those which were given and fixed, were re-initialized at random with the above specifications. After a heuristic was run, if a BDC-plan was found, the planning succeeded, otherwise, a failure was reported. To analyze the performance of each heuristic, the experiment was repeated multiple times and the metric *Planning Success Rate* (PSR) was used, as defined below:

$$PSR = 100 \times \frac{\text{number of times for which a BDC-plan was found}}{\text{number of total repeated times of experiment}} \tag{5}$$

Four sets of experiments were carried out. In the first one, ϕ_d and ϕ_b were fixed to be 0.5, while UR was varied for each resource from 0.0 to 0.6 in the step of 0.1 with the corresponding $ATL = 0.05 \times UR$. The experiment was repeated 500 times to observe how the existing load of resources affected the PSR of each heuristic. In the second set of experiments, UR was randomly generated in the interval [0.1, 0.4], and the ATL was computed correspondingly. ϕ_d and ϕ_b were selected from the set {0.25, 0.5, 0.75} to form 9 combinations which covered a wide spectrum of diverse user requests; the experiment was then repeated 500 times for each combination. Thus, the value of PSR was investigated under various constraints (from tight to relaxed). In the third set of experiments, we studied the same 9 combinations for user requests but for three specific values of UR. Finally, in the fourth experiment, the average running time of each heuristic to do planning was measured. This experiment was repeated 100 times for each workflow with various combinations of constraints.



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

(c) AIRSN, 53 nodes

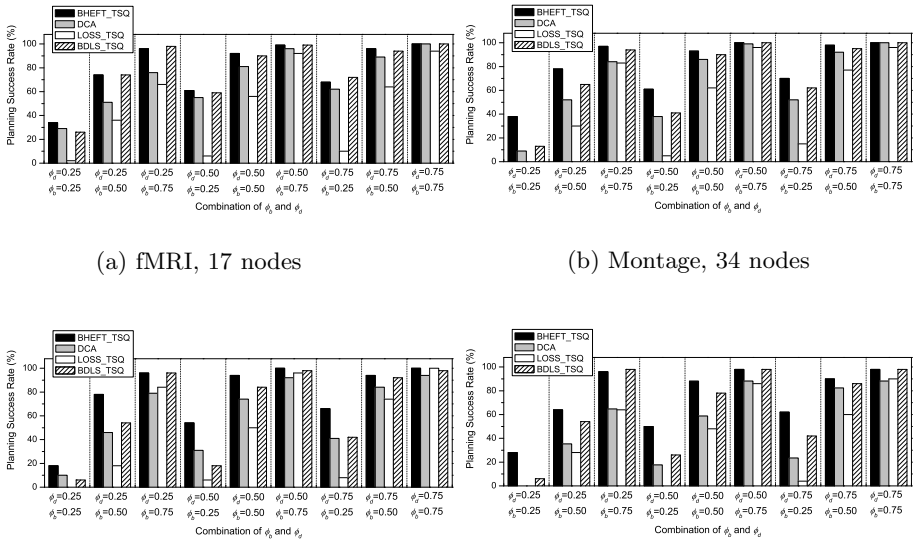
(d) LIGO, 77 nodes

Fig. 4. *PSR* with different utilization rate of resources

5.2 Experimental Results

First set of experiments: Figure 4 shows the results of the first set of experiments where the impact of the existing load of resources is investigated. Here, ϕ_d and ϕ_b are both fixed to be 0.5 to avoid unnecessary disturbance caused by setting the user constraints to be too tight or too relaxed. It can be seen from Figure 4 that the behaviour of the compared heuristics in terms of their *PSR* follows the same pattern regardless of the type of DAG. BHEFT almost always shows the best performance, with a notable exception in the case of fMRI, which could be attributed to its small number of nodes. As expected, all heuristics perform worse as *UR* increases. In such cases, the better performance exhibited by BHEFT is more profound in the graphs. Its performance is followed by BDLS, which appears to be second best outperforming LOSS and DCA. It is noted that this performance classification changes (or differences become less clear) when there is no existing load on resources (as also observed in [12] where LOSS seems to give better performance than BDLS).

Second set of experiments: In the second set of experiments, the performance of each heuristic was investigated under various circumstances of user constraints, from tight to relaxed. As already mentioned we considered nine combinations of different types of constraints. Figure 5 shows the value of *PSR* for different types of DAG and different budget-deadline constraints. The first observation is that when both the deadline constraint and the budget constraint are tight, for example, ϕ_d=0.25 and ϕ_b=0.25, all four heuristics obtain low *PSR*s; among them, BHEFT achieves the best *PSR* which is between 20% to 40%. When a small DAG (e.g., fMRI) is used, both DCA and BDLS obtain *PSR*s which are



(a) fMRI, 17 nodes

(b) Montage, 34 nodes

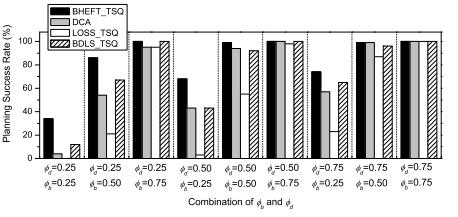
(c) AIRSN, 53 nodes

(d) LIGO, 77 nodes

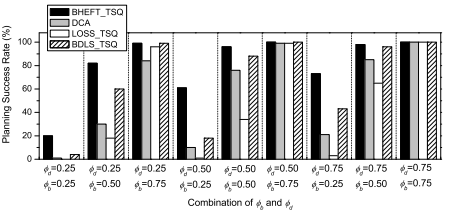
Fig. 5. *PSR* with different types of constraints

comparable to those of BHEFT. However, their *PSR*s turn significantly lower when a DAG such as LIGO is used. It is interesting to note that the performance of LOSS is particularly poor when the budget constraint is tight. This may be because the initial plan of LOSS, constructed using HEFT, usually has a small makespan regardless of the monetary cost; then, it may not be straightforward for LOSS to adjust the plan to meet the budget constraint with a limited number of local searches. BDLS can be almost as effective as BHEFT in many cases, for example, when both budget and deadline constraints are above 50%. In the case where a small DAG (e.g., fMRI) is used, BDLS can occasionally achieve a better *PSR* than BHEFT. However, overall, in most cases, BHEFT performs better than BDLS. The advantage of BHEFT is more profound when at least one of the constraints is tight and the used DAG has a large number of nodes.

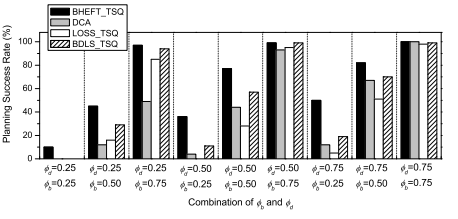
Third set of experiments: In order to consider the impact of the Utilization Rate in more detail, we studied the *PSR* for the nine different combinations of user constraints and three different values of utilization rate. The results, for two types of DAG, Montage and LIGO, are shown in Fig. 6. Once again, BHEFT performed the best among the competitive heuristics in most of the circumstances. The results highlight the impact that the existing load of resources may have on BDC-planning. As expected, when the Utilization Rate is low, that is, there is little existing load on resources, and the constraints for budget and deadline are relaxed (e.g., ϕ_d=0.75 and ϕ_b=0.75), all heuristics perform equally well.



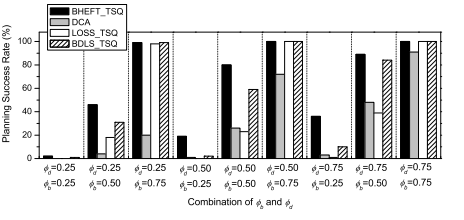
(a) Montage, UtilizationRate = 0.2



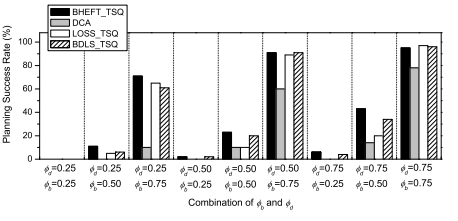
(b) LIGO, UtilizationRate = 0.2



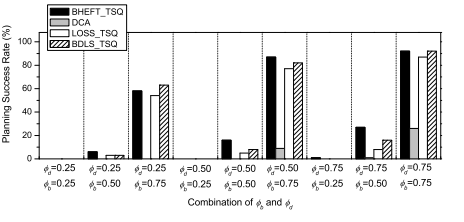
(c) Montage, UtilizationRate = 0.3



(d) LIGO, UtilizationRate = 0.3



(e) Montage, UtilizationRate = 0.4



(f) LIGO, UtilizationRate = 0.4

Fig. 6. *PSR* with different utilization rates and constraints for Montage and LIGO

Fourth experiment: In the fourth experiment, the execution time needed by each algorithm to obtain a planning result was studied. Figure 7 shows how the running time of each heuristic varies over diverse types of DAG and constraint settings. It is not surprising that, in most of the cases, LOSS has the highest time costs due to the overhead caused by numerous TSQs. It can be easily imagined that some other sophisticated algorithms, such as DCA or genetic algorithms, if using TSQ when scheduling, may need even more time compared to LOSS. Our results suggest that even LOSS is not scalable to large applications and too time-consuming for on-line workflow planning. Although not using TSQ, the DCA heuristic considered in the experiment still has an execution time comparable to BDLS, and this is significantly higher than BHEFT. The latter two algorithms are both based on list scheduling, whereas BHEFT needs evidently less running time than BDLS due to simpler computation and the fact that less communication is needed when making scheduling decisions. Moreover, BHEFT is the most scalable in terms of the growth of DAG size (and potentially the

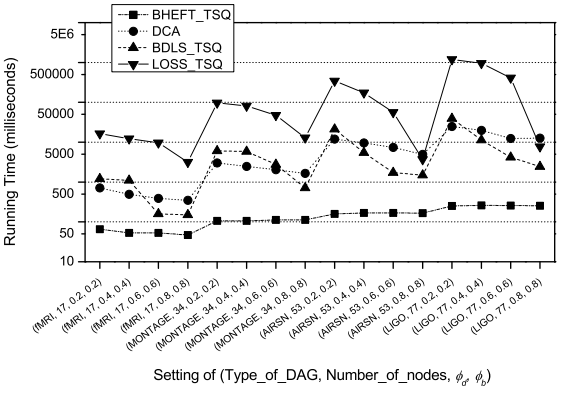


Fig. 7. Execution time for each heuristic with different DAGs and user constraints

number of resources which is considered constant in this experiment). As can be seen in the graph, when planning LIGO with 77 nodes on 6 resources, BHEFT only needs around 0.2 seconds on average. This suggests that BHEFT copes well with the real-time requirements of workflow planning.

Summary of observations: The results lead to the following observations:

- The existing load of resources may have significant impact on BDC-planning. Directly applying a heuristic not considering the existing load of resources in job planning (e.g., DCA) may result in a significant degradation of *PSR*. In contrast, BHEFT, which takes the existing load of resources into account, is able to achieve a significant improvement on the success rate of finding a BDC-plan which simultaneously satisfies deadline and budget constraints.
- Some guided local search heuristics (for example, LOSS) may be too sensitive regarding the existing load of resources and cannot perform reasonably well for BDC-planning, even when the existing load of resources is taking into account when making planning decisions.
- In the context of BDC-planning, simple list scheduling bi-criteria heuristics (for example, BHEFT and BDLS) may be as effective as more sophisticated heuristics based on extensive local search, such as DCA.
- With low running cost, BHEFT seems to be a good choice for BDC-planning.

6 Conclusion and Future Work

BDC-planning is required to establish an SLA in order to provide a certain level of QoS for workflow execution in market-based environments. This paper proposed BHEFT, a novel low-cost bi-criteria heuristic based on HEFT, to fulfill the specific requirements of BDC-planning. The experimental results suggest that BHEFT appears to be at least as effective, or even more so than other existing

sophisticated bi-criteria workflow scheduling heuristics, and has a lowest execution time cost and good scalability. It also appears that BHEFT can effectively and efficiently find a BDC-plan under various circumstances of constraints. This enables a quick admission control decision (i.e., a judgement of whether or not the submitted user request is acceptable), and provides the feasibility of automating the creation of an SLA over diverse user constraints. Based on the work in this paper, our future work will try more experiments using different applications and platforms and will consider the overestimation of task execution time in BDC-planning to cope with prediction uncertainty. In addition, we think it is worth investigating BDC-planning with more sophisticated pricing policies.

References

1. Almeida, J., Almeida, V., Ardagna, D., Cunha, I., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing 4(70), 344–362 (2010)
2. Berriman, G.B., Good, J.C., Laity, A.C., Bergou, A., Jacob, J., Katz, D.S., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Williams, R.: Montage: A grid enabled image mosaic service for the national virtual observatory. In: The Conference Series of Astronomical Data Analysis Software and Systems XIII, ADASS XIII (2004)
3. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems 25, 528–540 (2009)
4. Deelman, E., Kesselman, C., Mehta, G., Meshkat, L., Pearlman, L., Blackburn, K., Ehrens, P., Lazzarini, A., Williams, R., Koranda, S.: GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In: High Performance Distributed Computing (HPDC 2002), pp. 225–234 (2002)
5. Dõgan, A., Özgüner, R.: Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. The Computer Journal 3(48), 300–314 (2005)
6. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility grids: Time and cost trade-off management. In: Thirty-Second Australasian Computer Science Conference (ACSC 2009), vol. 91, pp. 139–147 (2009)
7. Garg, S.K., Buyya, R., Siegel, H.J.: Time and cost trade-off management for scheduling parallel applications on utility grids. Future Generation Computer Systems 8(26), 1344–1355 (2010)
8. Garg, S.K., Konugurthi, P., Buyya, R.: A linear programming driven genetic algorithm for meta-scheduling on utility grids. In: Proceedings of the 16th International Conference on Advanced Computing and Communication, ADCOM 2008 (2008)
9. Han, Y., Youn, C.: A new grid resource management mechanism with resource-aware policy administrator for SLA-constrained applications. Future Generation Computer Systems 7(25), 768–778 (2009)
10. Hiles, A.: Service level agreements: measuring cost and quality in service relationships. Chapman & Hall (1993)
11. Horn, J.V., Dobson, J., Woodward, J., Wilde, M., Zhao, Y., Voeckler, J., Foster, I.: Grid-based computing and the future of neuroscience computation. Methods in Mind (2005)
12. Prodan, R., Wieczorek, M.: Bi-criteria scheduling of scientific grid workflows. IEEE Transactions on Automation Science and Engineering 7, 364–376 (2010)

13. Quan, D.M.: Mapping Heavy Communication Workflows onto Grid Resources Within an SLA Context. In: Gerndt, M., Kranzlmüller, D. (eds.) HPCC 2006. LNCS, vol. 4208, pp. 727–736. Springer, Heidelberg (2006)
14. Quan, D.M., Kao, O.: Mapping Workflows onto Grid Resources Within an SLA Context. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 1107–1116. Springer, Heidelberg (2005)
15. Risch, M., Altmann, J., Guo, L., Fleming, A., Courcoubetis, C.: The GridEcon Platform: A Business Scenario Testbed for Commercial Cloud Services. In: Altmann, J., Buyya, R., Rana, O.F. (eds.) GECON 2009. LNCS, vol. 5745, pp. 46–59. Springer, Heidelberg (2009)
16. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in GRID Computing, pp. 189–202. Springer, Heidelberg (2007)
17. Siddiqui, M., Villazon, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized QoS. In: Proceedings of the 2006 IEEE/ACM Conference in Supercomputing (SC 2006), pp. 103–118 (2006)
18. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Systems 2(4), 175–187 (1993)
19. Singh, G., Kesselman, C., Deelman, E.: A provisioning model and its comparison with best-effort for performance-cost optimiazation in grids. In: Proceedings of the 16th International Symposium on High Performance Distributed Computing, pp. 117–126 (2007)
20. Talukder, A.K.M., Kirley, M., Buyya, R.: Multi-objective differential evolution for scheduling workflow applications on global grids. Concurrency and Computation: Practice and Experience 21(13), 1742–1756 (2009)
21. Topcuoglu, H., Hariri, S., Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 3(13), 260–274 (2002)
22. Wieczorek, M., Hoheisel, A., Prodan, R.: Taxonomies of the multi-criteria grid workflow scheduling problem. In: Proceedings of the CoreGRID Workshop on Grid Middleware (2007)
23. Yeo, C.S., Buyya, R.: Managing risk of inaccurate runtime estimates for deadline constrained job admission control in clusters. In: Proceedings of the 35th International Conference on Parallel Processing (ICPP 2006), pp. 451–458 (2006)
24. Yin, J., Wang, Y., Hu, M., Wu, C.: Predictive admission control algorithm for advance reservation in equipment grid. In: Proceedings of IEEE International Conference on Service Computing (SCC 2008), pp. 49–56 (2008)
25. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14, 217–230 (2006)
26. Yu, J., Buyya, R.: Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (2007)
27. Zhao, Y., Wilde, M., Foster, I., Voeckler, J., Jordan, T., Quigg, E., Dobson, J.: Grid middleware services for virtual data discovery, composition, and integration. In: The 2nd Workshop on Middleware for Grid Computing (2004)

Virtual Machine Placement for Predictable and Time-Constrained Peak Loads

Wubin Li, Johan Tordsson, and Erik Elmroth

Department of Computing Science and HPC2N,
Umeå University, SE-901 87 Umeå, Sweden
{wubin.li,tordsson,elmroth}@cs.umu.se
http://www.cloudresearch.se

Abstract. We present an approach to optimal virtual machine placement within datacenters for predicable and time-constrained load peaks. A method for optimal load balancing is developed, based on binary integer programming. For tradeoffs between quality of solution and computation time, we also introduce methods to pre-process the optimization problem before solving it. Upper bound based optimizations are used to reduce the time required to compute a final solution, enabling larger problems to be solved. For further scalability, we also present three approximation algorithms, based on heuristics and/or greedy formulations. The proposed algorithms are evaluated through simulations based on synthetic data sets. The evaluation suggests that our algorithms are feasible, and that these can be combined to achieve desired tradeoffs between quality of solution and execution time.

Keywords: Cloud Computing, Virtual Machine Placement, Binary Integer Programming, Off-line Scheduling, Load Balancing.

1 Introduction

Building on technologies such as distributed systems, autonomic computing, and virtualization, cloud computing emerges as a promising computing paradigm for providing configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [13]. A key feature of future cloud infrastructures is elasticity [2], i.e., the ability of the cloud to automatically and rapidly scale up or down the resources allocated to a service according to the workload demand while enforcing the Service Level Agreements (SLAs) specified.

In this paper, we focus on elasticity scenarios where workloads are predictable and to be deployed and scaled-out quickly through the rapid provisioning of Virtual Machines (VMs). Predictable workload scenarios are frequently occurring, e.g., online banking has regular peaks once a month, streaming video is consumed mostly during evenings, and video gaming workloads exhibit predictable daily and weekly changes [6], etc. Both the service and the cloud infrastructure

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 120–134, 2012.

can benefit from the predictability of the workloads, since placement schemes for VMs are possible to be pre-calculated and resources can be set up in advance.

To fulfil the service demand, the cloud infrastructure usually produces VM placement solutions improving criteria such as cost, performance, resource utilization, etc. However, from a cloud infrastructure perspective, physical machines usually have non-uniform capacities. Their respective utilizations may have high variances. Users of a service may suffer from high latency due to high utilizations of some physical servers. In other words, certain types of applications could benefit from keeping the utilization of individual machines as close as possible to the utilization of the entire system [15]. To tackle this problem, the worst case of individual physical machine utilization should be minimized and load balancing in the whole system should thus be optimized.

The VM placement problem can be generally formulated as a variant of the class constrained multiple-knapsack problem that is known to be NP hard [14]. Existing approximation algorithms can scale to at most a few hundred machines, and may produce placement solutions that are far from optimal when system resources are scarce [15]. In this paper, we focus on properties of the load balancing problem itself instead of proposing new generic approximation algorithms. We analyse how the studied problem differs from general VM placement problems, and present a linear programming formulation of the optimization problem along with some approximations. An evaluation based on synthetic workloads is used to investigate the feasibility of the algorithms.

The remainder of the paper is organized as follows. Section 2 briefly describes the problem and motivates our work. Section 3 presents the problem formulation, defines an optimal algorithm, as well as describes three problem-specific approximations. Section 4 presents an evaluation of our approach. Section 5 discusses related work. Finally, conclusions and future work are given in Section 6.

2 Problem Description

The studied scenario is illustrated in Figure 1. A set of physical machines with diverse capacities are used to execute VMs of different sizes. The VMs are grouped by VM sets, i.e., prepared bundles of, e.g., application servers, front ends, and data base replicas for managing peak loads of certain applications. These VM sets are to be deployed across the physical machines, i.e., PM_1, PM_2, ..., PM_m, which may have different background loads and non-uniform capacities. Each VM set is comprised of multiple VMs with various capacity requirements. The durations and sizes of VMs are known in advance. This life cycles of VM sets may be different, e.g., some may be provisioned longer than others, some may start to run earlier than others, etc.

The most significant aspect that could distinguish the VM placement for predictable peak loads from general placement problems is that the peak loads are time-constrained. After a certain period, the additional VMs are removed from the cloud infrastructure. During this period, multiple placement requests

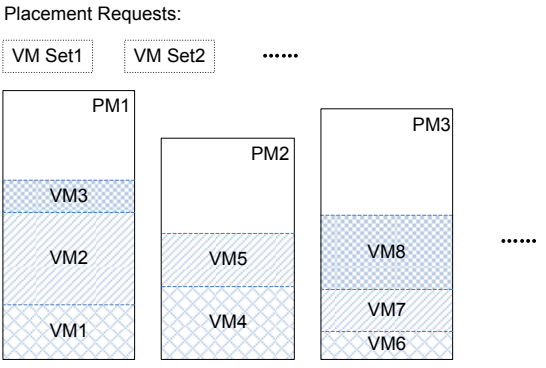


Fig. 1. Studied scenario illustration

may start or terminate. In this paper, the placement objective is load balancing, i.e., to minimize the highest utilization of any individual physical machine during this period.

3 Problem Analysis and Formulation

We use a quadruple $r =< id, s, e, VMSet >$ to uniquely identify a placement request, where s indicates when the request starts and e specifies the end-time. A placement request set can thus be represented by an array of quadruples temporally ordered by s. The *VMSet* is a collection of VMs, each of which may have different computation capacities.

Table 1. Hardware metrics for instance types

Instance Type	micro	small	medium	large	xlarge	...
CPU (# cores)	1	1	1	2	4	
CPU (GHz/core)	1	1	2	2	2	
Memory (GB)	0.613	1.7	3.5	7.5	15	...
Storage (GB)	50	160	300	850	1700	
Computing Capacity	**1**	**2**	**4**	**8**	**16**	**...**

To distinguish VMs with different computation capacities, we use the hardware discretization approach, used e.g., by Amazon EC2 as shown in Table 1. An example of placement request is <23, 2011-05-30 18:30, 2011-06-02 12:00, {4, 2, 1, 4, 16}>. This request has id 23, starts at 2011-05-30 18:30, ends at 2011-06-02 12:00 and demands 5 VMs with capacities 4, 2, 1, 4, and 16 respectively. All VMs in a request are to start and terminate at the same time.

Table 2. Symbols used in this paper

H	Time period that includes placement requests.
N	Number of placement requests.
N_i	Size of the VMSet in the ith request.
C_{ij}	Capacity requirement of the jth VM in the ith request.
M	Number of physical machines.
w_k	Existing load of the kth physical machine.
W_k	Total capacity of the kth physical machine.
x_{ijk}	The placement decision variable. $x_{ijk} = 1$ iff the jth VM in request i is placed on physical machine k, and 0 otherwise.
G	Number of overlap sets generated.
y_{ig}	$y_{ig} = 1$ if the ith placement request is in the gth overlap set, and 0 otherwise.

Table 2 contains an overview of the symbols used to formulate the load minimization placement problem. Now, for a given VM set in a request set R and a set of M physical machines, the highest utilization of any individual physical machine can be described by

$$Load(R) = \max_{k \in [1..M]} \frac{\sum_{i=1}^{N} \sum_{j=1}^{N_i} (x_{ijk} * C_{ij}) + \omega_k}{W_k}, \tag{1}$$

where x_{ijk} is the decision variables for placement, C_{ij} the VM capacity, and w_k and W_k the existing load and total capacity of the physical machines. For any allocation of VMs to physical machines, the following constraints apply:

$$\forall i \in [1..N], j \in [1..N_i] :$$
$$\sum_{k=1}^{M} x_{ijk} = 1 \tag{2}$$
$$\forall k \in [1..M]$$
$$\sum_{i=1}^{N} \sum_{j=1}^{N_i} (x_{ijk} * C_{ij}) + \omega_k \leq W_k. \tag{3}$$

Constraint (2) specifies that each VM in every placement request has to be assigned to exactly one physical machine, and constraint (3) describes how the total capacity of each physical machine cannot be exceeded.

There are multiple possible approaches to the placement request allocation problem for load minimization. Our first and simplest algorithm is a greedy formulation that for each VM in each VM set (in order by request start time) finds the placement that keeps the average load at a minimum. This is done by

finding the physical machines that provide the worst-fit for each VM, i.e., leaves the maximum residual capacity. Of course, before placing a certain request, previous requests that have terminated can be excluded and the physical machines reused. This algorithm, *Greedy Worst-Fit*, is defined in Algorithm 1.

Algorithm 1. Greedy Worst-Fit(R)

```
   Input: Placement request set R = {r_1, r_2, ..., r_n}.
   Output: Placement Scheme for R.
 1 Sort all the requests by start-time s;
 2 for 1 ≤ i ≤ n do
 3     for 1 ≤ j < i do
 4         if r_j is expired but still being provisioned then
 5             exclude r_j and release capacities of the physical machines that host
               the VMs of r_j;
 6         end
 7     end
 8     foreach vm in VMSet_i do
 9         pm_k ← the least loaded physical machine with highest residual capacity;
10         if vm can fit in pm_k then
11             assign vm to pm_k;
12         end
13         else
14             no feasible solution;
15             return;
16         end
17     end
18 end
```

Although the greedy formulation is fast to compute, it does not provide an optimal solution to the VM placement problem (with respect to load balancing), as VM placement is a version of the general assignment problem [7]. Our second algorithm operates in a similar manner to the Greedy Worst-Fit one in that it considers the placement requests sequentially in order of start time. However, instead of performing a greedy allocation, the second algorithm finds, for each point in time when a VM set is about to start, the allocation of all running VM sets, including the new one, that minimizes the average utilization. This method, (*Sequential*) is described in more detail in Algorithm 2 and the mathematical expressions for load minimization is given by equations (1), (2), and (3). Note that, in Algorithm 2, the minimization in each iteration (see line 8) treats only the active request sets.

As the complete set of VM requests are known in advance, we can, at the expense of additional complexity, solve the load balancing optimization problem not only for the currently running VMs, but for all VMs. This algorithm is a knapsack formulation, and is defined in Algorithm 3 (*Knapsack*).

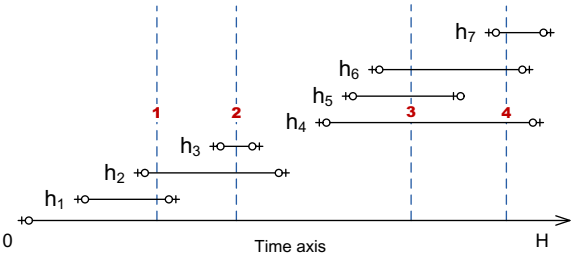


Fig. 2. Illustration for coexistence of placement requests

One key observation is that two VM sets may use the exact same physical resources if they do not overlap in runtime. More formally, two placement requests are coexistent if and only if their lifetimes overlap, i.e., placement request r_1 and r_2 are coexistent if and only if $s_2 \leq s_1 < e_2$ or $s_1 \leq s_2 < e_1$. Figure 2 shows an illustration of coexistence. In this figure, there are 7 placement requests whose start-times are h_i $(1 \leq i \leq 7)$ for request r_i, respectively. For a given placement request set R, we introduce the notion of *OverlapSets* to define a subset of R where any two requests in the subset are coexistent. Furthermore, there exists no request in R that is not in *OverlapSet* that is coexistent with every request in *OverlapSet*. For the example in Figure 2, we get $OverlapSets = \{\{r_1, r_2\}, \{r_2, r_3\}, \{r_4, r_5, r_6\}, \{r_4, r_6, r_7\}\}$.

In principle, to calculate the highest utilization of any individual physical machine during the whole period H, we must generate all *OverlapSets*, and compute the maximum load of machines in each *OverlapSet* after placing all VMs that run in that set. From the definition of the overlap sets, a straightforward recursive algorithm to generate the sets can be derived. However, this recursion results in an exponential runtime complexity. It is thus a very time-consuming task to complete generating all *OverlapSet*s when the number of placement requests is large. For example, in our experiments, the time required to generate all *OverlapSet*s varied from 0.01 second to 45 minutes.

Algorithm 2. *SequentialPlacement*(R)

Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$.
Output: Placement Scheme for R.

```
1 Sort all the requests by start-time s;
2 for 1 ≤ i ≤ n do
3     for 1 ≤ j < i do
4         if r_j is expired but still being provisioned then
5             exclude r_j and release capacities of the physical machines that host
              the VMs of r_j;
6         end
7     end
8     Minimize (1) with (2) and (3) as constraints;
9 end
```

Algorithm 3. *Knapsack*(R)

Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$.
Output: Placement Scheme for R.
1 Minimize (1) with (2) and (3) as constraints;

We instead use an approximation based on discrete time slots: The time from the earliest request start-time to the latest end-time is divided into T time slots. Every time slot is examined and placement requests in this slot are collected and considered as a potential element of *OverlapSet*s (see line 7 in Algorithm (4)). If a potential element is not a subset of some element in *OverlapSet*s, it is finally added to *OverlapSet*s after its subsets (if non-empty) are removed from *OverlapSet*s (lines $9 - 15$ in Algorithm (4)). Obviously, the quality of solution generated by this algorithm depends on T. If T is large enough, the solution is close to the one generated by the exact recursive method. Since the time complexity of Algorithm (4) is polynomial ($\Theta(T * n)$), it is much faster than the recursive formulation even when T and n are large. Through experiments, we note that it takes around 2 seconds to complete the generation process when $T = 10000$, $H = 24$ hours, and $n = 1000$, whereas with the recursive method, this problem size would take a day or more.

Algorithm 4. *GenerateOverlapSets*(R, T)

Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$, the number of time slots T.
Output: The *OverlapSets* of R.
1 $OverlapSets \leftarrow \{\}$;
2 Sort all the requests by start-time s;
3 $S = \underset{i \in [1..n]}{Min}\{s_i\}, E = \underset{i \in [1..n]}{Max}\{e_i\}$, $interval = (E - S)/T$;
4 **for** $1 \leq i \leq T$ **do**
5 $\quad ts \leftarrow S + (i-1) * interval$;
6 $\quad te \leftarrow S + i * interval$;
7 $\quad currentSet = \{r \in R \mid r$ starts in $[ts, te]\}$;
8 $\quad should\_add \leftarrow true$;
9 $\quad$ **foreach** P *in* $OverlapSets$ **do**
10 $\quad\quad$ **if** $P \subset currentSet$ **then**
11 $\quad\quad\quad OverlapSets \leftarrow OverlapSets \setminus P$;
12 $\quad\quad$ **end**
13 $\quad\quad$ **if** $currentSet \subseteq P$ **then**
14 $\quad\quad\quad should\_add \leftarrow false$;
15 $\quad\quad$ **end**
16 $\quad$ **end**
17 $\quad$ **if** *should_add* **then**
18 $\quad\quad OverlapSets \leftarrow OverlapSets \cup currentSet$;
19 $\quad$ **end**
20 **end**

Incorporating the concept of overlap sets, our knapsack algorithm can now be reformulated as:

$$\text{Minimize} \quad \{ \underset{R' \in GenerateOverlapSets(R)}{\text{Maximize}} \text{Load}(R') \quad \}$$

Subject to

$$\forall i \in [1..N], j \in [1..N_i] :$$

$$\sum_{k=1}^{M} x_{ijk} = 1 \tag{4}$$

$$\forall k \in [1..M], g \in [1..G] :$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N_i} (x_{ijk} * C_{ij} * y_{ig}) + \omega_k \leq W_k. \tag{5}$$

Here, y_{ig} is a decision variable for coexistence used to determine if two VMs can use the same physical resources i.e., if they do not overlap in time. Constraint (4) is the same as constraint (2) and specifies that each VM in every placement request has to be placed in exactly one physical machine. Constraint (5) is the capacity constraint for each physical machine, with the coexistence as an additional feature. This is a Min-Max optimization problem, which is non-linear. To transform this problem to a linear programming problem, we add μ to the list of unrestricted variables subject to the constraints

$$\forall R' \in GenerateOverlapSets(R) : \quad \text{Load}(R') \leq \mu \tag{6}$$

and try to minimize μ.

Two steps are required to solve the problem: generation of *OverlapSet*s from placement requests, and solving the model using the *OverlapSet*s as inputs. In principle, the solver must enumerate each possible placement scheme, check whether it is viable, and compare the μ to the minimum found so far. There are multiple potential optimizations to reduce the computation cost for generating *OverlapSet*s and solving this model. To reduce the search space, we can significantly improve the performance of the solver by identifying upper bounds that are easy to compute. Since Greedy Worst-Fit is polynomial and fast to complete, we use the approximated load calculated through Greedy Worst-Fit as an upper bound as shown in Equation (7):

$$\mu \leq \gamma, \tag{7}$$

where γ is the highest utilization of any individual physical machine as calculated by Greedy Worst-Fit algorithm. This optimization tends to reduce the time required to compute a solution drastically, thus improving scalability. We refer to this approach that combines upper bound optimizations and overlap sets as *Time-bound Knapsack*, as described in Algorithm 5. In this algorithm, Line 1 calculates the approximative placement using Greedy Worst-Fit algorithm. Lines 2-12 determine the upper bound value for the approximative placement, by finding the highest load for any physical machine that follows the greedy

placement scheme. Line 13 generates the overlap sets, and Line 14 minimizes the maximum load.

Algorithm 5. Time-bound Knapsack(R, T)

Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$, the number of time slots T.
Output: Placement Scheme for R.

```
1  Execute Greedy Worst-Fit algorithm to initialize variables x_ijk;
2  γ ← 0;
3  for 1 ≤ i ≤ n do
4      for 1 ≤ j < i do
5          if r_j is expired but still being provisioned then
6              exclude r_j and release capacities of the physical machines that host
               the VMs of r_j;
7          end
8      end
9      if γ < Load(r_i) then
10         γ ← Load(r_i);
11     end
12 end
13 OverlapSets ← GenerateOverlapSets(R,T);
14 Minimize μ with (4), (5), (6) and (7) as constraints;
```

4 Evaluation and Discussion

In this section, the four proposed algorithms are studied from three perspectives: how good they are at finding solutions to the placement problems, the quality of the found solutions, and the computational complexity. The experimental setup is a scenario with a cloud provider with 100 physical machines and 32 placement requests, each with between 1 and 8 VMs (uniformly distributed). As outlined in Table 3, VM capacity is uniformly distributed between micro (computing capacity 1) and xxlarge (computing capacity 32). The background load for each physical machine is uniformly distributed between 20% and 50%. The placement problems are encoded using the AMPL [9] modelling language and solved with the Gurobi [1] solver. All experiments are performed on a workstation with a 2.67 GHz quad-core CPU and 4 GB of memory.

To evaluate the performance of our approach with respect to quality of solution, we first perform 1000 experiments with groups of placement requests. We specify a one minute execution time limit for all algorithms. Even for very short term peak loads, e.g., hourly spikes, this one minute limit should be short enough to calculate a placement solution and configure the system accordingly.

Table 4 summarizes the 1000 experiments. We note that Sequential is able to solve most problems (994), followed by Time-bound Knapsack (923), Greedy Worst-Fit (870), and Knapsack trailing with 732 successfully solved problems. Looking closer at the unsuccessfully solved problems, we note that Time-bound

Table 3. Experiment Setup

H (experiment duration)	48 hours
Number of physical machines	100
Existing load for each physical machine	Uniform(20%, 50%)
Capacity for each physical machine	$2^{\text{Uniform}(0,7)}$
Number of placement requests	32
Number of VMs in a request	Uniform(1, 8)
VM capacity demands	$2^{\text{Uniform}(0,5)}$
Life-cycles of placement requests	Uniform(1,H)

Table 4. 1000 groups experiment with 1 minute execution time limitation

Algorithms	Feasible Solutions	No Solution	Time-out
Time-bound Knapsack	923	0	77
Knapsack	732	30	238
Sequential	994	4	2
Greedy Worst-Fit	870	130	0

Knapsack encounters no infeasible placements, whereas this happens 4 times for Sequential, 30 for Knapsack and 130 for Greedy. Considering the problem instances that could not be solved within feasible time (here selected as one minute), we note that Greedy always completes within this time, but Sequential fails in 2 cases, Time-bound Knapsack in 77, and Knapsack in 238 cases. When combining the two reasons for failing to solve the placement problems, Time-bound Knapsack and Sequential appear to be the most promising approaches.

Looking further into quality of solutions, we exclude, for each algorithm, the experiments that could not be solved successfully (or within a minute). The left part of Figure 3 shows the average load balance (i.e., the maximum load for any machine during the experiments), including Standard Deviation (SD), for the successfully solved instances for each algorithm. Here we note that Time-bound Knapsack result in the best load balance, $71.9\% \pm 6.1\%$, whereas the three other algorithms all result in loads above 80%, with Sequential the second best at $80.5\% \pm 7.6\%$. The right part of Figure 3 shows the average execution time, including deviation, for the successfully solved problems. As expected, the polynomial Greedy algorithm is the fastest with average execution time less than 0.5 seconds, as compared to 8 seconds for Time-bound Knapsack, 11 seconds for Sequential, and 13 seconds for Knapsack. For the last three algorithms, there are large deviations in execution time for successfully solved problems, also after excluding the experiments that failed to due exceeding the one minute threshold.

To understand the behaviour of the algorithms more in-depth, we focus on how the maximum load of any physical machine (the load balance) varies over the 48 hours experiment duration for one of the 1000 experiments. As illustrated in Figure 4, the Greedy algorithm results in volatile loads with large deviations over time, whereas and Sequential is more stable but still experiences fluctuations. In contrast, both Knapsack and Time-bound Knapsack are very stable, and keep the

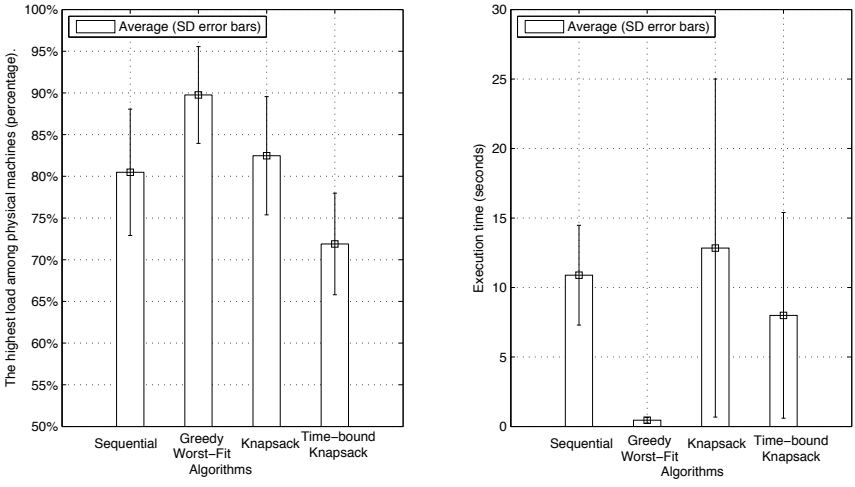


Fig. 3. Performance and execution time comparison for 1000 tests

maximum load constant for almost the full duration of the experiment. The low load very early and very late for all algorithms is due to there being few running VM sets at these points in time. Figure 4 also gives some insight into how often the algorithms cannot find feasible solutions. For complicated problems with many VMs, Greedy, Knapsack, and sometimes also Sequential may fail due to capacity constraints of the physical machines, whereas Time-bound Knapsack is more likely to find a solution.

To study the computational complexity (execution times) of the algorithms further, we perform a second experiment with 100 groups of placement requests where the execution time was unlimited. Here, we focus on the experiments where the placement took longer than one minute to solve. Table 5 presents the number of failures (experiments that ran for more than one minute) and their execution time deviations in the evaluated 100 tests. Here, we observe that Knapsack exceeds the time limit in 20% of all tests, Time-bound Knapsack in 4% of the tests, and Sequential in a single test, whereas Greedy always completes well within one minute. Looking at the average execution times for these tests, we note that Sequential requires 2.6 minutes, Time-bound Knapsack 95 ± 129 minutes, and Knapsack 346 ± 788 minutes, i.e., there are a few cases where the latter two algorithms required several hours to complete. A comparison of the required execution time and the percentage of problems successfully solved is shown in Figure 5. This figure illustrates that although the Knapsack and Time-bound Knapsack algorithms in a few specific cases can be very slow, they most often generate solutions within a few seconds, and allowing these to execute a couple of minutes improves the probability of finding a solution substantially.

To summarise these experiments, the Time-bound Knapsack algorithm generates the best solutions, i.e., finds the placement with the lowest average load, and is also able to find valid placements in complicated cases where the other algorithms fail. However, it can at times be very slow to execute. Conversely, the

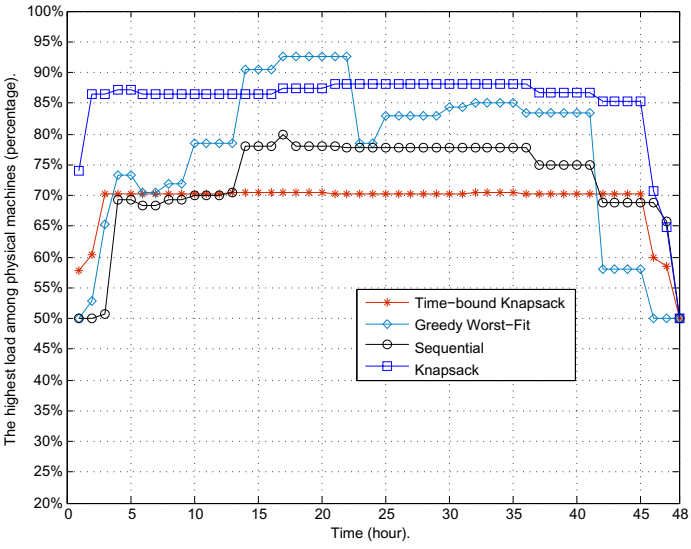


Fig. 4. Illustration of highest loads of the infrastructure evolve with time

Table 5. Number of failures (slow executions) and execution times for 100 tests

Algorithms	Failure	Failure execution time (minutes)
Greedy Worst-Fit	0	
Sequential	1	2.6 ± 0
Time-bound Knapsack	4	94.7 ± 128.6
Knapsack	20	345.5 ± 787.5

Greedy algorithm is very fast to compute and should scale well also for larger problem sizes due to its polynomial complexity. However, it generates placements with worse load balance and fails to find feasible solutions in some high workload scenarios. In comparison with these two algorithms, Knapsack performs worse in overall. Notably, Sequential can be a suitable compromise between quality of solution and execution time, although it does not excel in either.

5 Related Work

Virtual machine placement across physical servers has recently gained a lot of attraction. Our previous contributions within this area include integer programming methods to obtain optimal cost-performance tradeoffs in deploying VMs across multiple clouds [17] and methods to dynamically reschedule VMs (including modeling of VM migration overhead) upon changed conditions [12].

Other contributions to VM placement include a binary integer program formulation for cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads is proposed by den Bossche et al. [4]. It is demonstrated that this

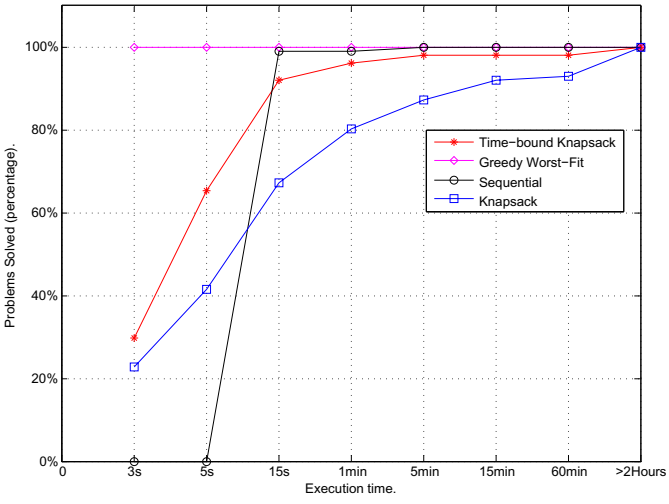


Fig. 5. Execution time vs. percentage of problems solved for 100 tests

approach results in a tractable solution for scheduling applications in a public cloud, but that the same method becomes much less feasible in a hybrid cloud setting due to sometimes having long solving time. Compared to our work, their approach also considers the life-cycles of workloads, but mainly focuses on cost-effective scheduling of applications in a hybrid cloud setting. Load balancing issues are not considered.

Bobroff et al. present a dynamic server migration and consolidation algorithm to minimize the number of working physical machines without violating SLAs [3]. This work takes only CPU demands into account and uses classification of workload signatures to identify the servers that benefit most from dynamic migration. Using adaptable forecasting techniques well suited for the classification, substantial improvement over static VM placement is shown, reducing the amount of required capacity and the rate of SLA violations.

A scalable application placement controller for enterprise data centres is proposed by Tang et al. [15]. The objective of this controller is to maximize the total satisfied application demand, to minimize the number of application starts and stops, and to balance the load across machines. Compared to existing state-of-the-art algorithms, this controller can produce within 30 seconds high-quality solutions for hard placement problems with thousands of machines and thousands of applications. This work is later extended to a binary search based framework striving to limit the worst case of each individual server's utilization by Tian et al. [16]. The system cost, defined as the weighted combination of both placement change and inter-application communication cost, can be also maintained at a low level. However, life-cycles of workloads remain unexplored.

Breitgand et al. [5] propose a multiple-choice multidimensional knapsack problem formulation for policy-driven service placement optimization in federated clouds, and a 2-approximation algorithm based on a greedy rounding of a linear

relaxation of the problem. The proposed placement algorithms aims at maximizing provider profit while protecting Quality of Service (QoS) as specified in SLAs of the workloads, and can be used to optimize power saving or load balancing internally in a cloud, as well as to minimize the cost for outsourcing workloads to external cloud providers. Breitgand et al. encode load balancing as the standard deviation of the residual capacity, which is a non-linear function. A binary search-based heuristic is used to minimize that function, and thus an optimal solution is not guaranteed.

6 Conclusions and Future Work

We study the VM placement problem for load balancing of predictable and time-constrained peak workloads. We formulate the problem as a Min-Max optimization one and present an algorithm based on binary integer programming, along with three approximations for tradeoffs in scalability and performance. An experimental study compares the proposed methods with respect to ratio of problems successfully solved, quality of solution, and computational complexity.

Future directions for our work include studies of other load balancing metrics, e.g., looking at how to minimize the average load over time instead of the maximum load. Another topic is how to refine the models and replace the one-dimensional computing capacity performance metric, e.g., with CPU, memory, disk, etc. as suggested by Breitgand et al. [5] and to incorporate inter-VM resources such as network bandwidth, as demonstrated by Lampe et al. [11]. Additionally, one interesting feature to consider in optimization is the grouping of VMs to hosts based on the interference and overhead that one VM causes on the other concurrently running VMs on the same physical host, as discussed by Kousiouris et al. [10].

Acknowledgments. We are grateful to the anonymous reviewers for their constructive and valuable feedback, improving the quality of this work. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreement no. 257115 (OPTIMIS [8]) and the Swedish Government's strategic effort eSSENCE.

References

1. Gurobi Optimization (2010), `http://www.gurobi.com` (visited October 2011)
2. 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. Communications of the ACM 53, 50–58 (2010)
3. Bobroff, N., Kochut, A., Beaty, K.A.: Dynamic placement of virtual machines for managing sla violations. In: Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, IM 2007, pp. 119–128 (2007)
4. den Bossche, R.V., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing, pp. 228–235 (2010)

5. Breitgand, D., Marashini, A., Tordsson, J.: Policy-driven service placement optimization in federated clouds. Tech. rep., IBM Haifa Labs (2011)
6. Chambers, C., Feng, W., Sahu, S., Saha, D.: Measurement-based characterization of a collection of on-line games. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, IMC 2005, pp. 1–14 (2005)
7. Chen, C., Tsai, K.: The server reassignment problem for load balancing in structured p2p systems. IEEE Transactions on Parallel and Distributed Processing 19(2), 234–246 (2008)
8. Ferrer, A.J., Hernádez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R.M., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S.K., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forgó, N., Sharif, T., Sheridan, C.: OPTIMIS: A holistic approach to cloud service provisioning. Future Generation Computer Systems 28(1), 66–77 (2012)
9. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. Duxbury Press (November 2002)
10. Kousiouris, G., Cucinotta, T., Varvarigou, T.: The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software 84(8), 1270–1291 (2011)
11. Lampe, U., Siebenhaar, M., Schuller, D., Steinmetz, R.: A cloud-oriented broker for cost-minimal software service distribution. In: Proceedings of the 2nd ServiceWave Workshop on Optimizing Cloud Services (2011)
12. Li, W., Tordsson, J., Elmroth, E.: Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011), pp. 163–171 (2011)
13. Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology, NIST (2011)
14. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica 29(3), 442–467 (2001)
15. Tang, C., Steinder, M., Spreitzer, M., Pacifici, G.: A scalable application placement controller for enterprise data centers. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 331–340. ACM (2007)
16. Tian, C., Jiang, H., Iyengar, A., Liu, X., Wu, Z., Chen, J., Liu, W., Wang, C.: Improving application placement for cluster-based web applications. IEEE Transactions on Network and Service Management 8(2), 104–115 (2011)
17. Tordsson, J., Montero, R., Moreno-Vozmediano, R., Llorente, I.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2), 358–367 (2012)

Risk Assessment in Service Provider Communities

Ioan Petri[1,2], Omer F. Rana[3],
Yacine Rezgui[1], and Gheorghe Cosmin Silaghi[2]

[1] School of Engineering, Cardiff University, UK
[2] Business Information Systems, Babes-Bolyai University, Romania
[3] School of Computer Science & Informatics,
Cardiff University, UK

Abstract. On-line service delivery undertaken between clients and service providers often incurs risks for both the client and the provider, especially when such an exchange takes place in the context of an electronic service market. For the client, the risk involves determining whether the requested service will be delivered on time and based on the previously agreed Service Level Agreement (SLA). Often risk to the client can be mitigated through the use of a penalty clause in an SLA. For the provider, the risk revolves around ensuring that the client will pay the advertised price and more importantly whether the provider will be able to deliver the advertised service to not incur the penalty identified in the SLA. This becomes more significant when the service providers outsource the actual enactment/execution to a data centre – a trend that has become dominant in recent years, with the emergence of infrastructure providers such as Amazon.com. In this work we investigate the notion of "risk" from a variety of different perspectives and demonstrate how risk to a service owner (who uses an external, third party data centre for service hosting) can be managed more effectively. A simulation based approach is used to validate our findings.

Keywords: Service Level Agreement, Risk Management, Fault Tolerance.

1 Background

With the emergence of Cloud computing it has become possible to differentiate between a software service owner (responsible for updating and managing a software capability encapsulated as a service) and an infrastructure provider (primarily offering computational, data and network resources that may be used to deploy the software service). A service owner can utilize the capability of one or more such infrastructure providers to offer the capability to clients, whereas an infrastructure provider looks for possible service owners to offer them managed access to resources, often at a pre-advertise price, at multiple capacities (small, medium and large instances in the case of Amazon.com, for instance) and with

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 135–147, 2012.

varying types of Service Level Agreements. Such differentiation between the service owner and infrastructure provider role is useful from a market perspective, as it enables different combinations of price-performance tradeoffs to be made available, thereby reducing the barrier to entry within a marketplace (as service owners no longer need to manage complex infrastructure which often incur significant capital cost) whilst also allowing specialist infrastructure providers to emerge on the market.

However utilizing external infrastructure to deploy services incurs risks for both the service owner and the infrastructure provider. Our focus is primarily on financial risk, evoking the notion of uncertainty and randomness within an exchange between a client and a provider. Significant literature exists about the notion of risk in financial markets, with this term being used synonymously with the "probability of a loss or gain arising from unexpected changes in market conditions" [7]. Although in a financial market risk is often associated with a change in market price of a product or derivative, in the context of this work, we associate risk with the likely financial loss that a service owner or infrastructure provider will incur due to their inability to deliver an advertised capability. It is therefore necessary for the service owner to consider one of the following three options: (i) *trust* the infrastructure provider and assume a certain degree of fault tolerance and resilience; (ii) *establish a Service Level Agreement (SLA)* to ensure that if a provider is unable to deliver the advertised capability, the infrastructure provider incurs a financial penalty that must be payed to the service owner; (iii) *utilize resilience mechanisms* directly to ensure that any possible faults that may arise can be overcome through a pre-identified strategy, thereby ensuring continued, fault free operation for clients. In (i) when dealing with trusted participants the process is simplified as there are already a number of approaches to ensure correct service provisioning. Trust may be established based on prior interaction with an infrastructure provider or based on the general reputation of the provider within the marketplace. This aspect has been investigated previously by a number of researchers [9,10]. On the other hand, in the context of untrusted environments ensuring fault free operation can be difficult due to a variety of possible outcomes that may arise during operation. This scenario is particularly prevalent when these parties are unknown to each other and therefore the level of risk associated with the transaction is considerably increased. Expanding on the three considerations identified above:

1. *Using Service Level Agreements* – this is applicable when the participants are unknown to each other – and therefore untrusted – with the behaviour of the participants being regulated through a previously agreed SLA. Such agreements are efficient instruments for mediating business transactions and can provide a useful reference point for monitoring capability exchanged between a client and provider (given that monitoring is carried out by either a trusted third party or through a pre-trusted component known to the client and the provider). An SLA may be used to specify Quality of Service (QoS) terms, the measurement criteria, reporting criteria and penalty/reward clauses between participants. Within an electronic market, an SLA may be used for: (i) an expression/proof of debts as

well as credits – debts to the client and credits to the service provider; (ii) as a token of exchange between participants; (iii) as an identification of responsibilities of participants involved (such as the client and service provider). Establishing an SLA between two parties (client & service provider) implies that the service provider has agreed to provide a particular capability to the client subject to some QoS constraints. In return, the client must provide a monetary payment (most often) or credit to the provider once the service has been delivered (subject to a penalty, often also monetary, in case the quality of service terms have not been adhered to) [1].

2. *Using trust mechanisms* – this is applicable when the environment is trusted and either: (i) clients and service providers have already interacted with each and have a history of prior (un)successful interactions; (ii) clients and service providers have access to feedback from other entities they trust – or through an aggregated reputation service they can access. Reputation can either be based solely on prior transactions, or be considered as a multi-dimensional characteristic involving technology, business preferences and usage/business policy – and their combinations [8]. With (ii), the feedback data provided by others to calculate the reputation may be misleading and/or sparse – thereby limiting its benefit. Hence, entities providing feedback can have different types of behaviours (both truth telling and deception), whereby feedback about a particular provider may be influenced by particular incentives that a client may have. By using existing trust mechanisms such malicious intent (based on incorrect feedback) can bias the overall trust establishment within a community of clients and service providers and trust values may change with the number of clients involved in the community and with those providing feedback [11].

3. *Using fault tolerance techniques* – this is applicable when dealing with unknown participants whose behaviour cannot be predetermined. Although a client (the service owner) may have an SLA with the provider, the client may still wish to minimise risk by ensuring that suitable fault tolerance strategies are available. For instance, establishing SLAs with entities that may exhibit faulty behaviours may represent a high risk. In order to mitigate these risks we propose a fault tolerance mechanism where various services are replicated among a number of peer-nodes.

The focus of this paper is to consider scenario (iii) where the service owner has to balance the loss in revenue incurred due to failure with the additional cost of replication. Determining the number of replicas to support needs to be balanced with the revenue achieved through each service instance and the likely penalty that may arise due to unavailability (arising from a failure). Section 2 discusses related work in risk management with a particular emphasis on financial risk, Section 3 describes the scenario we use to motivate this work and the overall methodology we employ to analyse risk. Section 4 provides an evaluation of the work through a number of experiments carried out on the PeerSim simulator.

2 Related Work

Risk assessment mechanisms are critical to increase the trust between clients and providers especially in distributed environments. The problem of risk management and associated cost mechanisms within a market of computational resources has been discussed by projects such as AssessGrid [5] and GridEcon [12]. The AssessGrid project proposed the development of a brokering mechanism that enabled *risk-aware* creation of SLAs between Grid service consumers and providers. The focus of the project was to offer a risk-aware decision support system allowing individuals to negotiate and consume Grid resources using SLAs. A utility computing business model which fits in an open market business model was used to monitor the system. The architecture of AssessGrid is divided into three layers, one for each of the actors: end-user, broker and provider. The end-user layer includes a portal which provides a number of abstract Grid applications which can interact with each other through an SLA Broker component (implemented using the WS-Agreement and WSRF (Web Services Resource Framework) specifications) [6]. The broker serves as the central actor of the system and can play the role of a mediator or a contractor on behalf of different participants. By investigating various scenarios and testing different roles that a provider can adopt, AssessGrid provides a risk management framework for supporting reliable service operations.

Multiple computing vendors such as HP, Amazon, Sun and IBM offer facilities for outsourcing service execution on commoditized servers using unit (small, medium and large) pricing models. These remotely located resources do not necessarily enable the end user to undertake specific analysis with particular requirements at a given time or to manage the risk of the system. Li et al. [7] try to predict availability by introducing risk analysis for Grids and propose new means to construct Service Level Agreements (SLAs) by reference to techniques of financial risk analysis. With this theoretical solution, prediction, quantification of risk, and consideration of liability in case of failure can be applied for the future provision of Grid Economics specifically, relating to the provision of SLAs through resource brokers, and comparable to markets in other commodities. In addition the model can be applicable to the configuration and management of related architectures such as those of P2P systems, Clouds and various kinds of network economics. For enriching the investigation, an analysis is performed on the potential formulation of a Grid Economy as a commodity market, and extended towards trading and hedging of risk, options, futures and structured products. The assumptions of this approach were constructed by collecting data regarding computational resource use on the UKs National Grid Service (NGS) and subsequently using this data in combination with approaches from computational finance (in particular the idea of Value at Risk (VaR)) to lead towards predicting availability of resources and associated insurance against losses.

Protector [2] is a probabilistic failure detector for cost-effective Peer-to-Peer storage. Protector, based on a SuperPeer overlay creation algorithm provides risk mitigation against transient failures. Protector presents applicability for group replication where all peers host replicas of the same object by detecting the

number of remaining replicas in a group, i.e., the number of replicas residing on online peers or peers experiencing transient failures. By using a failure prediction function calculated as the probability that a peer has permanently failed given an observed failure of d time units, Protector performs an aggregation of failure probabilities across all peers in the replica group in order to estimate the number of remaining replicas. Simulation is used to demonstrate that Protector enables the system to maintain objects in the most cost-efficient manner. Implementing a set of methods such as (i) leveraging prior failure statistics, and (ii) making estimates across a group of replicas which balance false positives for some peers against false negatives for others, Protector is validated by deploying a P2P storage system called *AmazingStore* – a storage sharing system that enables trusted users to exchange spare capacity with each other.

3 Methodology

When establishing an SLA, the service provider agrees to provide a particular capability to the client subject to some QoS constraints – referred to as Service Level Objectives in the WS-Agreement specification. In return, the client must provide a monetary payment (most often) or credit to the provider once the service has been delivered (subject to a penalty, often also monetary, in case the quality of service terms have not been adhered to). But when the service providers replicate their services (within one or more infrastructure provider(s) or data centre(s)), there are three important aspects to consider:

- Risk from the service owner's perspective: how many instances of each service should be replicated taking into account: (i) the cost associated with deploying each replica, and (ii) the penalty that must be paid if a service is unavailable (i.e. no working replica is available when a request is sent by a client).
- Risk from the infrastructure provider's perspective: determine how to optimise service replication in order to reduce deployment (hardware and software) costs and any penalties that may arise due to SLA non-compliance.
- Risk from the client's perspective: how to construct the SLA considering that a service owner may be unable to deliver the service.

Consider a server farm SF containing a collection of peer-nodes P=$[p_1, p_2, p_3, ..., p_n]$, some of which can be used for replicating a set of services S. S is a collection of services S=$[s_1, s_2, s_3, ..., s_m]$ deployed on the server farm SF. A subset $S_k \in S$ defines a collection of services S_k=$[s_1, s_2, s_3, ..., s_m]$,$m < n$ owned by a provider O_k where each service s_i has a number of replicated instances I_k. The set I_k identifies the number of instances for one service s_i, I_k=$[i_1, i_2, i_3, ..., i_k]$ and their associated costs; hence pairs (i_i, c_i) is associated with service s_i with c_i representing the cost of deploying instance i_i.

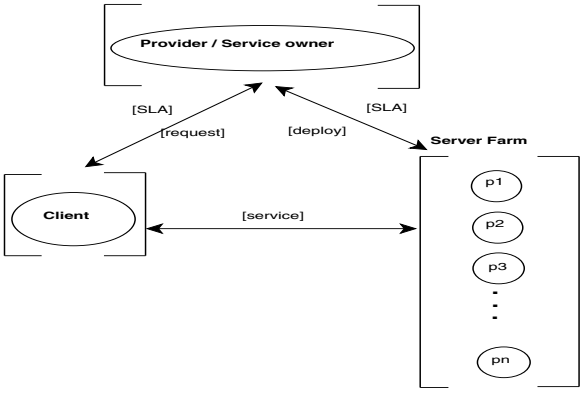


Fig. 1. Service Provision Scenario

3.1 Clients

A client searches for a service owner able to deliver a required capability, subject to a set of QoS constraints. We consider SLA_{Cl}^{Pr} identifying the provider commitment to deliver a service to the client, where the SLA encodes the particular constraints that have been agreed between the two parties. Such an SLA may encode characteristics for a single or multiple services from a provider. It is also necessary to identify the cost that a client must pay to the provider in order to acquire a needed capability s_i – identified as P, hence $C(Cl) = P(SLA_{Cl}^{Pr}(s_(i)))$.

3.2 Providers

For providers, deploying a single instance of s_i on the server farm incurs a cost c_i. As more instances are deployed, the cost can change based on the following function: $f(c) : I_k \rightarrow C_k$, where I_k is the definition domain of the function containing instances i_i used for the replication of s_i and C=$[c_1, c_2, c_3, ..., c_k]$ is domain of values identifying costs c_i of replication. The cost function $f(c)$ depends on the particular type of infrastructure that is being used in the server farm.

A server farm has a failure rate α calculated as the number of requests successfully processed within a time interval. In particular, we use an SLA (see figure 1) – SLA_{Cl}^{Pr} as an agreement between the client and the provider where the provider Pr seeks to minimise a cost function $f(c)$, whilst also maximising fault tolerance, to ensure that agreement SLA_{Cl}^{Pr} is complied with, while the client Cl seeks to receive the service capability described in SLA_{Cl}^{Pr}. In case of non-compliance with SLA_{Cl}^{Pr}, the penalty mechanism is applied for adjusting the difference in price payed based on terms agreed in the SLA. Hence, the cost of service provisioning from a provider's perspective can be expressed as: (i) the cost of access to resources within the server farm SP, and (ii) the price paid by the client Cl adjusted based on the penalty paid when the SLA cannot be complied with. *Cost for providers* are represented by: (i) The adjustments made due to possible penalties PEN – based on non-compliance with SLA_{Cl}^{Pr}; (ii) the

price P paid to the server farm owner Fo, SLA_{Pr}^{Fo} for deploying replicas. Hence, $C(Pr) = PEN(SLA_{Cl}^{Pr}(s_i)) + P(SLA_{Pr}^{Fo}(s_i))$.

Profit for providers is represented by the difference between the price of the SLA V and the cost affected for outsourcing the service into the server farm. Therefore, $P(Pr) = V(SLA_{Cl}^{Pr}(s_i)) - P(SLA_{Pr}^{Fo}(s_i)) - PEN((SLA_{Cl}^{Pr}(s_i)))$.

3.3 Server Farm Owner

Each server farm SF_i has a failure rate such as $SF = r_{k'}/r_k$ where $r_{k'}$ defines the number of successful requests of SF_i over the interval Δt and r_k represents the number of total requests. We consider that a provider Pr acts as a client for the server farm owner. Between the providers and farm owners we use SLA_{Pr}^{Fo} as an agreement which specifies terms and conditions for deploying service replicas on the server farm – hence: (i) the adjustments due to penalties PEN imposed by SLA_{Pr}^{Fo}; (ii) the cost incurred for deploying n instances based on SLA_{Pr}^{Fo}. This leads to the relation: $C(Fo) = PEN(SLA_{Cl}^{Pr}(s_i)) + n * \sum_{i=1}^{n}(c_i/i_i)$

Profit for farm owners is represented by the difference between the price of the SLA V and the cost affected with the number of instances needed for replication. Hence, $P(Fo) = V(SLA_{Pr}^{Fo}(s_i)) - n * \sum_{i=1}^{n}(c_i/i_i) - PEN(SLA_{Cl}^{Pr}(s_i)$ It is important to note that the cost of an instance in calculated as a product between the price with the instance p_i and the latency of response l: $c_i = p_i * l$. The protocol for calculating the latency of an instance is presented in algorithm 1.

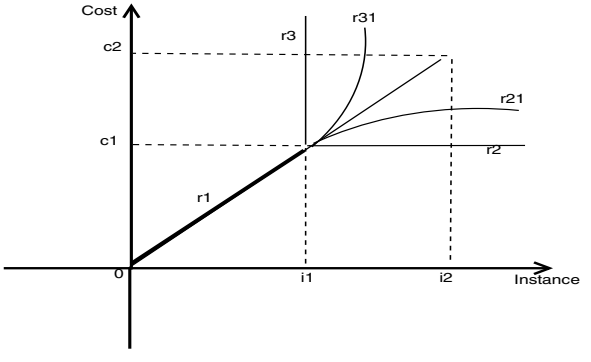


Fig. 2. Cost models

3.4 Cost Models

Depending on the number of replicas requested, the cost per replica can change – as illustrated in figure 2. Hence, the cost per replica can change based on a:

- Linear Model: defines a proportional increase in cost for each new instance deployed. In figure 2 this is identified by curve r_1. In the linear model an increase in the number of instances over intervals $[0, i_1]$ and $[i_1, i_2]$ produces an increase of costs identifies as $[0, c_1]$ and $[c_1, c_2]$.
- Decay Model: in this instance the cost of deploying new instances decrease (per instance) and eventually become stable after a threshold number of instances have been deployed. This implies that as more replicated instances are added for each service (up to the capacity available in the server farm), the system management and deployment costs do not increase. In figure 2 this is identified by curve r_{21}. In the decay model increasing instances over interval $[0, i_1]$ leads to an increase in costs over $[0, i_1]$, with a subsequent reduction of costs $[c_1, c_2]$ for instances $[i_1, i_2]$.
- Mixed Model: identifies a mixture between the linear model and the decay model. In figure 2 this is identified by curves r_2 and r_3. The mixed model can identify either an increase of costs $[c_1, c_2]$ when instances $[i_1, i_2]$ are requested –the case of r_3 or r_{31} or a reduction – r_2 or r_{21} of costs $[c_1, c_2]$ when instances $[i_1, i_2]$ are requested.

The particular cost model that is applicable depends on the infrastructure being used within a server farm. For instance, in a virtualized environment, adding more virtual machines (VMs) – up to a threshold limit – to each physical machine may not incur any additional cost (especially where replicas are being considered). There is an initial cost of transferring and instantiating a machine image, initiating and deploying the VM, etc. Once this has been done, additional VMs may incur less cost.

3.5 Configuration

In our approach we assume that each SLA_{Cl}^{Pr} identifies a request for a *single* service, however a server farm may be hosting multiple types of services (each of which may have multiple replicas). When a service request is submitted at time t_i a number of replicas within the system may fail. We investigate how the system reacts when: (i) a number of SLAs - SLA_{Cl}^{Pr} are executed in the server farms generating a load on the system. Service execution starts at a particular time interval defined by a frequency ν; (ii) during execution, k replicas of a service may fail with probability p; (iii) each provider replicates service s_i based on the cost function $f(c)$. We use latency as a metric to measure the reaction of the system in the context of a certain load determined by the number of SLAs being executed. For triggering failures on the peer-nodes, we use a fault probability p and an associated failure interval. Algorithm 1 explains how the latency is calculated. Latency is calculated based on the number of replicate instances and the number of requests submitted to the server farm. The averaged latency is then included in the calculation of the overall cost associated with an instance.

Algorithm 1. Enactment Protocol for Cost Calculation

```
1: for all i = 0; i < instancesNr; i++ do
2:    SELECT node n_k
3:    for all j = 0; j < totalRequests; j++ do
4:       requestIssuingTime = getClientIssuingTime();
5:       requestExecutionTime = getExecutionTime();
6:       requestExecutionLatency = requestExecutionTime − requestIssuingTime;
7:       if requestExecutionTime > 0 then
8:          averageRequestExecutionLatency+ = requestExecutionLatency;
9:          numberOfExecutedRequests++;
10:      else
11:         numberOfPendingRequests++;
12:      end if
13:   end for
14:   averageRequestExecutionLatency =
      averageRequestExecutionLatency/numberOfExecutedRequests
15: end for
```

4 Evaluation and Results

Validation of our approach has been carried out through simulation, using a P2P based resource sharing model. P2P systems present two important features: (i) scalability and (ii) dynamism. We make use of PeerSim – a scalable simulation environment that enables the definition of a number of different scenarios. In PeerSim, the protocols may either be implemented using a predefined PeerSim API or they can embedded into a real implementation [3]. PeerSim offers important modularity facilities as well as flexibility to support a variety of different system configurations. The P2P network is modelled as a collection of nodes, where each node has a list of associated protocols. The overall simulation is regulated through initializers and controls – that allow either events to be introduced into the simulation or to introduce particular capability at particular simulation time points.

We consider a community where clients, providers and server farm owners can establish SLAs to support service provisioning. We evaluate this community based on the cost incurred by each participant for: (i) acquiring the service–the primary action performed by clients, (ii) outsourcing the service–the primary focus of providers and (iii) instances of replication – the key activity carried out by server farm owners. Each client can request several services and each service has a number of replicated instances. Services are delivered on the basis of pre-established SLAs among peers. In our experiment we consider two metrics:

1. Cost such as: $C = C(Cl) + C(Pr) + C(Fo)$ where$C(Cl)$ represents the cost incurred by the client, $C(Pr)$ is the cost to the provider and $C(Fo)$ identifies the cost incurred by the server farm owner.

2. Profit such as: $P = P(Pr) + P(Fo)$ where $P(Pr)$ represents the profit of the provider and $P(Fo)$ represents the profit of the server farm owner.

We carry out a series of experiments to validate how the costs identified above are impacted by different types of faults in the system. Each experiment attempts to evaluate a particular objective.

Experiment 1. In this experiment we investigate how the overall cost and profit within the community is affected when a number of replicas fail – based on a failure rate parameter. Figure 3 illustrates how the cost and profit evolve with different failure rates. We can observe that the cost and the profit in the system are significantly affected when a high number of replicas fail. This experiment considers the failure rate within the following set: 0.01, 0.05, 0.09, 0.5.

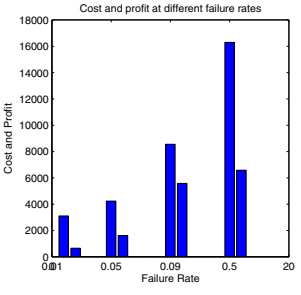


Fig. 3. Cost and Profit at different failure rates

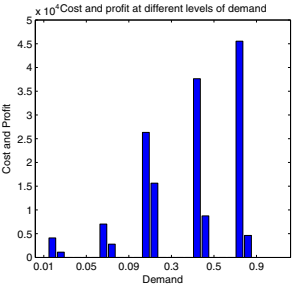


Fig. 4. Average latency at different levels of demand

It can be observed that the overall cost increases with fault probability. We observe the difference of cost when considering failure rates over the range 0.09-0.5. The cost is considerably increased for 0.5 while the profit remains stable. This arises because when multiple service instances fail, the penalties associated with SLA non-compliance are triggered, leading to an increase in the total cost.

Experiment 2. In this experiment we identify how the profit/loss are affected when the demand for services increases. Demand for services in this case is identified based on the number of service execution based on pre-defined SLAs. This experiment demonstrates how demand affects the overall community(service client, owner and server farm owner) in terms of cost and profit.

From figure 4 we observe how the distribution of cost and profit evolves in relation to different levels of demand. When using *a regular demand* in the system (demand=0.01), the cost and the profit remain stable. When increasing the demand to approximately 30 SLAs (demand=0.09) a significant impact of the cost and profit is identified. The highest impact in the distribution of cost and profit occurs when using a load of 120 SLAs (demand=0.09). From this experiment we can conclude that an increased demand in the system represents a factor of risk in the context of service deliveries.

Experiment 3. Previous experiments demonstrate how risk within the system increases when dealing with an increased demand. In this experiment we consider the rate at which new demand can be introduced into the system. It is important to note that the frequency of demand identifies the intervals between service executions. This experiment investigates how a variability in the *demand frequency* can affect the cost and the profit within the system.

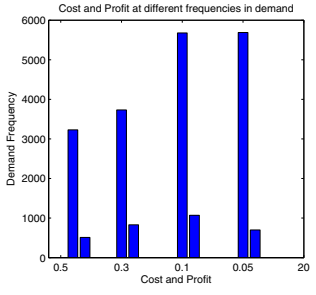


Fig. 5. Average latency when varying demand frequency

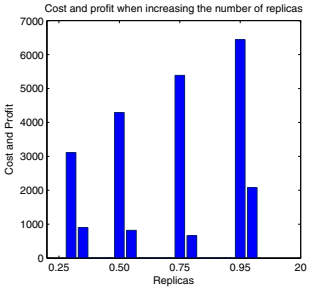


Fig. 6. Average latency when varying the cost

Increasing the demand frequency over a specific time interval represents another factor which can affect the cost and the profit of the system – illustrated in Figure 5. When demand frequency increases, the average latency increases, whereas when service executions are separated by longer intervals, this leads to lowered costs. We observe that the profit is stable while the cost increases in direct proportion to the demand frequency – as the latter implies a higher load on the server farm therefore a higher cost.

Experiment 4. Previous experiments demonstrate how the system changes when providers are dealing with a certain number of replicas (see section 3). In this experiment we extend the number of replicas and observe how the cost and profit are distributed. Figure 6 illustrates the distribution of cost and profit when increasing the number of replicas by 25%.

5 Conclusion

The emergence of Cloud computing deployment strategies enables us to differentiate between a service owner and an infrastructure provider, where a service owner may utilize the resources of an infrastructure provider to deploy a service. Where the relationship between these two actors (service owner and infrastructure provider) is not based on trust (i.e. based on experience gained in previous interactions), it is often necessary to establish an SLA. Such an agreement protects the service owner if the infrastructure provider is unable to deliver their advertised capability. We consider the financial risk that would be incurred by

both the service owner and the infrastructure provider, based on the price paid for the service by a client, the penalty incurred due to non-compliance with the SLA and the deployment cost for running multiple replicas on the infrastructure.

Through simulation we demonstrate that as the number of failures in the system increase, this can have significant impact on the distribution of cost and profit between the actors. We show that the demand for services represents another factor which can determine the level of profit/loss within the community. When an SLA is associated with each service execution, we demonstrate that the higher the frequency of SLA violations the higher the variation in costs – based on the deployment, penalties and replication costs present within the system. We increase the number of replicas in the system to increase service availability – focusing on a single server farm. The approach we present can be extended to multiple server farms – operating in different geographical data centres (an approach adopted by Amazon.com as part of their *Availability Zones*).

Acknowledgements. Gheorghe Cosmin Silaghi acknowledges support from the Romanian National Authority for Scientific Research under the project TE 316.

References

1. Petri, I., Rana, O., Cosmin Silaghi, G.: SLA as a Complementary Currency in Peer-2-Peer Markets. In: Altmann, J., Rana, O.F. (eds.) GECON 2010. LNCS, vol. 6296, pp. 141–152. Springer, Heidelberg (2010)
2. Yang, Z., Tian, J., Zhao, B.Y., Chen, W., Dai, Y.: Protector: A Probabilistic Failure Detector for Cost-Effective Peer-to-Peer Storage. IEEE Trans. Parallel Distrib. Syst. 22(9), 1514–1527 (2011)
3. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems 23(3), 219–252 (2005)
4. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3), 8 (2007)
5. Djemame, K., Padgett, J., Gourlay, I., Armstrong, D.: Brokering of risk-aware service level agreements in Grids. Concurrency and Computation: Practice and Experience 23(13), 1558–1582 (2011)
6. Djemame, K., Gourlay, I., Padgett, J., Birkenheuer, G., Hovestadt, M., Kao, O., Vo, K.: Introducing risk management into the grid. In: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (eScience 2006). IEEE Computer Society, Amsterdam (2006)
7. Li, B., Gillam, L.: Risk Informed Computer Economics. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009, pp. 526–531. IEEE Computer Society, Washington, DC (2009)
8. Eymann, T., Konig, S., Matros, R.: A Framework for Trust and Reputation in Grid Environments. Journal of Grid Computing 6(3), 225–237 (2008)
9. Li, X., Ling, L.: PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. IEEE Transactions on Knowledge and Data Engineering, 843–857 (2004)

10. Elwaer, A., Taylor, I., Rana, O.: Preference driven Server Selection in Peer-2-Peer Data Sharing Systems. In: 4th Int. Workshop on Data-Intensive Distributed Computing (DIDC) – Alongside the HPDC Conference, San Jose, California, USA (2011)
11. Petri, I., Rana, O., Rezgui, Y., Cosmin Silaghi, G.: Evaluating Trust in Peer-to-Peer Service Provider Communities. In: Proceedings of TrustCol Workshop, Orlando, Florida, USA (October 2011)
12. Risch, M., Altmann, J.: Cost Analysis of Current Grids and Its Implications for Future Grid Markets. In: Altmann, J., Neumann, D., Fahringer, T. (eds.) GECON 2008. LNCS, vol. 5206, pp. 13–27. Springer, Heidelberg (2008)

A Game-Theoretical Approach to the Benefits of Cloud Computing

Jörn Künsemöller[1] and Holger Karl[2]

[1] University of Paderborn, Graduate Program Automatisms, Warburger Str. 100, 33098 Paderborn, Germany
[2] University of Paderborn, Computer Networks Group, EIM-I, Pohlweg 47-49, 33098 Paderborn, Germany

Abstract. In order to answer the question whether or not to utilize the cloud for processing, this paper aims at identifying characteristics of potential cloud beneficiaries and advisable actions to actually gain financial benefits. A game-theoretic model of an *Infrastructure-as-a-Service (IaaS)* cloud market, covering dynamics of pricing and usage, is suggested. Incorporating the possibility of hybrid clouds (clouds plus own infrastructure) into this model turns out essential for cloud computing being significantly in favor of not only the provider but the client as well. Parameters like load profiles and economy of scale have a huge effect on likely future pricing as well as on a cost-optimal split-up of client demand between a client's own data center and a public cloud service.

Keywords: Cloud Computing, Markets, Game Theory.

1 Introduction

The question if cloud computing is a suitable alternative to a company-owned data center is often regarded as highly individual and general assessments as unapplicable in this context. Although a potential benefit of going to the cloud can be expressed in numbers, this is usually done in the form of case studies. While privacy concerns and tactical decisions are to be balanced against temporary cost efficiency in particular cases, a more analytical perspective on the topic provides a general understanding of the financial aspect. The development of a market model helps to better understand provider's and client's potential benefits from utilizing an *Infrastructure-as-a-Service (IaaS)* cloud. As these benefits depend on both participants' behavior, game theory can suggest likely or advisable behavior by contrasting their possible courses of actions.

By exploring crucial parameters, this paper aims at general conclusions of characteristics (load, efficiency, grade of automatization, ...) which a prospective cloud beneficiary may feature and attempts an estimation of future pricing. Such knowledge may shorten the market forming process as well as prevent wrong decisions taken under false presumptions on both sides. Unlike the usual case studies, decisions can be made not only under current conditions, but also under those that are likely to come.

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 148–160, 2012.

Based on a cost estimation stated in Section 3, several aspects are discussed and incorporated in a market model throughout Section 4. The final model and resulting market outcome are stated in Section 5.

2 Related Work

Several publications deal with the suitability of cloud services for different applications, especially as a substitute for on-site corporate IT. Guidelines for orderly decision making like in [8,10] usually include a financial comparison of feasible solutions. Calculation models for *total costs of ownership (TCO)* of a data center [3,9] can be taken as a basis for this. There also are ready-to-use calculators [1] for a direct comparison of expected costs based on specified demand. Other comparative work focuses more on technical than financial issues [7].

Some research addresses cloud related problems using game theory. Most of it focuses on algorithmic solutions in resource management, e.g. [11,18]. Applying game theory on cloud service negotiation is discussed in [19] by representing it as a bargaining problem. Cloud pricing is targeted by [2] using an evolutionary approach and is also covered in [15] using agent-based simulations in the setting of software services.

3 Cost Estimation in a Case Study

Several calculation models for comparing the cost structure of cloud services and local data centers exist [1,3,9]. The German IT magazine *iX*, for example, published a case study for a hypothetical company, estimating the TCO of a new data center or co-location setup versus Amazon's *EC2* service [3]. The case study chose values for critical factors that made EC2 win this cost comparison hands down. As those factors might change over time, the conclusion from this study also may change. In the study, costs per year for an owned data center consist of investment cost amortization and running costs. Investments are acquisition costs for server and network hardware and operation system licenses (3 years write-off) as well as infrastructure and building costs (15 years write-off). Running costs are maintenance, power, administration, and data transfer. EC2 costs, on the other hand, consist of instance costs to meet processing demand plus data transfer. Instance price is set to 0,22 €/h. The processing capability of each server is regarded as equivalent to two EC2 instances.

The cost model underlying this (and similar) case studies can be easily generalized into two linear relationships:

$$\mathrm{DC}_{\mathrm{costs}}(x) \approx 7150\,€ \cdot x \tag{1}$$

$$\begin{aligned} \mathrm{Cloud}_{\mathrm{costs}}(a, x) &= p \cdot a \cdot x \cdot C \cdot 24 \cdot 365 \\ &\approx 3850\,€ \cdot a \cdot x \end{aligned} \tag{2}$$

where x is a number of servers, p is the cloud instance price per per hour, $a \in [0 \ldots 1]$ is average load and C is the processing capability of a server in instances.[1]

The constant values in these equations are of course debatable; it is the point of this paper to make the discussion about cloud benefits less dependent on such constants. For example, the values here indicate that a cloud would always be cheaper, irrespective of the workload. In more general settings, this is of course not necessarily the case.

To keep things simple, co-location (buy servers and lease facilities and administration for operation) is not included as an option. It is not regarded as essentially different to an owned data center (buy servers and build facilities, employ administration personnel for operation) and briefly discussed in Section 6. Also, we shall use $\text{Cloud}_{\text{costs}}$ as a representative for all kinds of cloud offerings, not necessarily restricted to the Amazon cloud as such.

4 Towards a Market Model

4.1 Pricing

When one tries to anticipate the development of the cloud market, models as in Section 3 that assume constant prices are insufficient. As long as a cloud provider chooses a price that results in lower cost in the cloud compared to an own data center even at full workload ($a = 1$), we can expect the demand to stay constant. For any price p that causes $\text{Cloud}_{\text{costs}}(x, 1) > \text{DC}_{\text{costs}}(x)$ at full workload ($a = 1$), there is a *break-even workload* $a(p) < 1$ at which $\text{Cloud}_{\text{costs}}(x, a(p)) = \text{DC}_{\text{costs}}(x)$. This break-even workload is a decreasing function of the price p. For customers with a workload higher than the break-even workload (i.e., $a > a(p)$), it is unattractive to use the cloud at the given price. Similar to this, a *break-even price* $p(a)$ can be defined as the price at which $\text{Cloud}_{\text{costs}}(x, a) = \text{DC}_{\text{costs}}(x)$, which is a decreasing function of a given workload a. A price higher than break-even price makes the cloud more expensive than the data center option for a client with the given workload. Thus, pricing unsurprisingly affects sales volume as well as profit margin. Long-term pricing will in all likelihood maximize the product of these factors, as providers want to maximize their profit. How to determine the best price for more complex load situations and decision options is part of the question in dispute.

4.2 A Game-Theoretical Setup

An interaction model that maps both contractors' courses of action using game theory is hereby suggested; the goal is to estimate future pricing. To start simple, the model is set up as one player being the client under the case study's premises. The client has the options of building its own data center or use the cloud.

[1] More general cost models, e.g., a model where $\text{Cloud}_{\text{costs}}(a, x) = c_1 ax + c_2 a + c_3 x$ are easily conceived. We restrict the discussion in this paper to a linear modest; yet our approach should carry over to such affine cost models as well.

The cloud provider, as the opposing player, might have three discrete pricing possibilities (Fig. 1). Payoffs are based on client costs and provider revenues (in million euro per year). An *extensive form game* [13] is used, as the provider is making its offer and subsequently the client is free to accept it or not.

The provider will gain most when asking for the highest price at which cloud usage is anticipated. Although the game features several strategy combinations in a Nash equilibrium [12], most of them are incredible: The client will accede to the cloud agreement when an offered price causes lower costs and build a data center otherwise. Any other statement is an incredible thread and can be safely ignored. Incredible equilibria can be sorted out by asking for *subgame perfection* [16] which means that strategies in equilibrium have to remain best responses throughout all possible in-game situations. The highest possible price $p <$ break-even price in combination with the client using the cloud when $p <$ break-even price and building a data center when $p \geq$ break-even price is the only subgame perfect Nash equilibrium of the game. This equilibrium's outcome is highlighted in Fig. 1.

Any other offer in addition to the three price examples is possible, of course. To include all courses of action, continuous strategies come into play, with one continuum being the price between two discrete extreme choices. The price in subgame perfect equilibrium is right below the break-even price from Section 4.1. Any savings by using cloud services would be devoured by the provider in its pursuit of profit (Section 6).

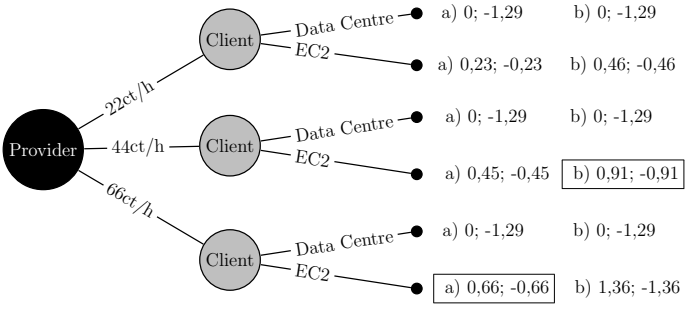


Fig. 1. A simple cloud model limited to three pricing possibilities. Payoffs are based on the case study with (a) 2 % and (b) 50 % peak load. (Order: Provider; Client; Highlights: Outcome of the subgame perfect Nash equilibrium)

4.3 Combining Cloud and Data Center

As implementations for local deployment of cloud services become available (e.g. *Eucalyptus* [4] or *Openstack* [14]) a client might want to combine a *private cloud* (build an own data center providing cloud services) and the *public cloud* (internet cloud services) to meet its demand. Usually, some capacity is always in use (base load) and some capacity is idle at times (peak load). The phrase *peak volume* refers to the total processing demand which exceeds base load (overall processing

demand minus base volume). Peak volume is disproportionally expensive to self-provide, as costs are only amortized over the time the necessary capacity is actually used. Thus, building a smaller data center to meet base load and buy instances from the cloud to meet peak demand could be a sensible choice. This also means that higher prices can be tolerated for public cloud instances.

From a client's perspective, all options seem to be reasonable in their own range of cloud pricing (Fig. 2): Naturally, complete outsourcing is most attractive when the service is really cheap. Instead of one break-even price up to which the cloud is more affordable (Section 4.1), there are two others: A $\text{Price}_{\text{low}} <$ break-even price, up to which the exclusive cloud solution is cheaper than a combination of cloud services with an own data center catering for base load. And a $\text{Price}_{\text{high}} >$ break-even price, up to which the combination is cheaper than an own data center covering all demand. The more peak volume the client demands, the less do $\text{Price}_{\text{low}}$ and $\text{Price}_{\text{high}}$ differ.

Both break-even prices give incentive to change the solution when exceeded. These solution changes come along with a break-in of demand and thus in total profit. Hence, there are two pricing candidates for the provider's profit maximization, when including this possibility of combining an own data center and public cloud services: The lower price is right below $\text{Price}_{\text{low}}$ the higher price is right below $\text{Price}_{\text{high}}$. While the latter depends on specific demand, $\text{Price}_{\text{low}}$ is not influenced by the load profile (only depends on data center costs of the client). It can be determined by identifying $\text{DC}_{\text{costs}}(x)$ with $\text{Cloud}_{\text{costs}}(1, x)$:

$$\text{Price}_{\text{low}} = \frac{\text{DC}_{\text{costs}}(1)}{24 \cdot 365 \cdot C} \tag{3}$$

where C is the capability of a server in instances.

At this price, the revenues generated by the client's peak volume are higher than those which could be gained from covering all volume at the lower price. Hence, there is no appeal to offer the latter. For a load profile that results in break-even in costs of cloud and the combined solution at higher annual costs (higher average load), this might turn out differently (Section 4.4). Either way, provider revenue is lower than data center costs and thus lower than they are without the client's possibility of combining private and public cloud. For the client, this possibility is only profitable if it does not have to be implemented (as the lower price is offered).

4.4 Different Load Profiles

Limiting a combined solution to processing base load in a private and all peak volume in the public cloud is too simple. Load can be described using a function $\text{time}(a)$, where a is some amount of load and the function returns the amount of time, the data center is under a load of a or more (Fig. 3). A resulting load curve usually features a smooth transition between peak and base load. Thus, several capacities of a data center might be reasonable for combination with a public cloud: Higher instance prices justify a larger data center when extra capacity

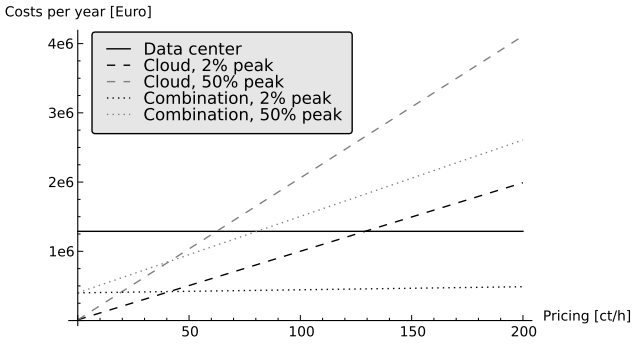


Fig. 2. From a client's perspective, all options can be worthwhile

is used effectually (meaning that substituting usage of this extra capacity with cloud instances would be more expensive). Those client possibilities are covered by another continuum: The fraction of overall capacity ($\mathrm{DC}_{\mathrm{frac}}$) that is met by a data center (private cloud). It extends from an exclusive use of an own data center to utilization of cloud instances for all demand. A value of 0.4 means an own data center capable of meeting 40% of peak capacity and using public cloud instances to meet any further demand. The cost optimal combination under certain pricing is given, when:

$$\begin{aligned} &\mathrm{DC}_{\mathrm{costs}}(x) = \mathrm{Cloud}_{\mathrm{costs}}(\mathrm{time}(DC_{frac}), x) \\ &\Rightarrow \mathrm{DC}_{\mathrm{frac}} = \mathrm{time}^{-1}\left(\frac{\mathrm{DC}_{\mathrm{costs}}(x)}{\mathrm{Cloud}_{\mathrm{costs}}(1, x)}\right) \end{aligned} \tag{4}$$

With $\mathrm{time}^{-1}(z)$ being 0 for an $z \geq 1$ (any price $\leq \mathrm{Price}_{\mathrm{low}}$). As the client is reducing the cloud share of its demand when price is increased, there is no break-even price of one specific combined solution and a data center ($\mathrm{Price}_{\mathrm{high}}$) as a credible price candidate. The best price has to be determined by identifying the most desireable of all combined solution that can be anticipated. This can be done by backwards induction: $\mathrm{Price}_{\mathrm{best}}$ is the price that maximizes revenue from the expected amount of usage under this pricing ($\mathrm{Price}_{\mathrm{best}}$ might be equal to $\mathrm{Price}_{\mathrm{low}}$):

$$\begin{aligned} &f'(\mathrm{Price}_{\mathrm{best}}) = 0 \quad \text{and } f''(\mathrm{Price}_{\mathrm{best}}) < 0 \\ &\text{where } f(\mathrm{price}) = \mathrm{price} \cdot \int_{\mathrm{DC}_{\mathrm{frac}}}^{1} \mathrm{time}(z)dz \end{aligned} \tag{5}$$

4.5 Provider Profit

Payoffs are based on provider's revenues so far. Although growth might be a worthy business objective, revenue as such is in vain if the business is not

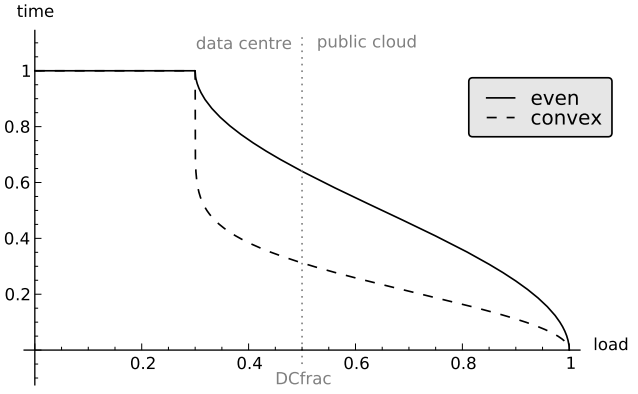


Fig. 3. Two prototypical load profiles. Base load (here: 30%) is always in use, higher utilization occurs more and more rarely.

profitable in the end. Thus, it is important to include provider costs into the model to have its payoffs based on profit. The provider actually has to build a data center itself to provide the service and provider costs can be estimated like the client's data center option in general.

In [3] the following factors are considered: Network hardware is assessed at 20% of server cost, maintenance at 10% of server and network hardware costs. Power consumption of a server is estimated at 50% of its specification, consumption of network gear at 44% of server consumption. For the total data center consumption these values are multiplied by a PUE (Power Usage Effectiveness) factor of 2,5 and assessed at 0,1€/kWh. The building and infrastructure investments are measured at 2024€/m^2 (2,84 m^2 per rack) and 16200€ per kW hardware power consumption (referring to uptime institute [17]). Administration costs are set to 73000€ a year per administrator, each one capable of covering 80 servers. Data transfer is included at a flat rate of 400€/month.

The provider has to operate equivalent hardware to provide a service of the same capabilities which a client's local data center would feature. Without some cost advantage in operation, a profitable offer at $\text{Price}_{\text{low}}$ would be impossible.

For a huge provider, there is a cost advantage due to economy of scale: According to James Hamilton from *Amazon Web Services*, the PUE of a large data center (50000+ servers) is between 1,2 and 1,5 and in comparison to a mid-sized data center (∼1000 servers) admin costs can be reduced by factor 7 due to automatization [6]. Compared to the case study's accounting, this reduces average costs per server by about 18%. A little over 20% would be theoretically possible with full automatization (no admin costs) and an ideal PUE of 1. Advantages of location like cheaper power or building costs might gain further benefits.

On the other hand, letting out instances on demand comes along with unpaid idle time. To compensate for the investment, time under utilization would have

to be more expensive than $\text{Price}_{\text{low}}$. But a provider is serving several clients (Section 5.2). Probably, these clients all have peaks but they are likely to be scattered over time. Hence, the provider does not have to operate a reserved amount of servers for each client: They can be overbooked (statistical multiplexing). There is no need to operate the same amount of servers a client would have to, but only the amount to meet average demand (in the best case). The provider runs the risk of not fulfilling its service level agreements; the probability depends on how much peaks are correlated in time. There is the risk of several client's peaks happening at the same time, of course, but the larger the number of clients the less variation from overall average demand has to be expected (law of large numbers). Demand correlation can be an issue, though (Section 6).

Costs of extending the provider's center to meet additional demand are:

$$\text{DC}_{\text{costsPr}}(a, x) = \text{DC}_{\text{costs}}(v \cdot a \cdot x) \cdot \text{EoS}_{\text{factor}} \tag{6}$$

where x is a number of servers, a is average load, v is the variation in all demand the provider meets (multiple of average demand expected at maximum) and $\text{EoS}_{\text{factor}}$ is the fraction of client data center costs that a provider has to spent on a data center (extension) of the same capabilities due to economies of scale.

As v goes to 1 when a provider serves a very large number of clients of huge diversity, a huge provider can offer $\text{Price}_{\text{low}}$ and still be profitable. A smaller provider with fewer clients, on the contrary, has to operate a larger amount of reserve capacity for variation and pass on costs to actually sold instances. $\text{Price}_{\text{low}}$ becomes unprofitable at a $v > (1/\text{EoS}_{\text{factor}})$.

5 The Final Game Model

Fig. 4 shows the game model which is developed in Section 4: The provider chooses a price and subsequently the client is free to use the service on these terms by any amount in combination with a private cloud.

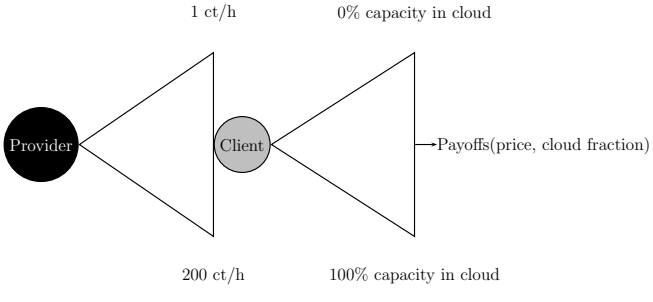


Fig. 4. A market model of the EC2 market using continuous strategies

5.1 Payoffs and Subgame Perfect Nash Equilibrium

Payoffs are determined as follows, where x is the number of necessary servers to meet peak demand, $\mathrm{DC}_{\mathrm{frac}}$ indicates the client's choice of what fraction of peak demand an owned data center handles and time(z) is load as described in Section 4.4. $\mathrm{DC}_{\mathrm{costs}}$ and $\mathrm{Cloud}_{\mathrm{costs}}$ are defined in Section 3 and $\mathrm{DC}_{\mathrm{costsPr}}$ in Section 4.5.

$$\begin{aligned}\text{Client's payoff} = &-\mathrm{DC}_{\mathrm{costs}}(x \cdot \mathrm{DC}_{\mathrm{frac}}) \\ &- \mathrm{Cloud}_{\mathrm{costs}}(\int_{\mathrm{DC}_{\mathrm{frac}}}^{1} \mathrm{time}(z)dz, x)\end{aligned} \tag{7}$$

$$\begin{aligned}\text{Provider's payoff} = &\ \mathrm{Cloud}_{\mathrm{costs}}(\int_{\mathrm{DC}_{\mathrm{frac}}}^{1} \mathrm{time}(z)dz, x) \\ &- \mathrm{DC}_{\mathrm{costsPr}}(x \cdot \int_{\mathrm{DC}_{\mathrm{frac}}}^{1} \mathrm{time}(z)dz)\end{aligned} \tag{8}$$

The game features a subgame perfect Nash equilibrium within the combination of $\mathrm{Price}_{\mathrm{best}}$ and the client combining public and private cloud services in a cost-optimal split-up (Section 4.4). $\mathrm{Price}_{\mathrm{best}}$ is equal to $\mathrm{Price}_{\mathrm{low}}$ at minimum.

5.2 Future Pricing

The former considerations are based on the demand of one client. To make a statement on probable long-term pricing, the dynamics between provider and all potential clients have to be reckoned. The single client is replaced by an aggregation of all potential clients. This meta-client has distinct processing demand, formed by combining all individual demand in the market: The demanded peak capacity x is the sum of all individual peak capacities and load can be described by a time(x) function which is a weighted (by individual capacity) average of individual load. The equilibrium price does not target a maximization of profit gain from any individual client. But $\mathrm{Price}_{\mathrm{best}}$ based on this average load curve is the best trade-off. While research on the distribution of load profiles and associated demand volume is needed to tell how its agglomeration would look like, what was said about the single-client-situation can be generalized to client aggregation: A client's best response on certain pricing remains unaltered by different load profiles up to instance costs which – when paid all along – correspond to the costs of a data center equivalent: $\mathrm{Price}_{\mathrm{low}}$. A higher price leads to a break-in of demand, as base load (lots of volume) is handled by owned data centers. Outsourcing processing peaks generates sloping demand (Fig. 5).

How exactly the demand wears off over price depends on the load profile. In consequence, the load defines if revenues at higher prices retain a volume which allows exceeding profit compared to $\mathrm{Price}_{\mathrm{low}}$. Profit at $\mathrm{Price}_{\mathrm{low}}$ is gathered from mass usage at a comparatively low margin. As discussed in Section 4.5, the data center cost advantage is determined by economies of scale and expected demand variation. At higher pricing, these factors become less important for the margin and thus better economies of scale make $\mathrm{Price}_{\mathrm{low}}$ more likely (Fig. 6).

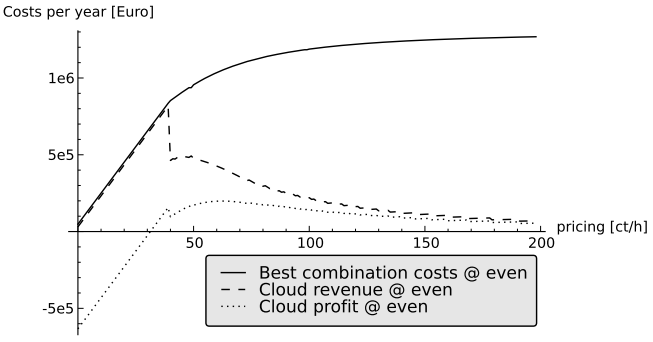


Fig. 5. Best response costs and their EC2 share and profit ($\text{EoS}_{\text{factor}} = 0{,}8$; $v = 1$)

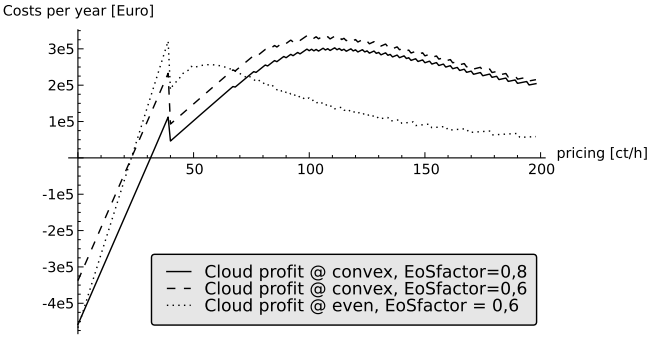


Fig. 6. Expected provider profit at different economies of scale and load ($v = 1$)

6 Discussion and Conclusions

Long-term pricing should be considered when decisions are taken. Cloud services might be cheap today, but things could look different as soon as cloud services are established and have to prove themselves as sustainably profitable. Likely pricing can be estimated based on data of average load in the market (the meta-client in Section 5.2). Each client can determine its individual, cost-optimal, combined solution based on this price. Not using the cloud at all is never favorable, although the suggested combined solution might contain a very small public cloud share. In general, all clients derive advantage from cloud computing, but those with higher-than-average load gain less than the average client.

Provided that the accounting assumptions are correct (especially that two EC2 instances are equivalent to one of the accounted servers), the EC2 service is currently not cost-effective. Instead, a price raise to 0,40 €/h is likely in the future. Assuming an average load in the market equal to *convex* (Fig. 3), an even higher instance price of over 1 €/h is more profitable (and is likely to be asked once the provider should achieve a monopoly position). At this instance price,

a client featuring *even* load would be best off building a data center capable of about 75% of its peak capacity and outsource exceeding demand to the cloud.

As stated before, demand is very robust up to $\text{Price}_{\text{low}}$ which is determined by the total costs of ownership of the clients' hypothetical data centers. Pricing that targets peak demand instead of mass usage depends on good market knowledge and is thus easily misapplied. Regarding this, a provider might as well refrain from this option if its gain is not considerable.

In order that both parties benefit from the cloud, it is important to easily combine instances from several sources (public and private clouds). For some applications, data exchange between these sources might be an issue. If a combination with arbitrary shares is not granted by the service, $\text{Price}_{\text{best}}$ is right below DC_{costs} (Section 4.2) and thus far more likely to be in equilibrium. Although this option is minimizing overall idle time, which is good for environmental reasons, the provider takes all the benefit. It is noteworthy, though, that such a setup (Section 4.2) is quite similar to the ultimatum game [5]. Results in experimental economics differ from theory regarding these games and a price which is not delivering enough benefit to the client might be perceived as somehow inappropriate and is thus rejected.

Section 4.5 mentioned that a smaller provider is more likely to have idle times than a huge provider due to suboptimal overbooking. Another reason might be a huge share of clients demanding resources at the same time (e.g., at daytime in a single timezone). Having to operate reserve capacity diminishes the profit margin and if savings due to economies of scale are exceeded, $\text{Price}_{\text{low}}$ becomes unprofitable. This not only means that a very large provider with clients scattered all over the globe can expect more benefits from the cloud than a small one serving regional clients. It also makes it very hard to establish a competitor in the market as massive investments can be expected. The whole aspect of competition needs a more in-depth analysis, of course. Regarding these observations, though, an oligopoly appears to be the most preferable market from a client's perspective as undercutting other offers keeps margins low, while costs of production are low due to the providers' size.

Looking at the market as a whole, the provider's high degree of capacity utilization is saving idle time and thus saves unnecessary hardware. This is the main reason for the huge general benefit of the cloud and it depends on which price is chosen, whether it is in favor of the client ($\text{Price}_{\text{low}}$ – leaving only economies of scale to the provider) or both parties ($\text{Price}_{\text{best}} \neq \text{Price}_{\text{low}}$). With $\text{Price}_{\text{low}}$ so depending on good economies of scale, it is important to understand which aspects are persistently cheaper on a larger scale. It is quite possible that technologies which provide a better PUE become available to smaller centers in the future as well, for instance.

In the case study, co-location was discussed as an option (Section 3). It was omitted in this paper as being quite similar to the data center option. As a facility that houses several client's servers might be very large, economies of scale effects are in effect as well as they are in favor of a cloud provider. Overbooking

is not possible, though, as the servers are reserved for a specific client. As a consequence, the benefit which is shared amongst provider and clients is much smaller in comparison to cloud utilization. Providing inferior gain to both parties, the whole business model of co-location is challenged by the existence of cloud computing.

References

1. Amazon Web Services: Amazon EC2 Cost Comparison Calculator, http://aws.amazon.com/en/economics/
2. An, B., Vasilakos, A.V., Lesser, V.: Evolutionary Stable Resource Pricing Strategies (poster paper). ACM SIGCOMM (August 2009)
3. Christmann, C., Falkner, J., Kopperger, D., Weisbecker, A.: Schein oder Sein. iX special: Cloud, Grid, Virtualisierung, pp. 6–9 (February 2010)
4. Eucalyptus Systems, http://www.eucalyptus.com/
5. Güth, W., Schmittberger, R., Schwarze, B.: An Experimental Analysis of Ultimatum Bargaining. Journal of Economic Behavior & Organization 3(4), 367–388 (1982)
6. Hamilton, J.: Cloud Computing Economies of Scale, MIX 2010 talk (2010), http://channel9.msdn.com/events/MIX/MIX10/EX01/
7. Hazelhurst, S.: Scientific Computing Using Virtual High-Performance Computing: A Case Study Using the Amazon Elastic Computing Cloud. In: Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology, SAICSIT 2008, pp. 94–103. ACM, New York (2008)
8. Klems, M., Nimis, J., Tai, S.: Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. In: Weinhardt, C., Luckner, S., Stößer, J. (eds.) WEB 2008. LNBIP, vol. 22, pp. 110–123. Springer, Heidelberg (2009)
9. Koomey, J.: A Simple Model for Determining True Total Cost of Ownership for Data Centers (White Paper). Uptime Institute (2007)
10. Lamberth, S., Weisbecker, A.: Wirtschaftlichkeitsbetrachtungen beim Einsatz von Cloud Computing. In: Pietsch, W., Krams, B. (eds.) Vom Projekt zum Produkt. LNI, vol. 178, pp. 123–136. GI (2010)
11. Londoño, J., Bestavros, A., Teng, S.H.: Colocation Games: And their Application to Distributed Resource Management. In: HotCloud 2009: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, p. 10. USENIX Association, Berkeley (2009)
12. Nash, J.F.: Equilibrium Points in n-Person Games. Proceedings of the National Academy of Sciences of the United States of America 36(1), 48–49 (1950)
13. von Neumann, J.: Zur Theorie der Gesellschaftsspiele. Mathematische Annalen 100(1), 295–320 (1928)
14. Openstack, http://www.openstack.org/
15. Rohitratana, J., Altmann, J.: Agent-Based Simulations of the Software Market under Different Pricing Schemes for Software-as-a-Service and Perpetual Software. In: Altmann, J., Rana, O.F. (eds.) GECON 2010. LNCS, vol. 6296, pp. 62–77. Springer, Heidelberg (2010)

16. Selten, R.: Reexamination of the Perfectness Concept for Equilibrium Points in Extensive Games. International Journal of Game Theory 4, 25–55 (1975)
17. Uptime Institute, http://www.uptimeinstitute.com/
18. Wei, G., Vasilakos, A., Zheng, Y., Xiong, N.: A Game-Theoretic Method of Fair Resource Allocation for Cloud Computing Services. The Journal of Supercomputing, 1–18 (2009)
19. Zheng, X., Martin, P., Powley, W., Brohman, K.: Applying Bargaining Game Theory to Web Services Negotiation. In: IEEE International Conference on Services Computing, pp. 218–225 (2010)

Planning for Optimal Multi-site Data Distribution for Disaster Recovery

Shubhashis Sengupta and K.M. Annervaz

Accenture Technology Labs
Bangalore, India
{shubhashis.sengupta,annervaz.k.m}@accenture.com

Abstract. In this paper, we present DDP-DR: a Data Distribution Planner for Disaster Recovery. DDP-DR provides an optimal way of backing-up critical business data into data centres (DCs) across several Geographic locations. DDP-DR provides a plan for replication of backup data across potentially large number of data centres so that (i) the client data is recoverable in the event of catastrophic failure at one or more data centres (disaster recovery) and, (ii) the client data is replicated and distributed in an optimal way taking into consideration major business criteria such as cost of storage, protection level against site failures, and other business and operational parameters like recovery point objective (RPO), and recovery time objective (RTO). The planner uses Erasure Coding (EC) to divide and codify data chunks into fragments and distribute the fragments across DR sites or storage zones so that failure of one or more site / zone can be tolerated and data can be regenerated.

Keywords: Disaster recovery, planning, data distribution, optimization.

1 Introduction

In today's enterprise computing, data centers generate an overwhelming volume of data to support applications such as particle physics and large LHC [1], storing of web pages and indexes [2], engineering data of pharmaceutical and semi-conductor companies, large multi-national technology companies and hosting service providers. Disaster Recovery (DR) and Business Continuity planning (BCP) mandate that critical enterprise data is backed up periodically and is kept in geographically separate and secure locations. In the event of operational disruption at the primary site, the operation can be resumed at an alternate site where the backed up data and log files are shipped and applications / services can be instantiated again.

1.1 Data Backup Technology

Traditionally, the data backup and archival is done using magnetic tapes which are transported to a remote location. However, such procedure is manual and cumbersome and rapid data restoration and service resumption is often not possible.

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 161–172, 2012.

Recently, with the advent of cheap disk-based storage and advances in networking, remote online backup options have become attractive [3]. The storage area network and virtualization technology has become sophisticated enough to create a storage volume snapshot to a remote site [4]. Increasingly, open-source technologies such as RSync are being used achieve the same goals; albeit with a lower efficiency.

1.2 Multi-site Data Backup and Replication

With Cloud computing and cheap storage technologies, backup architecture is now becoming distributed in nature. Data is now getting backed up in multiple geographically separated data centres to improve fault tolerance and availability [5]. It is also found that remote storing of backup data to a single third-party storage provider may not be good due to security / reliability / availability issues. It is advisable that an organization should hedge its bet by replicating data to multiple cloud locations (e.g. availability zones in Amazon) and data centers.

1.3 Optimal Data Distribution Plan for Multi-site Backup

Replication, however, increases data footprint in linear proportion to the number of sites. Eventually, the data footprint becomes quite large. Additionally, it is often seen that the strategy of data distribution is not driven from recovery standpoint – that is, there is no way of telling if the data recovery can happen on time. Therefore, there is a need for rationalizing and optimizing the backup data distribution with respect to data footprint, cost, security, availability and data recoverability (within time and cost). Disaster Recovery planning [6] often overlooks this critical issue.

In this paper, we propose a novel planning approach named Data Distribution Planner for Disaster Recovery (DDP-DR). The planner creates a plan for distributing data across heterogeneous storage sites in a manner so that the overall cost of storage can be kept below a specified budget, the integrity of backup data can survive failure of up-to a specified number of sites, and the distributed data can be recovered within a specified time bound. To reduce data replication footprint, we take recourse to data fragmentation through Erasure Coding (EC) [7]. We combine EC-based data encoding technique with Linear Programming (LP) based constraint satisfaction algorithm. EC breaks data files into multiple data and code fragments. Coding rate of EC determines how many encoded fragments will be created from a data file. In this work, we optimize the coding rate, and therefore, the data footprint through a set of customer-specific constraints. This data fragmentation technique creates considerably less data redundancy than conventional multi-site replication method.

The organization is the paper is as follows: in section 2, we describe the data backup and recovery processes associated with DR and also state our assumptions. We present the problem formulation in section 3. The solution description and key components of DDP-DR are described in section 4. In section 5, we compare the theoretical predictions with simulation generated results. Related works are described in section 6, followed by conclusion and possible extensions of the work.

2 Multi-site DR Process, Topology and Assumptions

2.1 DR Process and Topological Assumptions

In DR, a critical requirement is to keep the backup data as much synchronized as possible with the primary copy. This is because, in the event of failure, we want to resume operations with data which is as current as when the operation has failed. We treat the backup data as a single file even though it can be aggregate of multiple user and application data, log files, system configuration files etc. We call this file as *Level 0* or full copy backup (L_0). Subsequently, the customer can upload changes / additions to the backup data on periodic basis, say between time intervals T_i *and* T_{i-1}. We call these additional files as L_1 (delta) change.

In this context, two definitions are pertinent to DR planning.

- **RTO:** Recovery Time objective – this specifies an upper bound for the time limit within which, after disaster at the primary site, the entire backup data (L_0) needs to be restored at the recovery site to resume operations.
- **RPO:** Recovery Point Objective – this provides an upper bound of the time limit within which the L_1 backups are to be committed to the backup storage. This value, in effect, also determines how old the DR data is at the time of disaster.

In DDP-DR we model a multi-DC backup and disaster recovery scenario. One of the DC is the Primary site of a customer. Another DC may be chosen as Recovery site. Other DCs are used for storage of backup data. Each DC / storage zone may have different types of storage. We basically model three types of storage – SATA (serial access), iSCSI (internet Small Computer System Interface) and FC (Fiber Channel). These protocols provide different types of data write and read rates into and from the storage, and they vary in cost.

We also assume that each storage unit in the DC is divided into storage blocks (like object based storage) or Buckets. The data fragments get stored in these buckets.

One of key parameters of our planning is **Protection Level (PL)** – which is defined by degree (a numerical integer number) to which the customer requires guarantee against simultaneous data centre failures. For example, if there are *n* DCs, the maximum protection level that can be guaranteed to the customer is *n-1*.

Further, a customer can select if he wants a particular DC to be excluded or included from the list where the backup data can be placed. For example, a European bank may demand that customer data is only stored in DCs located in European Union for regulatory purposes. We capture this with **Placement Constraint (PC)** – which is taken as 0 if the data for a customer is not to be placed in a particular DC and 1 otherwise.

Finally, we note that in DR restoring data to the Recovery site is only the first step. This step is followed by other processes like infrastructure booting, application configuration, and service health-check etc. Those steps are beyond the scope of our current work.

2.2 Data Fragmentation Strategy

To reduce the footprint of data distributed across sites, we use Erasure Coding [7]. We use MDS coding to code m data chunks with k error-correcting code fragments to achieve $n = (m + k)$ fragments, with coding rate $r = m / (m + k)$. One distinguishing feature of EC is that we require any m fragments to get back the data. The backup file is broken into chunks and each chunk is then erasure coded and encoded blocks are then dispatched to different DCs. When required, any m of stored fragments can be transmitted back to the Recovery site and decoded to get the original file.

It has been established that EC allows lesser data footprint and lesser network bandwidth for similar resilience (system durability) than plain replication scheme [8]. Additionally, network coding schemes are found to be more efficient globally than many other local storage-level data protection schemes [9]. Complete treatise of these assertions is beyond the scope of the work. One of the drawbacks of EC is that to regenerate data from partial failures, one must recall all the remaining encoded fragments to decode, re-encode and redistribute. Newer network coding techniques [10] allow partial regeneration of data and, therefore, are more efficient.

3 DDP-DR Approach

In DDP-DR, an optimal plan for data distribution will be such so that the total cost of storage, the data backup time (RPO) and data recovery time (RTO) are minimized. The resultant decision problem is a multi-objective function. We try to break the multi-objective decision problem with single objectives at a time:

- *Objective 1*: Minimize cost of storage and replication for a customer while maintaining the RTO and RPO time bounds and other policy level constraints;
- *Objective 2*: Minimize the RTO (recovery time) for a customer while maintaining the cost bound and RPO time bound and other policy level constraints;
- *Objective 3*: Minimize the RPO (backup time) for a customer while maintaining the cost bound and RTO time bound and other policy level constraints.

In each case, customer policy level parameters such as Protection Level (PL) and, Placement Constraint (PC) are satisfied. The idea is to help the customer to select a suitable operating point after looking at the choices.

We consider that network links to the data center are of equal bandwidth and cost; and hence, they do not affect our solution. However, our plan formulation can be augmented with network considerations as well.

The distribution problem is a Mixed Integer Programming [11] problem. However, we have relaxed the problem into an LP and tried finding the feasibility in a convex hull. We found that the solutions do not go beyond a small bound beyond the actual MIP solution because of this relaxation.

3.1 Planning Parameters

We list some of the main parameters of our formulation in the table 1 below with explanations.

Table 1. Parameters for DDP-DR

Parameters	Explanation
F_{Cust}	Total backup file size (L_0) for Customer
B_{Cust}	Size of Delta backup (L_1) for Customer
Bucket_size	Bucket Size, presumed to be same across all storage units in DCs
m_e	Total number of input (to-be coded) fragments for the customer $\left\lceil F_{Cust} / bucket\_size \right\rceil$
n_e	Total number of output (encoded) fragments for the customer
e_r , d_r	Rate of encoding and rate of decoding at server
i={1,..,n}	Data centres
BW_{ij}	Available incoming bandwidth to data centre *j* from data centre *i*
IOPS	Data read-write rate in the group of storage servers in a zone (FC, iSCSI or SATA)
No_of_disks	Number of disks in a storage server in a storage zone
Segment_size	Size of the read/write segment in MB in the server
BW_{ijact}	Actual available bandwidth between data centers *i* and *j*
c_i	Weighted Average cost of storage in *i*-th data centre
X_i	Number of encoded fragments to be stored in data centre *i* ($n_e = \sum_{i=1}^{n} X_i$)
P_b	Backup time bound for the customer data of size B_{Cust} (equivalent to RPO)
T_b	Recovery time bound for the customer data of size F_{Cust} (equivalent to RTO)
C_b	Cost bound for the customer
S_i	Total available storage in *i-th* data centre

Known (input) parameters are: F_{Cust}, B_{Cust}, m_e, e_r, d_r, *bucket_size*, BW_{ij}, c_i, P_b, T_b, *IOPS, no_of_disks, segment_size,* and C_b

Parameters to be determined are: X_i, which is number of encoded fragments per data centre and, n_e therefore, Which is total number of encoded fragments for distribution.

3.2 Constraints

We now discuss the formulation of the constraints to the objective.

RPO Constraint: RPO constraint states that, for a customer *Cust*, encoding of the L_1 delta backup data and transferring of encoded fragments into storage DCs should happen within P_b time. We assume that data can be pushed simultaneously to all selected DCs. Mathematically,

$$(\frac{1}{e_r} * B_{Cust} + \max_{j=1}^{n}(\frac{1}{BW_{sjact}} * \frac{B_{Cust}}{m_e} * X_j)) \leq P_b$$

Re-writing,

$$\max_{j=1}^{n}(\frac{1}{BW_{sj_{act}}} * X_j) \leq ((\frac{P_b}{B_{Cust}} - \frac{1}{e_r}) * m_e$$

Re-writing again in standard LP form,

$$\forall j\{1,..n\}, \qquad (\frac{1}{BW_{sj_{act}}} * X_j) \leq ((\frac{P_b}{B_{Cust}} - \frac{1}{e_r}) * m_e$$

RTO Constraint: This constraint is connected with data recovery, i.e., in data recovery, encoded L_0 data for customer *Cust* is to be pulled out of storage data centres and decoded within time bound T_b. As in the previous case, here also we assume that the data is fetched simultaneously to Recovery site through channels from different DCs. Mathematically,

$$(\frac{1}{d_r} * F_{Cust} + \max_{j=1}^{n}(\frac{1}{BW_{jsact}} * \frac{F_{Cust}}{m_e} * X_j)) \leq T_b$$

Re-writing in standard LP form,

$$\max_{j=1}^{n}(\frac{1}{BW_{js_{act}}} * X_j) \leq ((\frac{T_b}{F_{Cust}} - \frac{1}{d_r}) * m_e)$$

$$\text{i.e, } \forall j\{1,..n\}, \qquad [(\frac{1}{BW_{jsact}} * X_j) \le ((\frac{T_b}{F_{Cust}} - \frac{1}{d_r}) * m_e)]$$

Storage Cost Constraint: The customer *Cust* may have a cost bound C_b The total cost for allocated storage across all DCs has to be within this bound. Mathematically,

$$\sum_{i=1}^{n} (F_{Cust} / m_e) * c_i * X_i \le C_b$$

PL Constraint: Enough encoded fragments are to be spread across the DCs so that a failure of up-to PL DCs may be tolerated. To support a failure of a single DC *j*, there should be enough decodable fragments available across other DCs.

$$\forall j\{1,..n\} \qquad \sum_{i=1, i\neq j}^{n} X_i \ge m_e$$

So, in order to support a protection level of simultaneous failures of up-to *k* DCs, there should be enough encoded fragments in rest (*n-k*) DCs. Thus,

$$S = \{1,..,n\}; \qquad \forall O \in \Re(S, n-k)[\sum_{i\in O} X_i \ge m_e]$$

Where, $\Re(S, n-k)$ is the combination from a data centres set *S* taken (*n-k*) at a time.

PC Constraint: The customer *Cust* may want to exclude a subset Q of the available *n* DCs to store any fragments of the backup data.

$$X_i = 0, i \in Q, Q \subset \{1,..,n\}$$

Storage Availability constraint: The total size of all of the fragments that are stored in DC *i* should be less than the space available in DC *i*:

$$\forall i \in \{1,..,n\}[\sum \frac{X_i * F_{Cust}}{M_e} < S_i]$$

Available Bandwidth constraint: The actual rate of data transfer from DC*i* to DC*j* may the smaller of the network bandwidth and the read/write rate of the storage unit. The read/write rate (in GB) of the storage unit can determined as (IOPS * *no_of_disks* * *segment_size* / 1024).

Mathematically, available bandwidth have been derived by

$$BW_{ij_{act}} = MIN(((IOPS * no\_of\_disks * segment\_size)/1024), BW_{ij})$$

3.3 Problem Formulation

As discussed earlier, we have broken the multi objective minimization problem into three sets of minimizations and allow users to choose from the available feasible

solutions. To that effect, we now discuss the corresponding Linear Programming formulations one after the other.

Cost Minimization: For a customer, the objective function for cost minimization may be written as the sum of the product of the size of each encoded fragment, the number of encoded fragments written to DC *i*, and the cost of storage per unit in DC *i* across all *n* DCs:

$$\text{Minimize} \quad \sum_{i=1}^{n} (F_{Cust} / m_e) * c_i * X_i \tag{A}$$

The minimization is subjected to RTO, RPO, PL, PC, Storage Availability and Bandwidth constraints.

RPO Minimization: The objective function for minimization of the RPO may involve minimizing the total time to encode and store the encoded fragments of incremental backup, L_1 across DCs:

$$\text{Minimize} \quad (\frac{1}{e_r} * B_{Cust} + \max_{j=1}^{n}(\frac{1}{BW_{sj_{act}}} * \frac{B_{Cust}}{m_e} * X_j))$$

This may be written as:

Minimize t such that,

$$\forall j\{1,..n\} \qquad [(\frac{1}{BW_{sj_{act}}} * \frac{B_{Cust}}{m_e} * X_j) + (\frac{1}{e_r} * B_{Cust}) < t] \tag{B}$$

The minimization is subject to Storage Cost, RTO, PL, PC, Storage Availability and Bandwidth constraints.

RTO Minimization: The objective function for minimizing the RTO will involve minimizing the time to retrieve the full backup, L_0, from each of the DCs that store coded fragments of the backup data:

$$\text{Minimize} \quad (\frac{1}{d_r} * F_{Cust} + \max_{j=1}^{n}(\frac{1}{BW_{js_{act}}} * \frac{F_{Cust}}{m_e} * X_j))$$

This may be written as

Minimize t such that:

$$\forall j\{1,..n\} \qquad [(\frac{1}{BW_{js_{act}}} * \frac{F_{Cust}}{m_e} * X_j) + (\frac{1}{d_r} * F_{Cust}) < t] \tag{C}$$

4 Implementation, Results and Simulation

DDP-DR has been implemented as a standalone software tool. The tool can accept the business SLAs through an input file as a set of values, bounds and parameters. For erasure coding, we use standard MDS module implementing Reed-Solomon technique. IBM ILOG CPLEX Optimizer was used as LP solver. We have taken extensive runs with number of DCs varying from 3 to 15. For infrastructure and customer parameters similar to the one discussed in the illustrative example to be discussed next; the problem formulation consisted an LP with less than 50 variables. On a machine with 2GB of RAM and 2.6GHz dual core processor, running Ubuntu 10.04, the results for each of the cases took less than few seconds. The number of iterations for solution to converge varied between 10 and 30, depending on the numerical values.

4.1 Discussion of Results

We discuss few sample results for illustration of the concept. We consider a hypothetical example with a customer C_1 and 6 DCs. The parameters specific to each of the customers are listed below in Table 2 and those specific to the storage DCs are presented in Table 3. We take *bucket_size* of the servers as 10 GB. Please note that we did not put any explicit Placement Constraint for customer data at any of the DCs.

Table 2. Customer Data

Customer	Incremental backup size (L_1)	Total backup size (L_0)	Protection Level
C_1	35.0GB	350.0GB	1

Table 3. Storage Properties

Data Centre	Storage Type	Free Storage	Per MB Storage Cost[1] ($)	IOPS
DC1 , DC4	iSCSI	3 TB	0.008	750
DC2 , DC5	SATA	3.9TB	0.0006	450
DC3 , DC6	FC	2TB	0.09	24000

The bandwidth of the network links was considered 10 MBPS. DC1 was chosen as Primary data centre and DC2 was chosen as Recovery site.

We run DDP-DR on this dataset. For cost minimization objective, we run the optimization routine (A) with progressive relaxed RPO and RTO bounds. As depicted in Table 4, we start with an RPO bound of 1 hr and RTO bound of 4 hrs and found that that a data distribution plan is not feasible (i.e., the solution does not converge). We progressively relax RTO bound to 4.6 hrs and we find a feasible distribution plan with following distribution pattern: DC1 till DC5 will hold 80 GB of data each while DC6 will hold 70 GB of data. The total storage cost for this plan is $15231, which is the

[1] Representative cost, not actual.

lowest cost. A total of 470 GB of storage is used to store 350 GB of backup data of customer C_1 (contrast this with pure Replication scheme where 700 GB will be required). Accordingly, the erasure coding rate selected by DDP-DR is n_e/m_e, which equals 47 /35. In practice, the L_0 data would be broken up into 10 GB chunks and then encoded into EC fragments with coding rate of 47 /35. We show selective iterations with different RTO and RPO bounds.

Table 4. Results for objective *A*

RPO Bound (Hrs)	RTO bound (Hrs)	Minimized Cost Objective ($)	*m*	*n*	Data Centre shares (DC1, DC2, DC3, DC4, DC5, DC6) of Fragments
1.0	4.0	Infeasible			
1.0	4.6	15231.0	35	47	8,8,8,8,8,7,
1.0	6.0	7289.0	35	46	10,10,6,10,10,0,
1.0	24.0	1763.0	35	53	17,17,0,2,17,0,

Table 5 describes few iterations where customer C_1's RPO was minimized while continuing to satisfy the specified RTO and cost bounds (objective *B*). Note that for an RTO bound of 24 hrs and cost bound of $1950, we get a distribution strategy {9,14,0,14,14,0} which minimizes the RPO (0.86 hrs). If we relax the cost bound to $2000, we get a better RPO and different fragment distribution (comparatively more iSCSI disks get used).

Table 5. Results of objective *B*

Minimized RPO (Hrs)	RTO bound (Hrs)	Cost Bound ($)	*m*	*n*	Data Centre shares (DC1, DC2, DC3, DC4, DC5, DC6) of Fragments
Infeasible	6.0	1950.0			
0.8628334	24.0	1950.0	35	51	9,14,0,14,14,0,
Infeasible	4.0	1950.0			
0.82172227	24.0	2000.0	35	50	11,13,0,13,13,0,

Table 6 describes few iterations where C_1's RTO was minimized while continuing to satisfy the specified RPO and cost bounds (objective *C*). Note that RTO is relatively insensitive to RPO bounds as the data involved in recovery is much more than in periodic backups.

Table 6. Results of objective *C*

RPO Bound (Hrs)	Minimized RTO (Hrs)	Cost Bound ($)	*m*	*n*	Data Centre shares (DC1, DC2, DC3, DC4, DC5, DC6) of Fragments
1.0	8.503	1950.0	35	51	9,14,0,14,14,0,
1.2	8.503	1950.0	35	51	9,14,0,14,14,0,
1.4	8.503	1950.0	35	51	9,14,0,14,14,0,

4.2 Simulation

To test the accuracy of the Planner, we carried out extensive simulation of the backup and recovery scenarios with ns2 network simulation tool [12], with a network having the multiple topologies. Generally, simulation results were well within with the Planner predicted results. In some cases, however, we get slightly higher values in simulation. Deviation is found within 5%. Our conjecture is that slightly lower network data transfer rate than theoretical limit in simulation has resulted in some delays. For brevity, we are omitting details of simulation results.

5 Related Work

There is little published literature on planning of distributed storage for DR for the purpose of data recoverability and meeting other SLAs. Minerva system, discussed in [13], deals with creating an array of storage nodes based on performance features such as I/O supported, disk characteristics etc. More recently, a study on storage planning for disaster recovery [14] has characterized the planning process for storage volume, paths, and DC zones. Data recovery scheduling after disaster has been studied in [15] by using heuristic (GA-based) techniques. In these works, authors have characterized the planning problem as an infrastructure provisioning problem (i.e., provisioning or sizing of storage infrastructure for DR). Our work, in contrast, looks at planning as constraint satisfaction problem (i.e., finding out if the distributed DR infrastructure can support customer SLAs at the time of recovery).

Some works on using Erasure like distributed coding for distributed fault tolerance in large-scale archival systems have been discussed in OceanStore [16], and Farsite [17]. One of the important distinctions of our work from these is that these works have an eventual recovery model, i.e., no recovery time objective SLA is set; while in our case the coding structure and data distribution strategy is strictly SLA dependent.

Multi-site replication of data for availability is also well investigated. There have been a series of work around data placement and replication in a data grid environment like Bell [18]. Data replication and job scheduling has been studied in [19]. QoS aware replication has been studies by [20]. Our problem is of different nature as we are studying replication related to archival fault-tolerance of data and not performance.

6 Conclusion and Future Work

With the advent of sophisticated online backup, multi-geography sites and cloud computing, multi-site DR is increasingly becoming popular for better reliability and availability of operational data. In this paper, we describe DDP-DR a novel data distribution plan for multi-site disaster recovery, where backup data can reside in multiple data centres including public cloud. The plan takes customer policy level constraints and infrastructural constraints into consideration to suggest a series of data distribution plans. The work tries to fit customer requirements into existing DR topologies as opposed to provisioning servers for DR as most of other DR planning

work has suggested. Initial verification of the plan with simulation suggests very encouraging results. As a future work, we will be looking at bringing network costs and other DR steps in overall optimization. We will also be looking at disaster recovery scenario with multiple customers (tenants) within same distributed DR architecture.

References

1. Schiers, J.: Multi-PB Distributed Databases. IT Division, DB group, CERN, presentation, http://jamie.web.cern.ch/jamie/LHC-overview.ppt
2. Gulli, A., Signorini, A.: The indexable web is more than 11.5 billion pages, http://www.divms.uiowa.edu/~asignori/papers/the-indexable-web-is-more-than-11.5-billion-pages/
3. Mozy Online Backup Storage at, http://www.mozy.com/
4. NetAppSnapMirror technical documentation at, http://www.symantec.com/business/support/resources/.../288533.pdf
5. Wood, T., Cecchet, E., Ramakrishnan, K.K., et al.: Disaster Recovery as a Cloud Service: Economic Benefits and Deployment Challenges. In: USENIX Hotcloud 2010 (2010)
6. Wallace, M., Webber, L.: The Disaster Recovery Handbook - A Step-by-Step Plan to Ensure Business Continuity and Protect Vital Operations, Facilities, and Assets (2007)
7. Plank, J.S.: Erasure codes for Storage Applications. In: FAST 2005: 4th Usenix Conference on File and Storage Technologies, San Francisco, CA (December 2005)
8. Weatherspoon, H., Kubiatowicz, J.D.: Erasure Coding Vs. Replication: A Quantitative Comparison. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 328–337. Springer, Heidelberg (2002)
9. Rodrigues, R., Liskov, B.: High Availability in DHTs: Erasure Coding vs. Replication. In: van Renesse, R. (ed.) IPTPS 2005. LNCS, vol. 3640, pp. 226–239. Springer, Heidelberg (2005)
10. Chen, Z., Wang, X., Jin, Y., Zhou, W.: Exploring Fault-tolerant Distributed Storage System using GE code. In: Proc. of Embedded Software and Systems, ICESS, pp. 142–148 (2008)
11. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. Wiley
12. NS-2 Project at, http://isi.edu/nsnam/ns/
13. Alvarez, G.A., Borowsky, B., et al.: Minerva: An automated resource provisioning tool for large-scale storage systems. ACM Transactions on Computer Systems, TOCS (2001)
14. Gopisetty, S., Butler, E., Jaquet, S., Korupolu, S., Seaman, M., et al.: Automated planners for storage provisioning and disaster recovery. IBM. J. Res. Dev. 52(4-5), 353–366 (2008)
15. Keeton, K., Beyer, D., et al.: On road to recovery – restoring data after disaster. In: Proceedings of European System Conference, EoroSys (2006)
16. Kubiatowicz, J., Bindel, D., Chen, Y., et al.: OceanStore: An Architecture for Global-Scale Persistent Storage. In: Proc. of ASPLOS (2000)
17. Adya, A., Bolosky, W.J., Castro, M., Cermak, G., et al.: FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In: Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI). USENIX (2002)
18. Bell, W.H., Cameron, D.G., Carvajal-Schiaffino, R., et al.: Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In: CCGrid. IEEE (2003)
19. Ranganathan, K., Foster, I.: Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. J. Grid Comput. 1(1), 53–62 (2003)
20. Tang, X., Xu, J.: QoS-Aware Replica placement for Content Distribution. IEEE Transactions on Parallel and Distributed System 16(10) (2005)

Saga: A Cost Efficient File System Based on Cloud Storage Service

Wei Shi, Dapeng Ju, and Dongsheng Wang

Department of Computer Science & Technology, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
wshi@wshi.net, {judapeng,wds}@tsinghua.edu.cn

Abstract. Cloud Storage service providers such as Amazon Simple Storage Service (S3) and Google Storage for Developers offer low-cost and highly available scale storage resource with a simple pay-as-you-go charging model. The cost of running storage systems on such a Cloud Storage service mainly depends on occupied storage space, number of requests and amount of data transfer. Traditional design of storage stack based on disk driver or tape didn't consider cost as a system metric, hence it brings considerable optimization space for the design of storage system based on Cloud Storage.

In this paper we propose Saga, a user mode file system based on Cloud Storage service, that is designed to support POSIX interface with the goal of minimizing cost. Saga is specially designed under the cost efficient principle that minimizes occupied storage space by store-one-copy and copy-on-write strategies and minimizes number of requests by distinguishing objects loaded by write or read requests. Saga is also efficient from a performance perspective and utilizes parallel characteristics of Cloud Storage to boost the performance. Experimental results show that Saga is cost efficient and works well with general-purpose I/O workloads.

Keywords: Cloud Storage, Cost Efficiency, POSIX, S3.

1 Introduction

Cloud Computing, as the next generation computation framework for hosting data and deploying software and services, has become more and more popular. As a part of Cloud Computing, Cloud Storage services, such as Amazon Simple Storage Service (S3) and Google Storage for Developers (Google Storage) offer performance and interface that differ substantially from those traditional storage devices in web hosting and scientific data analysis. Considering Cloud Storage as a sort of "physical device", then it has two new features comparing with traditional storage devices. One is the native parallel characteristics that data can be transferred by multiple threads using web service such as REST or SOAP. The other is the new economical characteristics that cloud resources are charged by a simple pay-as-you-go charging model.

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 173–184, 2012.

But for traditional storage system designs, there are two limitations when they facing this new type of "physical device". One problem is such "physical device" has a different interface from generic file system. So applications based on POSIX interface can't directly use Cloud Storage to store data and need to be rewritten. For the wide spread use of storage systems based on Cloud Storage, they need to support POSIX interface. The other problem is the charging model of Cloud Storage has made system cost as a critical metric. Storage systems must be designed with consideration to minimize the cost of Cloud Storage. Collectively, these changes have made existing storage systems designed for local storage device do not adapt with Cloud Storage correspondingly.

In this paper, we explore cost efficiency as the continuation of design space. We propose a file system based on Cloud Storage with the design goal of supporting POSIX interface and minimizing cost, called Saga. Saga aims to adapt the new opportunities and challenges created by the appearance of Cloud Storage. Rather than embed static assumptions that the cost of write operation is the same as the cost of read operation, write operation is much more expensive than read operation in Cloud Storage. So Saga makes a distinction between write and read operation. And Saga discriminates the data loaded by GET request and PUT request in its cache module named Dragon Orb. Dragon Orb manages a local cache which caches all the data fetched from Cloud Storage and going to store to Cloud Storage.

In the remainder of this paper, we first discuss prior work in storage system such as file system or database design based on Cloud Storage as well as the analysis and evaluations of Cloud Storage under different workloads. Then we give our cost analysis and evaluation of Cloud Storage. We then address the design of Saga and provide an in-depth description of its implementation. Then we evaluate Saga in performance and cost efficiency. We remark on conclude and future work in the last.

2 Related Work

Our research is motivated by the new economical characteristics of storage system introduced by Cloud Storage and aims to design a cost efficient storage system under different workloads and support most of existing applications. We review previous work based on Cloud Storage in the following wo aspects:

– Design of storage system based on Cloud Storage

Because of the easy interface and simple pay-as-you-go charging model of Cloud Storage, it is widely used to backup data or cache data. Cumulus[9]is a backup system built by Vrable et al., which can be used to backup whole file system to Cloud Storage. Cumulus is specifically designed under a thin cloud assumption and aims to minimize backup resource requirements, such as storage and network, and ongoing monetary costs. Cumulus aggregates data from small files and uses a segment cleaning strategy similar to Log-Structured File System(LFS)[8] to maintain storage efficiency. The design of Cumulus is quite different from that

of Saga. Cumulus is first a backup system while Saga is a generic network file system. Saga can randomly access any block of any file while Cumulus only can access the data of file after downloading and restoring the file at local file system.

Chiu at el. have utilized Cloud Storage as a sort of cache to accelerate service-oriented computations[3]. By using Cloud Storage to cache data output from services and proposing a self-adaption algorithm, their cache schema can scale up the cache system during peak querying times and back down to save cost. Using the elastic Cloud resource, a detailed evaluation showed that this cache system is capable obtaining minimal miss rates while utilizing far less nodes than statically allocated systems of fixed sizes in the span of the experiment.

Jungle Disk is a commercial software which has a design very similar to that of Saga. The desktop edition runs like a local file system with Amazon S3 as the backing store, and all read and write calls will redirect to Cloud Storage. Jungle Disk can also used for backup, keeping copies of old versions of files instead of delete them. At the backend, each file stored in Jungle Disk is an S3 object. So when we need to use a big file, we need to wait for downloading the whole file. Saga was inspired by Jungle Disk, but avoids this problem by chunk files into blocks and storing blocks as S3 objects.

Brantner at el. built a database on Amazon S3[2] and demonstrate the opportunities and limitations of using S3 as a storage system for general-purpose database applications which involve small objects and frequent updates. Experiment results showed that database costs more when the level of consistency is in higher level. For the highest level of consistency (Atomicity), the cost per transaction is almost twenty times as high as for the Naive approach which is used as a baseline.

– Analysis and evaluations of Cloud Storage

Several analysis and evaluations effort for estimating Cloud Storage whether be suitable for scientific applications have concentrated on the access performance, usability and low cost features of Cloud Computing.

Palankar at el. evaluated S3's ability to provide storage support to large-scale science projects and identify a set of additional functionalities that storage services targeting data-intensive science applications should support[5]. Experiment results showed the single-threaded GET bandwidth of S3 from an EC2 instance in the same region for objects sized 1B, 1KB, 1MB, 16MB and 100MB. But there is no evaluation of PUT bandwidth with different sized blocks. Concurrent performance of access bandwidth is also given through experiments. They also analyzed the data availability of S3 and cost and performance of S3 in scientific computing.

3 Design of Saga

Firstly, we will introduce the price schema of Cloud Storage. Amazon S3 and Google Storage are the two major Cloud Storage service providers. As Google Storage is very similar to Amazon S3, the desing of Saga will focus on S3, which

can also be applied to Google Storage. S3 only charges for occupied storage space, number of requests and data transfer. Occupied storage space is charged in the unit of Byte-Hours or GB-Months. S3 also charges for data request by number of requests. For different types of request, there are different types of fees. PUT, COPY, POST, or LIST, these four types of request cost $0.01 per 1,000 requests, while GET and all other requests cost $0.01 per 10,000 requests. This indicates that the GET request is cheaper than the PUT request. For data transfer, S3 makes a distinction whether it is in or out of the region. But for the COPY request within the same region, there is no data transfer fee. Furthermore, there is no data transfer fee for the transfer between S3 and Amazon EC2 instances within the same region.

Saga is designed to utilize Cloud Storage and meet the following four critical objectives.

- *A generic file system to support random access* To avoid rewriting the existing applications using POSIX interface, Saga is designed as a generic file system to add a layer between applications and Cloud Storage interface and support for POSIX interface and fine-grained operations such as random access.
- *Minimizing storage utilization* A critical strategy to make Saga truly cost-efficient in practice is to store only one copy for the objects with the same content. We maintain a mapping table to record the map from the logical objects to the physical objects stored in Cloud Storage. And we use the copy-on-write (COW) strategy when the content of the objects is changed.
- *Enhancing access performance* By using a differentiated object cache named Dragon Orb to cache objects at local file system, we can improve the access performance of Saga by masking the PUT and GET requests to Cloud Storage. Furthermore, Dragon Orb can significantly minimize the request fee through reducing PUT and GET requests.
- *Minimizing the cost of requests* According to the analysis in Section 3.1, we will find that PUT request is more expensive than GET request, so Saga is designed to adapt this feature of Cloud Storage, and to minimize the request fee by distinguishing the PUT and GET requests in the cache replacement algorithm. Moreover, Saga uses a lazy-write strategy for its object cache to reduce the requests.

3.1 Overview of Saga

Saga is designed as a user-space file system which supports POSIX interface with the goal of minimizing cost. Cloud Storage is charged by occupied storage space, number of requests and data transfer. For essential performance requirement of generic file system, Saga needs to access Cloud Storage with relatively stable network bandwidth. In our design, Saga is planned to deploy in EC2 instances which are in the same region with S3 buckets. So there is no data transfer fee for Saga. In Saga, we only consider occupied storage space and number of requests as the target to minimize cost.

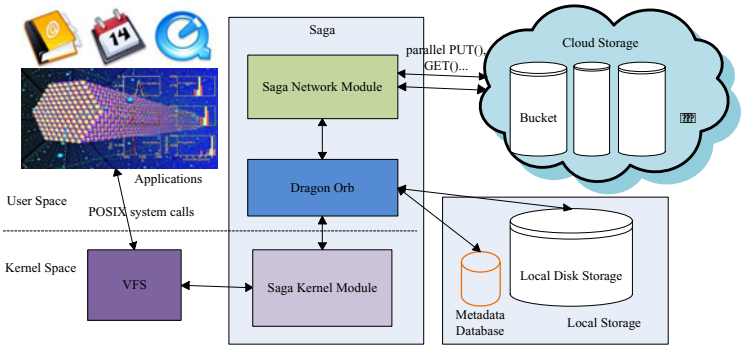


Fig. 1. Architecture of Saga

All files stored in Saga are chunked into fixed size blocks and all the fixed size blocks will be stored as objects in Cloud Storage. According to the functionalities, Saga can be divided into three modules: a cache module named Dragon Orb, a kernel module redirecting file system calls to Dragon Orb and a network module taking charge of writing data to Cloud Storage and reading data from Cloud Storage. The kernel module of Saga redirects all the file system calls to the user mode cache module Dragon Orb. And Dragon Orb manages a fixed size cache on local file system to store objects and utilizes a cache replacement algorithm called DO-LRU to evict objects when an object has to be loaded into the full object cache. If the evicted objects are dirty, they will be sent to the network module and then stored in Cloud Storage through the service providers' API.

Fig.1 illustrates the architecture of Saga. When the application needs to access files stored in Saga, (1) kernel module will redirect the file system calls to Dragon Orb. (2) Dragon Orb will firstly check the metadata and offset to find the needed objects, and then check whether to store the objects in object cache or not. (3) If the needed objects are stored in object cache, Dragon Orb will operate the relevant objects in object cache and return the result to the kernel module; otherwise, Dragon Orb will call the network module, get the relevant objects in Cloud Storage, store them in local objects cache, operate the objects and then return the result to the kernel module.

3.2 Saga Storage Policy

The analysis of S3's price schema in Section 3.1 shows that occupied storage space is an important factor to affect the cost of storage system based on Cloud Storage. To be cost-efficient, Saga needs to be designed with the goal of minimizing the occupied storage space. This goal can be archieved through compression algorithms to compress the data stored in Cloud Storage or some other methods.

Kulkarni et al. have showed that redundancy elimination at the block level is an efficient method to eliminate a broad spectrum of redundant data in a scalable and efficient manner[4]. Files stored in Saga are chunked into fixed size blocks and blocks are logical storage unit. The block size can be set to 512KB, 1MB, 2MB

or 4MB. There is a pointer for each block to point to the physical storage unit stored in Cloud Storage, i.e. objects. In order to save the occupied storage space in Cloud Storage, Saga is designed to allow the system to store only one copy and use pointers to the original object instead of creating redundant objects. When the content of an block in a file is changed, Saga utilizes a copy-on-write (COW) strategy to create a new object and use pointers to point to it.

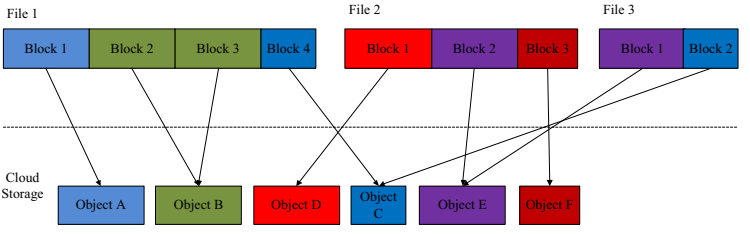


Fig. 2. An illustration of how files stored in Saga

Fig.2 illustrates the method how files being stored in Saga. Three files stored in Saga are chunked into blocks, which are stored in Cloud Storage. Object C and object E are shared by different files while there are two pointers in the same file pointing to object B. Because Saga chunks files into fixed-size blocks, the size of blocks are almost the same. But in the rear of some files, there will be a block size less than the fixed size. So the objects stored in Cloud Storage almost have the same size with a few objects less than the fixed-size. There is a reference counter for each object and it records the number of pointers point to this object. When the reference counter of an object decreases to zero, this object will be deleted.

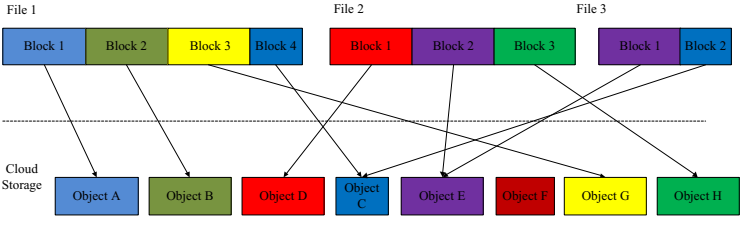


Fig. 3. An illustration of COW in Saga

Fig.3 illustrates the COW strategy of Saga. When the content of the third block of file1 is changed, a new object, object G, will be created and stored in Cloud Storage. Then the pointer of this block will change to point to object G. Similarly, the third block of file2 is changed and object H is created, when the pointer of the block is changed to point to object H there is no pointer point to object F and object F will be deleted.

Saga utilizes the store-one-copy strategy to save occupied storage space in Cloud Storage, but it also can be configured to compress the storage space using

zlib or *bzip2* compression algorithms. Saga allows the user to assign different compression policies to different file categories. A file category is determined by two essential attributes: one is the absolute path of the directory where the category's files locate in, and the other is files' extentions. To specify a file category, the path attribute is necessary, while the extention attribute is optional. Omitting the extention attribute means files with any extention are all included by the file category specified by the path attribute.

3.3 Dragon Orb: Differentiated Object Cache

In order to enhance the performance of accessing files, Saga exploits the temporal locality by the differentiated object cache module – Dragon Orb. When applications want to write or read a file, Saga will get affected objects from Cloud Storage. Dragon Orb will cache these objects in a fixed size local cache space on the local file system. The objective of Dragon Orb includes not only improving access performance but also minimizing the request cost of Cloud Storage. On the other aspect, Dragon Orb shields the fault caused by network by storing dirty objects on local file system firstly.

For flash-based SSDs, write operations are more expensive than read operations because write operations must be operated after erasure operations while erasure operations are expensive in time and lifetime of SSD. Park at el. addressed a replacement algorithm called CFLRU for flash memory and utilized the characteristics of flash memory that write cost is more expensive than read cost in the aspects of time and energy[6]. CFLRU is a variant of LRU and designed for page level cache. CFLRU set a window from the LRU position in LRU stack, clean pages will be evicted firstly in the window.

Inspired by CFLRU, Dragon Orb proposes a cache replacement algorithm called DO-LRU, which is also a variant of LRU and designed with the goal that objects loaded by write calls will have a longer lifetime than the objects loaded by read calls. DO-LRU set three Clean Mark Positions (CMP) to distinguish objects loaded by read calls from objects loaded by write calls in object cache. Objects loaded by read calls will be placed after a chosen CMP instead of MRU position of the cache stack. If the length of cache stack is L, the three CMPs are placed apart from the LRU end in a distance of $\lfloor \frac{L}{8} \rfloor$, $\lfloor \frac{L}{4} \rfloor$ and $\lfloor \frac{L}{2} \rfloor$, so LRU replacement algorithm could be considered with CMP placed apart from the LRU end in a distance of L. DO-LRU will perform a Clean Mark Dueling(CMD) method to choose the Clean Mark by simulating DO-LRU replacement algorithm with all three CMPs and LRU for a fixed time interval. CMD uses the set dueling mechanism[7][10] to gather runtime values of $Cost_{DO}$and choose an appropriate CMP. $Cost_{DO}$ is caculated as following:

$$Cost_{DO} = C_{PUT}/C_{GET} * N_{PUT} + N_{GET} \tag{1}$$

C_{PUT} and C_{GET} is the cost of PUT and GET per request. N_{PUT} is operation number of PUT requests and N_{GET} is operation number of GET requests.

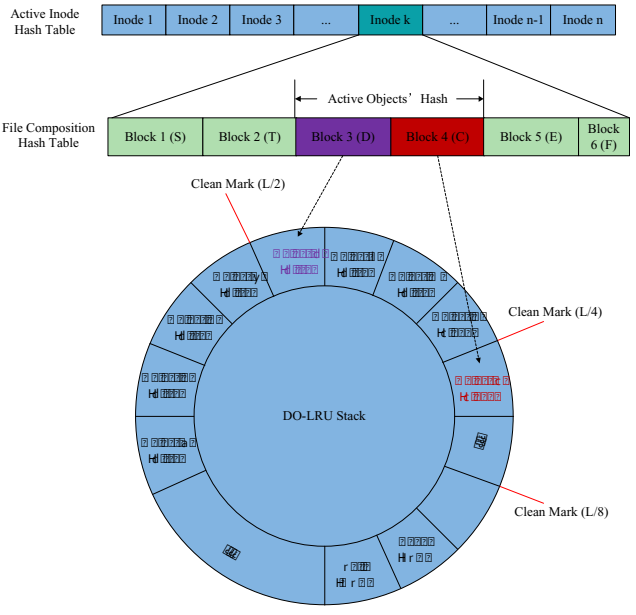


Fig. 4. Dragon Orb object cache structure

As illustrated by Fig.4, Dragon Orb utilizes a hash table for efficiently searching out the hash of blocks belonging to a file according to the inode number of the file. We refer the inodes of the files ever be opened as active inodes, and only active inodes will be stored in a hash table. According to the active inode, file composition information can be easily retrieved from the active inode hash table. File composition information includes a hash table which record the content hash of those blocks composed the file. Because objects are stored with their content hash as the key, it is easy to retrieve the object with its content hash. Those objects stored in the local cache called active objects. The number of active objects is limited by the size of local cache, and active inode hash table will only store those active inodes which have their object stored in local cache space. So the size of memory used to store active inode hash table and file composition hash table will be limited by the size of cache space, which is relatively small compared to the memory.

4 Implementation

We address details of the implementation of Saga in this section. Saga is the prototype of our design and we implemented Saga as a user mode file system based on FUSE[1] for simplicity. Our implementation of Saga is light weight with only 4800 lines of Python source code implementing the core file system functionality, along with 1400-line Python bindings for FUSE API. We chose S3

as the backend Cloud Storage service to implement Saga because it's the largest Cloud Storage service provider and its API has be used widely.

Saga kernel module is a python bindings for FUSE API, we use the ctypes package and libfuse to implement it. All file system calls to Saga will be redirected from the VFS layer to kernel module, and then be sent to the user space cache module Dragon Orb. Dragon Orb will perform the operations to the corresponding objects and return the result to the kernel module. Finally, kernel module will return the result to the applications.

We save all types of metadata of Saga in a single S3 object. When the file system is mounted, metadata object should firstly download from S3 and reconstruct the data structure in memory. We organized the metadata using table strucure within the metadata object. Specifically, each of the *file system hierarchy*, *file metadata information* and *file composition information* corresponds to a table in the metadata object. Appropriate indices are created in each table to quicken the lookups frequently queried by the *file system hierarchy* and the *file composition information*. The indices are stored back to the metadata object periodically and when the file system is unmounted. For metadata in cache, we organize them using two hash tables addressed in Section 3.3.

Compared with the traditional storage devices, Cloud Storage has a native parallel characteristics because data can be transfered by multiple threads using web service such as REST or SOAP. Two thread pools are built for sending objects to Cloud Storage and fetching them back, each of the thread pool has 10 threads.

5 Experiments and Evaluations

In this section, we evaluate the performance of Saga and its ability to save cost, as well as gain insight for future improvement of Saga design and implementation. All following experiments run on a basic 64-bit Amazon Linux instance with its Amazon Machine Image (AMI) id ami-8e1fece7 equipped with 2 Elastic Compute Unit (ECU), 613MB RAM, and 8GB Elastic Block Store (EBS). ECU was introduced by Amazon EC2 as an abstraction of computation resources. One ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

5.1 Continious Read and Write

The goal of evaluating Saga's continuously writing and reading performance is to understand the bottlenecks inherent in its current design by comparative analysis for different block size. To this end, a relatively large ISO file of around 700 MB is copied into and then out of Saga, and the continuously writing and reading rates are measured as the file size divided by the time it takes to copy the file. Fig.5 shows how the continuously writing and reading rates vary as the data segment size increases. It is observed that nearly in every scenario the continuously writing rate grows as the data segment becomes larger. But when Saga utilizes *bzip2* compression algorithm to compress data, the speed will not always grow. It shows that when we use *bzip2* to compress data, sometimes computing becomes the bottleneck except IO and *bzip2* speed curve has a jitter.

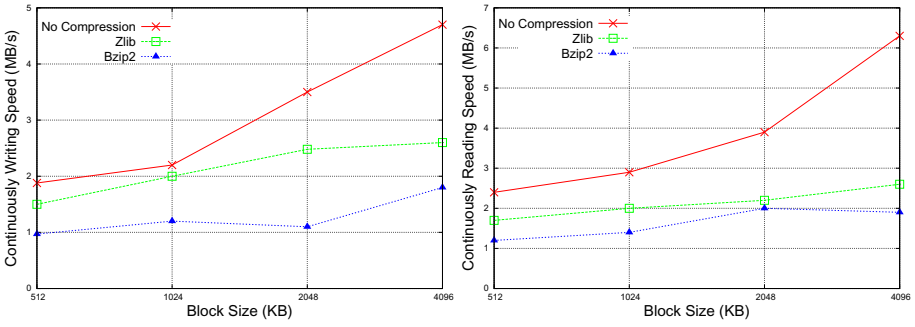


Fig. 5. Continuously writing and reading performance

5.2 Randomly Read and Write

Fig. 6 shows the time to perform 10,000 random *write* and *read* system calls respectively, 1 KB each time, against a binary file of around 900 MB. As can be seen, different from continuously writing, randomly writing appears to slow down as the data block size increases. The reason is that Saga performs operations in granularity of block. Even if the amount of data requested to be written are smaller than a block, Saga still needs to fetch at least one involved data block. As a result, the larger is the data block, the less efficiency Saga can carry out randomly writing and reading.

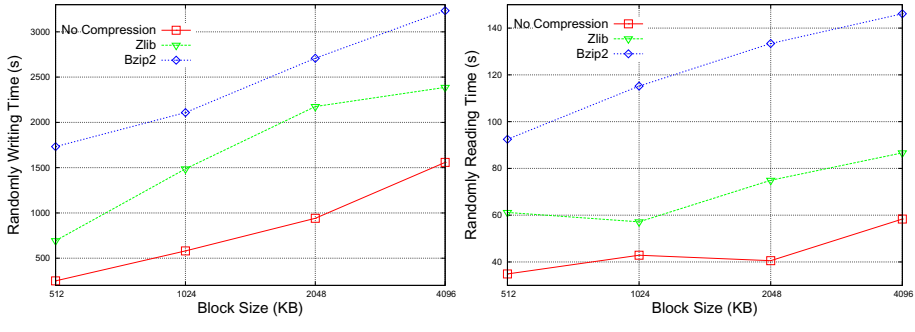


Fig. 6. Randomly writing and reading time

5.3 Cost of Different Block Size

With the goal of evaluating the cost of Saga, we compared the occupied storage space under different compression algorithms and compared the requests number between DO-LRU and LRU cache replacement algorithm. To this end, we run a bash script which is used to write data to Saga file system and fetch data back to local file system. We repeat this experiment for 6*4 times with different block size and different cache replacement algorithm. All these experiments run

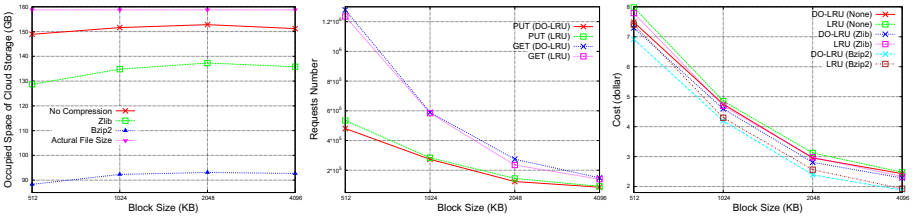


Fig. 7. Occupied storage space, requests number and cost under different environment

for 50 hours. Fig. 7 shows occupied storage space with different compression algorithms, requests number with different cache replacement algorithms and cost for different environment. It is observed that system cost decreases with the block size grow. For object cache replacement algorithm, DO-LRU is more cost-efficient than LRU.

6 Conclusions and Future Work

Cloud Storage service can offer low-cost and highly available scale storage resource with a simple pay-as-you-go charging model. Storage systems based on Cloud Storage should consider cost as an important metric beside bandwidth and latency. Compared with the transitional storage devices, Cloud Storage transfers data by web service and charges for occupied storage space, number of requests and data transfer.

Saga is a file system based on Cloud Storage and designed with the goal of minimizing cost and supporting POSIX interface. Saga is specially designed under the cost efficient principle which minimizes occupied storage space by store-one-copy and copy-on-write strategy and number of requests by distinguishing objects loaded by write or read requests. To the best of our knowledge, Saga is among the first effort towards minimizing cost and realizing it on the file system level. Preliminary evaluation shows that Saga is cost efficient and capable of providing a usable data access performance for upper applications.

Although Saga is already able to achieve usable performace, many improvements deserve to make. Saga is not a real network file system but a remote file system at present. Now we use metadata object to save all types of metadata on persistent layer. Moreover, we will build a metadata server utilized Cloud Computing resources to realize the network file system based on Cloud Storage and analyze the performance of concurrency and bandwidth.

References

1. Fuse: Filesystem in userspace, `http://fuse.sourceforge.net/`
2. Brantner, M., Florescu, D., Graf, D., Kossmann, D., Kraska, T.: Building a database on s3. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 251–264. ACM, New York (2008)

3. Chiu, D., Shetty, A., Agrawal, G.: Elastic cloud caches for accelerating service-oriented computations. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2010, pp. 1–11. IEEE Computer Society, Washington, DC, USA (2010)
4. Kulkarni, P., Douglis, F., LaVoie, J., Tracey, J.M.: Redundancy elimination within large collections of files. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC 2004. USENIX Association, Berkeley (2004)
5. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon s3 for science grids: a viable solution? In: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, DADC 2008, pp. 55–64. ACM, New York (2008)
6. Park, S.Y., Jung, D., Kang, J.u., Kim, J.s., Lee, J.: Cflru: a replacement algorithm for flash memory. In: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2006, pp. 234–241. ACM, New York (2006)
7. Qureshi, M.K., Jaleel, A., Patt, Y.N., Steely, S.C., Emer, J.: Adaptive insertion policies for high performance caching. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA 2007, pp. 381–391. ACM, New York (2007)
8. Rosenblum, M., Ousterhout, J.K.: The design and implementation of a log-structured file system. ACM Trans. Comput. Syst. 10, 26–52 (1992)
9. Vrable, M., Savage, S., Voelker, G.M.: Cumulus: filesystem backup to the cloud. ACM Transactions on Storage (TOS) 5(4) (December 2009)
10. Zhang, X., Li, C., Wang, H., Wang, D.: A cache replacement policy using adaptive insertion and re-reference prediction. In: Proceedings of the 2010 22nd International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2010, pp. 95–102. IEEE Computer Society, Washington, DC, USA (2010)

Developing a Cost-Effective Virtual Cluster on the Cloud

A. Stephen McGough

School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU United Kingdom
http://www.cs.ncl.ac.uk/people/Stephen.McGough

Abstract. The Cloud provides highly democratic access to computer services on a pay-per-use basis. A fact that has encouraged many researchers to adopt the Cloud for the processing of large computational tasks and data storage. This has been used in the past for single research endeavours or as mechanism for coping with excessive load on conventional computational resources (clusters). In this paper we investigate, through the use of simulation, the applicability of running an entire computer cluster on the Cloud. We investigate a number of policy decisions which can be made over such a virtual cluster to reduce the running cost and the effect these policies have on the users of the cluster.

Keywords: Clouds, IaaS, Cost Optimization, Cluster, Simulation.

1 Introduction

Cloud Computing [4] provides a new model for computational processing and data storage removing many of the access barriers to large-scale computing by eliminating the need for capital expenditure on large private infrastructures. Instead a user can 'rent' computational power or data space on a short-term basis – more than they could afford to buy though enough to meet their immediate needs – transferring expense to an operational cost. This approach tends to work best in scenarios with significant temporal variation in requirements – alternating between periods of little (or no) activity to periods of high activity.

This is in contrast to conventional resources available within organisations such as Universities or Companies – often in the form of a cluster of computers. Here capital expenditure is outlaid on a fixed number of computational resources and data storage. The size is dominated by two factors: the available budget, and the anticipated load on the cluster. The aim is to provision enough resources to deal with all but the exceptional load scenarios placed on the resources.

Like many institutions Newcastle University provides a computational cluster for researchers. This has the advantage of economy of scale – researchers share resources allowing each access to more than they could individually afford. Although researchers loose exclusive access to resources this is not seen as a problem as few utilise the resources 24/7. The Newcastle cluster is formed from the student access computers located around the campus. This has two

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 185–197, 2012.

disadvantages for cluster users: computers can be lost from the cluster at any time due to students logging in, no choice on operating system – student access mandates that computers run Windows, whilst most cluster users prefer Linux.

We have previously shown that ∼120MWh of energy was consumed in 2010 to power the Newcastle Condor cluster [14]. Although additional capital expenses for computer hardware and operational costs for computer maintenance exist as these computers are primarily for other purposes we do not currently take account of these. As the University is investigating the use of low powered thin clients for student access and direct charging for the energy used in the cluster this could lead to alternative approaches becoming more favourable in the future.

The advent of the Cloud, which removes capital cost and provides apparently infinite resources, has given researchers a new way to work – often in-spite of local resource availability. Large collections of resources can be provisioned in a short period of time, quicker than many institutions can offer, for a relatively small operational outlay, a fraction of the capital cost. A second approach, Cloud Bursting, has emerged where owners of clusters have exploited the Cloud to cope with excessive demand which exceeds the resources available in-house.

Here we explore an alternative use case – moving the entire cluster onto the Cloud. Investigating if the economy of scale benefits of a conventional cluster map onto a virtual cluster and the effectiveness of policies, applied to the virtual cluster, in terms of cost savings and impact on the cluster users. We evaluate the cost of using the Cloud in terms of the hours consumed on the Cloud and the impact on the cluster users as the effect on the average make-span for their jobs. Defining make-span as the time between job submission and job completion.

We use a high level trace-driven simulation [7], using trace logs from the Condor cluster [10] based at Newcastle University [13,14], to evaluate the effectiveness of our approach. Using just the submission times for jobs to the cluster and their execution times allows us to submit jobs into the simulated Cloud cluster where jobs will either receive service immediately, if virtual computational instances (refereed to here as *instances*) are idle, or enter a queue awaiting execution otherwise. Policy can then be enacted to determine if (and when) a new Cloud instance should be started or unused instances terminated. As the main focus of this paper is to comparatively evaluate a number of policies we do not concern ourselves with the appropriateness of these trace logs, using them only for comparison – real deployment would almost certainly alter usage patterns.

We adopt the Cloud model used by many providers (e.g. Amazon's EC2 [2]) allowing users to deploy virtual machine images onto servers owned by the provider – referred to as Infrastructure as a Service (IaaS) [18]. Billing is typically by the hour with partly used hours incurring a full hour charge. The start of a billing period varies between providers. Some charge from the start of the wall-clock hour in which the instance was invoked – billing from 7pm for an instance stated at 7:58pm – whilst others charge from the time the instance was invoked [9]. For clarity we refer to the former case as *wall-clock charging* and the latter as *exact charging*. It should be noted that although other billing intervals exist our results

are not invalidated by the use of shorter (or longer) periods, they merely alter the severity of the impacts that we seek to mitigate.

The rest of this paper is set out as follows. Section 2 discusses related research to the work we propose. In section 3 we describe in more detail the cluster that we are modelling. We present a number of policies for optimising the cost for using the Cloud in Section 4 along with the perceived benefits of these policies. The simulation environment is described in Section 5 with the simulation results being presented in Section 6. Finally our conclusions are presented in Section 7.

2 Related Work

There is currently great interest in Cloud Computing [4]. This has lead to a number of investigations into the applicability of the Cloud as a tool for aiding researchers in their work. A number of simulation approaches to model the benefits of Cloud computing have been performed. Deelman [8] evaluated the cost of using Amazon's Elastic Compute Cloud (EC2) [2] and Amazon's Simple Storage Service (S3) [3] to service the requirements of a single scientific application. Here we seek to service the requirements of multiple users and multiple applications.

de Assuncao [5] proposed the use of Cloud computing to extend existing clusters to deal with exceptional load. This work was further extended by Mattess [12] by proposing the use of Amazon spot instances, supply-and-demand driven pricing of instances, to further reduce the cost of Cloud Bursting. Our approach differs to these in the sense that we seek to deploy our entire cluster to the Cloud. The approach of using spot instances. however, could easily be included in our approach and would allow for the same cost reduction as proposed by Mattess. Van den Bossche [6] uses Binary Integer Programming to select which workflows should be bursted to the Cloud. This approach is computationally expensive to determine the optimal approach and does not address the issue of when to terminate instances. It may be naively assumed that the our approach here is no more than the degenerative case with no local resources. However, these papers discuss when Cloud resources should be brought in, whilst our work discusses how to optimally manage the invocation / termination of instances. These two approaches can therefore be seen as complementary.

Marshall [11] proposes policies for how to extend the number of cloud instances to use along with simulations of a small number of short running synthetic jobs to evaluate overhead times. Here we use a full trace log containing over half a million real jobs and evaluate for both overhead and Cloud cost.

Palankar [16] showed the criticality of data locality in the Cloud. We see that moving our data to the Cloud will help to reduce the data locality problem.

Additional functionality such as Amazon CloudWatch [1] allow instances to be brought up and down dependant on the characteristics of existing instances. The approaches we propose can be built into such a system.

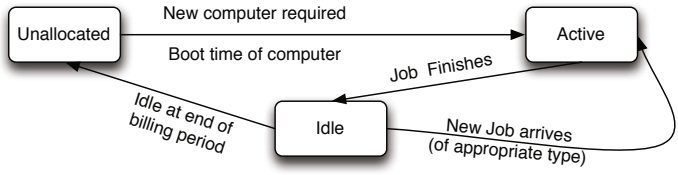


Fig. 1. The state diagram for Instances

3 Cloud Cluster Model

We discuss the general Cloud Cluster architecture that we are modelling. Each individual user is able to submit jobs to the Cluster at any time. A Job Management Service is used to deploy these jobs to dynamic pool of instances within the Cloud. This can be one of the many existing Cluster management tool such as Condor [10], PBS [17] or (Sun) Grid Engine [15]. Additional software is required to allow the cluster to add Cloud instances, when required, and terminate these when no longer needed. Instances within the Cloud Cluster can be seen as being in one of three states with interactions illustrated in Figure 1:

- **Unallocated**: those potential Cloud instances not currently under contract of the cluster – (effectively) an infinite set. The Job Management Service can 'hire' such an instance to run a job placing it in the Active state.
- **Active**: the instance is 'hired' by the cluster and is currently servicing a job for a user. On job completion the instance will enter the idle state.
- **Idle**: the instance is 'hired' by the cluster but not currently servicing a job. The instance will become active if the cluster allocates a job before the end of it's billing period otherwise it will be released into the unallocated state.

As an instance incurs the same charge irrespective of when it is terminated within a billing period it is always kept 'hired' until the end of this period – increasing the chance of there being an idle instance when a job arrives. Instances can either be provisioned for all users within a cluster or only a specific user.

Jobs are first matched against idle instances capable of accepting jobs from that user. Receiving continuous service from the active instance until completion when the instance will become idle. Jobs arriving to find no 'idle' instances capable of servicing them will cause a new instance to be provisioned, requiring time for the operating system and middleware to start, before running the job.

4 Policy

In this section we discuss a number of policies which can be applied to a Cloud based Cluster aimed at reducing the number of hours consumed by the Cluster in order to successfully complete all jobs. In each case we indicate how the policy could be realised and how we would expect the Cloud cluster to be affected.

P1: Limiting the number of Cloud instances: Although the Cloud offers (apparently) infinite availability each provider has thresholds over which prior

approval is required for more resources – EC2 is restricted to 20 instances per region, giving an overall limit of 200 instances. Limiting instances helps prevent excessive instance consumption when users submit large numbers of short jobs.

Jobs arriving to find no instances in the 'idle' state can either cause the invocation of a new instance, provided that the instance limit has not been reached, or be placed into a queue of pending jobs. Pending jobs are services in a FCFS manner as instances become 'idle'. This will reduce the number of hours consumed by the cluster at the expense of increasing the average make-span.

P2: Merging of different user's jobs: Allowing users to share Cloud instances could help reduce costs as less instances will be required and reduce make-span as jobs are more likely to discover usable idle instances. As the current cluster shares resources we are not reducing the security available to the user.

This can be implemented by having one central pool of Cloud instances with jobs being allocated to any 'idle' instance. This does, however, bring in the complexity of how to sub-charge for these 'shared' hours of Cloud usage. This can be done after an instance has been terminated using the following equation:

$$Cost_i = hours \times price \times \frac{\sum_{j=1}^{N_i} execution\_time_{i,j}}{\sum_{k=1}^{M}(\sum_{j=1}^{N_k} execution\_time_{k,j})} \quad (1)$$

Where *hours* is the number of hours the instance was active, *price* is the unit price per hour, N_i is the number of jobs from source i, M is the number of sources and $execution\_time_{i,j}$ is the execution time for the j'th job from source i. Thus each source's cost is based on the proportion of the overall time the source was active on the instance relative to all sources on this instance.

P3: Instance keep-alive: Experimentation has shown the time for an instance to initialise and start accepting jobs can range from 1 to 15 minutes, with high values being detrimental to overheads. This policy allows idle instances at the end of a billing period to remain 'hired' for the next period with probability p. To prevent a half-life decay an instance which is 'idle' for a full hour will always terminate. This policy many have a more impact on the make-span than on the cost saving, as an arriving job is more likely to find an 'idle' instance. The cost may go up due to instances running when no jobs are present.

P4: Delaying the start of Instances: This policy, like P1, aims to reduce the impact of short running jobs. Arriving jobs which cannot be allocated to an 'idle' instance are queued. If the job fails to obtain an instance within t minutes then a new instance will be created. This helps the overall cost for using the Cloud by reducing the chance of instances being brought up for short-running jobs. The average make-span will go up due to the extra waiting time.

P5: Removing the delay on starting an Instance: Policy P4 can be slow to react when large numbers of jobs are submitted. This throttling can be removed while the queue size exceeds a given proportion (r) of the maximum instance

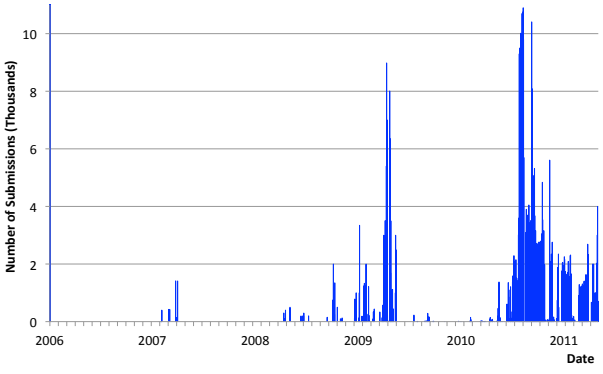


Fig. 2. Profile of job submissions

count. Although this is expected to increase the cost of using the Cloud it should reduce the average make-span.

P6: Waiting for the start of the next hour: Where a Cloud provider adopts a wall-clock charging model it may not be economical to start an instance just before the end of an hour. Jobs arriving within b minutes of the end of an hour are delayed until the start of the next hour. Although this will increase the make-span of the job it should decrease the cost to the Cloud.

5 Simulation Environment

Our simulations are based on trace logs from the Condor high-throughput cluster at Newcastle University [13,14]. The 1359 student access desktops, running Microsoft Windows XP, replaced on a four year rolling cycle. As the main focus of our work is the comparison of different polices for reducing the cost of using the Cloud we ignore the differences between local and Cloud performance and assume the Cloud execution time will match the original execution time.

Figure 2 depicts the profile for the 574,701 successful jobs made between 13th October 2005 to 13th March 2011 by 21 unique users requiring 228,688 hours to execute. Jobs which were terminated before completing by the submitting user have not been used for this simulation due to their lack of execution time.

6 Simulations and Results

We evaluate our policies in order to assess an optimal set of policies for our Cloud cluster. These evaluations could be performed on different cluster data and we believe that the conclusions from this work will be applicable to other similar clusters. As the cost per hour of different providers varies and even temporally within a provider we quote all values here in hours consumed. A simple multiplication of this value by the current hourly rate will yield the real cost.

Table 1 shows the results under the assumption of infinite instance availability. Exact charging gives a significant decrease in hours consumed over wall-clock charging. This equates to 97,000 hours or ∼23.3 minutes for each instance started. The make-span is almost identical with the discrepancy attributable to wall-clock instances (in general) powering down before exact charged instances, thus arriving jobs are less likely to find idle instances. The rest of the results are computed relative to the exact charge case to exemplify the relative benefits.

The following key letters are used to indicate the Cloud pricing model and source merging policy (P2) in the following graphs: **h** - wall-clock charging, **w** - exact charging, **m** - jobs can run on any instance, **s** - jobs can only be run on

Table 1. Baseline results for an infinite size Cloud Cluster

Charge Type	Hours Consumed	Average make-span
Exact charging	401,981	24.65 minutes
Wall-clock charging	472,571	24.75 minutes

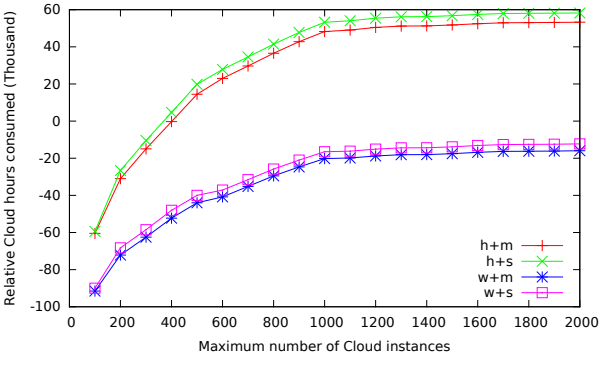


Fig. 3. Varying the maximum instance count on Cloud hours consumed

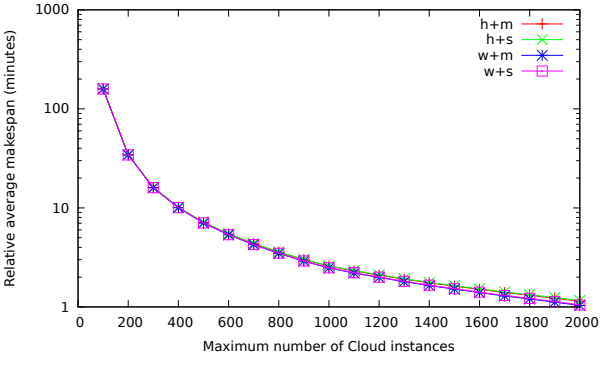


Fig. 4. Effect of varying the maximum instance count on average make-span

instances allocated to its own source. If present the number is the amount of time (in minutes) relative the experiment being performed.

Figures 3 and 4 exemplify policy P1. Increasing the maximum instances increases the hours consumed but reduces the average make-span. Exact charging remains much more optimal than wall-clock charging. The impact of merging jobs by different users (P2) appears to have only a marginal effect (1.3% ∼5,000 hours) on hours consumed, and no perceivable impact on make-span – a consequence of the cluster users working at different times, if more users were active at the same time this could lead to a significant reduction in hours. All subsequent experiments have been performed with a maximum of 500 instances.

In figures 5 and 6 we investigate the effect of start-up time for instances and whether it is beneficial to keep instances 'idle' in the absence of jobs (P3). As the startup time of instances increases so too does the number of hours consumed and the average make-span. Only for start-up times in excess of ten minutes is there a perceivable benefit to the make-span in increasing the chance of an

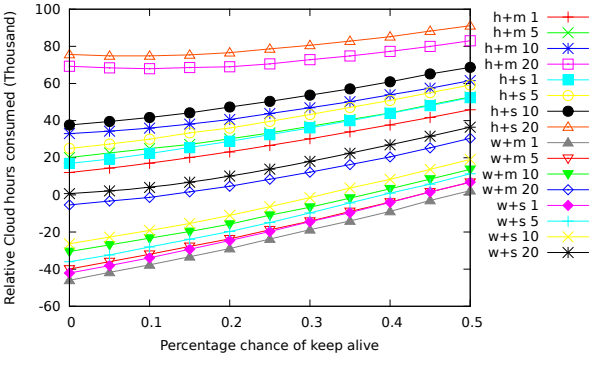


Fig. 5. Varying the boot time and chance of keep-alive on hours consumed

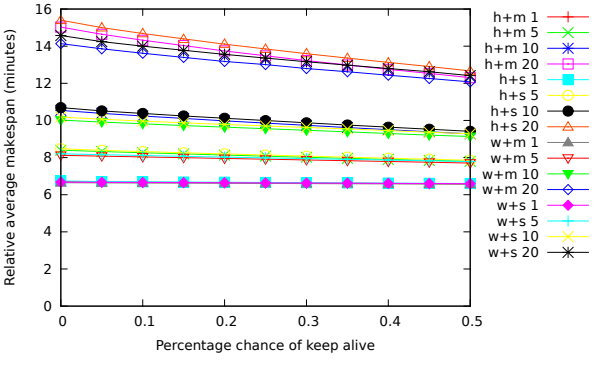


Fig. 6. Varying the boot time and chance of keep-alive on average make-span

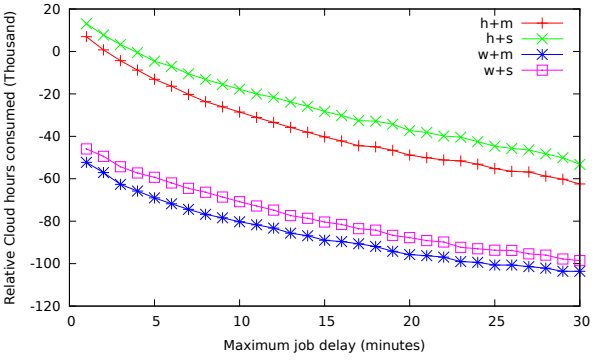


Fig. 7. Varying max instance delay on hours consumed

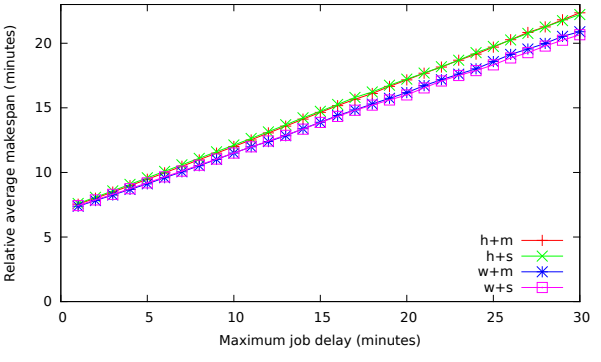


Fig. 8. Varying max instance delay on average make-span

instance remaining 'hired', though the hours consumed increase almost linearly as we increase the chance of an instance remaining 'hired'. Therefore using a policy to keep instances 'hired' in the absence of jobs only makes sense for boot times over ten minutes and with a probability of only around 10-15%.

Policy P4 is evaluated in figures 7 and 8 in which we vary the maximum delay time, for starting a new Cloud instance, in an attempt to reduce the hours consumed. As we increase the maximum delay the hours consumed decreases but the average make-span increases. As these two characteristics are inversely proportional it is necessary to balance maximum delay against increases in make-span. The reduction in hours consumed is slightly more pronounced for smaller values of maximum delay whilst the make-span is almost linear which would suggest that a small value for maximum job delay would be appropriate.

For figures 9 and 10 we investigate policy P5 in which we remove the delay on starting new instances (P4) when there is a high influx of jobs to the Cloud cluster. Here there is a clear distinction between the policies for merging or not merging different users jobs. For the hours consumed if the policy is not to merge sources then there is a benefit of having a 5% threshold on removing the delay

to starting resources. However, increasing this threshold has no further impact. If merging sources then the improvement isn't immediate though it does become better than the non-merged approach at around 10-20% capping. For make-span the non-merged policy reaches a maximum at 5% threshold whilst the merged policy approaches this as the threshold increases to 50%. Thus if used a capping of over 5% for the non-merged approach and around 5-15% for merged sources.

We explore the effect of delaying starting up new instances till the start of the next wall-clock hour (P6) in figures 11 and 12. The number of hours consumed decreases as we increase the number of minutes before the start of an hour. This is most significant for the cases of Cloud instances with wall-clock charging. The exact charging model also shows this reduction as we are producing a variation of policy P4 in which the maximum delay on instance creation is variable. When we look at the average make-span the value does increase, but only slowly, rising by only two minutes over the half hour range. Thus unless make-span is the overriding concern then this policy should be used with a high value.

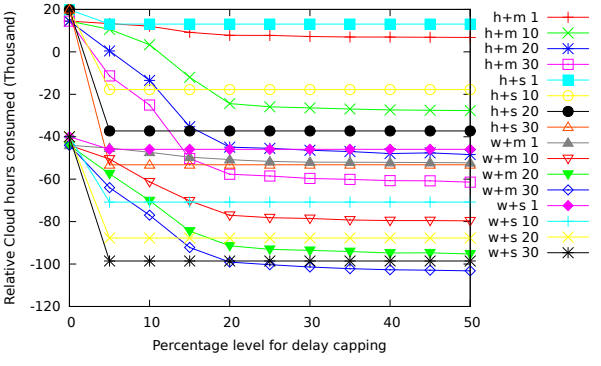


Fig. 9. Varying max job delay and job delay capping on hours consumed

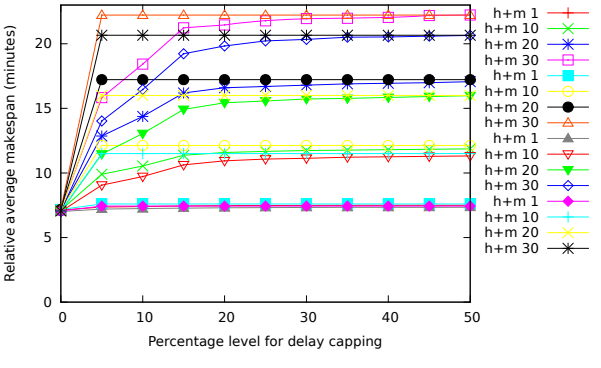


Fig. 10. Varying max job delay and job delay capping on average make-span

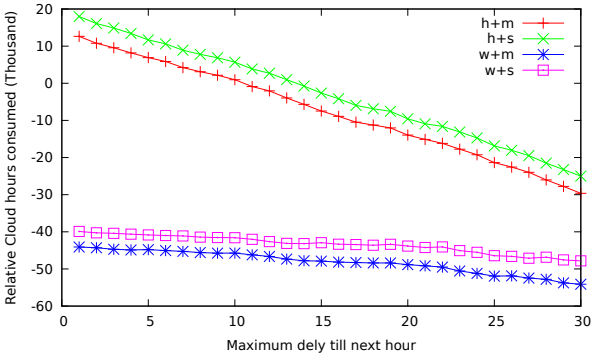


Fig. 11. Varying max delay to next hour on hours consumed

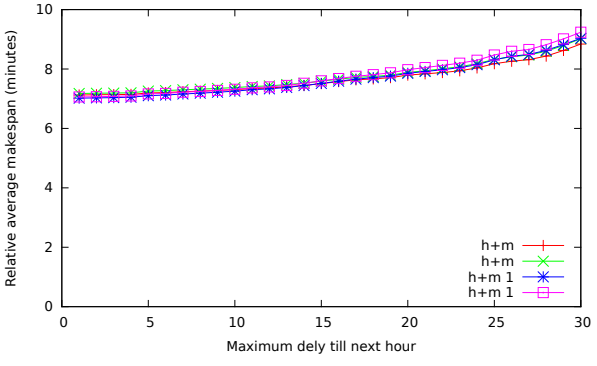


Fig. 12. Varying max delay to next hour capping on average make-span

7 Conclusions

In this paper we have demonstrated through the use of simulation how a Cluster can be deployed completely on the Cloud. We have demonstrated how policies over provisioning of instances can effect the overall cost of using the Cloud and the consequence this has on average make-span for users jobs. All of these policies have the potential to decrease the cost of using the Cloud at the expense of increasing the make-span. It is therefore important to weigh up these two considerations in order to select an optimal policy set for a given Cloud cluster.

The policies of delaying the start of instances (P4) and delaying the start of instances to the next hour (P6) appear to have the biggest impact on cost of using the Cloud with least impact on the job make-span. Especially in the latter case for resources with wall-clock charging. All the presented policies have the potential to be used together thus increasing the potential gain. As the policies effect when to start up instances and how long to wait before doing so a merging of the policies would require one policy to take precedence over another.

For example delaying jobs for at least ten minutes (P4) unless they are within twenty minutes of the start of the next hour (P6).

Although the Newcastle cluster is currently free it does have drawbacks: non-dedicated resources and imposed operating system. If electricity charges were introduced – 120MWh would currently equate to 335,000 hours on Amazon – although the cost of working in-house would still be cheeper the cumulative benefits for working on the Cloud would make it appear a much better option.

References

1. Amazon Web Services: CloudWatch, `http://aws.amazon.com/cloudwatch/`
2. Amazon Web Services: Elastic Compute Cloud, `http://aws.amazon.com/ec2/`
3. Amazon Web Services: Simple Storage Service, `http://aws.amazon.com/s3/`
4. 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. Commun. ACM 53(4), 50–58 (2010)
5. de Assuncao, M.D., di Costanzo, A., Buyya, R.: Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, HPDC 2009, pp. 141–150. ACM, New York (2009)
6. Van den Bossche, R., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 228–235 (July 2010)
7. Cheng, P.S.: Trace-driven system modeling. IBM Systems Journal 8(4), 280–289 (1969)
8. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the montage example. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC 2008, pp. 50:1–50:12. IEEE Press, Piscataway (2008)
9. Li, B., O'Loughlin, J., Gillam, L.: Fair benchmarking for cloud computing systems. Poster presentation, Cloud Workshop, UK All Hands Meeting (2011)
10. Litzkow, M., Livney, M., Mutka, M.W.: Condor-a hunter of idle workstations. In: 8th International Conference on Distributed Computing Systems, pp. 104–111 (1998)
11. Marshall, P., Keahey, K., Freeman, T.: Elastic site: Using clouds to elastically extend site resources. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), pp. 43–52 (May 2010)
12. Mattess, M., Vecchiola, C., Buyya, R.: Managing peak loads by leasing cloud infrastructure services from a spot market. In: Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications, HPCC 2010, pp. 180–188. IEEE Computer Society, Washington, DC, USA (2010)
13. McGough, A.S., Robinson, P., Gerrard, C., Haldane, P., Hamlander, S., Sharples, D., Swan, D., Wheater, S.: Intelligent power management over large clusters. In: International Conference on Green Computing and Communications (GreenCom 2010) (2010)
14. McGough, A.S., Gerrard, C., Noble, J., Robinson, P., Wheater, S.: Analysis of power-saving techniques over a large multi-use cluster. In: International Conference on Cloud and Green Computing (CGC 2011) (2011)

15. Oracle: (Sun) Grid Engine,
http://www.oracle.com/technetwork/oem/grid-engine-166852.html
16. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon s3 for science grids: a viable solution? In: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, DADC 2008, pp. 55–64. ACM, New York (2008)
17. Veridian Systems: Portable Batch Systems, http://www.openpbs.org
18. Youseff, L., Butrico, M., Da Silva, D.: Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop (GCE 2008), pp. 1–10 (2008)

Erratum: Risk Assessment in Service Provider Communities

Ioan Petri[1,2], Omer F. Rana[3],
Yacine Regzui[1], and Gheorghe Cosmin Silaghi[2]

[1] School of Engineering, Cardiff University, UK
[2] Business Information Systems, Babes-Bolyai University, Romania
[3] School of Computer Science & Informatics,
Cardiff University, UK

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 135–147, 2012.

DOI 10.1007/978-3-642-28675-9_15

In the originally published version of the paper "Risk Assessment in Service Provider Communities" the name of the author Yacine Rezgui was misspelled as Yacine Regzui.

The original online version for this chapter can be found at
http://dx.doi.org/10.1007/978-3-642-28675-9_10

Author Index