1-1-2015

Resource Management In Cloud And Big Data Systems

Lena Mashayekhy
Wayne State University,

RESOURCE MANAGEMENT IN CLOUD AND BIG DATA SYSTEMS

by

LENA MASHAYEKHY

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2015

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my parents and my sisters.

ACKNOWLEDGMENTS

I am heartily thankful to my advisor Dr. Daniel Grosu for his guidance and support throughout my Ph.D. studies. His mentorship, encouragement, and support were instrumental to my academic achievements. I am also grateful to the members of my dissertation committee Dr. Monica Brockmeyer, Dr. Nathan Fisher, and Dr. Ratna Babu Chinnam for providing me with excellent and detailed feedback on my work and dissertation.

I would like to express my appreciation to Dean Farshad Fotouhi, Dr. Weisong Shi, Dr. Loren Schwiebert, Dr. Ratna Babu Chinnam, Dr. Leslie Monplaisir, Dr. Sameena Shah, Dr. Mihaela van der Schaar, and Dr. Lorenzo Alvisi for their support, especially during my academic job search.

I am greatly thankful to my friends at Wayne State University for making this journey much more pleasant.

I owe special thanks to my father Aman and my mother Esmat for instilling in me a love of learning and a desire to excel, and for their continuous support of my goals. All I have and will accomplish are only possible due to their love and sacrifices. I would like to thank my sisters Hoda and Zahra for always being there for me and for being so energetic, optimistic, and supportive.

Finally, I would like to express my sincere appreciation to my brilliant and loving Mahyar. I am truly grateful to have him in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Clouds are large-scale distributed computing systems built around core concepts such as computing as utility, virtualization of resources, on demand access to computing resources, and outsourcing computing services [169]. These concepts have positioned the clouds as an attractive platform for businesses enabling them to outsource some of their IT operations. In fact, the clouds services market share in the IT business has rapidly increased, and it is estimated to reach $150 billion by 2015 [138]. Cloud services are offered as three main categories: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS), defined by the National Institute of Science and Technology (NIST) [114] as follows:

> SaaS: "The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings."
>
> PaaS: "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment."
>
> IaaS: "The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure

but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)."

In this dissertation, we focus on IaaS, where cloud providers offer different types of low level resources of their physical machines (PMs) in the form of virtual machine (VM) instances.

Cloud computing systems' ability to provide on demand access to always-on computing utilities has attracted many enterprises due to their cost-benefit ratios, leading to rapid growth of the cloud computing market. Such market, however, presents a host of new challenges due to the dynamic nature of users' demands and heterogeneity of the resources. The variability of users' demands increases when it comes to their requests for data-intensive applications. The ever-growing complexity of IaaS makes human administration and management inefficient and, in most of the cases, unfeasible. Therefore, avoiding direct management actions in resource allocation, VM provisioning, VM pricing, and monitoring, requires self-management and self-optimizing mechanisms. On the other hand, cloud users want to minimize their expenses while meeting their performance requirements. However, the challenge is how to allocate and price resources in a mutually optimal way with the lack of information sharing between cloud users and cloud providers. The aim of this dissertation is to design such mechanisms that facilitate autonomic provisioning of cloud resources considering the incentives of both cloud users and cloud providers. The proposed mechanisms can be incorporated in system tools for self-managing the cloud infrastructure. Designing mechanisms that are robust against manipulation of users are very critical for these systems. A resource management mechanism is responsible for deciding the allocation and the payment charged to the users for the allocated resources. Such a mechanism is composed of an allocation algorithm and a payment function. We propose several mechanisms in Chapter 1.2, Chapter 3, and Chapter 4 to address the problem of Virtual Machine (VM) provisioning, allocation, and pricing in clouds in different settings.

As cloud computing started growing rapidly, cloud systems comprising multiple cloud providers emerged. Such systems form coalitions to combine resources from the participating cloud providers in order to provide better computing power to them. However, these

cloud providers should not have incentives to leave their coalitions and join other coalitions. In Chapter 5, we design a mechanism for federation formation in clouds.

Several businesses and organizations are faced with an ever-growing need for analyzing the unprecedented amounts of available data. Such need challenges existing methods, and requires novel approaches and technologies in order to cope with the complexities of big data processing. One of the major challenges of processing data intensive applications is minimizing their energy costs. Electricity used in US data centers in 2010 accounted for about 2% of total electricity used nationwide [76]. In addition, the energy consumed by the data centers is growing at over 15% annually, and the energy costs make up about 42% of the data centers' operating costs [56]. Considering that server costs are consistently falling, it should be no surprise that in the near future a big percentage of the data centers' costs will be energy costs. Therefore, it is critical for the data centers to minimize their energy consumption when offering services to customers.

Big data applications run on large clusters within data centers, where their energy costs make energy efficiency of executing such applications a critical concern. MapReduce [36] and its open-source implementation, Hadoop [6], have emerged as the leading computing platforms for big data analytics. Improving energy efficiency of MapReduce applications leads to a significant reduction of the overall cost of data centers. In this dissertation, we address this issue by designing MapReduce scheduling algorithms that improve the energy efficiency of running each individual application, while satisfying the service level agreement (SLA). To address this problem, we propose two algorithms in Chapter 6.

1.1 Our Contributions

In this section, we present the outline of our dissertation and the summary of our contributions. We summarize below the five research projects that we accomplished as part of this dissertation.

- **A PTAS Mechanism for Provisioning and Allocation of Heterogeneous Cloud Resources.** Cloud providers provision their heterogeneous resources such as

CPUs, memory, and storage in the form of Virtual Machine (VM) instances which are then allocated to the users. One of the major challenges faced by the cloud providers is to allocate and provision these resources such that their profit is maximized, and the resources are utilized efficiently. Recently, cloud providers have introduced auction-based models which allow users to submit bids for their requested VMs. We address the problem of autonomic VM provisioning and allocation for the auction-based model considering multiple types of resources by designing an approximation mechanism. In addition, the mechanism determines the payment the users have to pay for using the allocated resources. This problem is computationally intractable, and our proposed mechanism is by far the strongest approximation result that can be achieved for this problem. We show that the proposed approximation mechanism is a Polynomial-Time Approximation Scheme (PTAS). Furthermore, our proposed mechanism drives the system into an equilibrium in which the users do not have incentives to manipulate the system by untruthfully reporting their VM bundle requests and valuations. We perform extensive experiments using real workload traces in order to investigate the performance of the proposed mechanism, PTAS-VMPAC. The results of this research were published in Proceedings of the ACM Cloud and Autonomic Computing Conference (CAC'13) [96] and an extended version of this paper has been accepted for publication in the IEEE Transactions on Parallel and Distributed Systems (TPDS) [101]. We present this work in detail in Chapter 1.2.

- **Physical Machine Resource Management in Clouds: A Mechanism Design Approach.** We address the problem of physical machine resource management in clouds considering multiple types of physical machines and resources. We formulate this problem in an auction-based setting and design optimal and approximate strategy-proof mechanisms that solve it. Our proposed mechanisms consist of a winner determination algorithm that selects the users, provisions the virtual machines (VMs) to physical machines (PM), and allocates them to the selected users; and a payment function that determines the amount that each selected user needs to pay to

the cloud provider. We prove that our proposed approximate winner determination algorithm satisfies the loser-independent property, making the approximate mechanism robust against strategic users who try to manipulate the system by changing other users' allocations. We show that our proposed mechanisms are strategy-proof, that is, the users do not have incentives to lie about their requested bundles of VM instances and their valuations. In addition, our proposed mechanisms are in alignment with green cloud computing strategies in which physical machines can be powered on or off to save energy. Our theoretical analysis shows that the proposed approximation mechanism has an approximation ratio of 3. We perform extensive experiments in order to investigate the performance of our proposed approximation mechanism compared to that of the optimal mechanism. The results of this research were published in the IEEE Transactions on Cloud Computing, Special Issue on Economics and Market Mechanisms for Cloud Computing [100]. We present this work in detail in Chapter 3.

- **An Online Mechanism for Resource Allocation and Pricing in Clouds.** Cloud providers provision their various resources such as CPUs, memory, and storage in the form of Virtual Machine (VM) instances which are then allocated to the users. The users are charged based on a pay-as-you-go model, and their payments should be determined by considering both their incentives and the incentives of the cloud providers. Auction markets can capture such incentives, where users name their own prices for their requested VMs. We design an auction-based online mechanism for VM provisioning, allocation, and pricing in clouds that consider several types of resources. Our proposed online mechanism makes no assumptions about future demand of VMs, which is the case in real cloud settings. The proposed online mechanism is invoked as soon as a user places a request or some of the allocated resources are released and become available. The mechanism allocates VM instances to selected users for the period they are requested for, and ensures that the users will continue using their VM instances for the entire requested period. In addition, the mechanism determines

the payment the users have to pay for using the allocated resources. We prove that the mechanism is incentive-compatible, that is, it gives incentives to the users to reveal their actual requests. We investigate the performance of our proposed mechanism through extensive experiments. A paper describing this research was published in the Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14) [104] and an extended version of this paper has been accepted for publication in the IEEE Transactions on Computers (TC) [103]. We present this work in detail in Chapter 4.

- **Cloud Federations in the Sky: Formation Game and Mechanism.** The amount of computing resources required by current and future data-intensive applications is expected to increase dramatically, creating high demands for cloud resources. The cloud providers' available resources may not be sufficient enough to cope with such demands. Therefore, the cloud providers need to reshape their business structures and seek to improve their dynamic resource scaling capabilities. Federated clouds offer a practical platform for addressing this service management issue. We introduce a cloud federation formation game that considers the cooperation of the cloud providers in offering cloud IaaS services. Based on the proposed federation formation game, we design a cloud federation formation mechanism that enables the cloud providers to dynamically form a cloud federation maximizing their profit. In addition, the proposed mechanism produces a stable cloud federation structure, that is, the participating cloud providers in the federation do not have incentives to break away from the federation. We analyze the performance of the proposed mechanism by performing extensive experiments. The results of the experiments show that the cloud federation obtained by our proposed mechanism is stable, yielding high profit for the participating cloud providers. A paper describing this research was published in the Proceedings of the 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'12) [90] and an extended version of this paper was published in the IEEE Transactions on Cloud Computing (TCC) [99]. We present this work in

detail in Chapter 5.

- **Energy-aware Scheduling of MapReduce Jobs for Big Data Applications.** The majority of large-scale data intensive applications executed by data centers are based on MapReduce or its open-source implementation, Hadoop. Such applications are executed on large clusters requiring large amounts of energy, making the energy costs a considerable fraction of the data center's overall costs. Therefore minimizing the energy consumption when executing each MapReduce job is a critical concern for data centers. We propose a framework for improving the energy efficiency of MapReduce applications, while satisfying the service level agreement (SLA). We first model the problem of energy-aware scheduling of a single MapReduce job as an Integer Program. We then propose two heuristic algorithms, called Energy-aware MapReduce Scheduling Algorithms (EMRSA-I and EMRSA-II), that find the assignments of map and reduce tasks to the machine slots in order to minimize the energy consumed when executing the application. We perform extensive experiments on a Hadoop cluster to determine the energy consumption and execution time for several workloads from the HiBench benchmark suite including TeraSort, PageRank, and K-means Clustering, and then use this data in an extensive simulation study to analyze the performance of the proposed algorithms. The results show that EMRSA-I and EMRSA-II are able to find near optimal job schedules consuming approximately 40% less energy on average than the schedules obtained by a common practice scheduler that minimizes the makespan. A paper describing this research was published in the Proceedings of the 3rd IEEE International Congress on Big Data (BigData'14) [102] and an extended version of this paper has been accepted for publication in the IEEE Transactions on Parallel and Distributed Systems (TPDS) [105]. We present this work in detail in Chapter 6.

1.2 Organization

The rest of this dissertation is organized as follows. In Chapter 1.2, we present our research on designing VM provisioning, allocation, and pricing mechanism for clouds. The mechanism is a PTAS (Polynomial-Time Approximation Scheme) and truthful. In Chapter 3, we present the design of a mechanism for solving the VM allocation and pricing problem considering the PMs. In Chapter 4, we present the design of online mechanisms for the VM allocation and pricing problem making no assumptions about the future demand for VMs. In Chapter 5, we present our research on designing a mechanism for cloud federation formation problem. In Chapter 6, we present our research on designing energy-aware schedulers for big data applications. In Chapter 7, we describe the possible future directions of our research, and conclude the dissertation.

CHAPTER 2: A PTAS MECHANISM FOR RESOURCE PROVISIONING AND ALLOCATION

2.1 Introduction

Cloud computing systems provide a large pool of abstracted, virtualized, and dynamically scalable resources to users as Infrastructure as a Service (IaaS). More specifically, the resources are offered to users as different types of virtual machine (VM) instances based on a pay-as-you-go model. For example, Amazon Elastic Compute Cloud (Amazon EC2) [1] is currently offering four types of VM instances: Medium (M), Large (L), Extra large (XL), and 2 Extra large (2XL), charging a fixed price per hour for each type of VM instance.

The ever-growing complexity of IaaS makes human administration and management inefficient and, in most of the cases, unfeasible. Therefore, avoiding direct management actions in resource allocation, VM provisioning, VM pricing, and monitoring, requires self-management and self-optimizing mechanisms. The aim of this chapter is to design such mechanisms that facilitate autonomic provisioning of cloud resources based on the user demand and the availability of resources. The proposed mechanisms can be incorporated in system tools for self-managing the cloud infrastructure [134].

Recently, cloud providers have introduced auction-based models when offering IaaS, which allow users to name their own price for their requested VM instances. Auctions have been proven to be effective market-based mechanisms for trading cloud services which not only benefit users by allowing them to obtain their requested resources at lower prices, but also allow cloud providers to utilize more resources and increase their profits. Mainstream cloud provider powerhouses such as Amazon have been offering cloud services in an auction market, the Amazon spot market, for several years. Based on Amazon's report [3], many of its users such as Scribd, So-net, Numerate, Backtype, and Fliptop saved more than 50% using its auction-based cloud market over its fixed-price market. In addition, a new initiative by Deutsche Börse Cloud Exchange, will launch in 2014 a vendor-neutral marketplace for cloud resources. This market will be a platform for offering, buying and deploying IaaS

in an auction setting [178].

We consider an auction market with a set of users and a set of heterogeneous VM instances, where each user can bid for any arbitrary combinations of VMs (bundle of VMs). Our setup and mechanisms are different from the Amazon spot market. The Amazon spot market allows requests only for individual VM instances and not for bundles of VM instances of different types. Therefore, users have to request each VM for their bundle individually. However, in such an auction-based setting, there is no guarantee of receiving the requested VMs all together. In our setting, we allow users to request bundles of heterogeneous VM instances such as requesting communication-intensive VMs and computation-intensive VMs together. In practice, there are many applications that require such heterogeneous bundles of VMs. For example, a social game application composed of three layers: front-end web server, load balancing, and back-end data storage, requires a bundle of heterogeneous VMs composed of communication-intensive VMs, computation-intensive VMs, and storage-intensive VMs, respectively [45].

Each user has a private value for her requested bundle. In our model, each user is interested in a single bundle of VM instances, and bid only for that bundle. Such a user obtains the specified value if she is allocated the whole bundle of VM instances (or any superset of it) and zero value, if she is allocated any other bundle. The users are also selfish in the sense that they want to maximize their own utilities. It may be beneficial for them to manipulate the system by declaring different bundles or bids from their actual requests. In an untruthful auction, users may bid much lower (than their actual valuations) which not only may hurt other users, but also may indirectly lead to profit losses for the cloud provider. Thus, unless strategy-proofness is enforced, maximizing the revenue may not be effective. In strategy-proof auctions, the dominating strategy for users is to bid truthfully, thereby eliminating the fear of market manipulations and the overhead of strategizing over others. When users report their true valuations, the cloud provider can allocate its resources efficiently to users who value them the most. However, allowing users to bid on bundles of VMs makes the design of strategy-proof mechanisms more challenging. This is due to the fact that by allowing bids on bundles, the dimensionality of the problem

increases. A major problem in such settings is determining the optimal allocation and protecting against manipulations by the users. Because finding the optimal allocation is computationally intractable [82], designing strategy-proof approximation mechanisms for solving the problem is of major interest.

In this chapter, we design a strategy-proof polynomial time approximation scheme (PTAS) mechanism that solves the VM instance provisioning and allocation problem. The goal is to find an allocation of resources to the users maximizing the social welfare, where the social welfare is the sum of users' valuations. We also design an exponential time strategy-proof optimal mechanism that will serve as a benchmark for the performance of the proposed PTAS mechanism. Our proposed PTAS mechanism is strategy-proof that is, the users do not have incentives to lie about their requested bundles of VM instances and their valuations. The proposed mechanism is designed to adapt to changing conditions (i.e., users requests) and to lead the system into an equilibrium in which users do not have incentives to manipulate the system by untruthfully reporting their resource requests and valuations.

2.1.1 Our Contributions

We address the problem of VM provisioning and allocation in clouds in the presence of multiple types of heterogeneous resources. First, we design a strategy-proof optimal mechanism that uses a dynamic programming algorithm to optimally select the winning users. We then design a strategy-proof approximation mechanism in spite the fact that approximation algorithms, in general, do not necessarily satisfy the properties required to guarantee strategy-proofness. In doing so, the allocation and payment determination of the proposed mechanisms are designed to satisfy the strategy-proofness property. Strategy-proof mechanisms drive the system into an equilibrium in which the users do not have incentives to manipulate the system by untruthfully reporting their VM bundle requests and valuations. We also show that the proposed approximation mechanism is a PTAS (Polynomial-Time Approximation Scheme) which is by far the strongest approximation result that can be achieved for this problem, unless $P = NP$. To the best of our knowledge, this is the

first study proposing a strategy-proof PTAS mechanism for solving the VM provisioning and allocation problem in clouds. The proposed mechanism allows dynamic provisioning of VMs, and does not require pre-provisioning the VMs. As a result, cloud providers can fulfill dynamic market demands efficiently. A key property of our proposed mechanism is the consideration of multiple types of heterogeneous resources for VMs which is the case in real cloud settings. We analyze the properties of the proposed mechanism and perform extensive experiments. The results show that the proposed PTAS mechanism determines near optimal allocations while satisfying the strategy-proofness property.

2.1.2 Related Work

The primary objective of mechanism design is to obtain system wide solutions for problems where multiple self-interested users with private information interact. Mechanism design provides a framework for designing mechanisms that align the system's incentives with those of the participants. For a comprehensive introduction to mechanism design, the reader is referred to [129].

Researchers approached the problem of VM provisioning in clouds from different points of view. Wood *et al.* [183] proposed an approach for dynamic provisioning of VMs by defining a unique metric based on the consumption of the three resources: CPU, network and memory. Ferrer *et al.* [43] proposed a toolkit for the cloud service and infrastructure providers. The toolkit aims to provide a foundation for a reliable cloud computing industry, by addressing the whole service life cycle. Casalicchio *et al.* [27] proposed a heuristic (hill-climbing) algorithm to maximize revenue in VM allocation problems satisfying capacity, availability, SLA, and VM migration constraints. Bjorkqvist *et al.* [18] proposed an opportunistic service provisioning strategy minimizing the service provisioning costs by provisioning a small number of faster VMs, while maintaining the target system utilization. Mashayekhy *et al.* [99] proposed a federation formation mechanism for resource provisioning and allocation in clouds considering several heterogeneous resources. Xiong *et al.* [185] considered an economical provisioning, where VMs are allocated to achieve a balanced resource allocation and a better overall performance. Sharma *et al.* [159] consid-

Table 2.1: Comparison of auction-based VM allocation methods.

Reference	heterogeneous VMs	strategy-proofness	optimality
[193]	x	in expectation	approx.
[196]	✓	in expectation	approx.
[195]	x	✓	approx.
[45]	✓	x	optimal
[124]	✓	✓	approx.
this study	✓	✓	optimal & PTAS approx.

ered minimizing the provisioning cost by reducing the time to transit to new configurations and optimizing the selection of virtual server configurations. Kokkinos *et al.* [75] proposed an algorithm to minimize the usage cost of Amazon EC2 instances and to maximize the utilization. The algorithm collects information regarding the current instances and then proposes a new set of instances that could be used for serving the same load. However, our main focus is on dynamic resource provisioning that achieves strategy-proofness and leads the system into an equilibrium. We also propose an approach for determining the prices of the bundles of VM instances.

Pricing and modeling of spot instances have been recently addressed by applying a variety of methodologies. Zhao *et al.* [198] developed two resource rental planning models, deterministic and stochastic, to minimize the operational cost of cloud applications on spot instances. Leslie *et al.* [83] proposed a resource allocation and job scheduling framework based on checkpointing. Huang *et al.* [63] proposed a tool for users to minimize their expenses of running applications in clouds while satisfying the deadlines. The tool automatically determines whether to choose on-demand or spot instances, and also the number of VMs. However, those studies have focused on users/SaaS sides to better use spot instances, while we focus on modeling the problem from IaaS' perspective. Toosi *et al.* [166] proposed pricing policies for federated clouds that increase utilization and profit. Lampe *et al.* [79] formulated the equilibrium price auction allocation problem for pricing and distribution of VMs across physical machines as a binary integer program. Based on this mathematical formulation, they proposed an optimal solution approach as well as a fast heuristic

approach. However, none of the above-mentioned studies guarantee strategy-proofness.

Designing mechanisms for auction-based cloud markets has attracted a great deal of attention. Zaman and Grosu [193] proposed a truthful-in-expectation auction-based mechanism to allocate VM instances that are statically provisioned. However, their mechanism does not consider heterogeneous VMs. In addition, truthfulness-in-expectation is a weaker notion of truthfulness (strategy-proof-ness). Zhang *et al.* [195] proposed a truthful auction-based mechanism for resource allocation in clouds in the presence of only one type of resources. Zhang *et al.* [196] proposed a randomized mechanism for VM allocation in clouds in an auction market considering heterogeneous VMs. Their proposed mechanism is truthful in expectation and is based on a pair of primal and dual LPs. Recently, Fu *et al.* [45] modeled the heterogeneous VM provisioning problem as a coalitional game, and proposed a core-based pricing method that obtains the optimal solution. Their method guarantees the optimal social welfare, at the expense of not obtaining strategy-proofness.

System heterogeneity plays an important role in determining the dynamics of strategy-proof mechanisms [177]. Our proposed PTAS mechanism takes into account the heterogeneity of the systems and that of user requests when making allocation decisions. In our previous work [123, 124], we proposed a family of strategy-proof greedy mechanisms for solving the VM instance provisioning and allocation problem considering multiple types of resources. These existing greedy mechanisms cannot guarantee near optimal solutions. We also proved that the approximation ratios of the greedy mechanisms is $\sqrt{NRC_{max}}$, where N is the number of users, R is the number of types of resources, and C_{max} is the highest capacity among all resources' capacities. We present a stylized example to show how far the solution obtained by a greedy mechanism can be from the optimal solution for the problem. Let us consider a cloud provider with one type of resource with capacity of 100. We consider two users submitting their requests, where the first one bids \$2 for 1 unit of the resource, while the second user bids \$100 for 100 units of the resources. All greedy mechanisms proposed in [124], choose the first user since it has the highest bid density with a value of 2, where the bid density is the ratio of bid and the amount of resources requested. However, the optimal mechanism chooses the second user with the value of 100.

This leads to $98 loss in revenue for the cloud provider if it chooses the greedy mechanism over the optimal mechanism. However, none of the above-mentioned studies proposed a mechanism guaranteeing a near optimal solution which is the case for our PTAS mechanism proposed in this chapter. Our proposed PTAS mechanism is by far the strongest approximation result that can be achieved for the VM provisioning and allocation problem, unless $P = NP$. Our proposed PTAS mechanism represents a big departure from the existing designs of VM allocation mechanisms that only provided a constant approximation guarantee. In Table 2.1, we summarize the differences between the existing auction-based VM allocation methods.

2.1.3 Organization

The rest of the chapter is organized as follows. In Section 2.2, we describe the VM provisioning and allocation problem in clouds, and introduce the basic concepts of mechanism design. In Section 2.3, we present the design of an optimal mechanism for VM provisioning and allocation. In Section 2.4, we present the proposed PTAS mechanism and characterize its properties. In Section 2.5, we evaluate the proposed mechanism by extensive experiments. In Section 2.6, we summarize our results.

2.2 Problem Statement and Preliminaries

We define the problem of VM provisioning and allocation in clouds (VMPAC) in the presence of multiple types of resources as follows. We consider a cloud provider offering a set of M types of VMs, $\mathcal{VM} = \{1, \ldots, M\}$, to users. Each VM consists of heterogeneous resources such as cores, memory, storage, etc. There are a set of R types of resources, $\mathcal{R} = \{1, \ldots, R\}$, where each VM of type $m \in \mathcal{VM}$ has a specific amount of resource of type r denoted by w_{mr}. The cloud provider has restricted capacity, C_r, on each resource $r \in \mathcal{R}$ available for allocation.

Table 2.2 shows the four types of VM instances offered by Amazon EC2 US West (Northern California) Region at the time of writing this chapter. If we consider that

CPU represents the type 1 resource, memory, the type 2 resource, and storage, the type 3 resource, we can characterize, for example, the XLarge instance ($m = 3$) by: $w_{31} = 4$, $w_{32} =$ 15 GB, and $w_{33} = 80$ GB.

We consider that the cloud provider receives requests for bundles of VM instances from a set $\mathcal{U}$ of N users. User $i \in \mathcal{U}$, $i = 1, \ldots, N$ submits a request (S_i, b_i), composed of a bundle of VM instances denoted by $S_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$, and a bid, denoted by b_i representing the maximum price the user is willing to pay for using the requested bundle if it is allocated. User i has a true valuation $v_i(S_i)$ for her requested bundle S_i. Note, that for user i, $v_i(S_i) = b_i$ is her true valuation for S_i. An example of a user request is $(< 2, 1, 4, 3 >, \$10)$, which means that the user is requesting 2 medium VM instances, 1 large VM instance, 4 xlarge VM instances, and 3 2xlarge VM instances, and her bid is \$10. We denote by $a_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$, the total amount of each resource of type r that user i has requested. Note that $a_{ir} > 0$, due to the fact that each VM instance includes all resources.

The goal of the cloud provider is to allocate VMs to users who value the VM instances the most, which can be achieved by maximizing the social welfare. We denote by V the *social welfare*, which is defined as the aggregation of users' valuations, i.e.,

$$V = \sum_{i \in \mathcal{U}} v_i(S_i) x_i \tag{2.1}$$

where x_i, $i \in \mathcal{U}$, are decision variables defined as follows:

$$x_i = \begin{cases} 1 & \text{if bundle } S_i \text{ is allocated to user } i, \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The problem of *VM provisioning and allocation in clouds (VMPAC)* is to find a subset of users who value the VM instances the most, such that the cloud provider fulfills their requested bundle of VMs along with determining their payments. In doing so, the cloud provider first finds such subset of users and then provisions the requested VMs for the

Table 2.2: VM instance types offered by Amazon EC2 - US West (Northern California)

	Medium $m = 1$	Large $m = 2$	XLarge $m = 3$	2XLarge $m = 4$
CPU	1	2	4	8
Memory (GB)	3.75	7.5	15	30
Storage (GB)	4	32	80	160

selected users. Finally, it bills the selected users based on all the submitted bids. We denote by $\mathcal{A}$ and $\mathcal{P}$, the allocation function and the payment function, respectively. The allocation function $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ determines which users receive their requested bundles, and the payment function $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ determines the amount that each user must pay to the cloud provider.

The request of a user i is denoted by $\theta_i = (S_i, b_i)$. We denote by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_N)$ the vector of requests of all N users. User i preferences are characterized by a *quasi-linear utility function* defined as the difference between her valuation and payment, $u_i(\boldsymbol{\theta}) = v_i(\mathcal{A}_i(\boldsymbol{\theta})) - \mathcal{P}_i(\boldsymbol{\theta})$, where $\mathcal{A}_i(\boldsymbol{\theta})$ is the allocated bundle to user i, and $\mathcal{P}_i(\boldsymbol{\theta})$ is the determined payment for user i. Each user's bundle and her valuation is private knowledge. However, users are selfish, and each user's goal is to maximize her utility, thus, she may manipulate the mechanism by submitting a different request from her true request to increase her utility. Since the request of a user is a pair of bundle and value, the user can lie about the value by submitting a higher bid in the hope to increase the likelihood of obtaining her requested bundle, or she can lie about her requested bundle. Such manipulations will lead to an inefficient allocation of VMs and will reduce the revenue obtained by the cloud provider if we do not prevent them by design. As a result, we resort to designing strategy-proof mechanisms that determine allocation and payment of users.

To distinguish user i's truthful request $\theta_i = (S_i, b_i)$ from the actual submitted request (that can be untruthful), we denote the actual request by $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$. We denote by $\boldsymbol{\theta}_{-i}$ the vector of all requests except user i's request (i.e., $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). A mechanism composed of an allocation and a payment function is *strategy-proof* if all users

have incentives to report their true requests.

Definition 1 (Strategy-proofness). *A mechanism consisting of an allocation function $\mathcal{A}$ and a payment function $\mathcal{P}$ is* strategy-proof *(or truthful) if for every user i with a true request θ_i and any other request $\hat{\theta}_i$, and for every request of the other users $\hat{\boldsymbol{\theta}}_{-i}$, it satisfies $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$.*

The definition implies that a mechanism is strategy-proof if truthful reporting of requests is a dominant strategy. As a result, the users maximize their utilities by truthful reporting of their requests irrespective of the requests of the other users. To design a strategy-proof mechanism the allocation function $\mathcal{A}$ must be monotone and the payment determination function $\mathcal{P}$ must be based on the critical payment [119].

To define monotonicity, we need to introduce a preference relation $\succeq$ on the set of requests as follows: $\hat{\theta}'_i \succeq \hat{\theta}_i$ if $\hat{b}'_i \geq \hat{b}_i$ and $\hat{S}_i =< \hat{k}_{i1}, \hat{k}_{i2}, \ldots, \hat{k}_{iM} >$, $\hat{S}'_i =< \hat{k}'_{i1}, \hat{k}'_{i2}, \ldots, \hat{k}'_{iM} >$ such that $\sum_{m \in \mathcal{VM}} \hat{k}'_{im} w_{mr} \leq \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}, \forall r \in \mathcal{R}$. That means request $\hat{\theta}'_i$ is more preferred than $\hat{\theta}_i$ if user i requests fewer resources of each type in her current bundle and/or submits a higher bid.

Definition 2 (Monotonicity). *An allocation function $\mathcal{A}$ is* monotone *if it allocates the resources to user i with $\hat{\theta}_i$ as her declared request, then it also allocates the resources to user i with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.*

The definition implies that any winning user who receives her requested bundle by submitting a request $\hat{\theta}_i$ will still be a winner if she requests a smaller bundle or submits a higher bid.

Definition 3 (Critical payment). *If $\mathcal{A}$ is a monotone allocation function, for every θ_i, there exist a unique value v_i^c, called* critical payment*, such that $\forall \hat{\theta}_i \succeq (S_i, v_i^c)$, $\hat{\theta}_i$ is a winning declaration and otherwise, is a losing declaration.*

A mechanism having the critical payment as a payment function will charge user i, $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = v_i^c$ if user i wins, and $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = 0$ otherwise.

In the design of our PTAS mechanism for solving VMPAC the allocation function needs to satisfy an additional property, called bitonicity.

Definition 4 (Bitonicity)**.** *A monotone allocation function $\mathcal{A}$ is* bitonic *if for any user i:*

(i) if $\mathcal{A}$ allocates the resources to the user i with $\hat{\theta}_i$ as her declared request, then $v_i(\mathcal{A}(\hat{\theta}'_i, \hat{\boldsymbol{\theta}}_{-i})) \geq v_i(\mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}))$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.

(ii) if $\mathcal{A}$ does not allocate the resources to the user i with $\hat{\theta}_i$ as her declared request, then $v_i(\mathcal{A}(\hat{\theta}'_i, \hat{\boldsymbol{\theta}}_{-i})) \geq v_i(\mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}))$, where $\hat{\theta}_i \succeq \hat{\theta}'_i$.

The allocation function $\mathcal{A}$ is bitonic with respect to $v_i()$. This requires that the welfare does not increase with $v_i()$ when user i loses ($\hat{b}_i < v_i^c$), and it does increase with $v_i()$ when user i wins ($\hat{b}_i > v_i^c$). In the next sections, we will design an optimal and a PTAS mechanism that solve the VMPAC problem. These mechanisms work as follows. They first receives the declared requests (bundles and bids) from each participating user and then based on the received requests determine the allocation, using their specific allocation function $\mathcal{A}$, and the payments, using their specific payment function $\mathcal{P}$.

2.3 Strategy-proof Optimal Mechanism for VM Provisioning and Allocation

In this section, we propose a Vickrey-Clarke-Groves (VCG)-based optimal mechanism that solves VMPAC. A VCG mechanism [35, 51, 172] is defined as follows:

Definition 5 (VCG mechanism)**.** *A mechanism is a Vickrey-Clarke-Groves (VCG) mechanism if*

(i) $\mathcal{A}$ is an optimal allocation function, and

(ii) $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = \sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}_{-i})) - \sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}))$,

where $\sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}_{-i}))$ is the optimal social welfare that would have been obtained had user i not participated, and $\sum_{j \in \mathcal{U} \setminus \{i\}} v_j(\mathcal{A}_j(\hat{\boldsymbol{\theta}}))$ is the sum of all users valuations except user i's.

Algorithm 1 VCG-VMPAC Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{U}$ **do**
3: Collect request $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: $(V^*, \mathbf{x}^*)$ = DP-VMPAC$(\hat{\boldsymbol{\theta}}, \mathbf{C})$
6: Provisions and allocates VM instances according to $\mathbf{x}^*$.
7: {Payment.}
8: $\mathcal{P}$ =VCG-PAY$(\hat{\boldsymbol{\theta}}, \mathbf{C}, V^*, \mathbf{x}^*)$

We define our proposed VCG-based mechanism that solves the VMPAC problem as follows:

Definition 6. *The VCG-VMPAC mechanism consists of the allocation algorithm DP-VMPAC and the payment function VCG-PAY defined by the VCG payment function (given in Definition 5 (ii)).*

Our proposed VCG-VMPAC mechanism is given in Algorithm 1. The mechanism is run periodically by the cloud provider. It collects the requests from the users, and it determines the allocation by calling the DP-VMPAC allocation algorithm. Once the allocation is determined the mechanism provisions the required number and types of VM instances and determines the payments by calling the VCG-PAY function. The users are then charged the amount determined by the mechanism.

In order to design a VCG-based mechanism for VMPAC, we need to design an algorithm that provides the optimal solution to VMPAC. The algorithm, called DP-VMPAC, is based on a dynamic programming approach, and it is given in Algorithm 2. The DP-VMPAC algorithm has two input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$) and the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$. The algorithm has two output parameters: V^*, the optimal social welfare and $\mathbf{x}^*$, the optimal allocation of VM instances to the users.

DP-VMPAC starts by determining $\hat{a}_{ir}$, the amount of each resource of type r requested by user i (lines 3-6). We denote by $\mathbf{A}_i$ the vector specifying the amount of all resource types requested by user i. We also denote by $V(j, \hat{\mathbf{C}})$ the optimal welfare for the subproblem that considers the first j users and the available capacity $\hat{\mathbf{C}}$. The algorithm calculates $V(1, \mathbf{C})$

Algorithm 2 DP-VMPAC: Optimal Allocation Algorithm

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
3: **for all** $i \in \mathcal{U}$ **do**
4: **for all** $r \in \mathcal{R}$ **do**
5: $\hat{a}_{ir} = \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}$, where $\hat{k}_{im} \in \hat{S}_i$
6: $\mathbf{A}_i = (\hat{a}_{i1}, \ldots, \hat{a}_{iR})$
7: $\hat{\mathbf{C}} = \mathbf{C}$
8: **if** $\hat{a}_{1r} \leq C_r, \forall r \in \mathcal{R}$ **then**
9: $V(1, \mathbf{C}) = \hat{b}_1$
10: $\hat{\mathbf{C}} = \mathbf{C} - \mathbf{A}_1$
11: **else**
12: $V(1, \mathbf{C}) = 0$
13: **for all** $j = 2, \ldots, N$ **do**
14: $V(j, \hat{\mathbf{C}}) = \max\{V(j-1, \hat{\mathbf{C}}), V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j\}$
15: $V^* = V(N, \mathbf{C})$
16: Find $\mathbf{x}^*$ by looking backward at $V(j, \hat{\mathbf{C}})$
17: **Output:** $V^*, \mathbf{x}^*$

(lines 8-12). Based on these values, it calculates $V(j, \hat{\mathbf{C}})$, where $j = 2, \ldots, N$ (lines 13-14) according to the following dynamic programming recurrence:

$$V(j, \hat{\mathbf{C}}) = \max\{V(j-1, \hat{\mathbf{C}}), V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j\} \tag{2.3}$$

The recurrence considers two cases, not allocating the bundle to j and allocating it to j. If allocating the requested bundle of the jth user increases the value $V(j-1, \hat{\mathbf{C}})$, the algorithm allocates the bundle to the jth user. The maximum between $V(j-1, \hat{\mathbf{C}})$ and $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$ gives the optimal value of $V(j, \hat{\mathbf{C}})$. Once the final value $V(N, \mathbf{C})$ is determined, the algorithm finds $\mathbf{x}^*$, the optimal allocation of VM instances, by looking backward at $V(j, \hat{\mathbf{C}})$, as follows. If $V(N, \mathbf{C}) = V(N-1, \mathbf{C})$, then it means that we did not select the N-th user (i.e., $x_N = 0$), and thus, we just recursively work backwards from $V(N-1, \mathbf{C})$. Otherwise, we select that user (i.e., $x_N = 1$), output the N-th request, and recursively work backwards from $V(N-1, \mathbf{C} - \mathbf{A}_N)$.

Theorem 1. *The DP-VMPAC algorithm finds the optimal solution to the VMPAC problem.*

Proof. To prove that $V(N, \mathbf{C})$ computed by DP-VMPAC is optimal, we need to show that

the subproblem $V(j, \hat{\mathbf{C}})$ is optimal for every j and $\hat{\mathbf{C}}$, where j is the number of users ($j \leq N$), and $\hat{\mathbf{C}}$ is the vector representing the available capacities of the resources. We consider two cases, according to the dynamic programming recurrence given in equation (2.3). In case one, we assume that not allocating the bundle to j is in the optimal solution for the subproblem $V(j, \hat{\mathbf{C}})$. Then, according to equation (2.3), $V^*(j, \hat{\mathbf{C}}) = V(j-1, \hat{\mathbf{C}})$. The proof in this case is by contradiction. We assume that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution for the subproblem. As a result, we need to check if $V(j-1, \hat{\mathbf{C}})$ is optimal. If $V(j-1, \hat{\mathbf{C}})$ is not optimal, then there would be a better solution $V'(j-1, \hat{\mathbf{C}}) > V(j-1, \hat{\mathbf{C}})$. Then, using the subproblem $V'(j-1, \hat{\mathbf{C}})$, we have: $V^*(j, \hat{\mathbf{C}}) < V'(j-1, \hat{\mathbf{C}})$, which contradicts the fact that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution. Therefore, $V(j-1, \hat{\mathbf{C}})$ is optimal.

In case two, we consider that allocating the bundle to j is in the optimal solution for the subproblem $V(j, \hat{\mathbf{C}})$. Then, according to equation (2.3), $V^*(j, \hat{\mathbf{C}}) = V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$. The proof in this case is by contradiction. We assume that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution for the subproblem. As a result, we need to check if $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is optimal. If $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is not optimal, then there would be a better solution $V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) > V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$. Then, using the subproblem $V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$, and the rest of the subproblems, $V^*(j, \hat{\mathbf{C}}) < V'(j-1, \hat{\mathbf{C}} - \mathbf{A}_j) + \hat{b}_j$, which contradicts the fact that $V^*(j, \hat{\mathbf{C}})$ is the optimal solution. Therefore, $V(j-1, \hat{\mathbf{C}} - \mathbf{A}_j)$ is optimal.

We conclude that $V^*(j, \hat{\mathbf{C}}) = V(j, \hat{\mathbf{C}})$, and that this property is maintained at all times thereafter. □

DP-VMPAC solves VMPAC optimally in time $O(N(C_{max})^R)$, where $C_{max} = \max_{r \in \mathcal{R}}\{C_r\}$. This is due to the fact that the dynamic programming builds a $(R+1)$-dimensional table, where the first dimension corresponds to the number of users and the other R dimensions correspond to the R types of resources.

The VCG-PAY function is given in Algorithm 3. VCG-PAY has four input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$), the vector of resource capacities $\mathbf{C}$, the optimal welfare V^*, and the optimal allocation given by $\mathbf{x}^*$. It has one output parameter: $\mathcal{P}$, the payment vector for the users. VCG-PAY calls DP-VMPAC to find the allocation and welfare obtained without user i's participation (line 6). Based on the optimal allocation

Algorithm 3 VCG-PAY: Payment Function

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C}$; vector of resource capacities
3: **Input:** V^*; optimal welfare
4: **Input:** $\mathbf{x}^*$; optimal allocation
5: **for all** $i \in \mathcal{U}$ **do**
6: $(V'^*, \mathbf{x}'^*) = \text{DP-VMPAC}(\hat{\boldsymbol{\theta}}_{-i}, \mathbf{C})$
7: $sum_1 = 0$
8: $sum_2 = 0$
9: **for all** $j \in \mathcal{U}, j \neq i$ **do**
10: $sum_1 = sum_1 + \hat{b}_j x'^*_j$
11: $sum_2 = sum_2 + \hat{b}_j x^*_j$
12: $\mathcal{P}_i = sum_1 - sum_2$
13: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

to the users with and without user i's participation, VCG-PAY finds the payment for user i, where sum_1 is the sum of all values without user i's participation in the mechanism, and sum_2 is the sum of all except user i's value in the optimal case (lines 7-12).

The VCG-VMPAC mechanism is strategy-proof, and it determines the optimal allocation. However, its execution time becomes prohibitive for large instances of VMPAC. More than this, the problem is strongly NP-hard (by a trivial reduction from the multidimensional knapsack problem [71]), and there is no Fully Polynomial Time Approximation Scheme (FPTAS) for solving it, unless $P = NP$. PTAS is by far the strongest approximation result that can be achieved for this problem, unless $P = NP$. In the next section, we design such a PTAS mechanism for VMPAC.

2.4 Strategy-proof PTAS Mechanism for VM Provisioning and Allocation

We now introduce our proposed strategy-proof PTAS mechanism, PTAS-VMPAC. The definition of a PTAS is as follows:

Definition 7. *A maximization problem has a PTAS if for every instance I and for every $\epsilon > 0$, it finds a solution S for I in time polynomial in the size of I that satisfies $S(I) \geq (1 - \epsilon)S^*(I)$, where $S^*(I)$ is the optimal value of a solution for I.*

We define the PTAS-VMPAC mechanism that solves the VMPAC problem as follows:

Algorithm 4 PTAS-VMPAC Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{U}$ **do**
3: Collect request $\hat{\theta}_i = (\hat{S}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: $(V, \mathbf{x})$ = PTAS-ALLOC$(\hat{\boldsymbol{\theta}}, \mathbf{C}, q)$
6: Provisions and allocates VM instances according to $\mathbf{x}$.
7: {Payment.}
8: $\mathcal{P}$ =C-PAY$(\hat{\boldsymbol{\theta}}, \mathbf{C}, q)$

Definition 8. *The PTAS-VMPAC mechanism consists of the allocation algorithm PTAS-ALLOC and the payment function C-PAY.*

The PTAS-VMPAC mechanism is given in Algorithm 4. The mechanism is run periodically by the cloud provider. It collects the requests from the users and determines the allocation by calling the PTAS-ALLOC algorithm. Once the allocation is determined the mechanism provisions the required number and types of VM instances, and then it determines the payments by calling the C-PAY function. The users are then charged the payment determined by the mechanism. PTAS-ALLOC and C-PAY are presented in the following subsections.

Our proposed mechanisms, VCG-VMPAC and PTAS-VMPAC, provision the VMs based on the requests of the users. The mechanisms can handle dynamic changes of heterogeneous user demands by supporting dynamic provisioning of cloud resources. As a result, cloud providers can employ our proposed mechanisms to fulfill dynamic market demands efficiently. For example, if a user releases a large VM, then the cloud provider can provision the released resources in the form of smaller VMs if there is a request for them.

2.4.1 PTAS-ALLOC: Allocation Algorithm of PTAS-VMPAC

Our proposed PTAS allocation algorithm, called PTAS-ALLOC, is given in Algorithm 5. PTAS-ALLOC has three input parameters: the vector of users declared requests $\hat{\boldsymbol{\theta}}$, the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$, and an integer q, where $q \leq N$. The parameter q controls how close the solution determined by PTAS-ALLOC is to the optimal solution. The PTAS-ALLOC algorithm has two output parameters: V, the total social

Algorithm 5 PTAS-ALLOC: Allocation algorithm for VMPAC

Input: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
Input: $\mathbf{C} = (C_1, \ldots, C_R)$; vector of resource capacities
Input: q;
$V = -\infty$
for all $\hat{\mathcal{U}} \subseteq \mathcal{U} : |\hat{\mathcal{U}}| \leq q$ **do**
  $\hat{\mathbf{x}} = \mathbf{0}$
  $\hat{V} = 0$
  $sum_r = 0, \forall r \in \mathcal{R}$
  for all $i \in \hat{\mathcal{U}}$ **do**
    $\hat{x}_i = 1$
    $\hat{V} = \hat{V} + \hat{b}_i$
    for all $r \in \mathcal{R}$ **do**
      $sum_r = sum_r + \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr} \hat{x}_i$
  if $C_r \geq sum_r, \forall r \in \mathcal{R}$ **then**
    $\tilde{\mathcal{U}} = \mathcal{U} \setminus \hat{\mathcal{U}}$
    $\hat{q} = |\hat{\mathcal{U}}|$
    for all $r \in \mathcal{R}$ **do**
      $d_r = C_r - \sum_{i \in \mathcal{U}} \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr} \hat{x}_i$
    $\mathbf{d} = (d_1, \ldots, d_R)$
    for all $i \in \tilde{\mathcal{U}}$ **do**
      for all $r \in \mathcal{R}$ **do**
        $\hat{a}_{ir} = \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}$
        $\tilde{a}_{ir} = \lceil \hat{a}_{ir} N^2 / d_r \rceil d_r / N^2$
      $\tilde{\mathbf{A}}_i = (\tilde{a}_{i1}, \ldots, \tilde{a}_{iR})$
    {DP to find $(\tilde{V}, \tilde{\mathbf{x}})$ for $(\tilde{\mathcal{U}}, \mathbf{d})$:}
    $\hat{\mathbf{d}} = \mathbf{d}$
    if $d_r \geq \tilde{a}_{1r}, \forall r \in \mathcal{R}$ **then**
      $V(1, \mathbf{d}) = \hat{b}_1$
      $\hat{\mathbf{d}} = \mathbf{d} - \tilde{\mathbf{A}}_1$
    else
      $V(1, \mathbf{d}) = 0$
    for all $j = 2, \ldots, N - \hat{q}$ **do**
      $V(j, \hat{\mathbf{d}}) = \max\{V(j-1, \hat{\mathbf{d}}), V(j-1, \hat{\mathbf{d}} - \tilde{\mathbf{A}}_j) + \hat{b}_j\}$
    $\tilde{V} = V(N - \hat{q}, \mathbf{d})$
    Find $\tilde{\mathbf{x}}$ by looking backward at $V(j, \mathbf{d})$
    if $V < (\hat{V} + \tilde{V})$ **then**
      $V = \hat{V} + \tilde{V}$
      $\mathbf{x} = \hat{\mathbf{x}} + \tilde{\mathbf{x}}$
Output: V, $\mathbf{x}$

welfare and $\mathbf{x}$, the allocation of VM instances to the users. Our approximation technique is inspired from Briest *et al.* [20] who proposed a strategy-proof approximation algorithm

for the generalized assignment problem where all bins have the same capacity.

The main idea in the design of PTAS-ALLOC, is finding a partial allocation first and then allocating through dynamic programming the remaining resources based on the rounded requests of the unallocated users. The partial allocation of fewer users is used as as a "seed" for the approximate solution and is also allowing us to control the solution error, ϵ. The more users we consider in the partial allocation (greater q), the better the error. In the most extreme case, when q is equal to the total number of users, the algorithm degenerates into an exhaustive search, producing the optimal allocation but making the algorithm computationally infeasible. In addition, as a consequence of dividing the allocation process into a partial allocation and an allocation of rounded requests of remaining users, the resulting overall allocation in each iteration of PTAS-ALLOC is bitonic and monotone. The algorithm chooses the maximum among the allocations obtained in each iteration, and thus, as we will show in Theorem 2, the allocation produced by PTAS-ALLOC is monotone.

The PTAS-ALLOC algorithm iterates over all subsets $\hat{\mathcal{U}}$ of at most q users (lines 5-38). For each such subset the algorithm finds a feasible partial allocation $\hat{\mathbf{x}}$ of at most q users (lines 5-14), determines the amount of partially allocated resources for each of the r types of resources (lines 17-19) and rounds the amount of requested resources by the unallocated users (set $\tilde{\mathcal{U}}$) for each of the r resources (lines 20-24). Then, it uses a dynamic programming approach to find an allocation of bundles based on the rounded requests $\tilde{a}_{ir}$, and the remaining unallocated capacities, d_r (lines 25-33). The algorithm determines the maximum welfare and the corresponding VM instance allocation $\mathbf{x}$ obtained over all iterations (lines 34-38).

We now describe the dynamic programming approach that finds the optimal allocation for the remaining users of the remaining capacities using the rounded requests of users (lines 25-33). In order to formulate the problem as a dynamic program, we consider the subproblem $V(j, \hat{\mathbf{d}})$ which includes the first j remaining users with the available capacity $\hat{\mathbf{d}}$ such that $V(j, \hat{\mathbf{d}})$ is the optimal value of the subproblem. The algorithm first calculates $V(1, \mathbf{d})$ (lines 26-31). Based on these values, it calculates $V(j, \hat{\mathbf{d}})$, where $j = 2, \ldots, N - \hat{q}$ (lines 32-33). The algorithm compares two cases, not allocating the bundle to j and allocating it

to j. If allocating the requested bundle of the jth user increases the value $V(j-1,\hat{\mathbf{d}})$, the algorithm allocates the bundle to the jth user. The maximum between $V(j-1,\hat{\mathbf{d}})$ and $V(j-1,\hat{\mathbf{d}}-\tilde{\mathbf{A}}_j)+\hat{b}_j$ gives the optimal value of $V(j,\hat{\mathbf{d}})$, where $\tilde{\mathbf{A}}_j$ is the vector of the rounded sizes of requested resources by user j. We can formulate the dynamic programming recursion as follows:

$$V(j,\hat{\mathbf{d}}) = \max\{V(j-1,\hat{\mathbf{d}}), V(j-1,\hat{\mathbf{d}}-\tilde{\mathbf{A}}_j)+\hat{b}_j\} \tag{2.4}$$

This dynamic programming formulation is the same as the one proposed for DP-VMPAC algorithm except that here the available capacity vector $\hat{\mathbf{C}}$ is replaced by the vector of remaining unallocated capacities $\hat{\mathbf{d}}$, and the vector specifying the amount of all resource types requested by user j, $\mathbf{A}_j$, is replaced by $\tilde{\mathbf{A}}_j$, the vector of the rounded sizes of requested resources by user j. Therefore, from Theorem 1, the dynamic programming approach finds the optimal solution for the allocation of the unallocated resources to the remaining users with rounded requests (corresponding to lines 25-33 of Algorithm 5).

The dynamic programming builds a table of size $(N-\hat{q})$ rows and N^2 columns, where $(N-\hat{q})$ is the number of users and N^2 is the number of possible different sizes for the resource capacities due to rounding of the sizes. As a result, the time complexity of the dynamic programming is $O(N(N^2)^R)$, where R is the number of resource types. The algorithm stores $V(N-\hat{q},\mathbf{d})$ to $\tilde{V}$ as the optimal welfare obtained by the dynamic programming for the selected $\tilde{\mathcal{U}}$, and the corresponding allocation to $\tilde{\mathbf{x}}$. Then, PTAS-ALLOC finds the maximum total social welfare, V across all iterations on the subsets of at most q users. It also finds the allocation $\mathbf{x}$ by $\hat{\mathbf{x}}+\tilde{\mathbf{x}}$ (lines 35-38).

Theorem 2. *PTAS-ALLOC is monotone.*

Proof. To prove that the PTAS-ALLOC is monotone, we need to show that each iteration of the main for loop provides a monotone and bitonic allocation. This is based on a result from [119] that states that if an algorithm A consists of applying the maximum operator among a set of allocation algorithms that are monotone and bitonic, then algorithm A is monotone. In our case, the allocation algorithms are basically the iterations of the main

for-loop in PTAS-ALLOC.

We show that one iteration is producing a monotone allocation by considering two cases. First, we consider a user i with declared request $\hat{\theta}_i$ is allocated her requested bundle, and she is in the first q users selected by the algorithm. If user i declares a request $\hat{\theta}'_i \succeq \hat{\theta}_i$ (a smaller bundle or higher bid), the allocation will not change. This satisfies the definition of the monotonicity property, where the winning user is among the first q users. Second, we consider that a user i with declared request $\hat{\theta}_i$ is allocated her requested bundle, and she is not in the first q users. In this case, if user i declares a request $\hat{\theta}'_i \succeq \hat{\theta}_i$, her allocation by dynamic programming will not change. This is due to the fact that she declares a more profitable request. As a result, user i remains among winning users which satisfies the monotonicity property, where the winning user is not among the first q users. This proves the monotonicity of each iteration.

To prove that PTAS-ALLOC is bitonic in each iteration, we consider two cases. First, user i is not among the first q users. If user i is a winning user, then by declaring a better request (a smaller bundle or higher bid), the social welfare can only be increased. If user i is not a winning user, then by declaring a larger bundle or less bid, the social welfare can not be increased. Second, user i is among the first q users. Thus, she is a winning user. If she declares a higher bid, the social welfare will increase. If she declares a smaller bundle, then the remaining capacities of each resource will increase. As a result, the social welfare can only increase. Thus, each iteration is bitonic.

The monotonicity and bitonicity properties of PTAS-ALLOC in each iteration, combined with the fact that the PTAS-ALLOC keeps the allocation that gives the maximum welfare among these iterations, proves the overall monotonicity of PTAS-ALLOC. □

We now show that our proposed allocation algorithm is a PTAS, that is, for every fixed ϵ, its running time is polynomial in the size of the input.

Theorem 3. *The PTAS-ALLOC algorithm is a PTAS.*

Proof. To prove that the algorithm is PTAS, we need to show that the solution determined by the algorithm is in a $(1-\epsilon)$ neighborhood of the optimal, and that the time complexity

of the algorithm is polynomial in N.

We first show that the solution is within $(1-\epsilon)$ of the optimal solution. Let $\mathbf{x}^*$ be the optimal allocation of the requested bundles, and V^* be the corresponding optimal value. Assume that PTAS-ALLOC determines an allocation $\mathbf{x}$ and a value V. Let $\hat{\mathbf{x}}$ be the optimal allocation when we consider only q users with the highest declared values in the first step. The second step of allocation is allocating the remaining resources given by $\mathbf{d}$ to the users who were not selected in the first step. The rounding procedure for the remaining users, in the second step, increases the size of the requested bundles of those users for each resource type. This may lead to an infeasible allocation of the bundles based on the new rounded sizes. Based on the rounding, the total increase in the size of the requested bundles for each resource is less than d_r/N. In order to make the allocation feasible, we can remove a requested bundle such that it satisfies the capacity constraints for each resource type while decreasing the least amount of value from the objective function. We find those allocated bundles in the second step where for all resource types their size is larger than d_r/N. Among those, we choose the bundle $\hat{S}_i$ with the smallest size. Since in the first step, we chose the q bundles with the highest values, the bundle $\hat{S}_i$ can be the $q+1$ most valuable bundle. Therefore, user i valuation for this bundle is $v_i(\hat{S}_i) \leq 1/(q+1)V^*$. Removing bundle $\hat{S}_i$ makes the obtained objective function between $(1-1/(q+1))V^*$ and V^*. Therefore, we have $(1-\epsilon)V^* \leq V \leq V^*$, where $\epsilon = 1/(q+1)$.

We now show that the time complexity of PTAS-ALLOC is polynomial in N. The running time depends on the partial allocation of q users and the dynamic programming. The time complexity of the dynamic programming is $O(N(N^2)^R)$, where N is the number of users and N^2 is the size of each resource based on the rounding. The exhaustive search to find a partial allocation is based on the total number of allocations of q users which is $\sum_{i=1}^{q} R\binom{N}{i} \leq qRN^q$. Thus, the time complexity of the algorithm is $O(qRN^{2R+q+1})$. This proves that the algorithm is PTAS. □

Algorithm 6 C-PAY: Critical Payment Function

1: **Input:** $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{C}$; vector of resource capacities
3: **Input:** q;
4: **Input:** $\mathbf{x}$; winning users
5: **for all** $i \in \mathcal{U}$ **do**
6: $\quad \mathcal{P}_i = 0$
7: $\quad$ **if** x_i **then**
8: $\quad\quad l = 0$
9: $\quad\quad h = \hat{b}_i$
10: $\quad\quad$ **while** $(h - l) \geq 1$ **do**
11: $\quad\quad\quad v_i^c = (h + l)/2$
12: $\quad\quad\quad \hat{\theta}_i^c = (\hat{S}_i, v_i^c)$
13: $\quad\quad\quad (V', \mathbf{x}') =$ PTAS-ALLOC $((\hat{\theta}_1, \ldots, \hat{\theta}_i^c, \ldots, \hat{\theta}_N), \mathbf{C}, q)$
14: $\quad\quad\quad$ **if** x_i' **then**
15: $\quad\quad\quad\quad$ {user i is winning by declaring v_i^c}
16: $\quad\quad\quad\quad h = v_i^c$
17: $\quad\quad\quad$ **else**
18: $\quad\quad\quad\quad l = v_i^c$
19: $\quad\quad \mathcal{P}_i = h$
20: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

2.4.2 C-PAY: Payment Algorithm of PTAS-VMPAC

The C-PAY function is given in Algorithm 6. The C-PAY function has four input parameters, the vector of users declared requests ($\hat{\boldsymbol{\theta}}$), the vector of resource capacities $\mathbf{C}$, the obtained allocation given by $\mathbf{x}$, and the integer q. It has one output parameter: $\mathcal{P}$, the payment vector for the users. The payments are based on the critical payments of the winning users. The payment of winning user i is v_i^c, where v_i^c is the critical payment of user i, if i wins and zero if i loses. Finding the critical payment is done by a binary search over values less than the declared value.

Theorem 4. *The payment algorithm, C-PAY, implements the critical payment.*

Proof. To prove that C-PAY determines the critical payment for the users, we need to show that $\mathcal{P}_i$ is the critical payment for user i, i.e., $\mathcal{P}_i$ is the minimum value that she can declare to obtain her requested bundle of VMs. The C-PAY algorithm finds the critical payment when the difference between upper and lower bound is less than 1 (line 10 of

C-PAY algorithm). In addition, the algorithm always sets the winning value as an upper bound h, and a losing value as a lower bound, l (line 16 and 18, respectively). We separate the proof into two parts depending on user i's declared value $\hat{b}'_i$, as follows:

i) User i declares a value $\hat{b}'_i$ greater than $\mathcal{P}_i$, (i.e., $\hat{b}'_i > \mathcal{P}_i$), and the algorithm finds a critical payment of $v_i^{c'}$. To prove that $\mathcal{P}_i$ is the critical payment for user i, we need to show that for every other critical payments (e.g., $v_i^{c'}$), the difference between $v_i^{c'}$ and $\mathcal{P}_i$ is at most one. We claim that $|v_i^{c'} - \mathcal{P}_i| \leq 1$. The proof is by contradiction, i.e., we assume $|v_i^{c'} - \mathcal{P}_i| > 1$. Therefore, there are two cases, either $v_i^{c'} - \mathcal{P}_i > 1$, or $\mathcal{P}_i - v_i^{c'} > 1$. In the first case, user i wins by declaring both $v_i^{c'}$ and $\mathcal{P}_i$, where $v_i^{c'} > \mathcal{P}_i$. Based on line 16 and line 19 of the C-PAY algorithm $v_i^{c'}$ is an upper bound on the payment, and thus, there exists a value l such that $v_i^{c'} - l < 1$ due to the convergence and termination of the binary search. Then, we have l as the lower bound and user i cannot win her bundle by declaring l based on line 18. As a result, $\mathcal{P}_i > l$, and thus $v_i^{c'} - \mathcal{P}_i < 1$, which contradicts the assumption. In the second case, user i wins by declaring both $v_i^{c'}$ and $\mathcal{P}_i$, where $\mathcal{P}_i > v_i^{c'}$. With the same argument, we have $\mathcal{P}_i$ as an upper bound for the payment, and thus, $\mathcal{P}_i - l < 1$ to terminate the binary search. Then, we have l as the lower bound and user i cannot win her bundle by declaring l. As a result, $v_i^{c'} > l$, and thus, $\mathcal{P}_i - v_i^{c'} < 1$, which contradicts the assumption.

ii) User i declares a value $\hat{b}'_i$ less than $\mathcal{P}_i$, (i.e., $\mathcal{P}_i - \hat{b}'_i > 1$). Since the algorithm converges when $\mathcal{P}_i - l < 1$, there exists l such that user i cannot win her bundle of VMs by declaring l. As a result, by declaring $\hat{b}'_i$, user i is not a winning user, and her payment is zero, thus, satisfying the properties of the critical payment.

These show that the payment $\mathcal{P}_i$ is the minimum valuation that user i must bid to obtain her requested bundle. As a result, the payment determined by C-PAY is the critical payment. □

We now show that the proposed mechanism is strategy-proof.

Theorem 5. *The PTAS-VMPAC mechanism is strategy-proof.*

Proof. The allocation algorithm PTAS-ALLOC is monotone (Theorem 2) and the pay-

Table 2.3: Users' true requests

User	1	2	3	4	5	6	7	8
k_{i1}	0	1	2	0	2	1	2	3
k_{i2}	0	2	0	0	0	0	0	2
k_{i3}	2	0	2	2	0	0	0	1
k_{i4}	0	0	3	1	2	3	1	1
b_i	30	18	95	10	5	15	7	80

Table 2.4: Different scenarios for user 8's request declaration

Case	$\hat{S}_8$	$\hat{b}_8$	Scenario	Status
I	$< 3, 2, 1, 1 >$	\$80	$\hat{b}_8 = b_8$, $\hat{S}_8 = S_8$	W
II	$< 3, 2, 1, 1 >$	\$90	$\hat{b}_8 > b_8$, $\hat{S}_8 = S_8$	W
III	$< 3, 2, 1, 1 >$	\$70	$\hat{b}_8 < b_8$, $\hat{S}_8 = S_8$	W
IV	$< 3, 2, 1, 1 >$	\$9	$\hat{b}_8 < b_8$, $\hat{S}_8 = S_8$	L
V	$< 3, 2, 1, 3 >$	\$80	$\hat{b}_8 = b_8$, $\hat{S}_8 > S_8$	W
VI	$< 3, 2, 1, 5 >$	\$80	$\hat{b}_8 = b_8$, $\hat{S}_8 > S_8$	L

ment function C-PAY determines the critical payment (Theorem 4). Therefore, according to [119], the PTAS-VMPAC mechanism is strategy-proof. □

2.4.3 Example

We now analyze the effect of untruthful reporting on the utility of the users participating in the PTAS-VMPAC mechanism by considering an example. Our goal is to show that our proposed mechanism, PTAS-VMPAC, is robust against manipulation by a user. The true requests of the eight users are shown in Table 2.3. We consider the capacities of the two resources as follows: 100 cores, and 1800 MB of storage. The PTAS-VMPAC ($\epsilon = 0.33$) allocates resources to user 1, 2, 3, 7 and 8 in the case where all users declare their true requests. The payments of the winning users based on C-PAY are 3, 3, 18, 0, and 10, respectively.

We assume that user 8 reports a different request, $\hat{\theta}_8$, from her true request $\theta_8 = (< 3, 2, 1, 1 >, \$80)$, where $S_8 =< 3, 2, 1, 1 >$ and $b_8 = 80$. As shown in Table 2.4, we analyze different scenarios, where user 8 submits different requests. In addition, Fig. 2.1 shows the

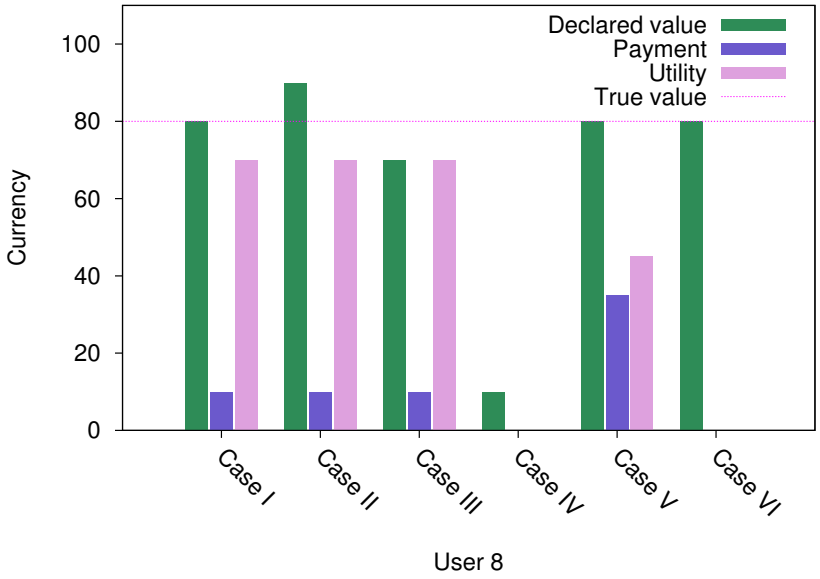


Figure 2.1: PTAS-VMPAC: Effect of untruthful declarations.

payment and utility of the user for all the cases. In Case I, user 8 submits her true request, that is, $\theta_8 = \hat{\theta}_8$. In this case, user 8 wins, and receives the requested bundle of VMs, S_8. The mechanism charges her \$10 for the bundle, and her utility is 80-10=70. In case II, user 8 submits a request with a higher bid $\hat{b}_8 = 90$. In this case, user 8 is still a winner and the mechanism determines the same payment for her as in case I, leading to a utility of 70. In case III, she submits a request with a lower bid $\hat{b}_8 = 70$, which is not less than the price determined by our mechanism (i.e., \$10). Thus, user 8 is still winning, and the mechanism charges her the same amount as in case I. However, if user 8 submits a request with a lower bid below the payment, she will not obtain her requested bundle, leading to zero utility. This is shown in case IV, where user 8 submits a bid $\hat{b}_8 = 9$. We now investigate scenarios in which user 8 requests a different bundle than her true bundle. In case V, she submits a larger bundle $\hat{S}_8 =< 3, 2, 1, 3 >$, where she requests 3 VM instances of type 2XL instead of 1 (the case of her true request, Case I). In this case, she obtains the bundle due to available capacities. However, she pays more than she pays in case I, II, and III. Thus, her utility decreases. In case VI, she submits a larger bundle $\hat{S}_8 =< 3, 2, 1, 5 >$, where she requests 5 VM instances of type 2XL instead of 1 VM instance (the case of her true request, Case I).

However, she becomes a loser since the cloud provider does not have enough resources to fulfill her request. We showed that if a user submits a request untruthfully, she can not increase her utility.

2.5 Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of the proposed mechanisms, PTAS-VMPAC, and the VCG-VMPAC. We also compare our proposed mechanisms with a greedy mechanism proposed in [124], called G-VMPAC-II. Here, we use a simpler name, G-VMPAC, to refer to G-VMPAC-II. G-VMPAC allocates the VM instances to users in decreasing order of their density metric (a metric based on the bid of a user and the scarcity of her requested resources). The auctions are generated using four workload logs from the Grid Workloads Archive [5] and the Parallel Workloads Archive [8]. VCG-VMPAC, PTAS-VMPAC, and G-VMPAC mechanisms are implemented in C++ and the experiments are conducted on Intel 2.93GHz Quad Proc Hexa Core nodes with 90GB RAM which are part of the Wayne State Grid System. In this section, we describe the experimental setup and analyze the experimental results.

2.5.1 Experimental Setup

In the absence of publicly available users requests data from cloud providers, we resort to the well studied and standardized workloads from both the Grid Workloads Archive and the Parallel Workloads Archive.

We selected three logs from the Grid Workloads Archive as follows: 1) NorduGrid traces from the NorduGrid system; 2) AuverGrid traces from the AuverGrid system; 3) SHARCNET traces from SHARCNET clusters installed at several academic institutions in Ontario, Canada. We also selected the following log from the Parallel Workloads Archive. 4) MetaCentrum from the national grid of the Czech republic. We selected these logs because of the availability of CPU and memory requests/usage recorded. We consider each hour of a log as one auction, where each job corresponds to a user request. For each log,

Table 2.5: Statistics of workload logs for the first 100 hours.

Logfile	Number of jobs	Range of CPU	Range of Storage (MB)	Available CPUs	Storage Capacity (MB)
GWA-T-3 NorduGrid	843	1	[1-630]	500	3,000
GWA-T-4 AuverGrid	1,524	1	[2-1,168]	500	10,000
METACENTRUM-2009-2	679	[1-32]	[1-26,755]	500	12,000
GWA-T-10 SHARCNET	2,938	[1-512]	[1-7,812]	10,000	20,000
GWA-T-10 SHARCNET (2)	6,629	[1-150]	[1-8,000]	10,000	20,000

except GWA-T-10 SHARCNET, we select 100 hours, while from GWA-T-10 SHARCNET, we select two 100 hours segments. This selection gives five 100-hour auctions for the experiments, and represents different input configurations such as available capacities and number of users. We present the statistics of the logs for the selected segments in Table 2.5. The number of users (jobs) for each log is given in the second column of Table 2.5. The total number of users (requests) is 12,613.

We consider each hour of a log as one auction to follow the standard practice in Amazon EC2. As a result, each log represents a series of auctions, where users submits their requests over time to a cloud provider. In each auction hour, the participants are the new users and the unserved users whose deadline has not been exceeded.

The following fields from the log files are selected in order to generate user requests: JobID, SubmitTime, RunTime, ReqNProcs, and Used Memory. JobID is the index of the job. SubmitTime and RunTime give the submission time of the job, and the time the job needs to complete its execution, respectively. ReqNProcs and Used Memory give the requested number of processors and the average used memory per processor, respectively.

Each job from the logs corresponds to a user request, where the job's resource usage represents the resources requested by the user. Therefore, each user request includes the requested number of CPUs and the amount of storage. For the bid of each user, we generate a random number between 1 and 10. In addition, we consider a deadline for each user request which is between 3 to 6 times of RunTime of each job. Users leave the auctions after their deadline. A user starts bidding for her requested bundle from the SubmitTime

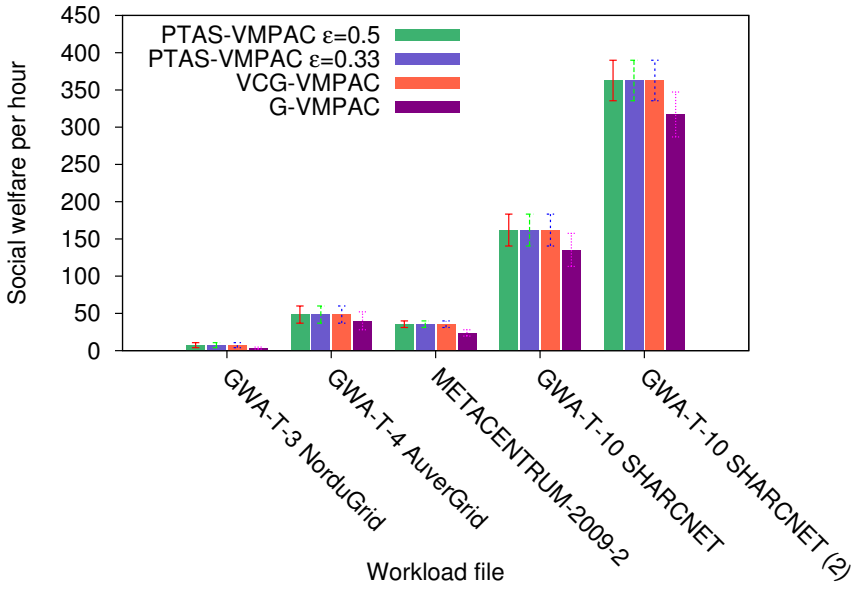


Figure 2.2: PTAS-VMPAC vs. VCG-VMPAC & G-VMPAC: Social welfare.

until her deadline.

2.5.2 Analysis of Results

We compare the performance of PTAS-VMPAC (for $\epsilon = 0.5$ and $\epsilon = 0.33$), G-VMPAC and VCG-VMPAC for different workloads in Figs. 2.2 and 2.4. The selected ϵ values correspond to q equal to 1 and 2. For each workload, we record the execution time, the social welfare, and the percentage of served users per hour for each mechanism.

Fig. 2.2 shows the average social welfare per hour for the logs. The reason that we only choose PTAS-VMPAC with $\epsilon = 0.5$ and $\epsilon = 0.33$, and did not select smaller values for ϵ is that for these cases PTAS-VMPAC obtained optimal results equivalent to the one obtained by the optimal VCG-VMPAC for all logs. Note that PTAS-VMPAC guarantees worst case performance, and it does not necessarily produce non-optimal solutions. The rounding procedure of PTAS makes the size of the requests larger than their actual size. For most cases, the total size of the requests in the optimal solutions is not equal to the available capacities. That means, there is extra remaining capacities even in the optimal allocation. As a result, the rounding of the optimal solution still fits in the available capac-

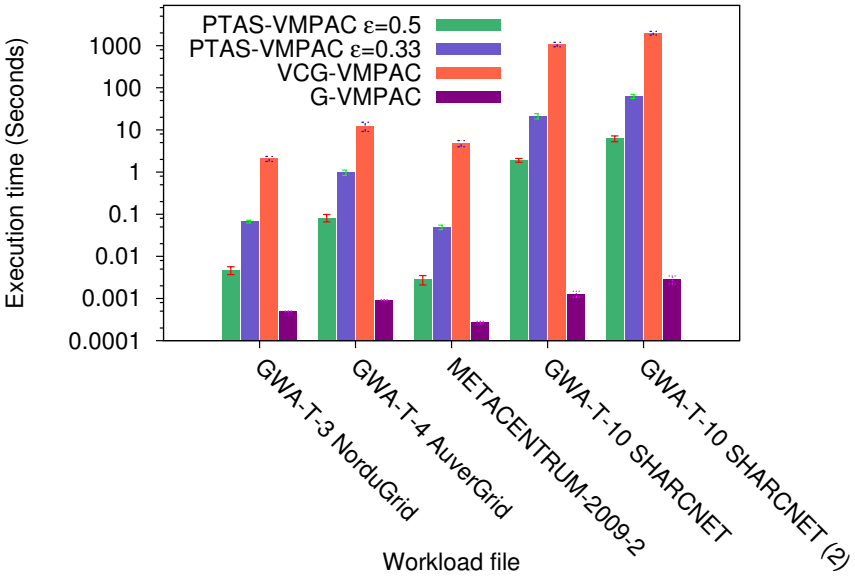


Figure 2.3: PTAS-VMPAC vs. VCG-VMPAC & G-VMPAC: Execution time.

ity. Note that such cases occur irrespective of the amount of aggregated users demands, which can be much higher than the available capacities. The optimality of PTAS-VMPAC depends on the total size of the requests in the optimal solution and the available capacities. Our proposed PTAS-VMPAC mechanism can find the optimal solution in such cases. In fact, there should be a specific configuration in the requested bundles and the available capacities in order to create the worst case scenario (in terms of performance guarantee of ϵ) for the PTAS-VMPAC mechanism. Later in this section, we provide a discussion on the cases where PTAS-VMPAC cannot achieve optimal social welfare. Since G-VMPAC considers only the bid densities when making allocation decisions, it obtains the lowest social welfare in general, which is far from the optimal social welfare. For example, for GWA-T-10 SHARCNET (2) the optimal social welfare is 362.70, while G-VMPAC obtains a social welfare of 317.05 leading to a 12.58% gap from the optimal solution obtained by our proposed PTAS-VMPAC.

Fig. 2.3 shows the average execution time of PTAS-VMPAC ($\epsilon = 0.5$ and $\epsilon = 0.33$), VCG-VMPAC, and G-VMPAC for the logs. In this set of experiments, PTAS-VMPAC with $\epsilon = 0.5$ not only achieves optimal social welfare, but it also has the lowest execution

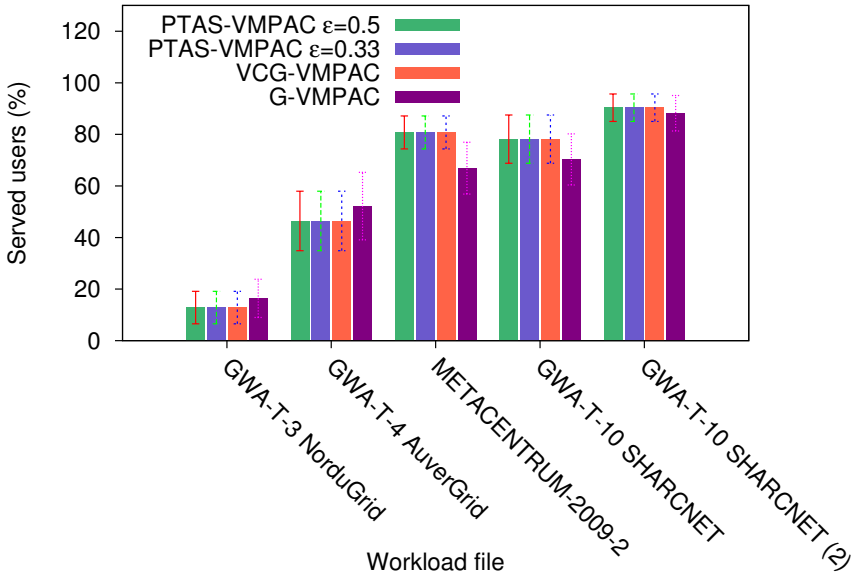


Figure 2.4: PTAS-VMPAC vs. VCG-VMPAC & G-VMPAC: Users served.

time among all the mechanisms obtaining optimal solutions. For example, in the case of METACENTRUM-2009-2, the execution time of PTAS-VMPAC with $\epsilon = 0.5$ is more than three orders of magnitude greater than that of VCG-VMPAC. VCG-VMPAC has the highest execution time in the GWA-T-10 SHARCNET (2) log since it has the highest available capacity and the highest number of requests 6,629. Note that the complexity of VCG-VMPAC depends on the number of requests and the available capacities. The results show that the execution time of the PTAS-VMPAC is polynomial in the number of requests. One key observation is that since PTAS-VMPAC with $\epsilon = 0.5$ can obtain optimal solutions very fast, thus it is beneficial for the cloud providers to use this mechanism rather than PTAS-VMPAC with other values for ϵ. For example, for the total of 2,938 and 6,629 requests, PTAS-VMPAC with $\epsilon = 0.5$ finds the optimal solutions in 1.90 and 6.23 seconds, respectively. G-VMPAC finds the results very fast since it is a greedy mechanism. However, it cannot guarantee a near optimal solution, which is the case for our proposed PTAS-VMPAC mechanism.

Fig. 2.4 shows the percentage of users that have been allocated by the mechanisms. Since the PTAS-VMPAC mechanism achieves the optimal solutions for all logs, it serves

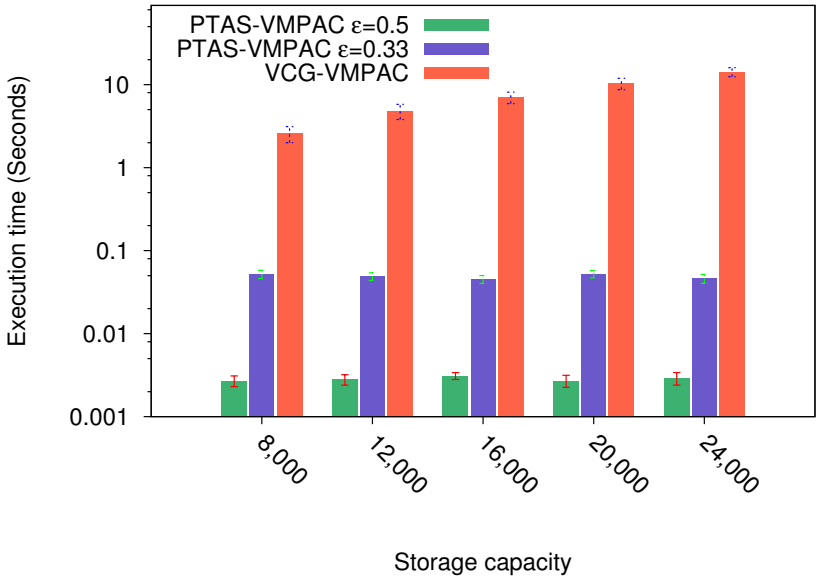


Figure 2.5: PTAS-VMPAC vs. VCG-VMPAC: Sensitivity analysis on execution time

the same percentage of users as VCG-VMPAC. Note that VCG-VMPAC does not serve a higher number of users than G-VMPAC. This is due to the fact that the optimal mechanism finds the most valuable subset of users in order to maximize the social welfare.

To show that the execution time of PTAS-VMPAC does not depend on the values of capacity, we perform sensitivity analysis with respect to storage capacity in Fig. 2.5. For this figure, we select the METACENTRUM-2009-2 log, and choose different storage capacities in each experiment. Fig. 2.5 shows that the execution time of PTAS-VMPAC mechanism does not change by increasing or decreasing the capacity. This is not the case for VCG-VMPAC where its execution time depends on the number of requests and the available capacities. For example, the execution times of VCG-VMPAC for storage capacity of 8,000 and 24,000 are 2.56 and 14.23 seconds, respectively.

We now investigate the cases where PTAS-VMPAC cannot achieve optimal social welfare. This happens in cases where the allocated amount of resources in the optimal solution is the same as the amount of available capacities. In such cases, the rounding procedure of PTAS-VMPAC needs extra amount for each request. Therefore, the optimal solution would not fit in the available capacities. As a result, at least one of the requests would

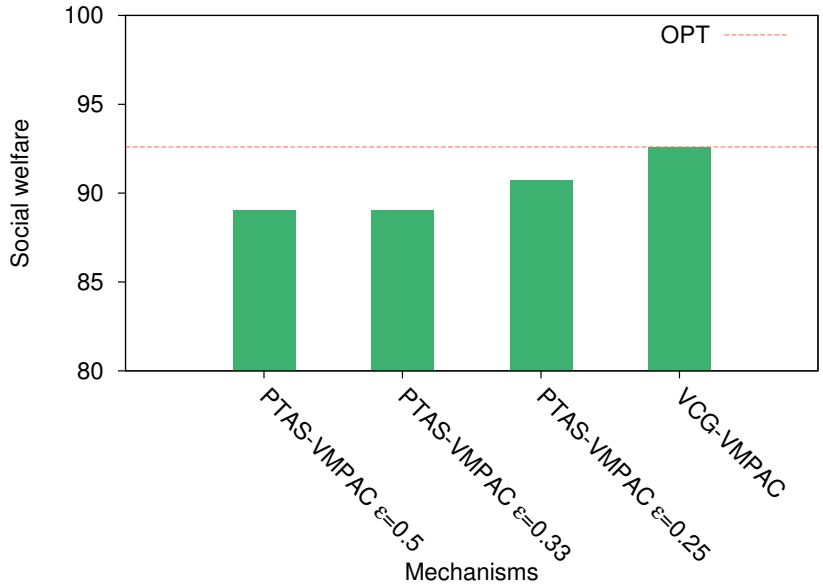


Figure 2.6: PTAS-VMPAC vs. VCG-VMPAC (special case): Social welfare.

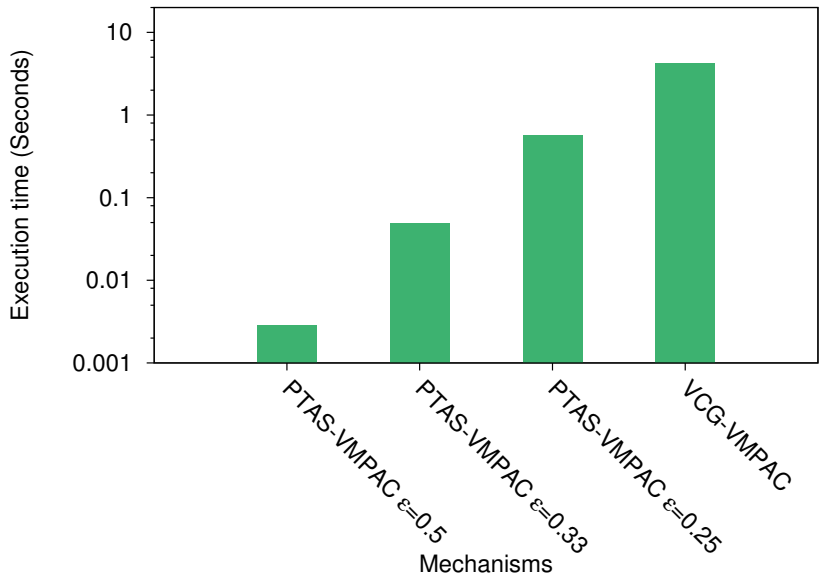


Figure 2.7: TAS-VMPAC vs. VCG-VMPAC (special case): Execution time.

not be fulfilled by PTAS-VMPAC, leading to a lower social welfare. Since in the logs, we do not have such cases, we designed a special experiment to show such a scenario. We select one auction of METACENTRUM-2009-2, and choose the capacities in a way that PTAS-VMPAC mechanisms do not necessarily find the optimal solution. Fig. 2.6 shows

the obtained social welfare based on the selected ϵ, where ϵ is 0.5, 0.33, 0.25 corresponding to q equal to 1, 2, 3, respectively. We also show the social welfare in the optimal case obtained by VCG-VMPAC. The results show that the obtained social welfare is within ϵ distance of the optimal social welfare. Fig. 2.7 shows the execution time of PTAS-VMPAC for the same selected ϵ, and the execution time of VCG-VMPAC. The results show that PTAS-VMPAC is able to find a near optimal social welfare in much shorter time. This is due to the fact the PTAS-VMPAC is a polynomial time approximation scheme.

From all the above results, we conclude that PTAS-VMPAC finds near-optimal solutions to the VMPAC problem and its execution time only depends on the number of users and the selected ϵ. These properties make PTAS-VMPAC a good candidate for deployment on the current cloud computing systems, where the capacities of the resources available for allocation are very large.

2.6 Conclusion

We addressed the problem of dynamic VM provisioning, allocation, and payment determination in clouds considering heterogeneous resources. We proposed a strategy-proof PTAS mechanism for autonomic resource allocation in clouds that provides incentives to the users to reveal their true valuations for the requested bundles of VM instances. We also designed a strategy-proof VCG-based mechanism using a dynamic programming approach. The objectives of the proposed mechanism are to maximize the social welfare in dynamic resource provisioning, to achieve strategy-proofness, and to lead the system into an equilibrium. We investigated the properties of our proposed PTAS mechanism by performing extensive experiments. The results showed that the proposed mechanism determines near optimal allocations while giving the users incentives to report their true valuations for the bundles of VM instances.

CHAPTER 3: PHYSICAL MACHINE RESOURCE MANAGEMENT IN CLOUDS

3.1 Introduction

The ever-growing demand for cloud resources from businesses and individuals places the cloud resource management at the heart of the cloud providers' decision-making process. A cloud provider offers infrastructure as a service (IaaS) by selling low level resources of its physical machines (PMs) in the form of virtual machines (VMs). These services are made available to users as utilities in a pay-as-you-go model, reducing the operational costs for the users. The cloud auction market follows the pay-as-you-go model, and it has proven to be beneficial for both users and cloud providers. This is due to the fact that in such market, cloud providers can attract more customers and better utilize their resources, while users can obtain services at a lower price than in the on-demand market.

We consider the physical machine resource management problem in the presence of multiple PMs and multiple types of resources (e.g., cores, memory, storage) in an auction-based setting, where each user bids for a bundle of heterogeneous VM instances. Bundles of heterogeneous VM instances are required by several types of applications, such as social game applications composed of three layers: front-end web server, load balancing, and back-end data storage. These types of applications require a bundle of heterogeneous VMs composed of communication-intensive VMs, computation-intensive VMs, and storage-intensive VMs, respectively [45]. The requests of the selected users are assigned to PMs, where a PM can be a server, a rack of servers, or a group of racks, and it may host several VMs. Each user has some private information about her requested bundle of VM instances and a private valuation for the bundle. This information is not publicly known by the cloud provider and the other users. The users are self-interested in a sense that they want to maximize their own utility. The cloud auction market could be vulnerable to such self-interested users' behaviors. It may be beneficial for the cloud users to manipulate the auction outcomes and gain unfair advantages by untruthfully revealing their requests (i.e.,

different VM bundles or bids from their actual request). Strategic behaviors of any user may hinder other qualified users, significantly reducing the auction efficiency, and discouraging users from participation. One of the goals in such settings is to design strategy-proof mechanisms, that is, mechanisms that induce the users to report their true requests and valuations.

Our proposed PM resource management mechanisms consist of three phases: winner determination, provisioning and allocation, and pricing. In the winner determination phase, the cloud provider decides which users receive their requested bundles. In the provisioning and allocation phase, the cloud provider provisions the amount of resources in the form of VM instances onto the PMs, and then allocates the requested bundles of VMs to the winning users. In the pricing phase, the cloud provider dynamically determines the price that the winning users should pay for their requests.

The winner determination phase of the PM resource management problem (PMRM) can be reduced to the multiple multidimensional knapsack problem (MMKP). In this setting, each PM is considered to be one multidimensional knapsack. The bundle of VMs from a user request is considered as an item. The aim is to select a subset of items for each knapsack maximizing the value. Chekuri and Khanna [32] showed that the multiple knapsack problem (MKP) is strongly NP-hard (even in the case of two knapsacks) using a reduction from the Partition problem. The MMKP problem is much harder than the MKP problem and is thus also strongly NP-hard. Sophisticated algorithms for solving MMKP do not necessarily satisfy the properties required to achieve strategy-proofness, and they need to be specifically designed to satisfy those properties. On the other hand, another desirable property of such algorithms in cloud auction settings is to have a very small execution time.

A major factor that a cloud provider needs to take into account when offering VM instances to users is pricing the VMs based on the market demand. Such pricing functions should consider the incentives of both cloud providers and users. Amazon reported that most users have saved between 50% and 66% by bidding in its spot market (Amazon auction market) compared to standard on demand market [2]. Dynamic pricing is an efficient way to improve cloud providers revenue [187]. Instead of exclusively selling VMs employing

a fixed-price model, cloud providers such as Amazon EC2 employ auction-based models, where users submit bids for their requested VM instances. The auction-based model allows the cloud providers to sell their VM instances at a price determined according to the real market demand. Note that in an untruthful auction, users may declare bids lower than their actual valuations which may hurt other users and indirectly lead to profit losses for the cloud provider. Thus, unless strategy-proofness is enforced, maximizing the revenue may not be effective. In the pricing phase, the cloud provider dynamically determines the price that users should pay for their requests.

We design an optimal and an approximation mechanism that motivates cloud users to reveal their requests truthfully. Our proposed mechanisms take the strategic behavior of individual users into account and simultaneously maximize the global performance objective of the system. In addition, both mechanisms place VMs in as few PMs as possible. Such approach has been recognized as an efficient way of reducing cost [110]. This is also in alignment with green cloud computing objectives [14], where the cloud provider determines which PMs to power on/off in order to save energy. The mechanisms allow a cloud provider to choose PMs configurations that are aligned with its power consumption policies. In addition, our proposed approximation mechanism iteratively provisions VMs on each PM. This iterative mechanism allows the cloud provider to power on/off PMs based on the user demands.

3.1.1 Our Contribution

We address the problem of cloud resource management in the presence of multiple PMs with multiple types of resources. We design a strategy-proof greedy mechanism, called G-PMRM. G-PMRM not only provisions and allocates resources, but also dynamically determines the price that users should pay for their requests. In order to guarantee strategy-proofness of G-PMRM, we design the winner determination algorithm, such that it determines loser-independent allocations on each PM. This property makes the G-PMRM mechanism robust against strategic users who try to manipulate the system by changing the allocations of other users. We prove that G-PMRM mechanism is a polynomial

time 3-approximation mechanism. We also design an optimal strategy-proof mechanism, VCG-PMRM, that we use as a benchmark when we investigate the performance of the G-PMRM mechanism. We perform extensive experiments in order to investigate the performance of the G-PMRM mechanism. The G-PMRM mechanism is fast and finds near optimal solutions, being very suitable for deployment in real cloud settings.

3.1.2 Related Work

Researchers approached the problem of VM placement in clouds considering different objectives and points of view. Dong et al. [38] proposed a method for VM placement considering multiple resource constraints using hierarchical clustering with best fit. Their goal is to improve resource utilization and reduce energy consumption by minimizing both the number of active physical servers and network elements. Ghribi et al. [48] proposed an allocation algorithm with a consolidation algorithm for VM placement in clouds in order to minimize overall energy consumption and migration cost. Maurer et al. [112] proposed a dynamic resource configuration to achieve high resource utilization and low service level agreement violation rates using knowledge management: case-based reasoning and a rule-based approach. Kesavan et al. [72] proposed a set of low-overhead management methods for managing the cloud infrastructure capacity to achieve a scalable capacity allocation for thousands of machines. Hu et al. [61] studied two time-cost optimization problems for provisioning resources and scheduling divisible loads with reserved instances in clouds. They formulated the problems as mixed integer programs. Tsai et al. [167] proposed a hyper-heuristic scheduling algorithm with the aim of reducing the makespan of task scheduling in clouds. Their approach uses two detection operators to determine when to change the low-level heuristic algorithm and a perturbation operator. Doyle et al. [39] proposed an algorithm to determine which data center requests should be routed, based on the relative priorities of the cloud operator. Such routing will reduce the latency, carbon emissions, and operational cost. Srikantaiah et al. [163] modeled the mapping of VMs to PMs as a multidimensional bin packing problem in which PMs are represented by bins, and each resource is considered as a dimension of the bin. They studied energy consumption and

resource utilization and proposed a heuristic algorithm based on the minimization of the sum of the Euclidean distances of the current allocations to the optimal point at each PM. Rodriguez and Buyya [152] proposed a meta-heuristic algorithm based on Particle Swarm Optimization for VM provisioning and scheduling strategies on IaaS that minimizes the overall workflow execution cost while meeting deadline constraints. Their approach considers dynamic provisioning, heterogeneity of unlimited computing resources, and VM performance variation. However, none of the above works proposed strategy-proof mechanisms for resource management in clouds.

Many researchers focused on reducing the operational costs of cloud providers through reducing energy consumption. Mastroianni et al. [110] proposed an approach for the consolidation of VMs on two resources, CPU and RAM, so that both resources are exploited efficiently. Their goal is to consolidate the VMs on as few PMs as possible and switch the other PMs off in order to minimize power consumption and carbon emissions, while ensuring a good level of QoS. In their proposed approach, decisions on the assignment and migration of VMs are driven by probabilistic processes and are based exclusively on local information. Mazzucco et al. [113] proposed policies based on dynamic estimates of users demand and models of system behavior in order to determine the minimum number of PMs that should be switched on to satisfy the demand with two objectives, reducing energy consumption and maximizing revenues. Polverini et al. [143] studied the problem of scheduling batch jobs on multiple geographically-distributed data centers. They considered the benefit of electricity price variations across time and locations. Their proposed algorithm schedules jobs when electricity prices are sufficiently low and to places where the energy cost per unit work is low. Mashayekhy et al. [105] proposed energy-aware scheduling algorithms for detailed task placement of MapReduce jobs. Their scheduling algorithms account for significant energy efficiency differences of different machines in a data center. Khosravi et al. [73] proposed a VM placement algorithm that increases the environmental sustainability by taking into account distributed data centers with different carbon footprint rates. Beloglazov et al. [15] investigated the challenges and architectural principles for energy-efficient management of cloud computing. They proposed energy-aware allo-

cation heuristics that provision resources to improve energy efficiency of the data center considering QoS requirements. For a survey on energy-efficient cloud computing systems, and their taxonomy, the reader is referred to [16]. For a survey of green computing performance metrics for data centers, such as power metrics, thermal metrics, and extended performance metrics, the reader is referred to [173]. However, in this study we focus on cloud auction markets which necessitate designing game theory based mechanisms to reach market equilibria.

Motivated by the recent introduction of cloud auctions by Amazon, resource allocation and pricing in clouds have been increasingly considered by several researchers. Yi et al. [189] proposed a resource provisioning approach to reduce the monetary cost of computation using Amazon spot instances. Their results show that by using an appropriate checkpointing scheme, the cost and task completion time can be reduced. Prasad et al. [145] proposed a cloud resource procurement and dynamic pricing approach in a reverse auction setting with several cloud providers. They proposed several strategy-proof mechanisms for resource procurement and pricing. Fu et al. [45] proposed a core-based pricing method using a coalitional game. However, they did not consider strategy-proofness. Iyer and Veeravalli [66] studied the problem of resource allocation and pricing strategies in cloud computing. They considered the Nash Bargaining Solution and Raiffa Bargaining Solution, and proposed optimal solutions for allocating virtual CPU instances for both independent tasks and workflow tasks. Kang and Wang [69] proposed an auction-based cloud resource allocation algorithm that considers the fitness between resources and services. Mihailescu and Teo [115] proposed a reverse auction-based mechanism for dynamic pricing of resources. A revenue sharing mechanism for multiple cloud providers using cooperative games was proposed by Niyato et al. [130]. Teng and Magoules [164] employed game theoretical techniques to solve the multi-user equilibrium allocation problem, and proposed a resource pricing and allocation policy where users can predict the future resource price. In our previous studies, we proposed truthful mechanisms for VM allocation in clouds without considering their placement onto PMs [96, 123].

Our work is different from all the previous works, since we address the cloud resource

management problem through an economic model by performing VM provisioning, placement, pricing, and considering possible energy savings. Moreover, we consider a realistic cloud setting with multiple heterogeneous PMs providing multiple types of resources, and users requesting different types of VM instances. We also provide worst case performance guarantee for our proposed mechanism.

3.1.3 Organization

The rest of the chapter is organized as follows. In Section 3.2, we describe the PM resource management problem in clouds. In Section 3.3, we present our proposed optimal mechanism. In Section 3.4, we describe our proposed approximation mechanism, and in Section 3.5, we characterize its properties. In Section 3.6, we evaluate the performance of the mechanisms by extensive experiments. In Section 3.7, we summarize our results.

3.2 Physical Machine Resource Management Problem

In this section, we present the system model and the problem of Physical Machine Resource Management (PMRM) in clouds.

We consider a cloud provider managing a public cloud consisting of P PMs, $\mathcal{PM} = \{1, \ldots, P\}$, offering a set $\mathcal{R} = \{1, \ldots, R\}$ of R types of resources such as cores, memory, and storage, to users in the form of VM instances. The information about cloud's physical resources is not known to the users. A PM can be a server, a rack of servers, or a group of racks. A key characteristic of our model is that it enables cloud providers to define PMs based on their resource configurations and user demands. This allows the cloud provider to treat the heterogeneous resources in a flexible way. Each PM p has restricted capacity, C_{pr}, for a resource $r \in \mathcal{R}$ available for allocation. We denote by $\mathcal{C}_p$ the vector of available capacities on each PM p. The cloud provider offers its heterogeneous resources to users in the form of M types of VMs. The set of VM types is denoted by $\mathcal{VM}$. Each VM of type $m \in \mathcal{VM}$ consists of a specific amount of each type of resource $r \in \mathcal{R}$. In addition, w_{mr} represents the amount of resources of type r that one VM instance of type m

Table 3.1: General purpose (M3) VM instance types offered by Amazon EC2.

	m3.medium $m=1$	m3.large $m=2$	m3.xlarge $m=3$	m3.2xlarge $m=4$
CPU	1	2	4	8
Memory (GB)	3.75	7.5	15	30
Storage (GB)	4	32	80	160

provides. Table 3.1 presents the four types of general purpose (M3) VM instances offered by Amazon EC2. By considering CPU, memory, and storage as type 1, type 2, and type 3 resources, respectively, for example, the m3.medium instance ($m = 1$) is characterized by: $w_{11} = 1$, $w_{12} = 3.75$ GB, and $w_{13} = 4$ GB.

We consider a set $\mathcal{N}$ of N users requesting a set of VM instances. Each user has a private valuation for obtaining her request consisting of VM instances. Bidding is the process of expressing user's valuation for a heterogeneous set of VMs and communicating it to the system. In general, it does not matter how the valuation is being encoded, as long as the system can understand the bid submitted by the user (bidder). Users use a bidding language to express their requests. We define a bidding language that can be used to express a user's request (which may or may not be their true request) and to report it to the system. Each user i, $i \in \mathcal{N}$, can submit a pair (ρ_i, b_i), where ρ_i is her requested bundle of VMs and b_i is the price that she is willing to pay for ρ_i. As a result, her valuation is defined as $v_i(\hat{\rho}_i) = b_i$ if $\rho_i \subseteq \hat{\rho}_i$ and $v_i(\hat{\rho}_i) = 0$, otherwise. Such a bid $\beta_i = (\rho_i, b_i)$ is called an atomic bid. Users with atomic bids are called single-minded bidders. User i's requested bundle is represented by $\rho_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$. It is worth noting that ρ_i can consist of one type of VM, while the request for the remaining types of VMs are zero. For example, request ($\langle 10, 0, 0, 5 \rangle$, \$25) represents a user requesting 10 m3.medium VM instances, 0 m3.large VM instance, 0 m3.xlarge VM instance, and 5 m3.2xlarge VM instances; and her bid is \$25. Table 3.2 summarizes the notation used throughout the chapter.

Given the above setting the problem of *Physical Machine Resource Management (PMRM)* in clouds is to determine the allocation of VM to PM simultaneously with the allocation of VM to users and the prices for the VM bundles such that the sum of users' valuations

Table 3.2: Notation

$\mathcal{PM}$	Set of physical machines $\{1, \ldots, P\}$
$\mathcal{VM}$	Set of virtual machine types $\{1, \ldots, M\}$
$\mathcal{R}$	Set of resources $\{1, \ldots, R\}$
w_{mr}	Amount of resources of type $r \in \mathcal{R}$ provided by a VM instance of type $m \in \mathcal{VM}$
C_{pr}	Capacity of $p \in \mathcal{PM}$ for a resource of type $r \in \mathcal{R}$
$\mathcal{N}$	Set of users $\{1, \ldots, N\}$
ρ_i	Bundle requested by user $i \in \mathcal{N}$
k_{im}	Number of requested VM instances of type $m \in \mathcal{VM}$ by user $i \in \mathcal{N}$
b_i	Bid of user $i \in \mathcal{N}$
v_i	Valuation function of user $i \in \mathcal{N}$
u_i	Utility function of user $i \in \mathcal{N}$
Π_i	Payment of user $i \in \mathcal{N}$

is maximized. A mechanism for solving the PMRM problem consists of three phases: winner determination, provisioning and allocation, and pricing. In the winner determination phase, the cloud provider determines which users receive their requested bundles. Based on the results of the winner determination phase, the cloud provider provisions the amount of resources in the form of VM instances onto the PMs, and then allocates the requested bundles of VMs to the winning users. Then, the cloud provider determines the unique amount that each winning user must pay based on the winner determination results. Note that the payment of a user is not greater than its submitted bid. The main building blocks of a PMRM mechanism include: a winner determination function $\mathcal{W}$ and a payment function Π.

Fig. 3.1 shows a high-level view of PMRM. For simplicity, we consider that only one type of resource is available. Four users submit their bids to the cloud provider, where two PMs are available to fulfill the users' requests. As an example, user 1 requests two VM_1 and one VM_2 as her bundle, and she submits a bid of \$0.50. The mechanism employed by the cloud provider collects the bids and then selects the users whose bundle would be provisioned. After it provisions the VMs on the PMs based on the selected users, it allocates the bundles to those users. The selected users pay the amount determined by the mechanism to the cloud provider.

User i has a *quasi-linear utility function* defined as the difference between her valuation and payment, $u_i = v_i(\mathcal{W}_i) - \Pi_i$, where $\mathcal{W}_i$ is the allocated bundle to user i, and Π_i is the

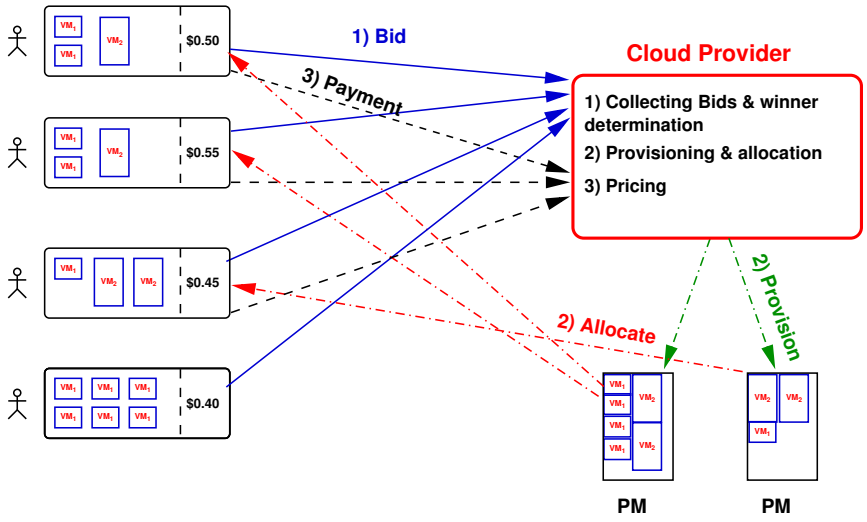


Figure 3.1: A high-level view of PMRM.

payment for user i. The users are self-interested, that is, they want to maximize their own utility. It may be beneficial for cloud users to manipulate the auction outcomes and gain unfair advantages via untruthfully revealing their requests. Since the request of a user is a pair of bundle and value, the user can declare a higher value in the hope to increase the likelihood of obtaining her requested bundle, or declare a different VM bundle from her actual request. Strategic behaviors of such users may hinder other qualified users, leading to reduced revenue and reputation of the cloud provider. Our goal is to design strategy-proof mechanisms that solve the PMRM problem and discourage users from gaming the system by untruthful reporting. The mechanism maximizes social welfare, the sum of users' valuations for the requested bundles of VMs.

3.3 Optimal Mechanism for PMRM

In this section, we propose an optimal strategy-proof mechanism for PMRM. For a detailed introduction on mechanism design the reader is referred to [129].

Cloud users may submit different requests from their true requests hoping to gain more utility. We denote by $\hat{\beta}_i = (\hat{\rho}_i, \hat{b}_i)$ user i's declared request. Note that $\beta_i = (\rho_i, b_i)$ is user i's true request. We denote by $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_N)$ the vector of requests of all users, and by $\boldsymbol{\beta}_{-i}$ the vector of all requests except user i's request (i.e., $\boldsymbol{\beta}_{-i} = (\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_N)$).

Users are rational in a sense that they do not want to pay more than their valuation for their requested bundles. A well-designed mechanism should give incentives to users to participate. Such a property of a mechanism is called individual rationality and is defined as follows:

Definition 9 (Individual rationality)**.** *A mechanism is* individually-rational *if for every user i with true request β_i and the set of other requests, we have $u_i(\beta_i) \geq 0$.*

In other words, a mechanism is individually-rational if a user can always achieve as much utility from participation as without participation. However, such mechanisms do not always give incentives to users to report their requests truthfully. Our goal is to design a mechanism that is strategy-proof, i.e., a mechanism that gives incentives to users to reveal their true requests.

Definition 10 (Strategy-proofness)**.** *A mechanism is* strategy-proof *(or incentive compatible) if $\forall i \in \mathcal{N}$ with a true request declaration β_i and any other declaration $\hat{\beta}_i$, and $\forall \hat{\boldsymbol{\beta}}_{-i}$, we have that $u_i(\beta_i, \hat{\boldsymbol{\beta}}_{-i}) \geq u_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i})$.*

The strategy-proofness property implies that truthful reporting is a dominant strategy for the users. As a result, it never pays off for any user to deviate from reporting her true request, irrespective of what the other users report as their requests.

Our first proposed strategy-proof mechanism is an optimal one and it is based on the Vickrey-Clarke-Groves (VCG) mechanism. An optimal winner determination function with VCG payments provides a strategy-proof mechanism [35, 51, 172]. We define our proposed optimal VCG-based mechanism for PMRM as follows:

Definition 11 (VCG-PMRM mechanism)**.** *The VCG-PMRM mechanism consists of winner determination function $\mathcal{W}$, and payment function Π, where*

i) $\mathcal{W}$ is an optimal winner determination function maximizing the social welfare, and

ii) $\Pi_i(\hat{\boldsymbol{\beta}}) = \sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}_{-i})) - \sum_{j \in \mathcal{N} \setminus \{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}))$,

such that $\sum_{j\in\mathcal{N}\setminus\{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}_{-i}))$ *is the optimal social welfare obtained when user* i *is excluded from participation, and* $\sum_{j\in\mathcal{N}\setminus\{i\}} v_j(\mathcal{W}_j(\hat{\boldsymbol{\beta}}))$ *is the sum of all users valuations in the optimal solution except user* i*'s value.*

The problem that needs to be solved in the winner determination phase of PMRM can be formulated as an integer program (called IP-PMRM), as follows:

$$\text{Maximize } V = \sum_{i\in\mathcal{N}} \sum_{p\in\mathcal{PM}} b_i \cdot X_{ip} \tag{3.1}$$

Subject to:

$$\sum_{p\in\mathcal{PM}} X_{ip} \leq 1, \forall i \in \mathcal{N} \tag{3.2}$$

$$\sum_{i\in\mathcal{N}} \sum_{m\in\mathcal{VM}} k_{im} w_{mr} X_{ip} \leq C_{pr}, \quad \forall p \in \mathcal{PM}, \forall r \in \mathcal{R} \tag{3.3}$$

$$X_{ip} = \{0, 1\} \tag{3.4}$$

The decision variables X_{ip} are defined as follows: $X_{ip} = 1$, if ρ_i is allocated to i on machine p; and 0 otherwise. The objective function is to maximize social welfare V. Constraints (3.2) ensure that the request of each user is fulfilled at most once. Constraints (3.3) guarantee that the allocation of each resource type does not exceed the available capacity of that resource for any PM. Constraints (3.4) represent the integrality requirements for the decision variables.

The winner determination phase of VCG-PMRM (implementing $\mathcal{W}$) consists of solving the IP-PMRM. The execution time of VCG-PMRM becomes prohibitive for large instances of the PMRM problem. As a result, we resort to designing a fast mechanism providing an approximate solution for the PMRM problem. The VCG-PMRM mechanism will be used in our experiments as a benchmark for the performance of the proposed approximation mechanism.

3.4 A Strategy-proof Approximation Mechanism for PMRM

In this section, we introduce our proposed strategy-proof greedy mechanism, G-PMRM. Greedy algorithms to solve PMRM do not necessarily satisfy the strategy-proofness property. To obtain a strategy-proof mechanism, the winner determination function $\mathcal{W}$ must be monotone, and the payment function Π must be based on the critical payment [119]. In addition, we design an iterative winner determination algorithm in the sense that, in each iteration, it determines the assignment of winning requests to their associated PM. This way the mechanism utilizes PMs one by one until all winning requests are assigned. This approach allows the cloud provider to power off unutilized PMs to save energy. In the following, we define the properties that our proposed mechanism needs to satisfy in order to guarantee strategy-proofness.

Definition 12 (Monotonicity)**.** *A winner determination function $\mathcal{W}$ is* monotone *if it selects user i with $\hat{\beta}_i$ as her declared request, then it also selects user i with a more preferred request $\hat{\beta}'_i$, i.e., $\hat{\beta}'_i \succeq \hat{\beta}_i$.*

That means, any winning user who receives her requested bundle by declaring a request $\hat{\beta}_i$ will still be a winner if she requests a more preferred request (i.e., smaller bundle and/or a higher bid). Formally, $\hat{\beta}'_i \succeq \hat{\beta}_i$ if $\hat{b}'_i \geq \hat{b}_i$ and $\hat{\rho}_i =< \hat{k}_{i1}, \hat{k}_{i2}, \ldots, \hat{k}_{iM} >$, $\hat{\rho}'_i =< \hat{k}'_{i1}, \hat{k}'_{i2}, \ldots, \hat{k}'_{iM} >$ such that $\sum_{m \in \mathcal{VM}} \hat{k}'_{im} w_{mr} \leq \sum_{m \in \mathcal{VM}} \hat{k}_{im} w_{mr}, \forall r \in \mathcal{R}$.

Definition 13 (Critical payment)**.** *Let $\mathcal{W}$ be a monotone winner determination function, then for every β_i, there exists a unique value v_i^c, called* critical payment*, such that $\forall \hat{\beta}_i \geq (\rho_i, v_i^c)$, $\hat{\beta}_i$ is a winning declaration, and $\forall \hat{\beta}_i < (\rho_i, v_i^c)$ is a losing declaration. $\Pi_i(\hat{\boldsymbol{\beta}}) = v_i^c$ if user i wins, and $\Pi_i(\hat{\boldsymbol{\beta}}) = 0$, otherwise.*

However, a key challenge in the design of our greedy mechanism in order to satisfy monotonicity is the presence of multiple PMs with multiple types of resources. Lucier and Borodin [86] and Chekuri and Gamzu [31] showed that loser-independent algorithms can be employed as sub-procedures in a greedy iterative approach to obtain monotonicity for the overall winner determination algorithm.

Algorithm 7 G-PMRM Mechanism

1: {Collect user requests.}
2: **for all** $i \in \mathcal{N}$ **do**
3: Collect user request $\hat{\beta}_i = (\hat{\rho}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: $(V, \mathbf{x})$ = G-PMRM-WIN($\hat{\boldsymbol{\beta}}$)
6: Provisions and allocates VM instances according to $\mathbf{x}$.
7: {Payment.}
8: Π =G-PMRM-PAY($\hat{\boldsymbol{\beta}}, \mathbf{x}$)

Definition 14 (Loser-independence)**.** *An algorithm $\mathcal{W}$ is* loser-independent *with respect to user i's request $\hat{\beta}_i$ in which $\mathcal{W}_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i}) = \emptyset$, if user i declares a request $\hat{\beta}'_i$, then either $\mathcal{W}_j(\hat{\beta}'_i, \hat{\boldsymbol{\beta}}_{-i}) = \mathcal{W}_j(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i})$, for all $j \neq i$, or $\mathcal{W}_i(\hat{\beta}_i, \hat{\boldsymbol{\beta}}_{-i}) \neq \emptyset$, where $\hat{\beta}'_i \succeq \hat{\beta}_i$.*

In other words, if user i was not selected by algorithm $\mathcal{W}$ when she declared request $\hat{\beta}_i$ and now she declares a new request $\hat{\beta}'_i$ while the declared requests of the rest of the users do not change, then the outcome of algorithm $\mathcal{W}$ changes only if user i becomes a winner by declaring a better request $\hat{\beta}'_i$.

If the bid of a not-selected user changes but her allocation stays the same then the allocations to all other users do not change. A key property of loser-independent algorithms is that if a user is not a winner, it guarantees the same output no matter her declaration. This property makes the algorithm robust against strategic users who try to manipulate the system by changing other users allocations. Note that if such a user tries to change the allocation determined by the algorithm, she should declare a request that will make her a winner.

Obtaining strategy-proofness requires the design of a loser-independent winner determination algorithm that allocates the resources of each PM individually. If the winner determination algorithm is loser-independent for each PM, then when the solutions for each individual machines are combined in an iterative fashion it will lead to a monotone overall winner determination algorithm. Having a monotone winner determination algorithm along with a critical value payment, makes the mechanism strategy-proof.

We define our proposed G-PMRM mechanism as follows:

Definition 15 (G-PMRM mechanism)**.** *The G-PMRM mechanism consists of the winner*

Algorithm 8 IS-FEASIBLE($\rho_i, \mathcal{C}_p$)

1: **for all** $r \in \mathcal{R}$ **do**
2: $\quad \sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
3: $flag \leftarrow$ TRUE
4: **for all** $r \in \mathcal{R}$ **do**
5: $\quad$ **if** $\sigma_{ir} > C_{pr}$ **then**
6: $\quad\quad flag \leftarrow$ FALSE
7: $\quad\quad$ **break;**
8: **Output:** $flag$

determination algorithm G-PMRM-WIN and the payment function G-PMRM-PAY.

The G-PMRM mechanism is given in Algorithm 7. The mechanism is run periodically by the cloud provider. It collects the requests from all users, and then it determines the winning users by calling the G-PMRM-WIN algorithm. Once the users are selected the mechanism provisions the required number and types of VM instances on the selected PMs, and then it determines the payments by calling the G-PMRM-PAY function. The users are then charged the payment determined by the mechanism. G-PMRM-WIN and G-PMRM-PAY are presented in the following.

Before describing the winner determination algorithm, we need to define a function, called IS-FEASIBLE(), that we call in our proposed winner determination algorithm. IS-FEASIBLE() is given in Algorithm 8. It checks the feasibility of allocating the requested bundle of VMs of a user on a specific PM, that is, it checks whether PM p has enough resources to fulfill a requested bundle of VMs. For user i with $\rho_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$, and PM p, the function computes σ_{ir}, the amount of resources of type r requested by user i, and then checks it against the available resource capacity C_{pr} for all types $r \in R$ of resources on PM p.

G-PMRM-WIN is given in Algorithm 9. G-PMRM-WIN has one input parameter, the vector of users declared requests $\hat{\boldsymbol{\beta}}$, and two output parameters: V, the total social welfare, and $\mathbf{x}$, the set of winning users.

The algorithm finds the total amount of each resource type requested by each user in $\mathcal{N}$ (lines 2-4). It also calculates the reserve price for each request (line 5). We consider reservation prices for the VMs to avoid non-profitable trade. The reserve price is often

a reflection of the VM cost. The cloud provider sets a reserve price o_m for each type of VM $m \in \mathcal{VM}$. These prices are denoted by a vector $\mathcal{O} =< o_1, \ldots, o_M >$. The reserve price (bundle-specific) of a user is calculated based on her requested bundle as follows: $rp_i = \sum_{m \in \mathcal{VM}} k_{im} o_m$, which is the weighted sum of reserve prices for all the requested VMs in the bundle of user i.

Users whose bids are above the reserve prices are included in $\mathcal{U}$, the set of users who are not yet selected to receive their requested bundles (lines 6-7). In the following, we call $\mathcal{U}$, the set of not-selected users. Then, the algorithm iterates over all PMs to find a subset of users whose requests should be assigned to each PM p, where the vector of resource capacities of PM p is $\mathcal{C}_p = (C_{p1}, \ldots, C_{pR})$ (lines 8-60). Each iteration of the algorithm (lines 8-60) consists of three phases. In the first phase, the algorithm finds the user with the maximum bid (lines 9-17). In the second phase, the algorithm finds the set of users based on their bid densities whose overall requested amount of resources is at least half of the capacities of the PM (lines 18-54). In the third phase, the algorithm finds the maximum social welfare V_p between the obtained social welfare of the first and second phase (lines 55-60). Based on the phase providing the maximum social welfare, the algorithm chooses the set of winning users $\mathbf{x}_p$ for PM p. The requested bundle of VMs of these users will be provisioned using the resources of PM p. Then, the algorithm updates the set of not-selected users $\mathcal{U}$ (line 60). After finding the set of winning users to be allocated to each PM $p, \forall p \in \mathcal{PM}$, the algorithm calculates the obtained social welfare V and the final set of winning users specified by vector $\mathbf{x}$ (line 61).

In the following, we discuss each phase in detail. In phase one, the algorithm first finds the set of users $\hat{\mathcal{U}}$ whose requests can be fulfilled by PM p (lines 9-14) by calling the IS-FEASIBLE() function that checks the feasibility of allocating the requested bundle of VMs of each user i on PM p. Then, it finds the maximum bid among the users in $\hat{\mathcal{U}}$ (line 15). It also finds the user associated with it as a winning user and updates $\hat{\mathbf{x}}$ (lines 16-17).

In the second phase, the algorithm finds the set of users $\tilde{\mathcal{U}}$, where each user's request is not greater than half of the available capacity of the PM p, for each resource by calling IS-FEASIBLE (lines 18-22). Then, the algorithm calculates the bid densities of users

Algorithm 9 G-PMRM-WIN($\hat{\boldsymbol{\beta}}$)

$\mathcal{U} = \emptyset$
for all $i \in \mathcal{N}$ **do**
  for all $r \in \mathcal{R}$ **do**
    $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
  $rp_i = \sum_{m \in \mathcal{VM}} k_{im} o_m$
  if $\hat{b}_i \geq rp_i$ **then**
    $\mathcal{U} = \mathcal{U} \cup \{i\}$
for all $p \in \mathcal{PM}$ **do**
  {*First phase*}
  $\mathcal{C}_p = (C_{p1}, \ldots, C_{pR})$
  $\hat{\mathcal{U}} \leftarrow \emptyset$; $\hat{\mathbf{x}} = \mathbf{0}$
  for all $i \in \mathcal{U}$ **do**
    if IS-FEASIBLE($\rho_i, \mathcal{C}_p$) **then**
      $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \cup \{i\}$
  $\hat{V} = \max_{i \in \hat{\mathcal{U}}} \hat{b}_i$
  $j = \text{argmax}_{i \in \hat{\mathcal{U}}} \hat{b}_i$
  $\hat{x}_j = 1$
  {*Second phase*}
  $\tilde{\mathcal{U}} \leftarrow \emptyset$; $\tilde{\mathbf{x}} = \mathbf{0}$
  for all $i \in \mathcal{U}$ **do**
    if IS-FEASIBLE($\rho_i, \mathcal{C}_p/2$) **then**
      $\tilde{\mathcal{U}} \leftarrow \tilde{\mathcal{U}} \cup \{i\}$
  for all $i \in \tilde{\mathcal{U}}$ **do**
    $d_i = \hat{b}_i / \sqrt{\sum_{r=1}^{R} \frac{\sigma_{ir}}{C_{pr}}}$
  Sort $\tilde{\mathcal{U}}$ in decreasing order of d_i
  $\bar{\mathcal{U}} \leftarrow \emptyset$; $\tilde{\mathcal{C}}_p = 0$; $flag \leftarrow$ `TRUE`
  while $\tilde{\mathcal{U}} \neq \emptyset$ and $flag$ **do**
    for all $r \in \mathcal{R}$ **do**
      if $\tilde{C}_{pr} > C_{pr}/2$ **then**
        $flag \leftarrow$ `FALSE`
    if $flag$ **then**
      $i \leftarrow \text{argmax}_{i \in \tilde{\mathcal{U}}} d_i$
      $\tilde{\mathcal{U}} = \tilde{\mathcal{U}} \setminus \{i\}$
      $\tilde{x}_i = 1$
      $\bar{\mathcal{U}} \leftarrow \bar{\mathcal{U}} \cup \{i\}$
      for all $r \in \mathcal{R}$ **do**
        $\tilde{C}_{pr} = \tilde{C}_{pr} + \sigma_{ir}$
  $\tilde{V} = 0$; $\bar{\mathcal{C}}_p = 0$
  if $\bar{\mathcal{U}} \neq \emptyset$ **then**
    for all $i \in \bar{\mathcal{U}}$ except the last user j added to $\bar{\mathcal{U}}$ **do**
      $\tilde{V} = \tilde{V} + \hat{b}_i$
      for all $r \in \mathcal{R}$ **do**
        $\bar{C}_{pr} = \bar{C}_{pr} + \sigma_{ir}$
    for all $r \in \mathcal{R}$ **do**
      $\bar{\sigma}_{jr} = C_{pr}/2 - \bar{C}_{pr}$
    $flag \leftarrow$ `TRUE`
    for all $r \in \mathcal{R}$ **do**
      if $\sigma_{jr} > \bar{\sigma}_{jr}$ **then**
        $flag \leftarrow$ `FALSE`
    if $flag$ **then**
      for all $r \in \mathcal{R}$ **do**
        $\bar{\sigma}_{jr} = \sigma_{jr}$
    $\bar{b}_j = d_j \sqrt{\sum_{r=1}^{R} \frac{\bar{\sigma}_{jr}}{C_{pr}}}$
    $\tilde{V} = \tilde{V} + \bar{b}_j$
  {*Third phase*}
  if $\hat{V} \geq \tilde{V}$ **then**
    $V_p = \hat{V}$; $\mathbf{x}_p = \hat{\mathbf{x}}$
  else
    $V_p = \tilde{V}$; $\mathbf{x}_p = \tilde{\mathbf{x}}$
  Update $\mathcal{U}$ to the unallocated users based on $\mathbf{x}_p$
$V = \sum_{p \in \mathcal{PM}} V_p$; $\mathbf{x} = \sum_{p \in \mathcal{PM}} \mathbf{x}_p$
Output: V, $\mathbf{x}$

in $\tilde{\mathcal{U}}$ (lines 23-24) according to a *density* metric defined as $d_i = \frac{\hat{b}_i}{\sqrt{\sum_{r=1}^{R} \frac{\sigma_{ir}}{C_{pr}}}}, \forall i \in \mathcal{U}$, where $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$ is the amount of each resource of type r requested by user i, and $\frac{1}{C_{pr}}$ is the *relevance factor* characterizing the scarcity of resources of type r. Next, the algorithm sorts the users in $\tilde{\mathcal{U}}$ based on the decreasing order of their bid densities (line 25). Then, the algorithm selects users based on their bid densities (lines 26-37). To do that it checks whether the total amount of requests of the current selected users are less than half of the available capacity of each resource in the PM (lines 28-30). If the total amount of requests do not reach the half, the algorithm selects a new user according to the bid densities, and updates the set $\bar{\mathcal{U}}$, $\bar{\mathbf{x}}$, and the total amount of requests assigned to PM p (lines 31-37). Then, G-PMRM-WIN finds the social welfare of the selected users in the second phase if $\bar{\mathcal{U}}$ is not an empty set (lines 38-54). It first, finds the social welfare $\tilde{V}$ of all the selected users except the last user (i.e., user j) added to the set $\bar{\mathcal{U}}$ (lines 40-43). Then, the algorithm finds the remaining capacities of each resource $\bar{\sigma}_{jr}$ from half of the capacities of the PM (lines 44-45). It also checks if the actual request of user j is less than the remaining capacities (lines 46-52). Next, the algorithm calculates the value $\bar{b}_j$ (line 53) based on either the remaining capacities (lines 44-45) or the actual user j's request if her request is less than the remaining capacities (lines 51-52). Finally, the algorithm updates the social welfare $\tilde{V}$ by adding $\bar{b}_j$ (line 54).

In phase three, the algorithm selects the maximum social welfare and the associated selected users from the two phases as the social welfare and the set of winning users whose requests will be provisioned on PM p (lines 55-60). The obtained social welfare on PM p is V_p, the maximum social welfare between the social welfare of the two phases. The set of winning users whose requests will be provisioned on PM p, $\mathbf{x}_p$, is the solution that gives V_p. Then, the algorithm updates the set of not-selected users $\mathcal{U}$ based on $\mathbf{x}_p$ to guarantee that each user is selected at most once (line 60). After finding the set of users to be allocated to each PM, the algorithm calculates the obtained social welfare V and the final set of winning users specified by vector $\mathbf{x}$ (line 61).

The G-PMRM-PAY function is given in Algorithm 10. The G-PMRM-PAY function has two input parameters, the vector of users declared requests ($\hat{\boldsymbol{\beta}}$), and the set of winning

Algorithm 10 G-PMRM-PAY: Critical Payment Function

1: **Input:** $\hat{\boldsymbol{\beta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_N)$; vector of requests (bundle, bid)
2: **Input:** $\mathbf{x}$; winning users
3: **for all** $i \in \mathcal{U}$ **do**
4: $\quad \Pi_i = 0$
5: $\quad$ **if** x_i **then**
6: $\quad\quad l = 0$
7: $\quad\quad$ **for all** $m \in \mathcal{VM}$ **do**
8: $\quad\quad\quad l = l + k_{im} o_m$
9: $\quad\quad h = \hat{b}_i$
10: $\quad\quad$ **while** $(h - l) \geq 1$ **do**
11: $\quad\quad\quad v_i^c = (h + l)/2$
12: $\quad\quad\quad \hat{\beta}_i^c = (\hat{\rho}_i, v_i^c)$
13: $\quad\quad\quad (V', \mathbf{x}') =$ G-PMRM-WIN $((\hat{\beta}_1, \ldots, \hat{\beta}_i^c, \ldots, \hat{\beta}_N))$
14: $\quad\quad\quad$ **if** x_i' **then**
15: $\quad\quad\quad\quad h = v_i^c$ {user i is winning by declaring v_i^c}
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad l = v_i^c$
18: $\quad\quad \Pi_i = h$
19: **Output:** $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_N)$

users given by $\mathbf{x}$. The payment of winning user i is v_i^c, where v_i^c is the critical payment of user i, if i wins and zero if i loses. Finding the critical payment is done by a binary search over values less than the declared value and above the reserve price. We consider reserve prices for the VMs in order to avoid non-profitable trade.

G-PMRM-PAY initializes the payment of each user to zero (line 4). Then for each winning user i (i.e., $x_i = 1$) it initializes the lower and upper bounds of the payment to the reserve price and the declared value of the user, respectively (lines 6-9). The reserve price of a user is calculated based on her requested bundle as follows: $l = \sum_{m \in \mathcal{VM}} k_{im} o_m$, which is the weighted sum of reserve prices for all the requested VMs in the bundle of user i.

G-PMRM-PAY sets the critical payment, v_i^c, as the average of the lower and upper bounds, and checks if user i would have won if she have had declared her bid as v_i^c (lines 11-13). If user i would have won by declaring v_i^c, G-PMRM-PAY sets the value as a new upper bound; otherwise, the value is a new lower bound (lines 14-17). G-PMRM-PAY tries to reduce the gap between upper and lower bound to 1. Then, the payment of user i, Π_i is set to the upper bound value (line 18). G-PMRM-PAY returns an output parameter, Π, the payment vector for the users.

Table 3.3: Example - Users requests.

User	requested storage	bid	bid density
I	512 GB	\$102	204
II	512 GB	\$102	204
III	768 GB	\$150	200
IV	256 GB	\$50	200
V	1024 GB	\$198	198
VI	1024 GB	\$198	198

Example. A key idea in the design of G-PMRM-WIN, is finding a partial allocation in the second phase in order to not only guarantee approximation ratio but also strategy-proofness. It is not possible to guarantee the strategy-proofness of the mechanism, if we allow G-PMRM-WIN to find an allocation considering the full capacity in the second phase. According to the Definition 4 (Monotonicity), any winning user who receives her requested bundle by declaring a request $\hat{\beta}_i$ will still be a winner if she requests a more preferred request (i.e., smaller bundle and/or a higher bid). In the following, we provide an example showing that considering the full capacity in the second phase will not satisfy the monotonicity property, that is, showing that if a winning user submits a higher bid, she becomes a loser.

We consider six users, where their requests, bids, and bid densities are shown in Table 3.3. We consider a cloud provider with only one type of resource (storage) with two PMs each with capacity of 1024 GB. The first phase of the mechanism selects users I and II for PM_1, and users III and IV for PM_2, where the solution has a total value of \$102+\$102+\$150+\$50=\$404. The second phase of the modified mechanism (i.e., considering the full capacity) selects user V for PM_1, and user VI for PM_2, where the solution has a total value of \$198+\$198=\$396. As a result, the mechanism selects the result of the first phase as the solution. Now, we consider that user IV submits a higher bid of \$52 instead of her actual value (\$50). Her bid density would change to 208, which is the highest density among all six users. Therefore, the first phase selects users IV and I for PM_1, and user II for PM_2, where the solution has a total value of \$102+\$102+\$52=\$256. The second phase (with full capacity) selects user V for PM_1, and user VI for PM_2, where the solution has a total value of \$198+\$198=\$396. As a result, the modified mechanism selects the results

of the second phase as the solution. This solution does not include user IV anymore, and the winner determination is not monotone, and thus the mechanism is not strategy-proof. This example shows that a user can lose by declaring a higher value which should not be a case.

In the next section, we prove that our proposed mechanism is strategy-proof and that its worst-case performance is well-bounded.

3.5 Properties of G-PMRM

In this section, we investigate the properties of G-PMRM. We first show that the mechanism is individually rational (i.e., truthful users will never incur a loss).

Theorem 6. *G-PMRM mechanism is individually rational.*

Proof. We consider two cases. In case one, we consider a truthful user i who does not win. Such user is not incurring a loss since she pays 0 (line 4 of Algorithm 10), and her utility is 0. In case two, we consider user i as a winning user. We need to prove that if user i reports her true request then her utility is non-negative. In line 18 of Algorithm 10, the payment for user i is set to h, where h is initially set to $\hat{b}_i$ as an upper bound. The determined payment of user i is less than the initial value of h due to binary search. As a result, G-PMRM-PAY always computes a payment $\Pi_i \leq \hat{b}_i$. The utility of user i (i.e., $u_i = \hat{b}_i - \Pi_i \geq 0$) is non-negative if she report truthfully (i.e., $\hat{b}_i = b_i$), and she never incurs a loss. This proves the individual-rationality of G-PMRM. □

We now prove that the allocation on each PM (obtained in each iteration of the G-PMRM-WIN algorithm) is loser-independent.

Theorem 7. *The allocation obtained by each iteration of G-PMRM-WIN is loser-independent.*

Proof. To prove the loser-independency property of each iteration of G-PMRM-WIN, we need to analyze the results of two scenarios, a current declaration and a new declaration. In the current declaration, user i submits a request $\hat{\beta}_i$, and G-PMRM-WIN on PM p finds $\hat{V}$ and $\tilde{V}$ as the social welfare obtained by the first and the second phase, respectively. In

the new declaration, user i submits a request $\hat{\beta}'_i \succeq \hat{\beta}_i$, and the rest of the users declare the same request as they declared in their current declarations. G-PMRM-WIN on PM p finds $\hat{V}'$ and $\tilde{V}'$ as the social welfare obtained in this scenario by the first and the second phase, respectively.

In order to prove that the allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent, we need to show that $\hat{V}' \geq \hat{V}$ and $\tilde{V}' \geq \tilde{V}$. In addition, if either $\hat{V}' > \hat{V}$ or $\tilde{V}' > \tilde{V}$, then user i has been selected, and if the obtained social welfare does not change, then either user i has been selected or the allocation of the users does not change. We separate the proof into two cases as follows.

i) $\hat{V}' \geq \hat{V}$, if one user increases her bid. If $\hat{V}' > \hat{V}$, then user i must be the user with the maximum bid, and thus, user i is in the solution (i.e., $\hat{x}'_i = 1$). If $\hat{V}' = \hat{V}$, then the allocation of the users does not change unless user i declares a bid $\hat{b}'_i = \hat{V}$, and she is selected by the algorithm.

ii) $\tilde{V}' \geq \tilde{V}$, we consider two subcases (a) and (b). In subcase (a), we consider that the overall amount of resource requests is less than half of the capacities of a PM. Then all users must be selected (i.e., $\tilde{x}'_i = 1, \forall i \in \tilde{\mathcal{U}}'$), where $\tilde{\mathcal{U}} \subseteq \tilde{\mathcal{U}}'$ since user i may declare a smaller bundle. As a result, $\tilde{V}' \geq \tilde{V}$. In subcase (b), we consider that the overall amount of resource requests is at least half of the capacities of a PM. Note that user i has a better bid density by declaring $\hat{\beta}'_i$. If $\tilde{x}'_i = 0$, then $\tilde{x}_i = 0$. If $\tilde{V}' = \tilde{V}$ and $\tilde{x}'_i = 0$, then the allocation would be the same. In addition, if $\tilde{V}' > \tilde{V}$, then $\tilde{x}'_i = 1$.

This proves that the allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent. □

Theorem 8. *G-PMRM-WIN is a monotone winner determination algorithm.*

Proof. The allocation obtained by each iteration of G-PMRM-WIN on each machine p is loser-independent with respect to the request of each user. If loser-independent algorithms are employed as sub-procedures in a greedy iterative approach, then the overall winner determination algorithm is monotone [86, 31]. Therefore, the overall allocation on all PMs obtained by G-PMRM-WIN is monotone. □

Theorem 9. *G-PMRM-PAY implements the critical payment.*

Proof. We need to prove that Π_i is the critical payment for user $i, \forall i \in \mathcal{N}$. We assume that user i is selected by G-PMRM-WIN (i.e., $x_i = 1$). If user i declares a higher value than Π_i, (i.e., $\hat{b}'_i > \Pi_i$), she wins and pays the same amount Π_i. This is due to the fact that if user i was selected in phase one, then declaring a higher value makes her a winner. In addition, if user i was selected in phase two, declaring a higher value increases her bid density, and thus, user i becomes a winner. If user i declares a lower value than Π_i, (i.e., $\hat{b}'_i < \Pi_i$), this leads to a lower bid density and a lower value. If user i was chosen based on either phase one or two, a lower bid density and a lower value for the user makes user i a non-winning user. These show that the payment Π_i is the minimum valuation that user i must bid to obtain her required bundle. This payment is between the reserve price and the declared value of the user. The critical payment property holds considering the reserve prices. In the case in which user i is not a winner, she pays 0, thus, satisfying the properties of the critical payment. As a result, the payment determined by G-PMRM-PAY is the critical payment. □

We now show that our proposed mechanism, G-PMRM, is strategy-proof.

Theorem 10. *G-PMRM mechanism is strategy-proof.*

Proof. The winner determination is monotone (Theorem 3) and the payment is the critical value payment (Theorem 4), therefore, according to [129], our proposed mechanism, G-PMRM, is strategy-proof. □

Theorem 11. *The time complexity of G-PMRM is polynomial.*

Proof. The time complexity of G-PMRM-WIN is $O(PN(\log N + MR))$. This is because sorting the requests requires $O(N \log N)$, while checking the feasibility of the allocation for each user on each PM requires $O(MR)$. The time complexity of G-PMRM-PAY is polynomial for similar reasons. As a result, the time complexity of G-PMRM is polynomial. □

We now prove that in the case of only one PM (i.e., $P = 1$), G-PMRM-WIN is a 2-approximation algorithm.

Theorem 12. *The approximation ratio of G-PMRM-WIN in the case of only one PM is* 2.

Proof. Let X^* be set of users in the optimal solution, and V^* be the optimal social welfare. Let X and V be the set of users and the social welfare in the obtained solution by G-PMRM, respectively. We need to prove that $V^* \leq V\alpha$, where α is the approximation ratio.

We consider two cases:

i) V is obtained by the second phase of the winner determination algorithm (i.e., $V = \tilde{V}$). If the amount of overall requests is less than half of the capacities of the physical machine, then all such users must be selected (i.e., $X^* = X$ and $V^* = V$). We now consider that the amount of overall allocated requests is at least one half the resource capacities of a physical machine.

In the optimal solution, for the remaining capacity of that resource, the social welfare is less than V since the second phase is based on bid densities. Thus, the first half contains the most valuable requests. Therefore, $V > V^*/2$.

ii) V is obtained by the first phase of the winner determination algorithm (i.e., $V = \hat{V}$). That means $\hat{V} \geq \tilde{V}$. In the optimal solution, the first half cannot have a better social welfare and the second half cannot have a better social welfare than the first half. As a result, $V > V^*/2$. □

Theorem 13. *The approximation ratio of G-PMRM in the case of multiple PMs is* 3.

Proof. To prove this theorem we use a result from [31], that states that if the winner determination algorithm is α-approximation on a bin, then the overall approximation ratio of the winner determination algorithm applied iteratively on multiple bins is $\alpha + 1$. Since we proved that the approximation ratio for G-PMRM-WIN on only one PM is 2, then it follows from [31] that the overall approximation ratio of G-PMRM on multiple PMs is 3. □

3.6 Experimental Results

We perform extensive experiments in order to investigate the performance of the proposed mechanism G-PMRM against the performance of the optimal VCG-PMRM mechanism. While it is desirable to compare G-PMRM with several other mechanisms, we found out that the existing mechanisms and approaches are not directly comparable to ours and decided to compare it with the optimal mechanism, VCG-PMRM. Therefore, we rely on the optimal results obtained by VCG-PMRM as a benchmark for our experiments. For the VCG-PMRM mechanism, we use the IBM ILOG CPLEX Optimization Studio Multiplatform Multilingual eAssembly to solve the PMRM problem optimally. The mechanisms are implemented in C++ and the experiments are conducted on AMD 2.4GHz Dual Proc Dual Core nodes with 16GB RAM which are part of the WSU Grid System. In this section, we describe the experimental setup and analyze the experimental results.

3.6.1 Experimental Setup

The generated requests are based on realistic data combining publicly available information provided by Amazon EC2 and Microsoft Azure as follows. We consider the same types of VM instances available to users as those offered by Amazon EC2. Each of these VM instances has specific resource demands with respect to two available resource types: cores and memory. We also set the amount of each resource type provided by a VM instance to be the same as in the specifications provided by Amazon Web Services for its Spot Instances and Elastic Compute Cloud (EC2) (See Table 3.1). Users can request a bundle of VMs, where for each VM type, they can request between 0 and 20 VM instances.

We generate bids based on Amazon Spot market report on users bidding strategies [2]. Amazon regularly updates its spot price history based on the past 90 days of activity. Amazon reported that most users bid between the price of reserved instances and on-demand prices. By doing so, these users saved between 50% and 66% compared to the on demand prices. The lowest price of the reserved instances is for the *Heavy Utilization Reserved Instances* which is $0.018 per hour for a medium VM instance of General Purpose

Table 3.4: Simulation Parameters

Param.	Description	Value(s)
N	Number of users	[50-600]
M	Number of VM instances	4 (M,L,XL,2XL)
R	Number of resource types	2 (Core, Memory)
PM	Number of PMs	100
C_1	Core capacity	512 cores
C_2	Memory capacity	1244.9 GB
w_{mr}	Amount of resource r provided by a VM instance m	as in Table 3.1
k_{im}	Number of requested VM of type m by user i	[0, 20]
b_i^0	bid of user i for a medium VM	$[0.018, 0.34]$
b_i	value of user i	$b_i^0 \sum_{m=1}^{M} 2^{m-1} k_{im}$

- Current Generation. However, the trade-off is that the user's requested bundles can be reclaimed by a cloud provider if the spot price exceeds their submitted bid prices. Thus, some users bid above on-demand prices and up to twice the on-demand prices in some cases. To generate bids, we generate a random number, b_i^0, for each user i from the range $[0.018, 0.34]$ for a medium VM instance, where the range is given by the lowest price for the reserved Amazon EC2 medium instance and the on-demand price for the medium instance. Then, we multiply the random number by the total weights of VMs in the user's requested bundle. The total weight of a VM instance for user i is $\sum_{m=1}^{M} 2^{m-1} k_{im}$. For both mechanisms G-PMRM and VCG-PMRM, we set the reserve prices of the VMs to the half of the posted prices for the Heavy Utilization Reserved Instances. Such reservation prices are reasonable considering the VM costs and a low profit margin. The parameters and their values used in the experiments are listed in Table 3.4.

We setup the PM configurations based on the specification of the Titan system [9] at Oak Ridge National Laboratory (No. 2 in Top500 [10]). Titan currently contains 299,008 CPU cores and 710 terabytes of memory. We consider 100 PMs available from Titan, where each PM has 512 cores and 1244.9 gigabytes of memory.

3.6.2 Analysis of Results

We compare the performance of G-PMRM and VCG-PMRM for different numbers of users, ranging from 50 to 600. For 500 and 600 users, the optimal mechanism, VCG-PMRM,

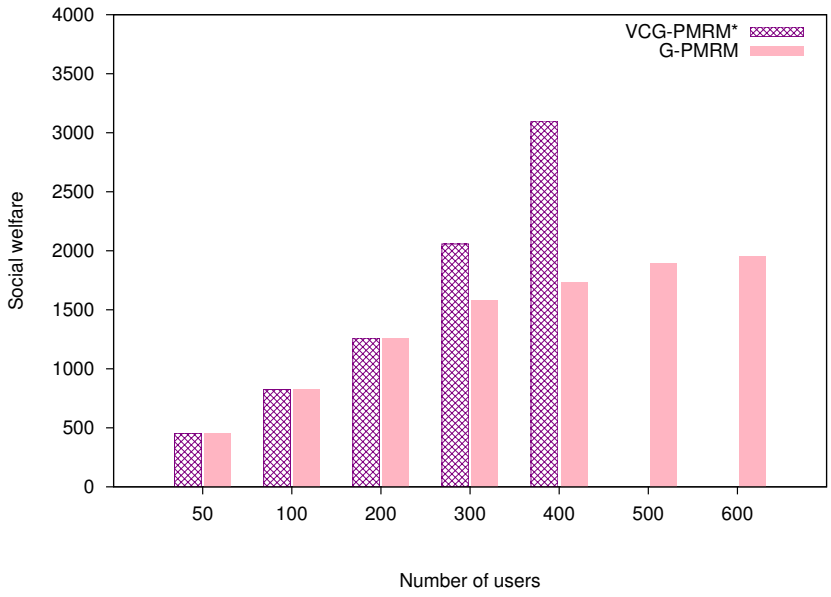


Figure 3.2: G-PMRM vs. VCG-PMRM: Social welfare

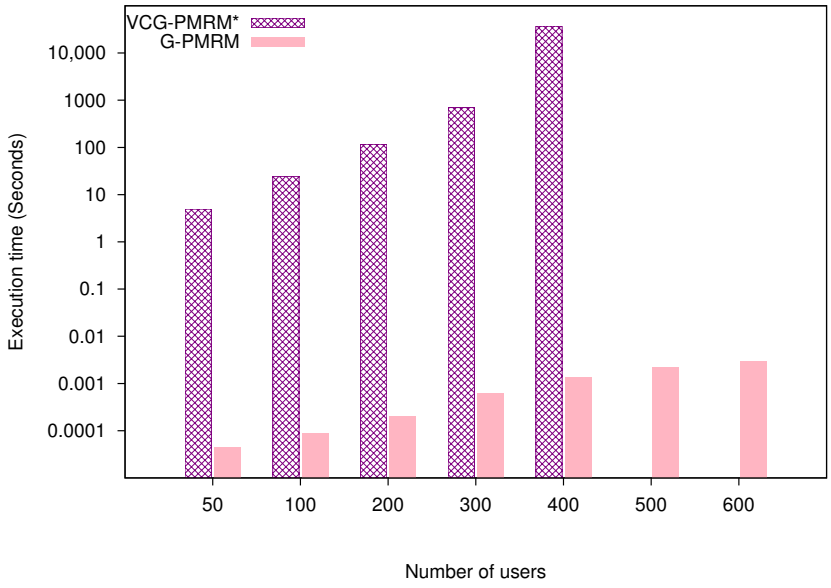


Figure 3.3: G-PMRM vs. VCG-PMRM: Execution time

could not find the solutions even after 24 hours. This is due to the fact that the problem is strongly NP-hard, and it is infeasible to solve for large instances.

Fig. 3.2 shows the social welfare obtained by the mechanisms for 50 to 600 users. The results show that G-PMRM obtains the optimal social welfare when user demand is low

compared to the available capacity. For example, for 200 users, G-PMRM and VCG-PMRM obtain the same social welfare of 1259.19. However, the results show that with the increase of the user demand, optimality gap increases. For example, for 300 users, the social welfare obtained by G-PMRM and VCG-PMRM is 1582.25 and 2056.26, respectively. For the first five groups of users for which the optimal mechanism could find solutions, the optimality gap is 23%. This gap is reasonable given the fact that the execution time of the VCG-PMRM is very large as we show in the next figure.

Fig. 3.3 shows the execution times of the mechanisms on a logarithmic scale. The execution time of VCG-PMRM is more than five orders of magnitude greater than that of G-PMRM. G-PMRM is very fast being suitable for real cloud settings, where the number of PMs and users are large. In addition, in auction-based mechanisms the response time should be very small. For example, Amazon runs its Spot Market auction-based mechanism every hour, and needs to find the winning users and their payments as soon as possible. In particular, the optimal VCG-PMRM mechanism is not feasible when the problem scales. VCG-PMRM was not able to determine the allocation for 500 and 600 users in feasible time, and thus, there are no bars in the plots for those cases. The results of Fig. 3.2 and 3.3 show that when the user demand is low, VCG-PMRM obtains the results in reasonable time. However, the execution time of VCG-PMRM is prohibitive when the demand is high. On the other hand, G-PMRM not only obtains the optimal results when the user demand is low, but also it obtains reasonable solutions very fast when the demand is high.

Fig. 3.4 shows the revenue obtained by the cloud provider using both mechanisms. The results show that G-PMRM and VCG-PMRM obtains the same revenue for the cloud provider when user demand is low compared to the available capacity. However, when the demand is high, G-PMRM obtains a higher revenue than VCG-PMRM. This is due to the fact that VCG-PMRM fulfills more requests, which in turn, leads to accepting more bids, and thus, reducing the price. Note that both mechanisms charge users bellow their submitted bids.

Fig. 3.5 shows the percentage of used PMs for the mechanisms. With an increase in the number of users, both mechanisms activate more PMs. For example for 100 users,

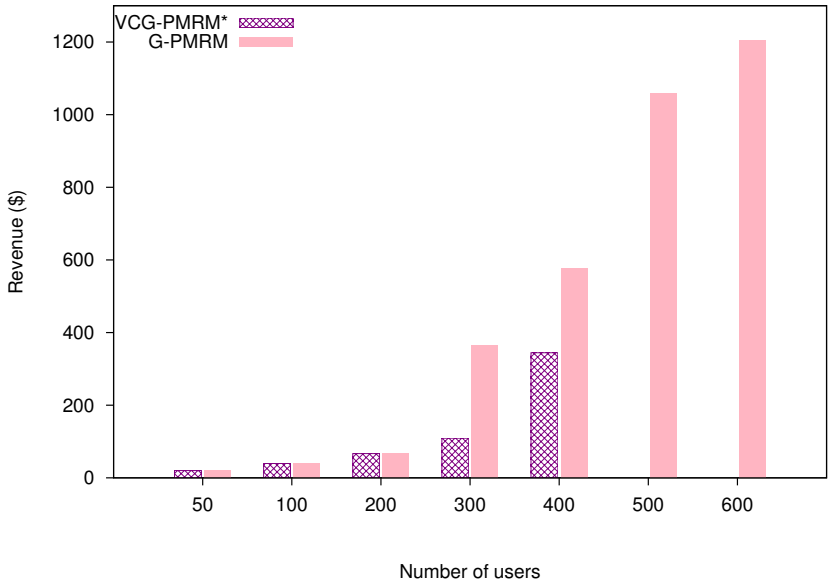


Figure 3.4: G-PMRM vs. VCG-PMRM: Revenue

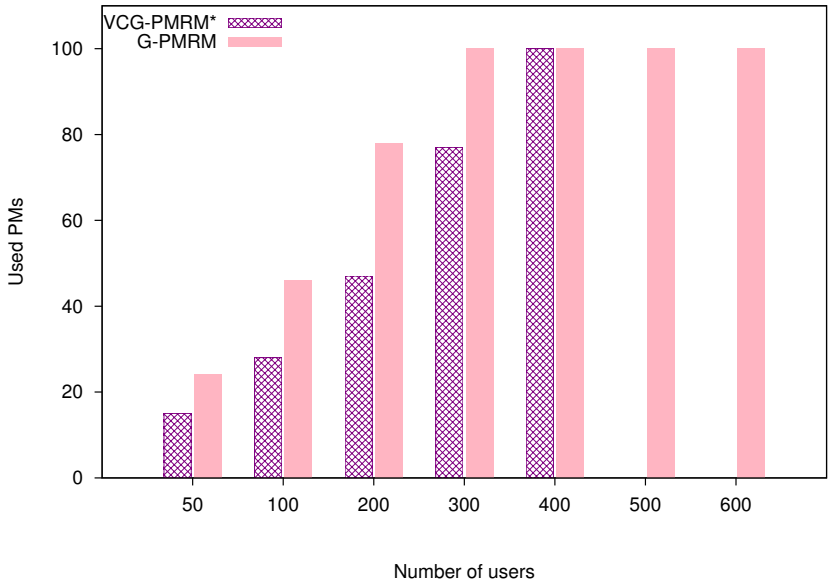


Figure 3.5: G-PMRM vs. VCG-PMRM: Used PMs

G-PMRM powers on about 46 percent of all the PMs, while VCG-PMRM powers about 28 percent of all the PMs. By increasing the number of users, all PMs are activated by both mechanisms. This can be seen for 400 users.

Fig. 3.6 shows the percentage of served users by the mechanisms. When the demand is

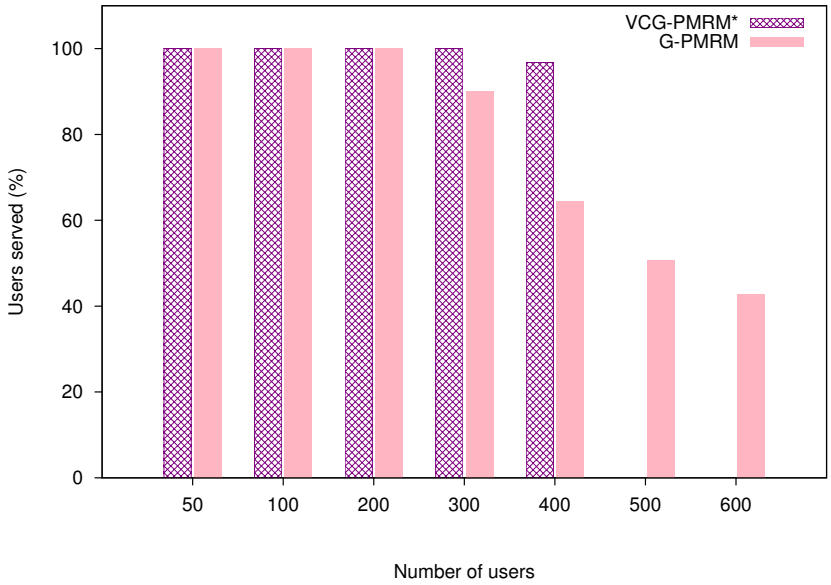


Figure 3.6: G-PMRM vs. VCG-PMRM: Users served

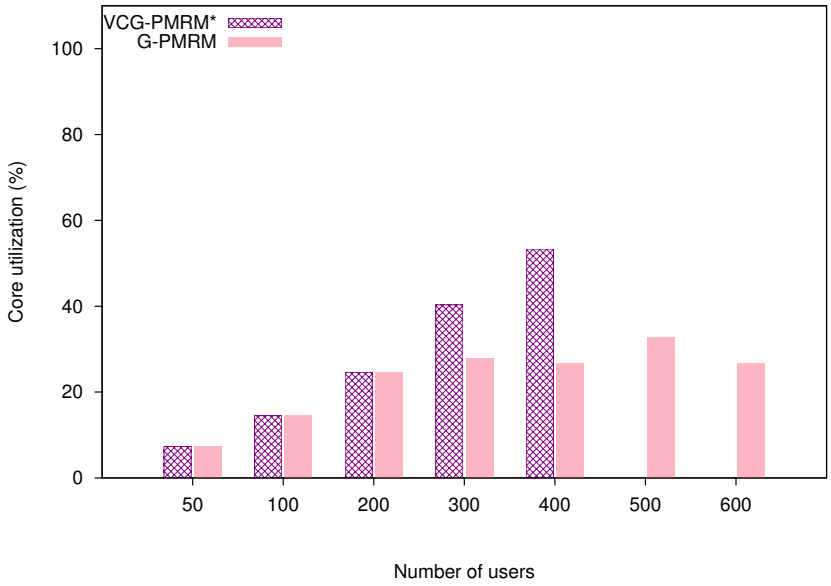


Figure 3.7: G-PMRM vs. VCG-PMRM: Core utilization

low, all users are served by both mechanisms. However, by increasing in the demand, the percentage of served users decreases. Note that a higher percentage of served users does not necessarily result in a higher revenue.

Figs. 3.7 and 3.8 show the percentage of resource utilization for both mechanisms. The

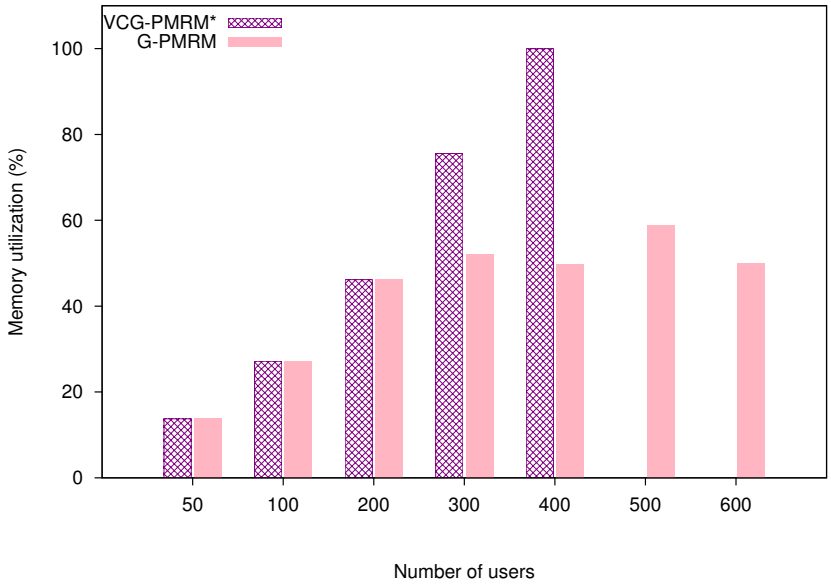


Figure 3.8: G-PMRM vs. VCG-PMRM: Memory utilization

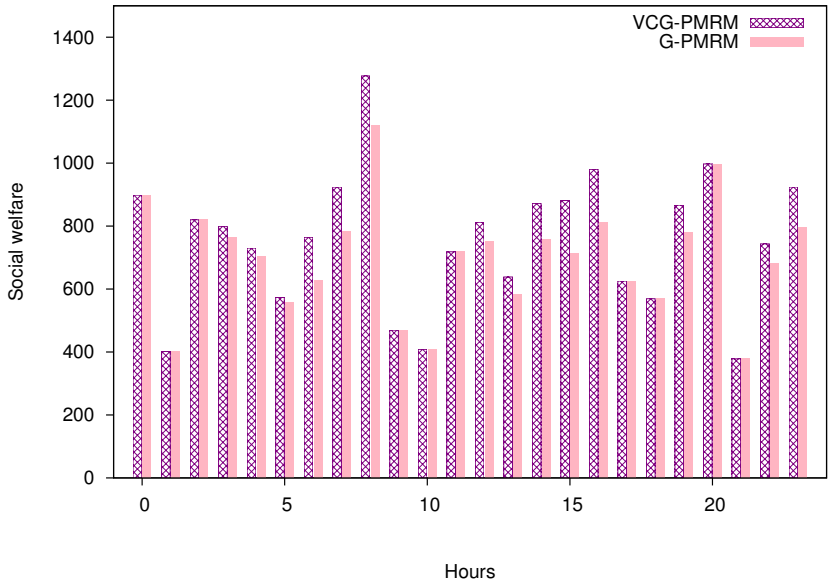


Figure 3.9: G-PMRM vs. VCG-PMRM: Social welfare over time.

percentage of core and memory utilization increases with the increase in the number of users. This is due to the fact that with more demand, the cloud provider can better utilize its resources.

We now analyze the performance of the mechanisms over an extended period of time

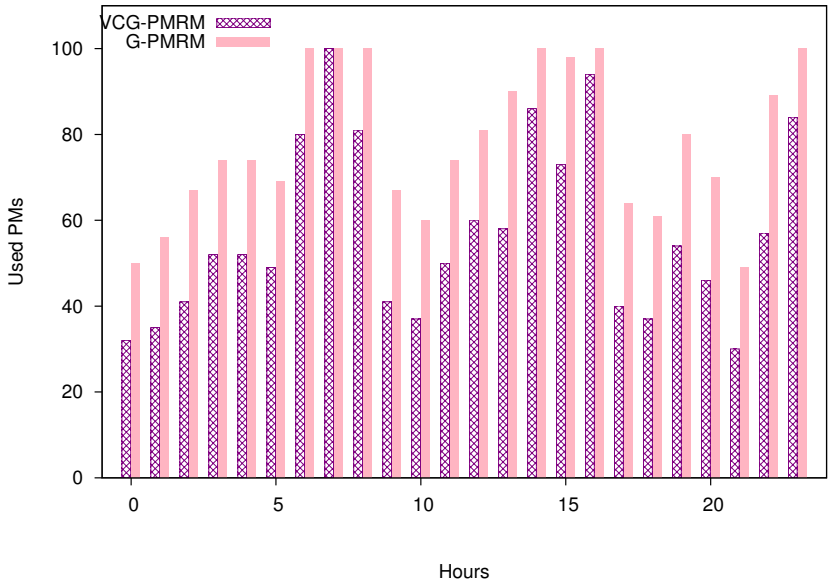


Figure 3.10: G-PMRM vs. VCG-PMRM: Used PMs over time.

of 24 hours, where users dynamically arrive and submit their requests. We consider that between 50 and 150 users arrive every hour and that each user requests resources for 1 to 4 hours. Fig. 3.9 shows the social welfare obtained by the mechanisms for each hour. The results show that G-PMRM obtains the optimal social welfare in several cases. When the number of users' requests accumulates, G-PMRM cannot always find the optimal solution. However, the optimality gap is small as guaranteed by the results of Theorem 8. Fig. 3.10 shows the percentage of PMs used in each hour. When the number of users' requests accumulates, both mechanisms use all the available PMs. However, with a decrease in the number of requests, both mechanisms can turn off several PMs to reduce the energy costs.

We also perform experiments to investigate the effect of untruthful reporting on the utility of the users. In these experiments, we consider the case with 300 users, and select one of the winning users (i.e., User A) with true request of 15 VMs of type m3.2xlarge and true valuation of 35.14. We consider User A's declarations that are different from her true request as shown in Table 3.5, where Case I represents User A true request. Fig. 3.11 shows the payment and the utility of User A for all these cases.

In case II, User A submits a request with a higher bid and G-PMRM selects User A as

Table 3.5: Different scenarios for User A's request declaration

Case	$\hat{\rho}_A$	$\hat{b}_A$	Scenario
I	$< 0, 0, 0, 15 >$	\$35.14	$\hat{\rho}_A = \rho_A$, $\hat{b}_A = b_A$
II	$< 0, 0, 0, 15 >$	\$50	$\hat{\rho}_A = \rho_A$, $\hat{b}_A > b_A$
III	$< 0, 0, 0, 15 >$	\$25	$\hat{\rho}_A = \rho_A$, $\hat{b}_A < b_A$
IV	$< 0, 0, 0, 15 >$	\$10	$\hat{\rho}_A = \rho_A$, $\hat{b}_A < b_A$
V	$< 0, 0, 0, 20 >$	\$35.14	$\hat{\rho}_A > \rho_A$, $\hat{b}_A = b_A$
VI	$< 0, 0, 0, 100 >$	\$35.14	$\hat{\rho}_A > \rho_A$, $\hat{b}_A = b_A$

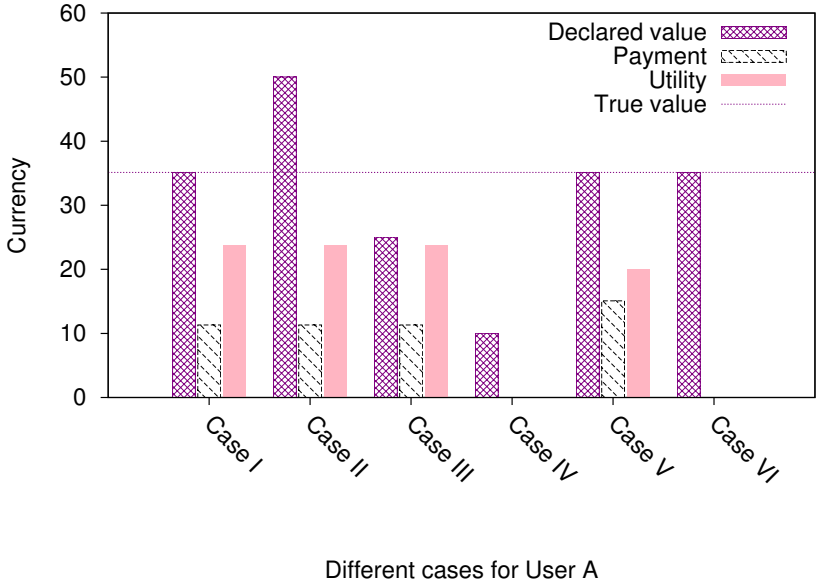


Figure 3.11: G-PMRM: Effect of untruthful declarations on the user.

a winner determining the same payment for her as in case I. In case III, User A submits a request with a lower bid, where the bid is not less than her payment determined by G-PMRM. In this case also, the user is selected as a winner, and her payment remains the same. In case IV, User A reports a lower bid than her bid in case I (the true valuation). G-PMRM does not select the user as a winner leading to zero utility for her. Therefore, User A did not increase her utility by this untruthful reporting. If User A requests a larger bundle as shown in case V, she obtains the bundle due to available capacities. However, her payment increases while her utility decreases due to requesting more VMs. If User A requests a larger bundle as in case VI, G-PMRM does not select the user as a winner leading to zero utility for her. These cases show that if any user submits an untruthful request, she can not increase her utility, that is the mechanism is strategy-proof.

From all the above results we can conclude that G-PMRM decides the allocation much faster than VCG-PMRM, achieves a social welfare close to the optimal, and obtains a higher revenue than VCG-PMRM. The performance of G-PMRM scales very well with the number of users.

3.7 Conclusion

We proposed optimal and approximate strategy-proof mechanisms for resource management in clouds in the presence of multiple PMs and multiple types of resources that give incentives to the users to reveal their true valuations for the requested bundles of VM instances. Therefore, our mechanisms do not put the burden on users to compute complex strategies of how to best interact with the mechanisms. We investigated the properties of our proposed mechanisms by performing extensive experiments. The results showed that the performance of our proposed approximation mechanism scales very well with the number of users.

CHAPTER 4: AN ONLINE MECHANISM FOR RESOURCE ALLOCATION AND PRICING

4.1 Introduction

Cloud computing is gaining more market share in the IT industry by adding flexibility on resources acquisition, enabling individuals and enterprises to pay only for the resources and services they use. Cloud providers offer their services based on the pay-as-you-go model, enabling the reduction of enterprises' capital and operational costs. One of the major problems in offering such services is designing efficient mechanisms for Virtual Machine (VM) provisioning, allocation, and pricing. Such mechanisms should consider the economic incentives of both cloud users and cloud providers in finding the market equilibrium [24]. Current cloud providers such as Amazon EC2 and Microsoft Azure employ fixed-price and auction-based mechanisms in order to provision resources in the form of VM instances and sell them to the users. The auction-based mechanisms complement the fixed-price models, potentially providing the most cost-effective option for obtaining cloud resources. Obtaining the VMs in an auction market can significantly lower users' computing costs for their jobs [1]. The auction-based mechanisms provide incentives to the users to adjust consumption patterns according to availability, price, and other factors. Existing resource allocation and pricing mechanisms are offline, and thus, they need to collect the information about all users' requests and then decide the allocation of VM instances to users and the prices they need to pay. However, cloud users request VM instances over time, thus, creating an online setting for the provisioning, allocation, and pricing problem. Therefore, cloud providers need to design online mechanisms suitable for such settings in order to provide faster services and to efficiently allocate and price their resources.

One of the challenges in designing online mechanisms is dynamic pricing. A price determination function should consider the incentives of both cloud providers and users. In doing so, it should increase revenue, facilitate healthy competition among users, and increase the efficiency of resource usage. A cloud provider may increase the price to generate

more profit. However, in a competitive environment, if the increase in the price is too high, the cloud provider may lose its potential users leading to a profit loss. On the other hand, if the cloud provider sets the price too low, it may become overwhelmed by high demand from users. Since the available capacity is limited, the cloud provider can serve a limited number of users with low price, leading to loss in both profit and reputation. The challenge is how the cloud provider should determine the price to maximize its profit in such competitive markets. Mechanism design considers the incentives of the participants when deciding the allocation and payment. In doing so, the price determination function should determine the payments of the users based on the value the users derive from the services [22]. A fundamental problem with significant economic implications is how the cloud should price its heterogeneous resources at different times under dynamic demand such that its overall profit is maximized.

We consider an online market with multiple self-interested users who are competing for cloud resources. In online settings, all users arrive and depart dynamically requiring making decisions without having information about the future. Each user name her own price for a bundle of VM instances, and specifies the amount of time the bundle must be allocated and a deadline. Each user has private information about her requested bundle, and this information is not necessarily reflected in her submitted request. This is due to the fact that the users are self-interested, and they may manipulate the system in order to maximize their utility. A key property of our proposed mechanism, called incentive-compatibility, is to give incentives to users to reveal their actual requests including the price, the VM bundle, the length of time of using the requested VM bundle, and the deadline for their requested bundles. The objective of the mechanism is to allocate cloud resources to the users who value them the most. The mechanism also calculates the price that each user must pay to the cloud provider. The allocation and pricing mechanisms used by the current cloud computing providers do not require the users to explicitly specify a length of time for using a VM at the time of submitting their requests. These cloud providers charge the users a fixed price per hour for each VM instance used. Our proposed mechanism provides more flexibility allowing the users to specify the length of time for which they would like

to acquire the bundle of VMs and a deadline. We believe that our setting provides more opportunities to providers, to optimize their operating costs and to increase their profits, as well as to users, who will be able to better express their requirements in order to maximize their utilities.

In this chapter, we design an online mechanism for the VM allocation and pricing problem in clouds in the presence of multiple types of resources (e.g., cores, memory, storage, etc.). Our proposed mechanism is online and thus, makes no assumptions about future demand and supply of VMs, which is the case in real cloud settings. Our proposed online mechanism calculates the allocation and payment as users arrive at the system and place their requests. Our proposed mechanism demonstrates the benefits of quick response, revenue maximization, and incentive compatibility which are critical when providing online cloud services.

4.1.1 Our Contribution

We address the problem of online VM provisioning, allocation, and pricing in clouds in the presence of multiple types of resources. We design an offline incentive-compatible mechanism and an online incentive-compatible mechanism for VM allocation and pricing that give incentives to the users to reveal their actual true requests. Our proposed offline mechanism is optimal given that the information on all the future requests is known a priori. However, our proposed online mechanism makes no assumptions about the future demand for VMs, which is the case in real cloud settings. Our proposed online mechanism is invoked as soon as a user places a request or some allocated resources are released and become available. The mechanism not only provisions and allocates resources dynamically, but also determines the users' payments such that the incentive-compatibility property is guaranteed. We compare the performance of the optimal mechanism with that of online mechanism. The proposed online mechanism provides very fast solutions making it suitable for execution in real-time settings. We perform extensive experiments showing that the proposed online mechanism is able to find near optimal solutions.

4.1.2 Related Work

Mechanism design [128] is a sub-field of game theory aiming at reaching systems' equilibria having desired properties such as high revenue [128]. There is a rich body of work on mechanism design considering *static systems* in which all participants are present and a one-time decision is made to find a solution [37, 41, 57]. Such systems are considered in an offline setting, whereas in online mechanism design, all participants arrive and depart dynamically, requiring making decisions without having information about the future. The problem of online mechanism design was introduced by Friedman and Parkes [44]. They proposed strategy-proof online mechanisms, where truthful revelation of a user's valuation is a dominant strategy equilibrium. For an introduction to online mechanism design, the reader is referred to Parkes [139]. Several online variants of Vickrey-Clarke-Groves (VCG) mechanisms were proposed by Gershkov and Moldovanu [46] and by Parkes and Singh [140]. These mechanisms focus on Bayesian-Nash incentive compatibility. However, these studies rely on a model of future availability, as well as future supply. Hajiaghayi et al. [55] designed online mechanisms for auctioning identical items, where users have three parameters as private information: value, arrival time, and departure time. However, they assumed that the number of users is known in advance. Hajiaghayi et al. [54] investigated online mechanisms for re-usable items in which items can be allocated to different users at different time slots. They mainly focused on unit-length requests. Porter [144] studied the problem of online scheduling of a re-usable resource in model-free setting, and characterized the monotonicity properties.

Researchers approached the problem of resource provisioning and allocation in clouds from different points of view [68, 77, 174, 186]. Jangjaimon and Tzeng [68] designed an enhanced adaptive incremental checkpointing mechanism for multithreaded applications on resource-as-a-service clouds under spot instance pricing. The objective of their approach is to reduce the expected job turn-around time and the cost. Kuo et al. [77] proposed a 3-approximation algorithm for the VM placement problem to minimize the maximum access latency. Xiao et al. [184] studied the automatic scaling problem in clouds. They

proposed a color set algorithm to decide the application placement and load distribution. Their proposed algorithm is invoked periodically, and it reduces the number of instances in order to save energy. Lee and Zomaya [81] proposed two energy-conscious task consolidation heuristics for clouds with the goal of maximizing resource utilization considering both active and idle energy consumption. Papagianni et al. [137] tackled the problem of providing a unified resource allocation framework for networked clouds with the goal of minimizing the cost of resource mapping procedures. Ghazar and Samaan [47] proposed a pricing mechanism for virtual network services to regulate the demand for their shared substrate network resources. Guazzone et al. [52] proposed a framework for dynamic management of computing resources in order to achieve suitable QoS levels and to reduce the amount of energy consumption for providing services. HoseinyFarahabady et al. [60] studied the problem of task assignment on hybrid-clouds. They proposed two approximation methods for two different cases of known and unknown running time of available tasks. More specifically, they designed a fully polynomial-time randomized approximation scheme based on a Monte Carlo sampling method for the case of unknown running time. Leslie et al. [83] proposed a framework for resource allocation and job scheduling of VMs aiming to cost efficiently execute deadline-constrained jobs. Their proposed framework ensures quality of service in terms of cost, deadline compliance and service reliability. Cao et al. [25] proposed a pricing model to maximize profit considering different factors of a cloud such as the amount of services, the workload of an application, the cost of renting, and the cost of energy consumption. In addition, they proposed a queuing model in order to find optimal configuration of a multiserver system. All of these prior works assume that the information is publicly known, and none of them considers a competitive setting, in which the requests characteristics are private to the users.

Recently, the concepts of game theory and mechanism design have been employed in the design of cloud resource management mechanisms [42, 179, 196]. Feng et al. [42] proposed a game theoretic approach considering multiple competing cloud providers. They proposed iterative price determination algorithms for cloud providers to maximize their profits when offering IaaS (Infrastructure as a Service). Zhang et al. [196] proposed a ran-

domized mechanism for VM allocation in clouds in an auction market. Their proposed mechanism is truthful in expectation and is based on a pair of primal and dual LPs (Linear Programs). It considers a different settings than ours in which the the requests from the users do not specify the duration of the time the VM bundle is requested, the arrival time, and the deadline; it only specifies the bundle of VM and its valuation. Prasad et al. [145] proposed a cloud resource procurement approach which not only automates the selection of cloud providers but also implements dynamic pricing. They proposed a strategy-proof mechanism based on VCG, a Bayesian mechanism, and an optimal mechanism for resource procurement where a user performs a reverse auction for procuring resources from cloud providers. Wang et al. [175] proposed a generalized dominant resource fairness mechanism for the multi-resource allocation problem, where there are multiple heterogeneous servers. Their proposed mechanism improves the resource utilization leading to shorter job completion times. In our previous studies, we proposed truthful mechanisms for VM allocation in clouds in periodic-time (offline) settings [96, 101, 124]. However, none of these studies consider online settings.

Online resource management in clouds has recently attracted a great deal of attention. Hua et al. [62] proposed a scalable distributed scheme in cloud data centers considering the network architecture design and data placement. Their proposed network scheme leverages the off-line precomputation to improve online cloud services. Zhang et al. [197] proposed a bandwidth cost minimization approach for uploading deferral big data to a cloud or a federation of clouds. In doing so, they designed a heuristic smoothing algorithm and an efficient distributed randomized online algorithm. Abbasi et al. [12] proposed an online algorithm to minimize operational cost of a set of geo-distributed data centers. Song et al. [161] proposed an online bin packing approach that uses virtualization technology to allocate cloud resources dynamically based on application demands. Their proposed approach supports green computing by optimizing the number of servers used. Zhao et al. [199] proposed an online algorithm for dynamic VM pricing across data-centers in a geo-distributed cloud in order to maximize the overall profit. Zhang et al. [195] proposed an online auction mechanism for resource allocation in clouds in the presence of only one type

of resources. They assumed that job lengths and bids are within known intervals. Zaman and Grosu [192] proposed a truthful online mechanism for provisioning and allocation of VM instances in clouds. However, their mechanism assumes that the cloud provider offers only one type of resources, computational resources. The current work is different from the two above-mentioned studies since it considers the existence of several resource types, being more suitable for use in real cloud settings. Note that considering one resource makes the problem NP-hard, while in our study, we tackle a much more challenging problem which is strongly NP-hard. Therefore, satisfying incentive-compatibility in our settings brings about more challenges. In addition, unlike the above-mentioned studies we do not consider any assumptions on the bids and their distributions, and thus, creating a general framework for the online setting.

4.1.3 Organization

The rest of the chapter is organized as follows. In Section 4.2, we describe the online VM allocation and pricing problem in clouds. In Section 4.3, we introduce the basic concepts of mechanism design, and present our proposed offline optimal mechanism. In Section 4.4, we present the proposed online mechanism, and characterize its properties. In Section 4.5, we evaluate the mechanisms by extensive experiments. In Section 4.6, we summarize our results.

4.2 VM Allocation and Pricing Problem

In this section, we model the online VM allocation and pricing (OVMAP) problem in the presence of multiple types of resources. A cloud provider offers R different types of resources, $\mathcal{R} = \{1, \ldots, R\}$, such as cores, memory, storage, etc. These resources are provisioned in the form of M types of VM instances $\mathcal{VM} = \{1, \ldots, M\}$ and then offered to the users. Each VM instance of type $m \in \mathcal{VM}$ has a specific amount of each type of resource $r \in \mathcal{R}$, denoted by w_{mr}. The capacity C_r for each resource $r \in \mathcal{R}$ available for allocation is limited. In Table 4.1, we show the four types of VM instances offered by

Table 4.1: VM instance types offered by Amazon EC2.

	Small $m=1$	Medium $m=2$	Large $m=3$	Extralarge $m=4$
CPU	1	2	4	8
Memory (GB)	1.7	3.75	7.5	15
Storage (GB)	160	410	850	1690

Amazon EC2 US West (Northern California) Region. If we consider that CPU represents the type 1 resource, memory, the type 2 resource, and storage, the type 3 resource, we can characterize, for example, the Medium instance ($m = 2$) by: $w_{21} = 2$, $w_{22} = 3.75$ GB, and $w_{23} = 410$ GB.

A set $\mathcal{U}$ of N users are requesting a set of VM instances for a certain amount of time in order to execute their jobs on the cloud. User i, $i \in \mathcal{U}$, requests a bundle $S_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$ of M types of VM instances, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$. In addition, she specifies a bid b_i for her requested bundle S_i. User i's request is denoted by $\theta_i = (S_i, a_i, l_i, d_i, b_i)$, where a_i is the arrival time of her request, l_i is the amount of time for which the requested bundle must be allocated, and d_i is the deadline for her job completion. For example, request $(\langle 4, 3, 1, 2 \rangle, 2, 1, 7, \$15)$ represents a user requesting 4 Small VM instances, 3 Medium VM instances, 1 Large VM instance, and 2 Extra large VM instances; the request arrives at time 2, needs 1 unit of time to execute, expires at time 7, and her bid is \$15. We denote by $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$, the total amount of each resource of type r that user i has requested.

We define $\delta_i = d_i - l_i$ as the time by which S_i must be allocated to user i in order for her job to complete its execution. If the cloud provider allocates a requested bundle, the request is never preempted. User i values her requested bundle S_i at b_i, which is the maximum price a user is willing to pay for using the requested bundle if it is allocated within time window $[a_i, \delta_i]$. The users are assumed to be *single-minded*. That means, user i desires only S_i and derives a value of b_i if she gets S_i, or any superset of it, for the specified time before its deadline, and zero value, otherwise.

The standard objective of mechanism design is to maximize welfare [129], which can help a cloud provider increase its revenue. This is due to the fact that the mechanism

allocates the VMs to the users who value them the most. The *welfare*, V, is the sum of users' valuations, $V = \sum_{i \in \mathcal{U}} b_i \cdot x_i$, where x_i, $i \in \mathcal{U}$, are decision variables defined as follows: $x_i = 1$, if bundle S_i is allocated to user i within time window $[a_i, \delta_i]$; and $x_i = 0$, otherwise. Our goal is to design an online incentive-compatible mechanism maximizing V, that is, a mechanism that solves OVMAP.

We also define the offline version of OVMAP, called VMAP, which considers that the information on all the future requests is known a priori. In order to formulate VMAP as an integer program we define the decision variables over time $t \in \mathcal{T}$ as follows:

$$X_{it} = \begin{cases} 1 & \text{if } S_i \text{ is allocated to } i \text{ at } t, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

In addition, we define indicator parameters as follows:

$$y_{it} = \begin{cases} 1 & \text{if } a_i \leq t \leq \delta_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

The feasibility of the allocation to user i is indicated by y_{it}. This indicator parameter ensures that the allocation of the requested bundle is within time window $[a_i, \delta_i]$.

We formulate the problem of offline VM allocation and pricing (VMAP) as an Integer

Program (called VMAP-IP) as follows:

$$\text{Maximize} \sum_{i \in \mathcal{U}} \sum_{t \in \mathcal{T}} b_i \cdot y_{it} \cdot X_{it} \tag{4.3}$$

Subject to:

$$\sum_{t \in \mathcal{T}} X_{it} \leq 1, \forall i \in \mathcal{U} \tag{4.4}$$

$$\sum_{i \in \mathcal{U}} \sum_{\omega=t-l_i+1}^{t} \sum_{m \in \mathcal{VM}} k_{im} w_{mr} y_{i\omega} X_{i\omega} \leq C_r,$$

$$\forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{4.5}$$

$$X_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{4.6}$$

$$y_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{4.7}$$

The objective function is to maximize welfare V, where $x_i = \sum_{t \in \mathcal{T}} y_{it} \cdot X_{it}$. Constraints (4.4) ensure that the request of each user is fulfilled at most once. Constraints (4.5) guarantee that the allocation of each resource type does not exceed the available capacity of that resource for any given time. Constraints (4.6) and (4.7) represent the integrality requirements for the decision variables and indicator parameters. These constraints force the cloud provider to provision the whole bundle of VM instances and to allocate bundles to the selected users. The VMAP problem is strongly NP-hard by a simple reduction from the multidimensional knapsack problem [71]. Note that VMAP-IP assumes that the information about all users' requests is available at the time of solving it. As a result, if solved, VMAP-IP finds the optimal allocation of cloud resources in an offline setting. However, in an online setting, we do not have the information about future requests (such as arrivals), and thus, we have to rely on online mechanisms that solve the OVMAP problem. Our goal is to design such an online incentive-compatible mechanism that solve the OVMAP problem.

4.3 Mechanism Design Framework

In this section, we first present the basic concepts of mechanism design and then propose an offline optimal mechanism.

4.3.1 Preliminaries of Mechanism Design

In general, a deterministic mechanism $\mathcal{M}$, is defined as a tuple $(\mathcal{A}, \mathcal{P})$, where $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ is the allocation function that determines which users receive their requested bundles, and $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ is the payment rule that determines the amount that each user must pay for the allocated bundles. In our model, each user $i \in \mathcal{U}$ is characterized by her actual request denoted by θ_i. Each user's request is private knowledge. The users may submit different requests from their actual (true) requests. We denote by $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$ user i's submitted request. Note that $\theta_i = (S_i, a_i, l_i, d_i, b_i)$ is user i's actual request. The valuation function $v_i(\hat{\theta}_i)$ of user i is defined as follows:

$$v_i(\hat{\theta}_i) = \begin{cases} b_i & \text{if } \hat{S}_i \text{ is allocated by } \mathcal{A} \\ & \quad \wedge (S_i \subseteq \hat{S}_i) \wedge (t_i \leq \delta_i) \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where t_i is the time at which $\hat{S}_i$ has been allocated to user i. The goal is to design incentive-compatible mechanisms that maximize the welfare V, where $V = \sum_{i \in \mathcal{U}} v_i(\hat{\theta}_i) \cdot x_i$.

We denote by $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$ the vector of requests of all users. In addition, $\hat{\boldsymbol{\theta}}_{-i}$ is the vector of all requests except user i's request (i.e., $\hat{\boldsymbol{\theta}}_{-i} = (\hat{\theta}_1, \ldots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \ldots, \hat{\theta}_N)$). The utility function of user i is *quasi-linear*, and thus, it is defined as the difference between her valuation and payment, $u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}) = v_i(\hat{\theta}_i) - \mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$, where $\mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$ is the payment for user i calculated by the mechanism using the payment rule $\mathcal{P}$.

Definition 16 (Individual rationality)**.** *A mechanism is* individually-rational *if for every user i reporting her actual request θ_i we have $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq 0$, for all other users requests $\hat{\boldsymbol{\theta}}_{-i}$.*

In other words, a mechanism is individually-rational if a truthful user can always achieve as much utility from participation as without participation. Therefore, users reporting truthfully their requests will never incur losses (i.e., negative utility) by participating in the mechanism. However, such mechanisms do not give incentives to users to report their requests truthfully. The goal of a self-interested user is to maximize her utility, and she may manipulate the mechanism by lying about her actual request. In our case, the request of a user consists of a bundle, an arrival time, an amount of time for which the requested bundle must be allocated, a deadline, and a value. As a result, a user can lie about any of these parameters in the hope to increase her utility. These manipulations may reduce the revenue of the cloud provider. Our goal is to prevent such manipulations by designing incentive-compatible mechanisms for solving OVMAP. A mechanism is *incentive-compatible* if all users have incentives to reveal their actual requests.

Definition 17 (Incentive compatibility)**.** *A mechanism $\mathcal{M}$ is* incentive-compatible *(or truthful) if for every user i, for every submitted requests of the other users $\hat{\boldsymbol{\theta}}_{-i}$, an actual request θ_i and any other submitted request $\hat{\theta}_i$ of user i, we have that $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$.*

In an incentive-compatible mechanism, truthful reporting is a dominant strategy for the users. In other words, it is in the users best interest to submit their actual request irrespective of other users requests. To design an incentive-compatible mechanism, we need to design a monotone allocation function $\mathcal{A}$, while the payment rule must be based on the critical payment [118].

For our model, we define monotonicity in terms of the following preference relation $\succeq$ on the set of requests: $\hat{\theta}'_i \succeq \hat{\theta}_i$ if $\hat{S}_i \succeq \hat{S}'_i$, $\hat{a}'_i \leq \hat{a}_i$, $\hat{l}'_i \leq \hat{l}_i$, $\hat{d}'_i \geq \hat{d}_i$, and $\hat{b}'_i \geq \hat{b}_i$ for user i. Moreover, $\hat{S}'_i \succeq \hat{S}_i$ if $\sigma'_{ir} \leq \sigma_{ir}$, $\forall r \in \mathcal{R}$. That means the request $\hat{\theta}'_i$ is more preferred than $\hat{\theta}_i$ if user i requests a smaller bundle, submits an earlier request, the bundle for a shorter time period, a later deadline, and submits a higher value. In our setting, users cannot report an earlier arrival (i.e., $\hat{a}_i \leq a_i$), a shorter length (i.e., $\hat{l}_i \leq l_i$), or a later deadline (i.e., $\hat{d}_i \geq d_i$) than their true arrival time, true length, and true deadline. There is no reason for a user to submit her request earlier than when her job is ready for execution. Declaring a shorter

length does not allow the completion of the job. Reporting a later deadline may result in getting her bundle too late to complete her job on time.

Definition 18 (Monotonicity)**.** *If a* monotone *allocation function $\mathcal{A}$ allocates the resources to user i with $\hat{\theta}_i$, then it also allocates the resources to that user with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.*

In other words, $\mathcal{A}$ is monotone if any winning user who receives her requested bundle by declaring a request $\hat{\theta}_i$ is still wining if she submits a more preferred request.

In addition to a monotone allocation function $\mathcal{A}$, any incentive-compatible mechanism $\mathcal{M}$ has a payment rule $\mathcal{P}$. To avoid manipulations and satisfy incentive-compatibility, the payment $\mathcal{P}_i$ of any user i, must be independent of her request [129]. In this setting, a payment rule that satisfies the critical payment property along with a monotone allocation function are sufficient conditions to obtain an incentive-compatible mechanism [129]. In the following, we describe the critical payment property.

Definition 19 (Critical payment)**.** *If $\mathcal{A}$ is monotone, for every θ_i, there exist a unique value b_i^c, called* critical payment*, such that $\forall \hat{\theta}_i \succeq (S_i, a_i, l_i, d_i, b_i^c)$, $\hat{\theta}_i$ is a winning declaration, and $\forall \hat{\theta}_i \prec (S_i, a_i, l_i, d_i, b_i^c)$, $\hat{\theta}_i$ is a losing declaration.*

For given requests $\hat{\boldsymbol{\theta}}_{-i}$ and allocation function $\mathcal{A}$, $\hat{\theta}_i$ is a winning declaration if $i \in \mathcal{A}(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$ (i.e., $x_i = 1$); otherwise we say that $\hat{\theta}_i$ is a losing declaration.

We define the payment rule $\mathcal{P}$ based on the critical payment as follows. $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = b_i^c$, if user i is a winning user, and $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = 0$, otherwise. A winning user is a user who is selected by the allocation function to receive her request (i.e., $x_i = 1$). We denote by b_i^c, the critical payment of user i.

In the next subsection, we incorporate our proposed VMAP-IP in the design of a Vickrey-Clarke-Groves (VCG)-based optimal mechanism which computes the allocation and payment offline.

4.3.2 Incentive-Compatible Offline Optimal Mechanism

In this section, we present a VCG-based optimal mechanism that solves VMAP, the offline version of OVMAP problem. Since the setting is offline, our proposed mechanism has all

Algorithm 11 Optimal Offline Mechanism: VCG-VMAP (**C**)

1: **for all** $i \in \mathcal{U}$ **do**
2: Collect user request $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$
3: $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_N)$
4: $(V^*, \mathbf{x}^*)$ = Solution of IP-VMAP($\hat{\boldsymbol{\theta}}$)
5: Provisions and allocates VM instances based on $\mathbf{x}^*$.
6: **for all** $i \in \mathcal{U}$ **do**
7: $(V'^*, \mathbf{x}'^*)$ = Solution of IP-VMAP($\hat{\boldsymbol{\theta}}_{-i}$)
8: $\mathcal{P}_i = \sum_{j \in \mathcal{U}, j \neq i} \hat{b}_j (x'^*_j - x^*_j)$
9: **Output:** V^*; $\mathbf{x}^*$; $\mathcal{P}$

the information about the users such as their arrival, deadline, requested time, requested bundle, etc, and thus, it finds the optimal solution. Any VCG-based mechanism [129] requires an optimal allocation algorithm implementing the allocation function $\mathcal{A}$. A VCG mechanism is defined as follows. A mechanism is a Vickrey-Clarke-Groves (VCG) mechanism if the allocation function $\mathcal{A}$ maximizes V, and the payment function $\mathcal{P}$ is defined as follows:

$$\mathcal{P}_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}) = \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} v_j(\hat{\theta}_j) - \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} v_j(\hat{\theta}_j), \forall i \in \mathcal{U}, \tag{4.9}$$

where $\sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} v_j(\hat{\theta}_j)$ is the optimal welfare that would have been obtained had user i not participated, and $\sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} v_j(\hat{\theta}_j)$ is the aggregated users' valuations except user i's.

We design a VCG-based mechanism, called VCG-VMAP, that solves the VMAP problem, by incorporating our proposed VMAP-IP and its optimal solution along with the above-mentioned VCG payment rule. The optimal offline VCG-VMAP mechanism is shown in Algorithm 11. The mechanism has as input the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$. VCG-VMAP collects all the requests (lines 1-3), and when it has all the information about the requests, it determines the optimal allocation of resources to users by solving the IP-VMAP given in Equations (4.3) to (4.7) (line 4). Then, the mechanism provisions the resources in the form of VM instances based on the requested number and types of VM instances of winning users (line 7). Finally, the mechanism determines the payment of each user (lines 6-8). In doing so, VCG-VMAP finds the allocation and welfare obtained without each user's participation (line 7). Then, the mechanism charges

each user based on the welfare obtained with and without her participation (line 8). The mechanism returns three parameters: V^*, the optimal welfare, $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_N^*)$, the optimal allocation of VM instances to the users, and $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$ their payments.

Because VCG-VMAP is designed with an optimal allocation function and uses the VCG payment rule, it is incentive-compatible [129]. However, VMAP is strongly NP-hard, and thus, the execution time of VCG-VMAP becomes prohibitive for large instances of VMAP. In addition, VCG-VMAP computes the allocation and payment offline since it has all the information about future demands. However, in a real settings this information is not available to the cloud providers and requires designing online mechanisms. In Section 4.4, we introduce our proposed online mechanism.

4.4 Online Mechanism for VM Allocation and Pricing

Our goal is to design an incentive-compatible greedy mechanism that solves the OVMAP problem in online settings.

The VM instances have R dimensions, where the R dimensions correspond to the R types of resources. Since the cloud provider provisions resources in the form of VM instances, any bundle of VMs can be seen as one R-dimensional item. Without loss of generality, we consider that the smallest item in the R-dimensional space contains one unit of each resources. This assumption does not restrict our proposed model since the resource capacities can be normalized to their units. As a result, the total volume of available items to allocate to the users is $\prod_{r \in \mathcal{R}} C_r$. In this section, we present an incentive-compatible online mechanism for the OVMAP problem, called OVMAP.

4.4.1 OVMAP Mechanism

The OVMAP mechanism is given in Algorithm 12. OVMAP is an event handler, that is, it is invoked when a new user request arrives or some allocated VM instances become available. OVMAP takes as input an Event, the current allocation set $\mathcal{A}$, and the payment set $\mathcal{P}$. An Event is either a release of resources or an arrival of a user request. In lines 1

Algorithm 12 OVMAP Mechanism (Event, $\mathcal{A}, \mathcal{P}$)

1: $t \leftarrow$ Current time
2: $\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}$, i has not been allocated$\}$
3: $\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}$, ($i$ has been allocated) $\wedge$
(its job has not finished yet)$\}$
4: **for all** $i \in \mathcal{U}$ **do**
5: **for all** $r \in \mathcal{R}$ **do**
6: $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$
7: **for all** $r \in \mathcal{R}$ **do**
8: $C_r^t \leftarrow C_r - \sum_{i | \hat{\theta}_i \in \tilde{\mathcal{Q}}^t} \sigma_{ir}$
9: $\mathcal{C}^t \leftarrow (C_1^t, \ldots, C_R^t)$; vector of resource capacities at time t
10: **if** $\mathcal{Q}^t = \emptyset$ or $\mathcal{C}^t = \mathbf{0}$ **then**
11: **return**
12: $\mathcal{A}^t \leftarrow$ OVMAP-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$
13: $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}^t$
14: $\mathcal{P} \leftarrow \mathcal{P} \cup \{\hat{b}_i | (\hat{\theta}_i, t) \in \mathcal{A}^t\}$
15: $\mathcal{P} \leftarrow$ OVMAP-PAY$(t, \mathcal{Q}^t, \mathcal{A}, \mathcal{P}, \mathcal{C}^t)$
16: **return** $\mathcal{A}, \mathcal{P}$

to 8, OVMAP sets the current time to t and initializes four variables as follows:

$\mathcal{Q}^t$: the set of requests of the users that have not been allocated. Formally,

$\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}, t \leq \hat{\delta}_i \wedge \nexists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A}\}$;

$\tilde{\mathcal{Q}}^t$: the set of requests of the users that have been allocated and their jobs have not finished yet. Formally,

$\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U} \wedge \exists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A} \wedge t_i + \hat{l}_i > t\}$;

σ_{ir}: the amount of each resource of type r requested by user i; and,

C_r^t: the available capacity of the resource r at time t.

The mechanism stores the resource capacities at time t as a vector $\mathcal{C}^t$ (line 9). Then, it proceeds only if resources and requests are available. OVMAP determines the allocation by calling OVMAP-ALLOC. The allocation function OVMAP-ALLOC returns $\mathcal{A}^t$, the set of users who would receive their requested bundles at time t (line 12). The mechanism then updates the overall allocation set $\mathcal{A}$ using the newly determined set $\mathcal{A}^t$. Then, the mechanism determines the payment of users. The payment of users in $\mathcal{A}^t$ are inserted into

Algorithm 13 OVMAP-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$

1: $\mathcal{A}^t \leftarrow \emptyset$
2: **for all** $i|\hat{\theta}_i \in \mathcal{Q}^t$ **do**
3: $\quad f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$
4: Sort all $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of f_i
5: **for all** $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of f_i **do**
6: $\quad \hat{\mathcal{C}} = \mathcal{C}^t$
7: $\quad flag \leftarrow$ `TRUE`
8: $\quad$ **for all** $r \in \mathcal{R}$ **do**
9: $\quad\quad \hat{C}_r = \hat{C}_r - \sigma_{ir}$
10: $\quad\quad$ **if** $\hat{C}_r < 0$ **then**
11: $\quad\quad\quad flag \leftarrow$ `FALSE`
12: $\quad\quad\quad$ **break**;
13: $\quad$ **if** $flag$ **then**
14: $\quad\quad \mathcal{C}^t = \hat{\mathcal{C}}$
15: $\quad\quad \mathcal{A}^t \leftarrow \mathcal{A}^t \cup (\hat{\theta}_i, t)$
16: **Output:** $\mathcal{A}^t$

the payment set with $\mathcal{P}_i = \hat{b}_i$ as their initial payment (line 14). The payment function OVMAP-PAY returns updated set $\mathcal{P}$ containing the payment of users at time t (line 15). The payment of user i is going to be updated several times until $t = \delta_i$, i.e., until the time instance the user must obtain the requested bundle. OVMAP-PAY calculates the payments for these users and updates the payment set $\mathcal{P}$.

4.4.2 Allocation algorithm of OVMAP

The allocation algorithm OVMAP-ALLOC is given in Algorithm 13. We define a metric called the *bid density*. OVMAP-ALLOC algorithm allocates the VM instances to users in decreasing order of their bid densities. OVMAP-ALLOC considers the setting in which a set $\mathcal{U}$ of N users are requesting a heterogeneous set of VM instances for *any length* of time in order to execute their applications/jobs on the cloud. It also considers a continuous-time model such that $t \in [0, T]$. Note that the request time length for any user i is $\hat{l}_i \geq 1$. The bid density is defined as follows:

$$f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}} \tag{4.10}$$

The bid of user i for a bundle of VM instances for time $\hat{l}_i$ can be interpreted as requesting a hyper-rectangle with volume $\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$ in the $(R+1)$-dimensional space defined by the R resource types and the time. User i values this volume at $\hat{b}_i$, if allocated. Hence, f_i represents how much user i values one unit of volume from the $(R+1)$-dimensional space.

OVMAP-ALLOC sorts all requests in non-increasing order of bid densities, f_i (line 4). Then the algorithm allocates bundles requested by the sorted users in $\mathcal{Q}^t$ while resources last (lines 5-15). OVMAP-ALLOC checks if it can fulfill the request of user i (lines 8-12). If there are not enough resources, user i will not be selected, and her request will be rejected after the current time passes δ_i (by removing user i from $\mathcal{Q}^t$). If there are enough resources, user i will be allocated (line 15) and the amount of available resources will be updated (line 14).

The mechanism uses the non-increasing order of bid densities for allocation because the cloud provider is interested in users who want to pay more per unit of their resources per unit of time. OVMAP-ALLOC tries to maximize the sum of the reported values of the users who get their requested bundles. Finally, OVMAP-ALLOC returns the set $\mathcal{A}^t$ of users who are selected for allocation at time t.

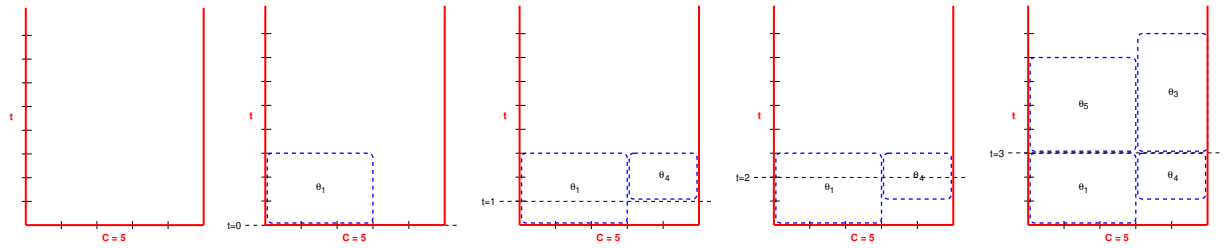
4.4.3 Payment function of OVMAP

The payment function OVMAP-PAY is given in Algorithm 14. This function calculates the *critical payment* of each user i if her requested bundle is allocated at t. The critical payment of user i is the minimum value that she must report to get her requested bundle at time t. OVMAP-PAY determines the set $\mathcal{W}$ of requests of users who are allocated and still feasible for allocation at t (line 1). Then, it determines the set $\hat{\mathcal{Q}}$ of requests of users who are allocated or not allocated at t (line 2). OVMAP-PAY calculates f_i for all users in $\hat{\mathcal{Q}}$ (lines 3-4). Then, OVMAP-PAY determines the payment for all users that have been allocated at time t (lines 5-17). The payment of user i is calculated based on the critical value payment. To determine the critical payment, we eliminate user i from the system, add back to C^r the resources allocated to user i, and identify a losing user that becomes a winner because of user i elimination. The value reported by this losing

Algorithm 14 OVMAP-PAY$(t, \mathcal{Q}^t, \mathcal{A}, \mathcal{P}, \mathcal{C}^t)$

1: $\mathcal{W} = \{\hat{\theta}_i | \exists t' \leq t : (\hat{\theta}_i, t') \in \mathcal{A} \wedge t \leq \hat{\delta}_i\}$
2: $\hat{\mathcal{Q}} = \mathcal{Q}^t \cup \mathcal{W}$
3: **for all** $i | \hat{\theta}_i \in \hat{\mathcal{Q}}$ **do**
4: $\quad f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$
5: **for all** $\hat{\theta}_i \in \mathcal{W}$ in non-increasing order of f_i **do**
6: $\quad \hat{\mathcal{C}} \leftarrow \mathcal{C}^t$
7: $\quad$ **for all** $r \in \mathcal{R}$ **do**
8: $\quad\quad \hat{C}_r = \hat{C}_r + \sigma_{ir}$
9: $\quad q = -1;$
10: $\quad \bar{\mathcal{A}} \leftarrow$ OVMAP-ALLOC$(t, \mathcal{Q}^t \setminus \hat{\theta}_i, \hat{\mathcal{C}})$
11: $\quad$ **for all** $\hat{\theta}_j \in \mathcal{Q}^t \cap \{\hat{\theta}_j | (\hat{\theta}_j, t) \notin \mathcal{A}^t \wedge (\hat{\theta}_j, t) \in \bar{\mathcal{A}}\}$
$\quad\quad\quad$ in non-increasing order of f_j, where $f_j < f_i$ **do**
12: $\quad\quad q = j;$
13: $\quad\quad$ **break**;
14: $\quad$ **if** q **then**
15: $\quad\quad \mathcal{P}_i \leftarrow \min(f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}, \mathcal{P}_i)$
16: $\quad$ **else**
17: $\quad\quad \mathcal{P}_i \leftarrow 0$
18: **Output:** $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

user is the critical value of user i. Thus, only the resources allocated to user i are placed back into C^r (lines 7-8). Then, it calls the allocation algorithm, OVMAP-ALLOC, without considering the participation of user i (line 10). Then, OVMAP-PAY tries to find a user j who had not been allocated at t when user i participated, and would have been allocated at t if user i did not participate (lines 11-17). If OVMAP-PAY finds such a user, it stores her index q (line 12), and it determines the payment of user i based on the density of user q (line 15); otherwise user i pays 0 (line 17). In other words, the payment of user i is calculated by multiplying $\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$ with the highest density among losing users, (i.e., that of user q), who would win if user i would not participate. This is the minimum value to be reported by user i such that she gets her requested bundle. Finally, the set $\mathcal{P}^t$ is returned to the mechanism.



(a) Initial state (b) $t = 0$ (c) $t = 1$ (d) $t = 2$ (e) $t = 3$

Figure 4.1: Execution of OVMAP

4.4.4 Example of OVMAP Execution

We show the execution of the mechanism by considering a setting in which the users bid as shown in Table 4.2. For simplicity, we consider $R = 1$, that is only one resource type is available (e.g., core). As a result, σ_{i1} is the total amount of resources that user i has requested. For example, user 1's bid $\hat{\theta}_1$, contains the following information: her requested resource is $\hat{\sigma}_{11} = 3$, she submits her bid at $\hat{a}_1 = 0$, she requests the bundle for $\hat{l}_1 = 3$ time units, her deadline is $\hat{d}_1 = 5$, and she values the allocation of the bundle for the entire time at $\hat{b}_1 = 5$. We also show for each user, the value of $\hat{\delta}_i = \hat{d}_i - \hat{l}_i$, the time by which the bundle must be allocated to meet the deadline, and $f_i = \frac{\hat{b}_i}{\hat{l}_i \times \hat{\sigma}_{i1}}$, the bid density.

We assume that the available capacity of resource 1 for allocation is 5 units. We show the execution of OVMAP for this setting in Figure 4.1 and Table 4.3. In Figure 4.1a, we show the initial state of the system, in Figure 4.1b, we show the system state at time $t = 0$, and continue with $t = 1, 2, 3$ in the subsequent figures. We also show the values of C_1^t and sets of $\mathcal{Q}^t$, $\tilde{\mathcal{Q}}^t$, $\mathcal{A}$, and $\mathcal{P}$ for each of the above time instances as a time diagram in

Table 4.2: User bids

$\hat{\theta}_i$	$\hat{\sigma}_{i1}$	$\hat{a}_i$	$\hat{l}_i$	$\hat{d}_i$	$\hat{b}_i$	$\hat{\delta}_i$	f_i
$\hat{\theta}_1$	3	0	3	5	5	2	0.56
$\hat{\theta}_2$	3	0	3	4	4	1	0.44
$\hat{\theta}_3$	2	1	5	8	6	3	0.60
$\hat{\theta}_4$	2	1	2	5	3	3	0.75
$\hat{\theta}_5$	3	3	4	9	8	5	0.67
$\hat{\theta}_6$	3	3	6	10	9	4	0.50

Table 4.3: Execution of OVMAP

t	$t=0\_$	$t=0$	$t=1$	$t=2$	$t=3$
$\mathcal{Q}^t$	$\emptyset$	$\{\hat{\theta}_1, \hat{\theta}_2\}$	$\{\hat{\theta}_4, \hat{\theta}_3, \hat{\theta}_2\}$	$\{\hat{\theta}_3\}$	$\{\hat{\theta}_5, \hat{\theta}_3, \hat{\theta}_6\}$
$\tilde{\mathcal{Q}}^t$	$\emptyset$	$\{\hat{\theta}_1\}$	$\{\hat{\theta}_4, \hat{\theta}_1\}$	$\{\hat{\theta}_4, \hat{\theta}_1\}$	$\{\hat{\theta}_5, \hat{\theta}_3\}$
C_1^t	5	2	0	0	0
$\mathcal{A}$	$\emptyset$	$\{(\hat{\theta}_1, 0)\}$	$\{(\hat{\theta}_1, 0), (\hat{\theta}_4, 1)\}$	$\{(\hat{\theta}_1, 0), (\hat{\theta}_4, 1)\}$	$\{(\hat{\theta}_1, 0), (\hat{\theta}_4, 1), (\hat{\theta}_5, 3), (\hat{\theta}_3, 3)\}$
$\mathcal{P}$	$\emptyset$	$(4, -)$	$(4, -, -, 2.4)$	$(4, 0, -, 2.4)$	$(4, 0, 0, 2.4, 6, -)$

Table 4.3. As a reminder, $\mathcal{Q}^t$ is the set of bids of users that participate at time t, $\tilde{\mathcal{Q}}^t$ is the set of bids of users that are holding some resources at time t (including those who win their bids at time t), C_1^t is the amount of resources available after allocation at time t, and $\mathcal{A}$ and $\mathcal{P}$ are the allocation and payment sets.

In the second column of Table 4.3, we show the initial system state and the subsequent columns represent the state of the system at time $t = 0, 1, 2, 3$, respectively. Figure 4.1a shows the initial state, where all resources are available and there are no outstanding bids. In column 2 of Table 4.3, we see that all sets are empty and $C_1^{0-} = 5$, since all resources are available for allocation. Users 1 and 2 submit their bids at $t = 0$ and hence OVMAP is invoked. Now, $\mathcal{Q}^t = \{\hat{\theta}_1, \hat{\theta}_2\}$, since both users will participate in the mechanism. As shown in Table 4.2, $f_1 > f_2$, therefore user 1 is allocated a bundle of size $\hat{\sigma}_{11} = 3$. User 2's request ($\hat{\sigma}_{21} = 3$) cannot be satisfied by the remaining resources ($C_1^t = 2$), thus $\hat{\theta}_2$ remains in set $\mathcal{Q}^t$. $\hat{\theta}_1$ is included in the set $\tilde{\mathcal{Q}}^t$, since user 1 is now receiving some resources. Finally, $(\hat{\theta}_1, 0)$ is added to set $\mathcal{A}$ and $(\hat{\theta}_1, 4)$ is added to set $\mathcal{A}$, since the value of the payment for user 1 determined in line 15 of OVMAP-PAY is $\mathcal{P}_1^t = f_2 \cdot \hat{l}_1 \cdot \hat{\sigma}_{11} = 4$.

At time $t = 1$, users 3 and 4 submit their bids, and their bids are included in the set $\mathcal{Q}^t$. User 4 has the highest bid density (i.e., $f_4 > f_3 > f_2$) and $\hat{\sigma}_{41} \leq C_1^t$, therefore user 4 obtains the requested bundle. At this time, should user 4 not participate, user 3 would have received her requested bundle, therefore $(\hat{\theta}_4, 2.4)$ is added to the set $\mathcal{P}$, where the payment of 2.4 is the product $f_3 \cdot \hat{l}_4 \cdot \hat{\sigma}_{41}$. The payment for user 1 does not change, since user 2 would still obtain her required bundle should user 1 not participate. Figure 4.1c shows the allocation.

At $t = 2$, $C_1^t = 0$, thus OVMAP is not invoked. However, user 2's deadline for allocation

$\delta_2 = 1$ has passed and she leaves the system. Since user 2 did not obtained her bundle, her final payment is zero. Figure 4.1d shows the allocation of resources at $t = 2$.

At $t = 3$, both users 1 and 4 complete their jobs and bids $\hat{\theta}_5$ and $\hat{\theta}_6$ are submitted. User 3's request is still not allocated. According to the ordering on the bid densities, users 5 and 3 obtain their requested bundles. User 3's payment is zero, since the remaining user 6 would not obtain her bundle even if user 3 would have not participated. In a scenario with reserve price, user 3's payment will be set to the reserve price. Since $\delta_1 = 2$ has passed, user 1's payment will not change. We show the outcome in Figure 4.1e. The process continues like this, as more users submit their requests.

To show the importance of incentive compatibility, we consider the following small example with two users, i and j, arriving at the same time and competing for a unit of resource. The true valuations of users i and j for the unit of resource are \$5 and \$3, respectively. We consider a scenario in which the cloud provider implements a fist-price type auction to allocate the resources. That is, the user with the highest bid wins and pays the price she bids. If the users bid truthfully (bid the same as their valuations), user i wins and pays \$5. The revenue of the provider is \$5. Since user i wants to maximize her utility (decrease her payment), she may misreport. If she submits \$3.01 as her bid, she still wins, and she pays \$3.01. However, the revenue of the provider reduces from \$5 to \$3.01. As a result, user i can benefit by missreporting, thus obtaining a higher utility. If user i submits \$2 as her bid, she will not win. In this scenario, a user needs to know how other users bid in order to strategize on how to bid to maximize her utility. Now we consider a scenario using our proposed mechanism. In the case that users i and j submit their actual true bids, our mechanism selects user i as the winning user, and charge her \$3 based on the critical payment (bid of the losing user). The revenue of the provider is \$3. If user i misreports and submits \$4 as her bid, she still wins. However, she still pays \$3, and the revenue of the provider does not change. As a result, user i cannot benefit by missreporting and change the provider revenue. This scenario shows that a user cannot change the revenue of the cloud provider by missreporting her true valuations. This leads to more stable revenues for the cloud provider. Another benefit is that the user does not need to strategize since she

will maximize her utility only by reporting her true valuation.

4.4.5 Properties of OVMAP

In this section, we investigate the properties of OVMAP. We first show that the OVMAP mechanism is *individually rational* (i.e., truthful users will never incur a loss).

Theorem 14. *OVMAP mechanism is individually rational.*

Proof. We consider user i as a winning user. We need to prove that if user i reports her true request then her utility is non-negative. This can be easily seen from the structure of the OVMAP mechanism. In line 15 of Algorithm 4, the payment for user i is set to $\mathcal{P}_i = f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$, where user q is the user who would have won if user i did not participate. Since user q appears after user i in the decreasing order of the density metric, we have, $f_q \leq f_i$, and thus, OVMAP-PAY always computes a payment $\mathcal{P}_i \leq b_i$. As a result, the utility of user i (i.e., $u_i = b_i - \mathcal{P}_i \geq 0$) is non-negative, and she never incurs a loss. In addition, a truthful user who does not win is not incurring a loss since she obtains 0 utility. This proves the individual-rationality of OVMAP mechanism. □

We now prove that the OVMAP mechanism is incentive-compatible. In order to prove that the mechanism is incentive-compatible, we need to show that the allocation algorithm is monotone, and the payment function is based on the critical payment.

Theorem 15. *OVMAP mechanism is incentive-compatible.*

Proof. We first show that the allocation algorithm OVMAP-ALLOC is monotone. If user i wins by reporting $\hat{\theta}_i$, then she will also win if she reports a more preferred request $\hat{\theta}'_i \geq \hat{\theta}_i$. Clearly, if user i reports $\hat{b}'_i \geq \hat{b}_i$, her bid $\hat{\theta}'_i$ will be allocated if $\hat{\theta}_i$ is also allocated. Similarly, if a user gets the allocation by reporting $\hat{d}_i$, she will also get it by reporting $\hat{d}'_i \geq \hat{d}_i$. Similar reasoning applies for the other parameters in the request of the user.

We now prove that the payment function implemented by OVMAP-PAY is based on the critical payment. In doing so, we need to show that $\mathcal{P}_i$ determined by OVMAP-PAY is the minimum value that user i must report to get the allocation. User i's payment is

$\mathcal{P}_i = f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}$ (line 15), where q is the index of user q appearing after user i based on the non-increasing order of the bid density (line 11), and she would have won if user i did not participate. We consider that user i submits a lower value $\hat{b}'_i < \mathcal{P}_i$. User i's new bid density is $f'_i = \frac{\hat{b}'_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}} < \frac{\mathcal{P}_i}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$. By replacing $\mathcal{P}_i$, we have $f'_i < \frac{f_q \cdot \hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}{\hat{l}_i \cdot \prod_{r \in \mathcal{R}} \sigma_{ir}}$. Thus, we have $f'_i < f_q$, that is, user i will appear after user q who did not win. As a result, if user i reports a bid below the minimum value (i.e., $\mathcal{P}_i$), she loses; otherwise she wins. This unique value is the critical payment for user i. This, together with the fact that losing users pay zero, show that the payment function implemented by OVMAP is the critical payment.

Since the payment is the critical payment and the allocation function is monotone, it follows from Parkes [139] that OVMAP is incentive-compatible. □

Theorem 16. *The time complexity of OVMAP mechanism is polynomial.*

Proof. The time complexity of OVMAP-ALLOC is $O(N(\log N + MR))$. This is because sorting the requests requires $O(N \log N)$, while checking the feasibility of the allocation for each user requires $O(NMR)$. Similar reasoning applies to OVMAP-PAY. As a result, the time complexity of OVMAP mechanism is polynomial. □

4.5 Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of our proposed online mechanism, and the offline optimal VCG-VMAP mechanism. For the VCG-VMAP mechanism, we use the CPLEX 12 solver [7] to solve the VMAP problem optimally. The data that drives our experiments consists of six workload logs from the Grid Workloads Archive [5] and the Parallel Workloads Archive [8]. The mechanisms are implemented in C++ and the experiments are conducted on AMD 2.4GHz Dual Proc Dual Core nodes with 16GB RAM which are part of the university grid system. In this section, we describe the experimental setup and analyze the experimental results.

Table 4.4: Statistics of workload logs.

Logfile	Avg jobs per hour	Range of CPU	Range of memory (MB)	Range of Storage (MB)	Available CPUs	Memory Capacity (MB)	Storage Capacity (MB)	Avg CPU per job	Avg memory per job (MB)	Avg storage per job (MB)
GWA-T-1 DAS-2	81	[1-128]	[1-4,295]	[10-51,070]	50	100	100	4.37	46.96	43.95
GWA-T-3 NorduGrid	34	1	[1-2,147]	[10-1,053,072]	24	1,400	50,000	1	595.6	93,888.77
GWA-T-4 AuverGrid	33	1	[1.7-3,668]	[10-259,316]	7	8,800	640,000	1	374.36	27,805.86
GWA-T-10 SHARCNET	147	[1-3000]	[1-32,021]	[10-2,087,029]	85	2,000	1,000	2.9	94.49	39.43
METACENTRUM-2009-2	42	[1-60]	[1-61,538]	[10-2,592,130]	44	100	20,000	1.55	325.14	21,189.11
PIK-IPLEX-2009-1	36	[1-2560]	[1-29,360]	[10-4,815,007]	88	89,000	4,700	12.15	3,528.22	18,716.06

4.5.1 Experimental Setup

Since real users request data have not been publicly released by cloud providers yet, we rely on well studied and standardized workloads from both the Grid Workloads Archive [5] and the Parallel Workloads Archive [8]. We selected the following six logs based on the availability of both recorded CPU and memory requests/usage: i) DAS-2 traces from a research grid at the Advanced School for Computing and Imaging in Netherlands; ii) NorduGrid traces from the NorduGrid system; iii) AuverGrid traces from the AuverGrid system; iv) SHARCNET traces from SHARCNET clusters installed at several academic institutions in Ontario, Canada. v) MetaCentrum from the national grid of the Czech republic; vi) IBM iDataPlex Cluster log from the Potsdam Institute for Climate Impact Research (PIK) in Germany. In our experiments, a user request is represented by a job in a log. We present statistics of the logs in Table 4.4.

Each log represents a series of requests, where the users arrive over time, and they can submit their requests to a cloud provider. The following fields of the log files are extracted to represent different features of the users' requests. (1) JobID: the user's identifier; (2) SubmitTime: the arrival time of the request; (3) RunTime: the amount time for which the requested bundle must be allocated; (4) ReqNProcs: the requested number of processors; (5) Used Memory: the requested amount of memory. Since the amount of storage usage was

not recorded in the workloads, we generate the requested storage as shown in Table 4.4. We consider these resource usage as values for the requested σ_{ir}, the amount of each resource of type r requested by user i, where i is a job in a log, and r is a type of resource. We generate a random number b_i between 1 and 10 to represent user i's bid. For a deadline of a request, we choose a random number between 3 to 6 times the job's runtime. We use the job's runtime as the requested length of the job. We select 100 hours of the logs containing 706, 842, 1523, 1865, 677, and 416 requests for the afore-mentioned logs, respectively.

4.5.2 Analysis of Results

We compare the performance of OVMAP and VCG-VMAP for different workloads. For each workload, we record the execution time, the welfare, the revenue, the percent of users served and the utilization for each mechanism. Users served is the number of users who received their requests for their entire requested time. The utilization of each resource type is calculated as the percentage of allocated resource out of the total capacity of that resource over the entire time. We now present the results obtained by OVMAP for the selected logs.

We analyze the performance of OVMAP and VCG-VMAP in terms of welfare, execution time, the percent of users served, resource utilization, and revenue. In this case, users are requesting a heterogeneous set of VM instances for a length of time. The optimal mechanism, VCG-VMAP, could not find the solutions even after 72 hours for three out of the six logs. This is due to the fact that the problem becomes more complex for different job lengths, higher number of requests, and greater available capacity. Fig. 4.2 shows the welfare achieved by the mechanisms. VCG-VMAP is not able to determine the allocation for GWA-T-3 NorduGrid, GWA-T-4 AuverGrid, and GWA-T-10 SHARCNET in feasible time, and thus, there are no bars in the plots for those cases. For the remaining logs, the results show that OVMAP obtains a welfare very close to that obtained by the optimal VCG-VMAP mechanism. On average the optimality gap is 3.7%. For example, OVMAP and VCG-VMAP obtain welfares of 1233.57 and 1274.25 for the METACENTRUM-2009-2 log, respectively, leading to a 3.19% optimality gap. Such results are very promising given

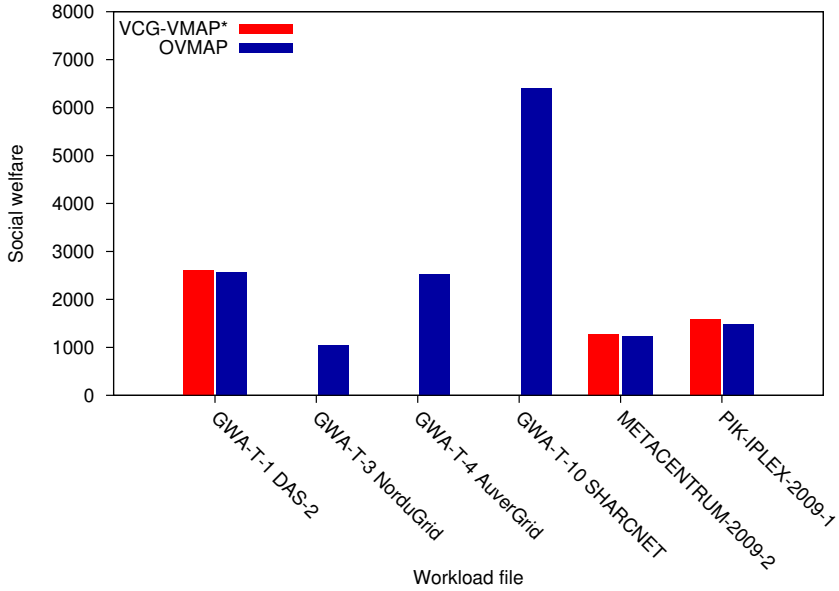


Figure 4.2: OVMAP vs. VCG-VMAP: Welfare

the fact that OVMAP is an online mechanism which does not have any information about future demand. However, VCG-VMAP is an offline mechanism and has all the information available a priori. Fig. 4.3 shows the execution times of the mechanisms on a logarithmic scale. As we expected from the time complexity of the mechanism, the execution time of OVMAP is very small. However, the execution time of the optimal offline mechanism, VCG-VMAP, is more than six order of magnitudes greater than that of OVMAP for each of the logs. Note that the online setting requires mechanisms with very small execution times. Since OVMAP obtains close to optimal welfare and is very fast it is very suitable for solving the allocation and pricing problem is online settings. We measured the execution time of the mechanism for processing each of the requests from the traces and calculated its average. The average execution time of the mechanism for processing a request is 1.27 microseconds. This shows that the mechanism is very fast and can be used in online settings. Since for each request the system will need to instantiate a VM to serve the request, the total time required to process and serve a request is given by the sum of the time to instantiate a VM and the time to run the mechanism. Since the time to instantiate a VM is in the order of tens of seconds, the contribution of the mechanism to the total time

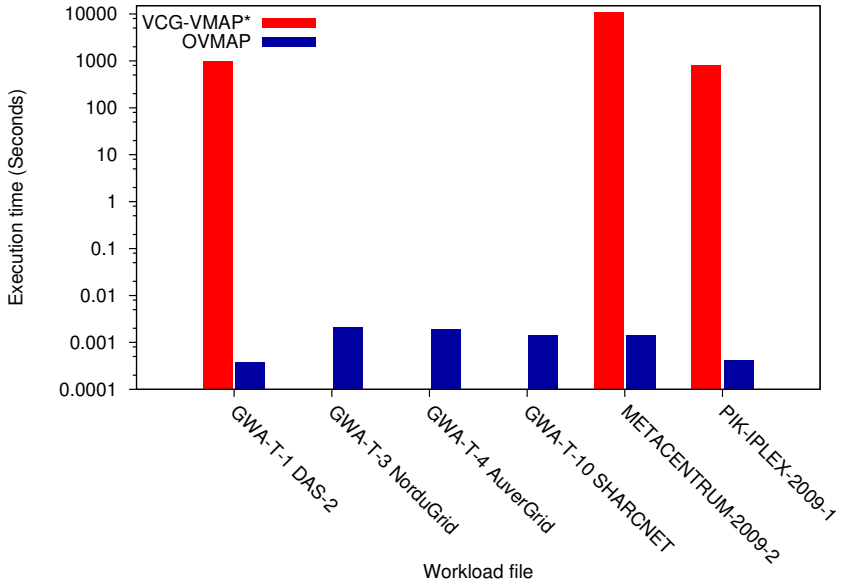


Figure 4.3: OVMAP vs. VCG-VMAP: Execution time

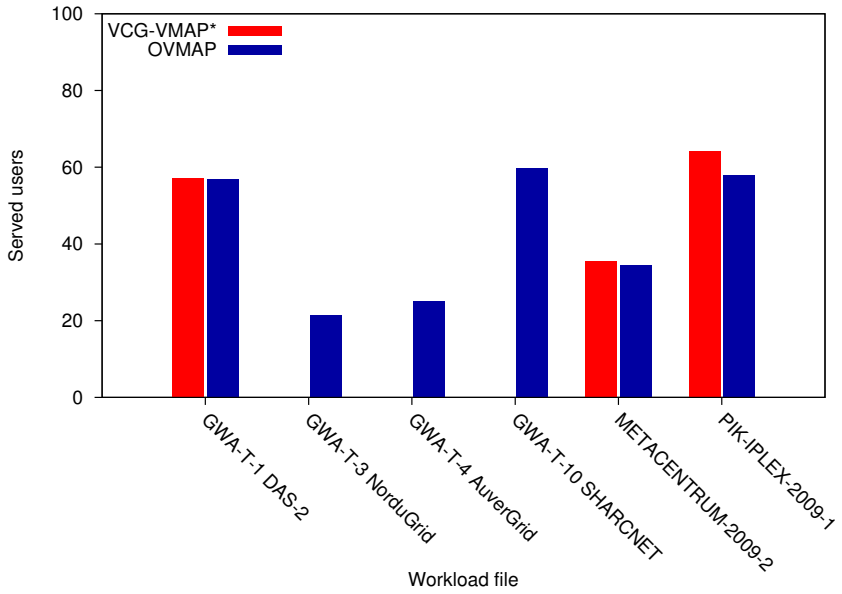


Figure 4.4: OVMAP vs. VCG-VMAP: Users served

to process a request is negligible. The total time to process a request will basically impose an upper bound on the arrival rate of requests. However, since not all of the requests will be allocated (no VM instantiated) the upper bound on the arrival rate will be much grater than the bound imposed by the total time to process a request.

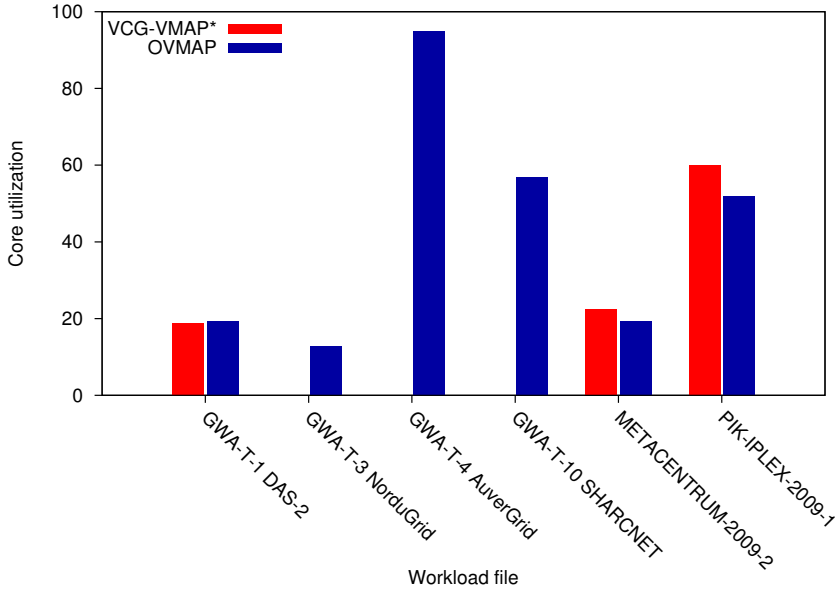


Figure 4.5: OVMAP vs. VCG-VMAP: Core utilization

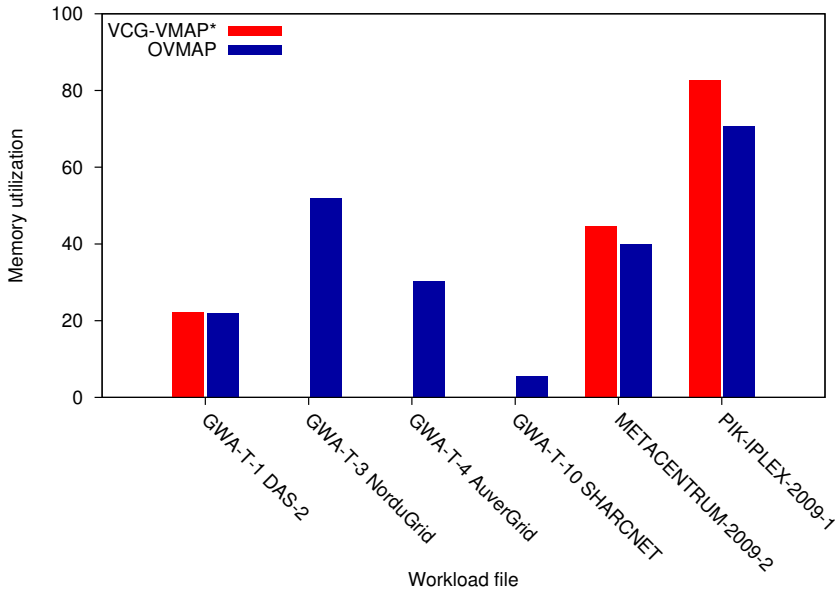


Figure 4.6: OVMAP vs. VCG-VMAP: Memory utilization

Fig. 4.4 shows the percentage of served users for the mechanisms. The percentage of served users obtained by OVMAP is very close to that of VCG-VMAP due to its close to optimal solution. This is due to the fact that the solution determined by OVMAP is very close to the optimal solution. Figs. 4.5 to 4.7 show the utilization of cores, memory

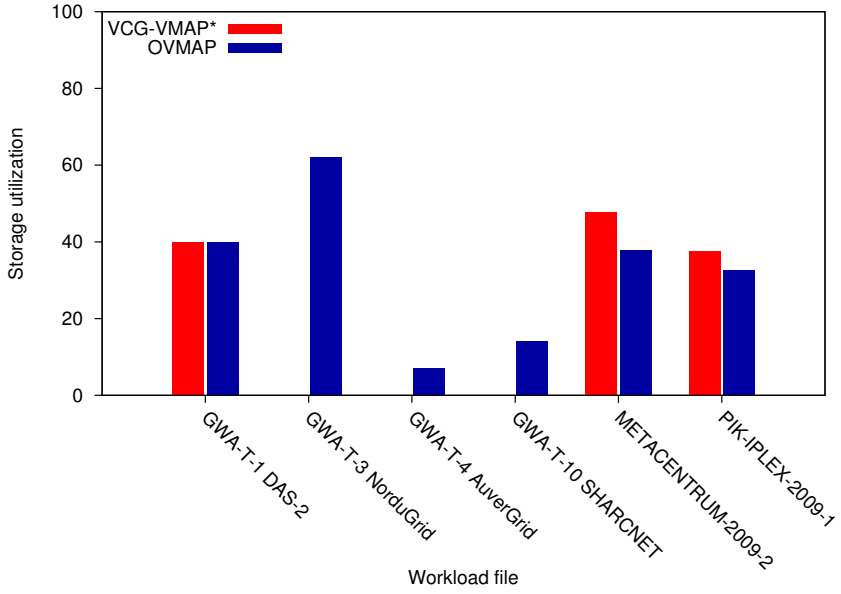


Figure 4.7: OVMAP vs. VCG-VMAP: Storage utilization

and storage, respectively. Note that a higher utilization does not show the effectiveness of the mechanisms. The objective of all the mechanisms is maximizing the welfare not the utilization of the resources. Fig. 4.8 shows the revenue achieved by the cloud provider when using the mechanisms. OVMAP is able to obtain a higher revenue than that of the VCG-VMAP.

For example, for log GWA-T-1 DAS-2, OVMAP and VCG-VMAP obtain total revenues of \$916.25 and \$783.67, respectively, corresponding to 16.91% higher revenue using OVMAP. Note that the VCG-VMAP is optimal in terms of welfare and not the revenue. This is due to the fact that VCG-VMAP fulfills more requests (i.e., more users are allocated), that is, it accepts more bids. Accepting more bids, reduces the price charged to users and implicitly the revenue.

From the above results we can conclude that OVMAP decides the allocation and pricing much faster than VCG-VMAP and achieves a welfare closer to the optimal. As a result, OVMAP is suitable for making allocation decisions and price determination in real-time.

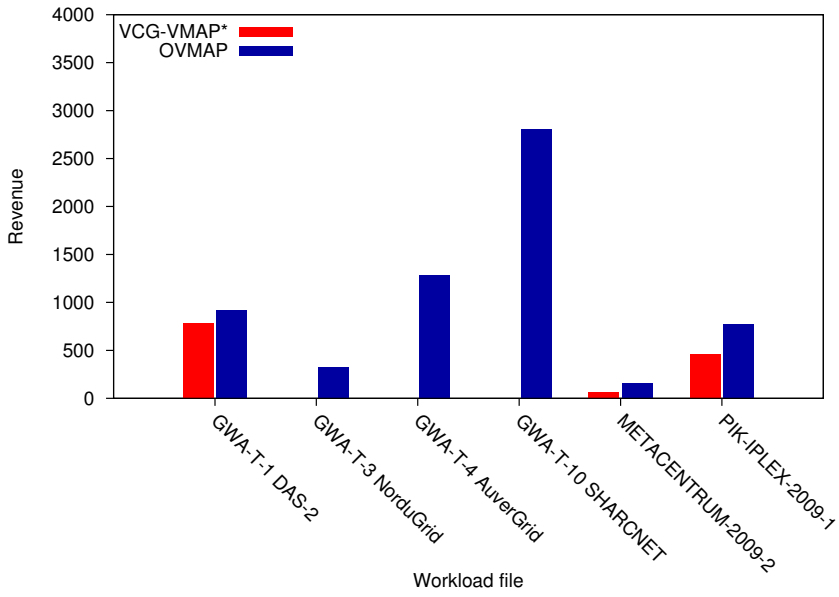


Figure 4.8: OVMAP vs. VCG-VMAP: Revenue

4.6 Conclusion

The nature and dynamics of users' demand in cloud markets necessitates designing online mechanisms. Such online mechanisms make no assumptions about future demands. In this chapter, we proposed an online incentive-compatible mechanism, OVMAP, for VM allocation and pricing in clouds. The OVMAP mechanism not only provisions and allocates resources dynamically, but also determines the price that users must pay for their requested VMs. Our proposed mechanism provides incentives to the users to reveal their actual requests facilitating a healthy competition among users. We proved that OVMAP is individually-rational and incentive-compatible. In addition, we proposed an optimal offline mechanism in order to compare its performance with our proposed online mechanism. The experimental results showed that the proposed online mechanism obtains better revenue and decides the allocation much faster than the offline mechanism.

CHAPTER 5: CLOUD FEDERATIONS IN THE SKY: FORMATION GAME AND MECHANISM

5.1 Introduction

Clouds are large-scale distributed computing systems built around core concepts such as computing as utility, virtualization of resources, on demand access to computing resources, and outsourcing computing services [169]. These concepts have positioned the clouds as an attractive platform for businesses enabling them to outsource some of their IT operations. In fact, the clouds services market share in the IT business has rapidly increased, and it is estimated to reach $150 billion by 2015 [138]. Cloud services are offered as three main categories: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). In this chapter, we focus on IaaS, where cloud providers offer different types of resources in the form of virtual machine (VM) instances.

Cloud computing systems' ability to provide on demand access to always-on computing utilities has attracted many enterprises due to their cost-benefit ratios, leading to rapid growth of the cloud computing market. Such market, however, presents a host of new challenges due to the dynamic nature of users' demands. The variability of users' demands increases when it comes to their requests for data-intensive applications. The amount of computing resources that data-intensive applications require can dramatically increase, and cloud providers' available resources may not be sufficient enough to cope with such demands. This emerging service management problem in cloud computing necessitates that cloud providers reshape their business structures and seek to improve their dynamic resource scaling capabilities. Federated clouds offer a practical platform for addressing this service management problem. A cloud provider can dynamically scale-up its resource capabilities by forming a cloud federation with other cloud providers. On the other hand, other cloud providers that have unused capacities can make profit by participating in a federation. Users' requests can be satisfied by federating resources belonging to several cloud providers [149, 151]. A cloud federation is a collection of cloud providers that co-

operate in order to provide the resources requested by users. Forming cloud federations helps achieve greater scalability and performance. If a cloud provider does not have enough resources to provide all the requested resources to the customer, it will reject the requests which leads not only to profit loses, but also to reputation losses. However, by forming a federation with other cloud providers, it can provide part of the requested resources to make some profit. In addition, the federation may provide the resources at a lower cost. Employing only one cloud provider may lead to issues such as lock-in and a single point of failure. Interoperability among clouds can eliminate the single point of failure problem [59]. Federating cloud resources can be a solution for interoperability among multiple clouds, enabling the formation of a pool of resources used for providing on demand services.

Virtualization is a major breakthrough enabling cloud providers to abstract the physical infrastructure, and to hide the complexity of underlying resources. They create a pool of virtualized resources which are offered to users as different types of VM instances. In this chapter, we model the cloud federation formation as a coalitional game, where cloud providers decide to form a coalition (cloud federation) to allocate VMs dynamically, based on users' requests. The cloud federation tries to maximize the total profit obtained by serving the users' requests. The model that we consider consists of a set of cloud providers and a user that submits a request consisting of a number of different VM instances. A subset of cloud providers will form a federation in order to provide the requested VM instances. Based on the proposed federation formation game we design a cloud federation formation mechanism.

5.1.1 Our Contribution

We model the cloud federation formation as a hedonic game, a type of coalitional game, satisfying fairness and stability properties. We define the federation preference relations for the cloud federation formation game and design a cloud federation formation mechanism which allows the cloud providers to make their own decisions to form a federation yielding the highest total profit. In the proposed mechanism, federations of cloud providers decide to merge and split in order to form a federation providing requested resources as a service

to the user. The mechanism also determines the individual profit of each participating cloud provider in the federation. Each cloud provider covers its incurred costs, and obtains a profit based on its market power. The mechanism provides a stable federation structure, that is, none of the cloud providers has incentives to merge to another federation or split from a federation to form another federation. We analyze the properties of our proposed cloud federation formation mechanism and perform extensive experiments to investigate its properties.

5.1.2 Related Work

The primary requirements for forming federations of cloud providers are discussed by Rochwerger *et al.* [150]. In order to support these requirements, Rochwerger *et al.* [149] introduced the Reservoir (resources and services virtualization without boundaries) model which allows two or more cloud providers to pool their resources together in order to provide services as a federated cloud. However, Reservoir does not provide any mechanism for forming cloud federations. Buyya *et al.* [23] presented the vision, challenges, and architectural elements of federated cloud computing environments. Their proposed framework supports scaling of applications across multiple cloud providers. Celesti *et al.* [28] introduced a cloud architecture that allows a cloud to build a federation with other clouds. Their model considers two types of clouds, home and foreign, where a home cloud is the cloud that is unable to fulfill its users' requests and forwards the requests to foreign clouds. Their model assumes that the foreign clouds could provide resources. It does not consider the incentives of the foreign clouds for providing resources to the home cloud. However, in this chapter we consider the incentives of the cloud providers for forming federations and provide a mechanism for sharing the profit among the cloud providers in a federation. Goiri *et al.* [49] provided models that assist a cloud provider in making decisions on forming federations with public clouds in order to maximize its profit. Their study does not consider the incentives of the other clouds for providing resources. In addition, they did not consider different types of VMs and their heterogeneous resources. However, our work takes into account the incentives of all the participating cloud providers and the heterogeneity of

VMs and cloud resources. Toosi *et al.* [166] proposed several resource provisioning policies helping the cloud providers increase their resource utilization and profit. They considered a model where the cloud providers can terminate the VMs, called spot VMs, whenever the profit for running such VMs is negative. Van den Bossche *et al.* [168] proposed a binary integer program formulation that minimizes the cost of outsourcing using a mix of public and private clouds. Their mechanism tries to maximize the utilization of the private cloud. However, their proposed method cannot find the solution on hybrid clouds in reasonable amount of time. Nordal *et al.* [131] proposed a system for managing computations in federated clouds. They also considered the applications' confidentiality constraints. Bin *et al.* [17] proposed a VM placement approach in a cloud federation considering multiple data privacy constraints. However, they did not consider the cost of outsourcing in the objective of their method. Chaisiri *et al.* [29] proposed an optimal VM provisioning algorithm using stochastic programming considering several cloud providers with the objective of maximizing profit. Bruneo [21] proposed performance evaluation techniques based on stochastic reward nets for federated clouds to predict and quantify the cost-benefit of a strategy portfolio and the corresponding quality of service experienced by users. Mihailescu and Teo [116] evaluated the impact of users' rationality in a federated cloud. Yang *et al.* [188] proposed a business-oriented federated cloud computing architecture for a specific type of applications, the real-time online interactive applications, such as multi-player online computer games. Their model is built on the concept of computation migration instead of VMs, and it does not consider the federation formation problem.

Researches approached the cloud resource management problem considering different objectives and points of view. Kesavan *et al.* [72] proposed a set of low-overhead management methods for managing the cloud infrastructure capacity to achieve a scalable capacity allocation for thousands of machines. Rodriguez and Buyya [152] proposed a meta-heuristic algorithm based on Particle Swarm Optimization for VM provisioning and scheduling strategies on IaaS. The proposed strategies minimize the overall workflow execution cost while meeting deadline constraints. Their approach considers dynamic provisioning, the heterogeneity of computing resources, and the variations in the VMs performance.

Doyle et al. [39] proposed an algorithm that determines to which data center the user requests should be routed, based on the relative priorities of the cloud operator. Their algorithm reduces the latency, carbon emissions, and operational costs. Mastroianni *et al.* [110] proposed an approach for the consolidation of VMs on two resources, that minimizes the power consumption while ensuring a good level of QoS. In their approach, decisions on the assignment and migration of VMs are driven by probabilistic processes and are based exclusively on local information. All these works addressed the resource management issues within a single cloud and not within federations of clouds.

Game theory-based resource allocation mechanisms for single clouds were proposed by Wei *et al.* [180] and Jalaparti *et al.* [67]. Zhang *et al.* [195] proposed online auction mechanisms for resource allocation in clouds. In our previous studies [96, 123], we proposed strategy-proof mechanisms for VM provisioning and allocation in clouds in order to maximize the profit. A game theoretic solution for dynamic resource allocation in a cloud federation was proposed by Hassan *et al.* [58]. The authors defined a price function for a cloud provider that gives incentives to other clouds to contribute resources and to form a federation. Mihailescu and Teo [115] proposed a strategy-proof dynamic pricing scheme for federated cloud environments. A revenue sharing mechanism for multiple cloud providers using stochastic linear programming games was proposed by Niyato *et al.* [130]. Their mechanism does not consider the cost that each cloud provider incurs. As a result, the solution of the revenue sharing is in the core of the proposed game. The model considers a fixed cooperation cost for each cloud provider. A cloud provider decides to join or not to join the federation based on the cooperation cost. Mashayekhy and Grosu [93] proposed a mechanism for solving the virtual organization formation problem in grids. The mechanism considers the incentives of the grid service providers while providing the required capabilities to execute the user's application. They also proposed a distributed mechanism for dynamic virtual organization formation in grids [91]. These mechanisms cannot be employed in cloud settings because the unique characteristics of cloud systems bring about new problems. Clouds necessitate the design of novel mechanisms considering virtualization, VM provisioning, VM allocation, and profit sharing among cloud providers. Li *et*

al. [85] investigated profit maximization strategies in cloud federations, where VMs are sold through auctions. They proposed a truthful double auction-based mechanism for trading VMs within a federation, where clouds can buy and sell their resources. Samaan [154] proposed an economic model based on repeated games, to regulate capacity sharing in a cloud federation. Cloud providers' objective is to sell their unused capacity in the spot market, but they are uncertain of future workload fluctuations. Our study is different from these studies since we consider the federation formation problem, where our proposed mechanism determines how cloud providers should provide the resources to fulfill users' requests.

5.1.3 Organization

This chapter is organized as follows. In Section 5.2, we describe the cloud federation formation problem and the system model we consider. In Section 5.3, we describe the game theoretic framework used to design the proposed cloud federation formation mechanism. Then, we present the proposed mechanism and characterize its properties. In Section 5.4, we evaluate the mechanism by extensive simulation experiments. In Section 5.5, we summarize our results.

5.2 Cloud Federation Framework

In this section, we describe the model of the system and introduce the problem of maximizing the profit within a cloud federation. We also introduce a coalitional game, called the cloud federation game, that serves as a basis for the development of our proposed cloud federation formation game and mechanism that will be presented in Section 5.3.

5.2.1 System Model

We first describe the system model consisting of a set of cloud providers, a broker as a mediator, and several cloud users. The broker is a trusted third party responsible for handling the federation formation tasks such as, receiving requests, executing the federation formation mechanism, receiving the payment from users, and dividing the profit among

Table 5.1: Notation

$\mathcal{I}$	Set of cloud providers $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$
N_i	Number of available cores of $\mathcal{C}_i \in \mathcal{I}$
M_i	Amount of available memory of $\mathcal{C}_i \in \mathcal{I}$
S_i	Amount of available storage of $\mathcal{C}_i \in \mathcal{I}$
$\mathcal{VM}$	Set of VMs $\{VM_1, \ldots, VM_n\}$
w_j^c	Number of cores in $VM_j \in \mathcal{VM}$
w_j^m	Amount of memory in $VM_j \in \mathcal{VM}$
w_j^s	Amount of storage in $VM_j \in \mathcal{VM}$
p_j	Price for an instance of type $VM_j \in \mathcal{VM}$
c_{ij}	Cost of an instance of type VM_j provided by $\mathcal{C}_i \in \mathcal{I}$
r_j	Number of requested VM instances of type VM_j
$\mathcal{R}$	User's request, $\mathcal{R} = \{r_1, \ldots, r_n\}$

participating providers. We assume that a set of cloud providers $\mathcal{I} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ is available to provide resources in the form of VM instances to cloud users. The cloud providers offer n types of VM instances: $\mathcal{VM} = \{VM_1, \ldots, VM_n\}$, where each instance provides a specific number of cores, amount of memory, and amount of storage. The VM instance of type VM_j $(j = 1, \ldots, n)$ is characterized by: (i) the number of cores, w_j^c; (ii) the amount of memory, w_j^m; and (iii) the amount of storage provided, w_j^s.

Each cloud provider $\mathcal{C}_i \in \mathcal{I}$ has a specific number of cores, amount of memory, and amount of storage available to share in a federation. Note that each provider reserves a specific capacity for its own users, and specifies the available capacity to be shared in the federation based on its load. We denote by N_i, the number of available cores of cloud provider $\mathcal{C}_i$, by M_i, the amount of available memory of cloud provider $\mathcal{C}_i$, and by S_i, the amount of available storage of cloud provider $\mathcal{C}_i$. Each provider $\mathcal{C}_i$ incurs cost when providing resources. For a cloud provider $\mathcal{C}_i$, we denote by c_{ij}, the cost associated with each VM instance of type VM_j, where $j = 1, \ldots, n$.

A user sends a request to a broker, consisting of the number of VM instances of each type needed. A request is denoted by $\mathcal{R} = \{r_1, \ldots, r_n\}$, where r_j is the number of requested VM instances of type VM_j, $j = 1, \ldots, n$. The broker bills a user based on the allocated VM instances. To do so, the broker sets a price p_j on each type of VM instance VM_j,

where $j = 1, \ldots, n$. The final price that the user pays for her request is independent of the cloud provider providing the VM instances. The final price paid by the user for each of the r_j VM instances of type VM_j is $r_j p_j$, where p_j is a fixed price for an instance of type VM_j. A broker has all the information about cloud providers such as their available resources and associated cost, and it is responsible for forming the federation. Table 5.1 summarizes the notation used throughout the chapter.

A cloud federation $\mathcal{F}$ is a set of cloud providers, i.e., $\mathcal{F} \subseteq \mathcal{I}$. The objective of a cloud federation $\mathcal{F}$ is to maximize its profit. We formulate the cloud federation profit maximization problem for a given federation $\mathcal{F}$ as an integer program (IP), called IP-CFPM, as follows:

$$\text{Maximize} \sum_{\mathcal{C}_i \in \mathcal{F}} \sum_{j=1}^{n} x_{ij}(p_j - c_{ij}), \tag{5.1}$$

Subject to:

$$\sum_{j=1}^{n} w_j^c x_{ij} \leq N_i, \ (\forall \mathcal{C}_i \in \mathcal{F}), \tag{5.2}$$

$$\sum_{j=1}^{n} w_j^m x_{ij} \leq M_i, \ (\forall \mathcal{C}_i \in \mathcal{F}), \tag{5.3}$$

$$\sum_{j=1}^{n} w_j^s x_{ij} \leq S_i, \ (\forall \mathcal{C}_i \in \mathcal{F}), \tag{5.4}$$

$$\sum_{\mathcal{C}_i \in \mathcal{F}} x_{ij} = r_j, \ (\forall j = 1, \ldots, n), \tag{5.5}$$

$$\sum_{j=1}^{n} x_{ij} \geq 1, \ (\forall \mathcal{C}_i \in \mathcal{F}), \tag{5.6}$$

$$x_{ij} \geq 0, \text{ and is integer}$$
$$(\forall \mathcal{C}_i \in \mathcal{F} \text{ and } \forall j = 1, \ldots, n), \tag{5.7}$$

The decision variables x_{ij} represent the number of VM instances of type VM_j provided by cloud provider $\mathcal{C}_i$. The objective function (5.1) is the total profit obtained by the participating cloud providers in the federation $\mathcal{F}$. The total profit is equal to the revenue received from the user minus the cost incurred by the cloud providers. Constraints (5.2) ensure that the number of cores provided by a cloud provider participating in the federation is less than its available number of cores. Constraints (5.3) guarantee that the amount of memory provided by a cloud provider participating in the federation is less than the amount of its available memory. Constraints (5.4) ensure that the amount of storage provided by a cloud provider participating in the federation is less than the amount of its available storage. Constraints (5.5) guarantee that the number of VM instances assigned to the user for each type of VM by all cloud providers is exactly the number of VM instances requested by the user for that type of VM instance. Constraints (5.6) ensure that each cloud provider in the federation contributes at least one VM instance. These constraints force the cloud providers to contribute resources to the federation. Constraints (5.7) represent the integrality requirements for the decision variables.

5.2.2 Cloud Federation Game

In this section, we introduce the *cloud federation game*, a coalitional game that allows us to model federations and investigate the stability of different federation structures. This game model will be extended in Section 3 into a hedonic game, called the cloud federation formation game, that characterizes the process of federation formation and will serve as the basis for the design of our proposed federation formation mechanism. The reader is referred to [133] for preliminaries on coalitional game theory. We define a *cloud federation game* $(\mathcal{I}, v)$, as a coalitional game with transferable utility, where each cloud provider in $\mathcal{I}$ is a player in the game, and v is the *characteristic function*, defined on $\mathcal{F} \subseteq \mathcal{I}$. The characteristic function is the profit obtained when the cloud providers of federation $\mathcal{F}$ cooperate as a coalition. This function is a real-valued function such that $v : \mathcal{F} \rightarrow \mathbb{R}^+$ and $v(\emptyset) = 0$. We consider each cloud federation $\mathcal{F} \subseteq \mathcal{I}$ as a *coalition*. If all the cloud providers form a federation, i.e., $\mathcal{F} = \mathcal{I}$, we call it the *grand federation*.

We define the characteristic function for our proposed cloud federation game as follows:

$$v(\mathcal{F}) = \begin{cases} 0 & \text{if } |\mathcal{F}| = 0 \text{ or IP-CFPM is not feasible,} \\ P & \text{if } |\mathcal{F}| > 0 \text{ and IP-CFPM is feasible,} \end{cases} \tag{5.8}$$

where $|\mathcal{F}|$ is the cardinality of $\mathcal{F}$, and P is the total profit obtained by the federation (i.e., the value of IP-CFPM objective function).

A cloud federation game should satisfy two main properties, fairness and stability. The profit obtained by a federation should be *fairly* divided among the participating cloud providers. A federation should be *stable*, that is, the participating cloud providers should not have incentives to leave the federation. In the following, we explain these two properties of the proposed game in more details.

The value $v(\mathcal{F})$ of the federation $\mathcal{F}$ must be divided among its participating cloud providers based on a given rule that satisfies fairness. In this study, we consider a fair profit division rule based on the market share of the cloud providers. A cloud provider that contributes more resources in all the possible federations in which it participates should

receive higher profit regardless of its resource allocation in the selected federation. We define, $\psi_{\mathcal{C}_i}(\mathcal{F})$, the *payoff* or the *share* of cloud provider $\mathcal{C}_i$ that is part of federation $\mathcal{F}$. In order to determine the payoff $\psi_{\mathcal{C}_i}(\mathcal{F})$, we employ the normalized Banzhaf value [132]. The *Banzhaf value* is a division of payoffs for the grand federation that takes into account the power of the players. The Banzhaf value of cloud provider $\mathcal{C}_i$ in the cloud federation game $(\mathcal{I}, v)$ is defined as follows:

$$\beta_{\mathcal{C}_i}(\mathcal{I}) = \frac{1}{2^{m-1}} \sum_{\mathcal{F} \subseteq \mathcal{I} \setminus \{\mathcal{C}_i\}} [v(\mathcal{F} \cup \{\mathcal{C}_i\}) - v(\mathcal{F})]. \tag{5.9}$$

The Banzhaf value represents the average marginal contribution of cloud provider $\mathcal{C}_i$ over all possible federations containing $\mathcal{C}_i$. The marginal contribution of $\mathcal{C}_i$ in a federation $\mathcal{F}$ is $v(\mathcal{F} \cup \{\mathcal{C}_i\}) - v(\mathcal{F})$, i.e., the difference between the value of a federation with and without $\mathcal{C}_i$. The *normalized Banzhaf value* is defined as

$$\mathcal{B}_{\mathcal{C}_i}(\mathcal{I}) = \frac{\beta_{\mathcal{C}_i}(\mathcal{I})}{\sum_{\mathcal{C}_j \in \mathcal{I}} \beta_{\mathcal{C}_j}(\mathcal{I})}. \tag{5.10}$$

The normalized Banzhaf value gives a fair way of dividing the grand federation's profit among its members. The profit that each member $\mathcal{C}_i$ receives in the grand federation is calculated as follows:

$$\psi_{\mathcal{C}_i}(\mathcal{I}) = \mathcal{B}_{\mathcal{C}_i}(\mathcal{I}) v(\mathcal{I}). \tag{5.11}$$

The payoff vector $\Psi(\mathcal{I}) = (\psi_{\mathcal{C}_1}(\mathcal{I}), \cdots, \psi_{\mathcal{C}_m}(\mathcal{I}))$ gives the payoff division for the grand federation. Computing the Banzhaf value for a game with a large number of players is NP-hard [111].

We now define, $\psi_{\mathcal{C}_i}(\mathcal{F})$, the payoff of cloud provider $\mathcal{C}_i$ participating in federation $\mathcal{F}$, as follows:

Table 5.2: The characteristics of available VM instances.

	Small VM_1	Medium VM_2	Large VM_3	Extralarge VM_4
w_j^c (1.6GHz CPU)	1	2	4	8
w_j^m (GB Memory)	1.7	3.75	7.5	15
w_j^s (TB Storage)	0.22	0.48	0.98	1.99
p_j (price)	0.12	0.24	0.48	0.96

$$\psi_{\mathcal{C}_i}(\mathcal{F}) = \frac{\psi_{\mathcal{C}_i}(\mathcal{I})}{\sum_{\forall \mathcal{C}_j \in \mathcal{F}} \psi_{\mathcal{C}_j}(\mathcal{I})} v(\mathcal{F}). \tag{5.12}$$

In our setting, using the Banzhaf value for payoff division is more reasonable than using the Shapley value. The Shapley value [158] considers the order of the players entering the federations, when determining the payoffs. However, in our case such an order does not affect the value of the federations. The Banzhaf value assumes that each player is equally likely to join any federation. That means, each federation will form with equal probability.

We analyze the stability of the grand federation using a solution concept for coalitional games, called the *core*. To define the core, we first need to introduce the concept of imputation, as follows.

Definition 20 (Imputation)**.** *An* imputation *is a payoff vector* $(\psi_{\mathcal{C}_1}(\mathcal{I}), \cdots, \psi_{\mathcal{C}_m}(\mathcal{I}))$ *satisfying:*

i) $\psi_{\mathcal{C}_i}(\mathcal{I}) \geq v(\mathcal{C}_i)$, $\forall \mathcal{C}_i \in \mathcal{I}$, *and*

ii) $\sum_{\mathcal{C}_i \in \mathcal{I}} \psi_{\mathcal{C}_i}(\mathcal{I}) = v(\mathcal{I})$.

Condition (i) guarantees that the profit obtained by each cloud provider $\mathcal{C}_i$ participating in the grand federation is not less than its profit obtained by acting alone. Condition (ii) ensures that the entire profit of the grand federation is divided among all cloud providers.

Definition 21 (Core)**.** *The* core *is a set of imputations satisfying* $\sum_{\mathcal{C}_i \in \mathcal{F}} \psi_{\mathcal{C}_i}(\mathcal{I}) \geq v(\mathcal{F}), \forall \mathcal{F} \subseteq \mathcal{I}$.

Table 5.3: The cloud providers' settings.

	N_i	M_i	S_i	$\mathcal{VM}$	c_j	p_j
$\mathcal{C}_i$	8	16 GB	2000 GB	VM_1	0.072	0.12
				VM_2	0.168	0.24
				VM_3	0.378	0.48
				VM_4	0.839	0.96

That means, the profit of any federation is not greater than the sum of the payoffs of its participating cloud providers in the grand federation. The existence of a payoff vector in the core shows that the grand federation is stable. As a result, a payoff division is in the core if there is no incentive for any cloud provider to leave the grand federation to join another federation. In the case that the core does not exist (i.e., the grand federation is not stable), independent and disjoint federations would form.

In the following we consider an example that shows that the core of the proposed cloud federation game can be empty. We consider three cloud providers $\mathcal{I} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$, and four types of VM instances $\mathcal{VM} = \{VM_1, VM_2, VM_3, VM_4\}$ representing small, medium, large, and extra large VM instances, respectively. The description of the VM instances is provided in Table 5.2. The instance types and pricing are similar to the ones used by Microsoft Azure [11].

We consider that a user requests one VM instance of type small, one VM instance of type medium, and one VM instance of type extra large. That is, the request of the user is $\mathcal{R} = \{1, 1, 0, 1\}$. In Table 5.3, we give the computing and storage capacity of each cloud provider, the cost and the price of each type of VM instance. For simplicity, we assume that all cloud providers have the same specifications. As an example, $\mathcal{C}_1$ incurs a cost of \$0.072 to provide one VM of type VM_1. The computing, memory, and storage capacity of $\mathcal{C}_1$ is 8 cores, 16 GB, and 2000 GB, respectively. Based on the request, the user pays \$1.32 to the federation, that is \$0.12+\$0.24+\$0.96.

If $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ provide resources individually, then none of them can satisfy the user's request. The values $v(\mathcal{F})$ for all possible federations are given in Table 5.4. If the sum of the

Table 5.4: The value for each federation.

$\mathcal{F}$	$v(\mathcal{F})$
$\{\mathcal{C}_1\}$	0
$\{\mathcal{C}_2\}$	0
$\{\mathcal{C}_3\}$	0
$\{\mathcal{C}_1, \mathcal{C}_2\}$	0.24
$\{\mathcal{C}_1, \mathcal{C}_3\}$	0.24
$\{\mathcal{C}_2, \mathcal{C}_3\}$	0.24
$\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$	0.24

payoffs to individual cloud providers $\{\mathcal{C}_1, \mathcal{C}_2\}$, $\{\mathcal{C}_1, \mathcal{C}_3\}$, and $\{\mathcal{C}_2, \mathcal{C}_3\}$ is less than 0.24 in the grand federation $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$, then the set of cloud providers has incentive to deviate from the grand federation. Since $\psi_{\mathcal{C}_1}(\mathcal{I}) + \psi_{\mathcal{C}_2}(\mathcal{I}) \geq v(\{\mathcal{C}_1, \mathcal{C}_2\})$, $\psi_{\mathcal{C}_1}(\mathcal{I}) + \psi_{\mathcal{C}_3}(\mathcal{I}) \geq v(\{\mathcal{C}_1, \mathcal{C}_3\})$, and $\psi_{\mathcal{C}_2}(\mathcal{I}) + \psi_{\mathcal{C}_3}(\mathcal{I}) \geq v(\{\mathcal{C}_2, \mathcal{C}_3\})$, are not simultaneously satisfied, there is no payoff vector in the core, and thus, the core of the cloud federation game is empty. Therefore, the grand federation would not form.

Since the grand federation may not be stable, we propose a cloud federation formation mechanism in order to find a stable cloud federation. In the next section, we introduce the cloud federation formation game, where the focus is on how to form independent and disjoint federations. The cloud federation formation game will be the basis for the design of our proposed cloud federation formation mechanism.

5.3 Cloud Federation Formation Mechanism

As we showed in the previous section, the core of the cloud federation game can be empty. If the grand federation does not form, independent and disjoint federations would form. In this section, we introduce the proposed cloud federation formation game, present the proposed cloud federation formation mechanism and characterize its properties.

5.3.1 Federation Formation Framework

In this section, we investigate the coalitional structures in the cloud federation game when the grand federation does not form, i.e., the grand federation is not stable. A federation structure $\mathcal{FS} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_h\}$ is a partition of the set of cloud providers $\mathcal{I}$ such that each provider is a member of exactly one federation, *i.e.*, $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset$ for all i and j, where $i \neq j$ and $\bigcup_{\mathcal{F}_i \in \mathcal{CF}} \mathcal{F}_i = \mathcal{I}$. We denote by Π the set of all federation structures. The total number of federation structures is B_m, where B_m is the m-th Bell number [74], and $m = |\mathcal{I}|$. Thus, finding the optimal federation structure via exhaustive search through all federation structures is not feasible.

The coalition formation [13] investigates the partitioning of the players into disjoint sets. In general, the problem of finding the optimal coalition structure in coalition formation is NP-complete [157]. Note that, only one of the federations in the federation structure is selected to provide the resources requested by the user. The remaining cloud providers may form other federations to service other requests of users. As a result, the formation of other federations with cloud providers outside of the selected federation does not affect the decision of the cloud providers participating in the selected federation.

To be able to model the cloud federation formation process, we augment the cloud federation game presented in subsection 2.2 with a preference relation over federations. The newly obtained game, called the *cloud federation formation game*, is a hedonic game [19], that is, a special type of coalitional game that considers players' preferences over coalitions. In hedonic games, players have preferences over coalitions. In the following, we present the definition of a hedonic game.

Definition 22 (Hedonic game)**.** *A hedonic game is a tuple $(\mathcal{I}, \succeq)$, where $\mathcal{I}$ is the set of players in the game, $\succeq_i$ is a reflexive, complete, and transitive preference relation defined on Π_i for player i, and Π_i is the set of subsets in $\mathcal{I}$ containing player i.*

Each player knows whether it prefers to be in company of some players rather than others. If $A \succeq_i B$, player i prefers coalition B at most as much as coalition A. As in the case of the cloud federation game, the core does not exist for the cloud federation formation

game.

Definition 23 (Cloud federation formation game)**.** *A cloud federation formation game is a pair* $(\mathcal{I}, \succeq)$*, where* $\succeq_i$ *is a reflexive, complete, and transitive binary relation on* Π_i*, and* Π_i *is the set of federations in* $\mathcal{I}$ *containing* $\mathcal{C}_i$*.*

We define the *federation preference relation* $\succeq_i$ for each $\mathcal{C}_i$. This allows $\mathcal{C}_i$ to compare two federations and to indicate its preference to be a part of one of them. $A \succeq_i B$ implies that $\mathcal{C}_i$ prefers to be a member of federation A than to be a member of federation B, or at least it prefers both federations equally. In addition, $A \succ_i B$ indicates that $\mathcal{C}_i$ strictly prefers to be a member of A than a member of B.

To model the cloud federation formation as a hedonic game, we need to define the federation preference relation over all pairs of federations in Π_i. For all $\mathcal{C}_i \in \mathcal{I}$ and for all $\mathcal{F}, \mathcal{F}' \in \Pi_i$, we define $\succeq_i$ as

$$\mathcal{F} \succeq_i \mathcal{F}' \iff v(\mathcal{F}) \geq v(\mathcal{F}'). \tag{5.13}$$

That means a cloud provider prefers the federation that gives the higher profit. Using this preference relation, every cloud provider can evaluate its preferences over the set of possible federations that the cloud provider can be a member of. Therefore, the objective of each cloud provider is to determine the membership in a federation that gives the highest profit.

Next, we define two comparison relations based on the preference relation $\succeq_i$. These comparison relations will be used in the design of our proposed cloud federation formation mechanism. The two comparison relations, called *merge comparison* $\gg_m$ and *split comparison* $\gg_s$, will allow us to decide if a federation is more preferred than other federations.

The *merge comparison* $\gg_m$ is defined as follows:

$$\begin{aligned} \{\mathcal{F} \cup \mathcal{F}'\} \ \gg_m \{\mathcal{F}, \mathcal{F}'\} \iff & \\ \{\forall \mathcal{C}_i \in \mathcal{F}; \{\mathcal{F} \cup \mathcal{F}'\} \succ_i \mathcal{F} \text{ and} & \\ \forall \mathcal{C}_j \in \mathcal{F}'; \{\mathcal{F} \cup \mathcal{F}'\} \succ_j \mathcal{F}'\} & \end{aligned} \tag{5.14}$$

Equation (5.14) implies that federation $\{\mathcal{F} \cup \mathcal{F}'\}$ is preferred over two disjoint federations

$\{\mathcal{F}, \mathcal{F}'\}$, if the profit obtained by federation $\{\mathcal{F} \cup \mathcal{F}'\}$ is greater than the profit obtained by the providers in $\mathcal{F}$, and it is greater than the profit obtained by the providers in $\mathcal{F}'$. As a result, all providers are able to improve the total profit.

The *split comparison* $\gg_s$ is defined as follows:

$$\begin{aligned} \{\mathcal{F}, \mathcal{F}'\} \ \gg_s \{\mathcal{F} \cup \mathcal{F}'\} \iff \\ \{\exists \mathcal{C}_i \in \mathcal{F}; \mathcal{F} \succeq_i \{\mathcal{F} \cup \mathcal{F}'\} \text{ or} \\ \exists \mathcal{C}_j \in \mathcal{F}'; \mathcal{F}' \succeq_j \{\mathcal{F} \cup \mathcal{F}'\}\} \end{aligned} \tag{5.15}$$

Equation (5.15) implies that $\{\mathcal{F}, \mathcal{F}'\}$ is preferred over $\{\mathcal{F} \cup \mathcal{F}'\}$, if at least one federation is able to keep the same amount of profit or to increase the profit of its members. Such a split preference is irrespective of the other cloud providers' preferences outside of that federation.

The two comparison relations defined above induce two rules [13] that will be used in the design of our proposed cloud federation formation mechanism:

Merge Rule: For any pair of federations $\mathcal{F}$ and $\mathcal{F}'$:
$\{\mathcal{F} \cup \mathcal{F}'\} \gg_m \{\mathcal{F}, \mathcal{F}'\} \Rightarrow$ Merge $\mathcal{F}$ and $\mathcal{F}'$.

Split Rule: For any federation $\{\mathcal{F} \cup \mathcal{F}'\}$:
$\{\mathcal{F}, \mathcal{F}'\} \gg_s \{\mathcal{F} \cup \mathcal{F}'\} \Rightarrow$ Split $\{\mathcal{F} \cup \mathcal{F}'\}$.

The merge rule implies that two federations join to form a larger federation if operating all of their cloud providers together strictly improves the total profit. The split rule implies that a federation splits only if there exists one sub-federation that obtains at least the same total profit with its constituent cloud providers. A split happens irrespective of the effect on the profit of the other sub-federations, i.e., their profit may decrease. Note that only one of the federations in the federation structure is selected to provide the resources requested by the user. Therefore, the goal of the cloud providers is to find a federation with maximum profit. This federation is obtained through an iterative application of the merge and the split rules.

As we mentioned before, computing the Banzhaf value for a game is NP-hard. However, through the iterative application of merge and split rules some of the possible federations are checked and their values are calculated. Based on those values, we define the *estimated Banzhaf value* of $\mathcal{C}_i$ as follows:

$$E_{\mathcal{C}_i}(\mathcal{I}) = \frac{1}{\lambda} \sum_{\substack{\mathcal{F} \subseteq \mathcal{I} \setminus \{\mathcal{C}_i\} \\ \mathcal{F} \in \mathcal{V} \\ \mathcal{F} \cup \mathcal{C}_i \in \mathcal{V}}} [v(\mathcal{F} \cup \{\mathcal{C}_i\}) - v(\mathcal{F})]. \tag{5.16}$$

where $\mathcal{V}$ is the set of all checked federations (i.e., federations that were already produced during the merge and split iterations), and λ is the total number of checked federations containing $\mathcal{C}_i$. That means, $\lambda = 2^{m-1} - \alpha$, where α is the number of non-checked federations. The estimated Banzhaf value is based only on the value of federations that are checked during the merge and split process. The *normalized estimated Banzhaf value* is defined as follows:

$$\mathcal{E}_{\mathcal{C}_i}(\mathcal{I}) = \frac{E_{\mathcal{C}_i}(\mathcal{I})}{\sum_{\mathcal{C}_j \in \mathcal{I}} E_{\mathcal{C}_j}(\mathcal{I})}. \tag{5.17}$$

The profit that each member $\mathcal{C}_i$ receives in the grand federation is calculated as follows:

$$\psi_{\mathcal{C}_i}(\mathcal{I}) = \mathcal{E}_{\mathcal{C}_i}(\mathcal{I}) v(\mathcal{I}). \tag{5.18}$$

The payoff vector $\Psi(\mathcal{I}) = (\psi_{\mathcal{C}_1}(\mathcal{I}), \cdots, \psi_{\mathcal{C}_m}(\mathcal{I}))$ gives the payoff divisions of the grand federation. We define $\psi_{\mathcal{C}_i}(\mathcal{F})$, the payoff of cloud provider $\mathcal{C}_i \in \mathcal{F}$, as follows:

$$\psi_{\mathcal{C}_i}(\mathcal{F}) = \frac{\psi_{\mathcal{C}_i}(\mathcal{I})}{\sum_{\forall \mathcal{C}_j \in \mathcal{F}} \psi_{\mathcal{C}_j}(\mathcal{I})} v(\mathcal{F}). \tag{5.19}$$

During the merge-and-split we estimate the Banzhaf value for each provider based only on the federations that were already explored. The profit obtained by the federation is divided among participating cloud providers in proportion to their power in the federation.

We define a new concept of stability similar to the stability of a coalition structure

Algorithm 15 Cloud Federation Formation Mechanism (CFFM)

Input: Request $\mathcal{R}$
$\mathcal{V} = \emptyset$
$\mathcal{FS} = \{\{\mathcal{C}_1\}, \cdots, \{\mathcal{C}_m\}\}$
for all $\mathcal{F}_i \in \mathcal{FS}$ **do**
    $v(\mathcal{F}_i)$ = Solve IP-CFPM($\mathcal{F}_i$)
    $\mathcal{V} = \mathcal{V} \cup \mathcal{F}_i$
repeat
    MergeFederations();
    SplitFederation();
until No split happens
Find $\mathcal{F}_k = \arg\max_{\mathcal{F}_i \in \mathcal{FS}} \{v(\mathcal{F}_i)\}$
for all $\mathcal{C}_i \in \mathcal{F}_k$ **do**
    Calculate $\psi_{\mathcal{C}_i}(\mathcal{F}_k)$ based on $\mathcal{V}$
$\mathcal{F}_k$ allocates and provides the requested VM instances.

defined in the context of the hedonic games [19]. The difference from the stability concept in hedonic games is that we consider only one federation instead of a federation structure. This is due to the fact that only one federation of cloud providers is needed to form in order to fulfill a user request. As a result, our proposed stability notion is defined on the participating cloud providers in the obtained federation by the proposed mechanism. We define the *individual federation stability* as follows.

Definition 24 (Individual federation stability)**.** *A federation $\mathcal{F}$ is* individually federation stable *if there is no member $\mathcal{C}_i \in \mathcal{F}$ such that $\mathcal{F} \setminus \{\mathcal{C}_i\} \succeq_j \mathcal{F}$ for all $j \in \mathcal{F}$.*

In other words, a federation $\mathcal{F}$ is individually federation stable if there is no cloud provider $\mathcal{C}_i \in \mathcal{F}$ that can leave $\mathcal{F}$ without making at least one cloud provider $\mathcal{C}_j \in \mathcal{F}$ unhappy.

In the next section, we introduce our proposed cloud federation formation mechanism and prove that it produces individually stable federations.

5.3.2 Cloud Federation Formation Mechanism (CFFM)

The proposed cloud federation formation mechanism (CFFM), presented in Algorithm 15, relies on the merge and split rules defined in the previous section. The mechanism is executed by a broker.

Algorithm 16 MergeFederations()

1: **repeat**
2: Select two non-checked federations $\mathcal{F}_i, \mathcal{F}_j \in \mathcal{FS}$
3: $v(\mathcal{F}_i \cup \mathcal{F}_j)$ = Solve IP-CFPM($\mathcal{F}_i \cup \mathcal{F}_j$)
4: $\mathcal{V} = \mathcal{V} \cup \{\mathcal{F}_i \cup \mathcal{F}_j\}$
5: **if** $\mathcal{F}_i \cup \mathcal{F}_j \gg_m \{\mathcal{F}_i, \mathcal{F}_j\}$ **then**
6: $\mathcal{F}_i \leftarrow \mathcal{F}_i \cup \mathcal{F}_j$
7: $\mathcal{F}_j \leftarrow \emptyset$ {$\mathcal{F}_j$ is removed from $\mathcal{FS}$}
8: **until** No merge happens

CFFM receives a user request as input (line 1). CFFM uses $\mathcal{V}$ to store all checked federations during the merge-and-split process. The algorithm sets $\mathcal{V}$ to the empty set (line 2). First, an initial federation structure $\mathcal{FS}$ in which every individual cloud provider $\mathcal{C}_i \in \mathcal{I}$ is a federation $\mathcal{F}_i$, is formed (line 3). Then, CFFM solves IP-CFPM to find $v(\mathcal{F}_i)$ for each federation $\mathcal{F}_i$ (line 5). Note that if IP-CFPM is not feasible for a federation, i.e., IP does not have a solution, the value of the federation is zero. All singleton federations are added to $\mathcal{V}$ (line 6).

CFFM iteratively calls MergeFederations() and SplitFederation() functions (lines 7-10). CFFM exits from the merge-and-split process when there is no possibility for a further merge or a further split. Then, CFFM finds a federation, $\mathcal{F}_k$, with the highest total profit among all federations in the final federation structure (line 11). CFFM calculates the normalized estimated Banzhaf value as the individual profit for each participating cloud provider in $\mathcal{F}_k$ based on its marginal contributions in the set of checked federations, $\mathcal{V}$ (lines 12-13). The selected federation, $\mathcal{F}_k$, provides the requested VM instances to the user (line 14).

The MergeFederations() procedure is presented in Algorithm 16. This procedure checks all merge possibilities of any pair of federations in the federation structure. First, MergeFederations() randomly selects two non-checked federations in $\mathcal{FS}$, e.g., $\mathcal{F}_i$ and $\mathcal{F}_j$ (line 2). Then, it solves IP-CFPM to find the value of a new federation $\{\mathcal{F}_i \cup \mathcal{F}_j\}$, and adds the new federation to the set $\mathcal{V}$ (lines 3-4). If merge is more preferred, then federations $\mathcal{F}_i$ and $\mathcal{F}_j$ are merged. The new federation is saved in $\mathcal{F}_i$, and $\mathcal{F}_j$ is removed from $\mathcal{FS}$ (lines 5-7). Note that the members of $\mathcal{F}_i$ have changed, and thus, $\mathcal{F}_i$ can be selected again for the

Algorithm 17 SplitFederation()

1: **for all** $\mathcal{F}_i \in \mathcal{FS}$ where $|\mathcal{F}_i| > 1$ **do**
2: **for all** partitions $\{\mathcal{F}_j, \mathcal{F}_k\}$ of $\mathcal{F}_i$,
where $\mathcal{F}_i = \mathcal{F}_j \cup \mathcal{F}_k, \mathcal{F}_j \cap \mathcal{F}_k = \emptyset$ **do**
3: $v(\mathcal{F}_j)$ = Solve IP-CFPM($\mathcal{F}_j$)
4: $v(\mathcal{F}_k)$ = Solve IP-CFPM($\mathcal{F}_k$)
5: $\mathcal{V} = \mathcal{V} \cup \mathcal{F}_j$
6: $\mathcal{V} = \mathcal{V} \cup \mathcal{F}_k$
7: **if** $\{\mathcal{F}_j, \mathcal{F}_k\} \gg_s \mathcal{F}_i$ **then**
8: $\mathcal{F}_i \leftarrow \mathcal{F}_j$
9: $\mathcal{FS} = \mathcal{FS} \cup \mathcal{F}_k$
10: **break**

merge. The MergeFederations() procedure tries to find another pair of non-checked federations in the federation structure by repeating the procedure. The MergeFederations() procedure terminates if all pairs of federations are checked and a merge does not happen, or the grand federation forms.

The SplitFederation() procedure is presented in Algorithm 17. This procedure checks all split possibilities of any federation with more than one member in the federation structure. The SplitFederation() procedure tries to split a federation, e.g., $\mathcal{F}_i$, into two disjoint federations $\mathcal{F}_j$ and $\mathcal{F}_k$, where $\mathcal{F}_j \cup \mathcal{F}_k = \mathcal{F}_i$. Then, it solves IP-CFPM twice to find the value of $\mathcal{F}_j$ and $\mathcal{F}_k$ (lines 3-4). In addition, it adds the two federations into the set $\mathcal{V}$ since their values are calculated (lines 5-6). If the split is more preferred, then one of the splited federations, $\mathcal{F}_j$, is saved in $\mathcal{F}_i$, and the other splited federation, $\mathcal{F}_k$, is added to the federation structure (lines 7-10). Since the federation structure has changed after the split happened, CFFM executes another iteration of merge-and-split. Note that if none of the existing federations splits then CFFM terminates. This is due to the fact that the existing federations already have been checked for the merge in the MergeFederations() procedure. If only the MergeFederations() procedure is applied without applying the SplitFederation() procedure, then the mechanism converges very fast, but the obtained solution would be far from the optimal, since it does not allow the formation of smaller intermediate federations that can later on lead to better federations. However, by applying the SplitFederation() procedure, the mechanism iteratively improves the solution until it finds one that is closer

to the optimal.

For a given user request, only one federation will form. When another request arrives and the cloud providers participating in some already formed federations have resources available, they can participate again along with other cloud providers to form a federation in order to serve the request. If a cloud provider does not have any resources available, it waits until the federation that it belongs to dissolves, and then participates again in the federation formation mechanism to serve future requests.

5.3.3 CFFM Properties

In this section, we show that CFFM converges to a stable federation structure and analyze CFFM's time complexity. First, we prove that the proposed CFFM converges to a final federation structure.

Theorem 17. *CFFM converges to a federation structure composed of disjoint federations of cloud providers.*

Proof. Based on the proposed preference relation $\gg_m$, the resulting federation after each merge is more preferred than previous federations. This is also true for the resulting federations after each split using the proposed preference relation $\gg_s$. As a result, if a federation structure is created during the merge-and-split iterations, the mechanism cannot create that federation structure again by any further merge-and-split iterations. This is due to the fact that by any merge-and-split iteration, the mechanism finds more preferred federations. In addition, the total number of federation structures is finite. Therefore, the merge-and-split iterations terminate, and the final federation structure cannot be subject to any further merge and split. As a result, CFFM always converges. □

Since CFFM converges to a final federation structure, there is no possibility for further merge and split. Therefore, the final federation structure cannot be subject to any further change, that is, none of the federations in the federation structure can merge to another federation (or split into sub-federations) to form another federation structure.

Now, we prove that the final federation satisfies the individual stability property.

Theorem 18. *CFFM produces an individually stable federation.*

Proof. A federation $\mathcal{F}$ is individually federation stable if there is no cloud provider $\mathcal{C}_i \in \mathcal{F}$ that can leave $\mathcal{F}$ without making at least one cloud provider $\mathcal{C}_j \in \mathcal{F}$ unhappy. In the split procedure, CFFM checks if any cloud provider in a federation wants to leave its current federation by checking all the possibilities for split. If it finds such a cloud provider, CFFM applies the split rule. As a result, no cloud provider that is part of the final federation has incentive to leave the federation. □

Solving the federation formation problem optimally has the same complexity as performing an exhaustive search on all the possible partitions of the set of cloud providers. However, since CFFM is using the merge and split procedures, it does not perform an exhaustive search on the set of partitions. The time complexity of CFFM is determined by the number of merge and split operations and the size of the sub-partitions. In the worst case scenario, each federation needs to make a merge attempt with all the other federations in $\mathcal{FS}$. In the initial federation structure, where each of the m individual cloud providers is a federation, the first merge occurs after $\frac{m(m-1)}{2}$ attempts in the worst case. The second merge requires $\frac{(m-1)(m-2)}{2}$ attempts and so on. As a result, the total worst case number of merges is in $O(m^3)$. In the worst case scenario, splitting a federation F is in $O(2^{|F|})$, which involves finding all the possible partitions of size two of the participating federations.

5.4 Experimental Results

We perform a set of simulation experiments which allows us to investigate how effective the proposed federation formation mechanism is in producing stable cloud federations.

5.4.1 Experimental Setup

For our experiments, we consider that eight independent cloud providers are participating. We set the costs and the types of VMs offered by each of these cloud providers to the types and costs of VMs offered by Amazon EC2 [1] in each of its eight regions. These cloud providers offer four types of VM instances as presented in Table 5.2. For the cost of

Table 5.5: Cost

Amazon EC2 Regions	US East (N. Virginia)	US West (Oregon)	US West (Northern California)	EU (Ireland)	South America (Sao Paulo)	Asia Pacific (Singapore)	Asia Pacific (Sydney)	Asia Pacific (Tokyo)
Cloud Provider	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$
VM_1 (Small)	$0.03	$0.045	$0.048	$0.033	$0.055	$0.04	$0.058	$0.044
VM_2 (Medium)	$0.06	$0.091	$0.096	$0.065	$0.111	$0.08	$0.115	$0.088
VM_3 (Large)	$0.12	$0.182	$0.192	$0.130	$0.222	$0.16	$0.230	$0.175
VM_4 (Extralarge)	$0.24	$0.364	$0.384	$0.260	$0.444	$0.32	$0.460	$0.350

VMs, we use the VM prices of On-Demand Instance Prices offered by Amazon EC2. We set the cost of VM instances to the half of the actual VM prices of Amazon EC2 regions. The detailed information regarding the cost of VMs are presented in Table 5.5. We relied only on EC2 information for its different regions because it is publicly available. Each of the cloud providers considered in the simulation are independent and the cost information from each of the Amazon EC2 regions is used only to setup their cost structure. Following Samaan [154] and Toosi *et al.* [166], we consider 1024 cores, 1740 GB of memory, and 225 TB of storage as the average of capacities of the cloud providers from which 40% is available for the federation. We generate 100 requests such that requests with less than 15%, 25%, 35%, and 45% of the total available capacity belong to the small, medium, large, and extra-large classes, respectively. All requests cannot be served by only one cloud provider and they need to form a federation in order to serve the user. Each class contains 25 requests, and in the plots we represent the average of the obtained results. The parameters used in our experiments and their values are listed in Table 5.6.

While it is desirable to compare our proposed mechanism with several mechanisms, we

Table 5.6: Parameters

Param.	Description	Value(s)
m	Number of cloud providers	8
n	Number of VM types	4
N_i	Number of cores	[512, 1536]
M_i	Memory (GB)	[870, 2610]
S_i	Storage (TB)	[112, 338]
c_j	VM cost vector (4)	Based on Amazon Regions
p_j	VM price vector (4)	Based on Microsoft Azure

found out that the existing mechanisms and approaches are not directly comparable to ours and decided to compare it with only two other mechanisms, Optimal Cloud Federation Mechanism (OCFM), and Random Cloud Federation Mechanism (RCFM). The OCFM mechanism finds the optimal solution to the federation formation problem, that is, it finds a cloud federation with maximum profit. This is achieved by exhaustively enumerating all the possible federations and solving IP-CFPM optimally for each of these federations. We rely on the optimal results obtained by OCFM as a benchmark for our experiments. We use the IBM ILOG Concert Technology APIs [7] in C++ to solve the integer program IP-CFPM associated with the mechanism. IBM ILOG provides optimization APIs and its engine is the CPLEX Optimizer that solves integer programming problems. The RCFM mechanism selects several cloud providers randomly and forms a federation. All the mechanisms are implemented in C++, and the experiments are conducted on AMD 2.93GHz hexa-core dual-processor systems with 90GB of RAM which are part of the Wayne State Grid System.

5.4.2 Analysis of Results

In Fig. 5.1, we compare the total profit obtained by CFFM with that obtained by the other two mechanisms. In all cases CFFM yields the highest profit which is very close to the optimal profit obtained by OCFM. These results show that RCFM achieves profits that are not even half the profits obtained by the other two mechanisms. In addition, the obtained

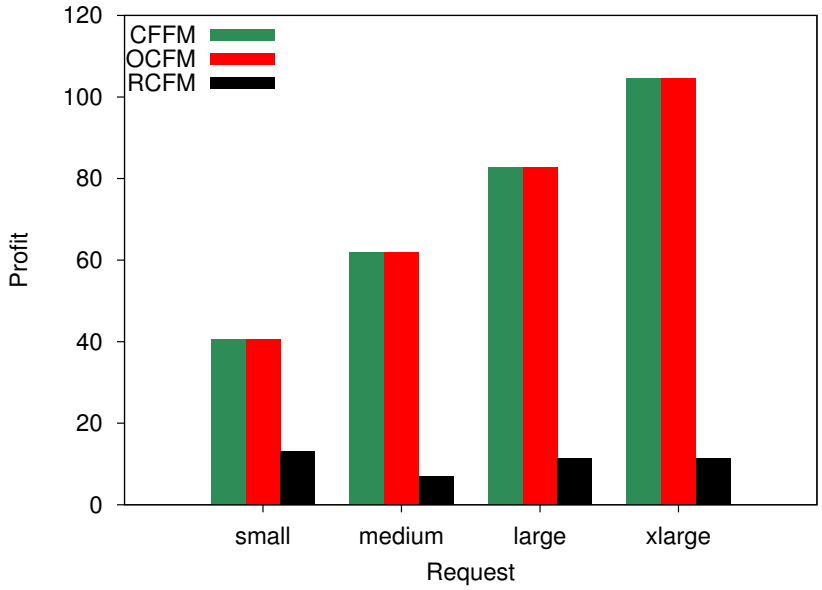


Figure 5.1: Total profit of the cloud federation

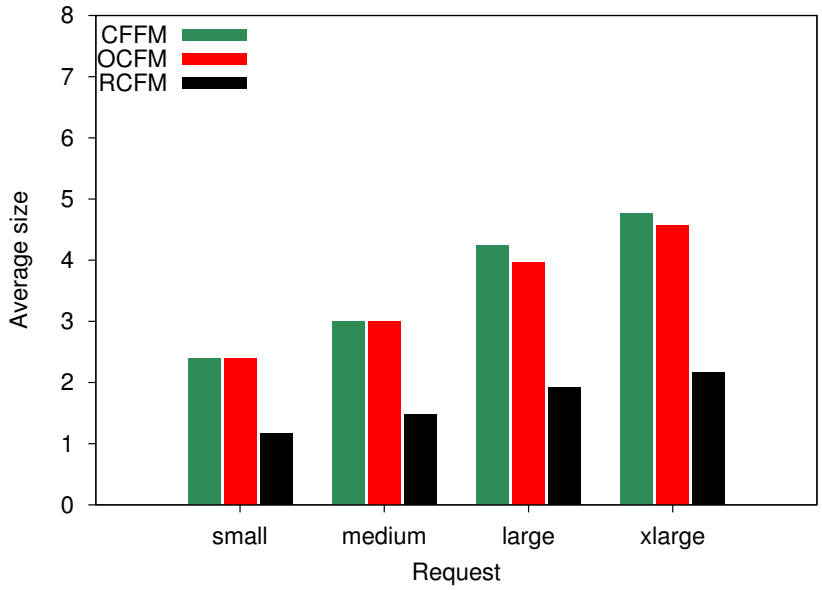


Figure 5.2: Average size of the cloud federation

total profit obtained by CFFM and OCFM increases with the increase in the size of the requests.

Fig. 5.2 shows the average number of cloud providers participating in the federation for each class of requests. As it is expected, with the increase in the size of the requests,

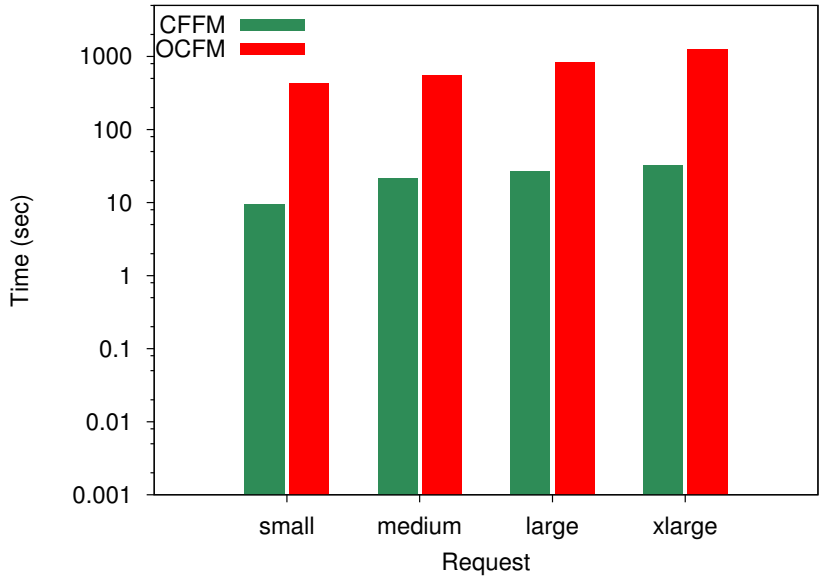


Figure 5.3: Execution time of the mechanisms

more cloud providers need to participate in the federation to serve the requests. The size of the formed federations obtained by our proposed mechanism, CFFM, is close to that of the optimal solution. Note that in RCFM some of the federations are not feasible (i.e., the randomly chosen cloud providers cannot fulfill the request). Therefore, the average federation size of RCFM is very small.

Fig. 5.3 shows the execution time of the two mechanisms. The execution times of RCFM are negligible compared to that of CFFM and OCFM, and thus, we chose not to present them in the figure. From 255 federations that the eight cloud providers could form, CFFM only considers some of them in the merge-and-split process based on the merge and split rules. On average, CFFM explores 48 federations until it finds the final federation. As a result, the execution time of CFFM is a lot less than that of OCFM which goes through all the federations. The mechanisms require more time for larger requests. The execution time of our proposed mechanism, CFFM, is about two orders of magnitude less than that of the optimal mechanism OCFM.

Figs. 5.4 to 5.7 show the individual profit of each participating cloud provider in the federation, separately, for each class of requests. The mechanisms use different profit division

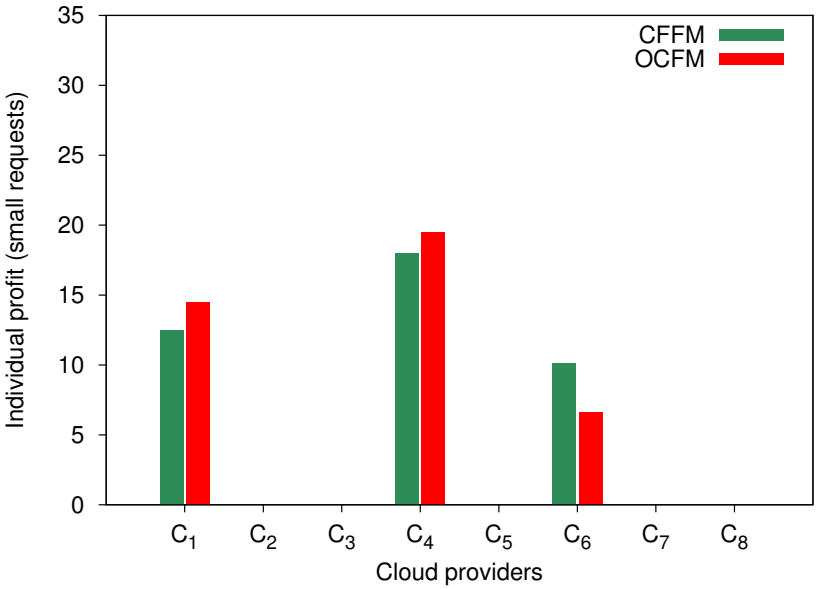


Figure 5.4: Profit of cloud providers (small requests)

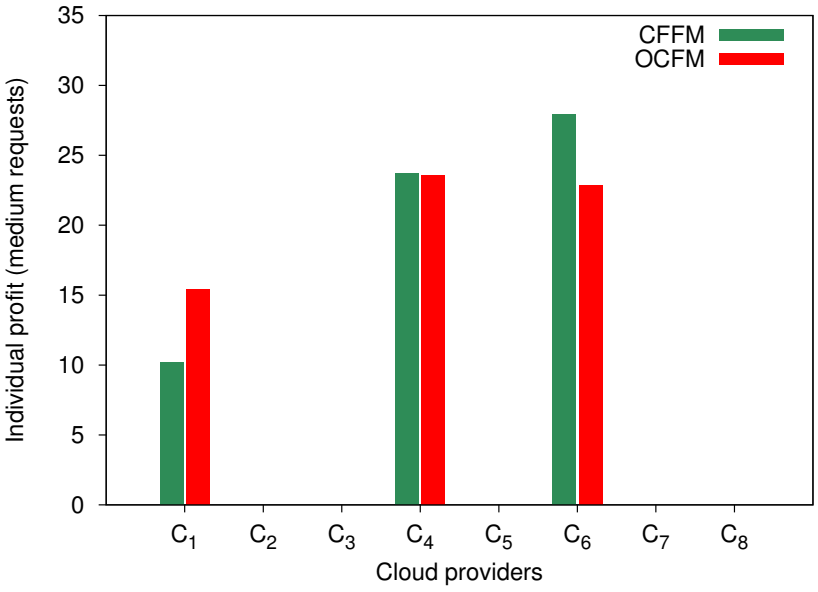


Figure 5.5: Profit of cloud providers (medium requests)

rules as follows. CFFM uses the estimated normalized Banzhaf value, while the OCFM uses the normalized Banzhaf value. As it is shown in the figures, the individual profits of the participating cloud providers are very close in CFFM and OCFM. The difference between the average individual profit of CFFM and OCFM is 17%, 17%, 17% and 6% for

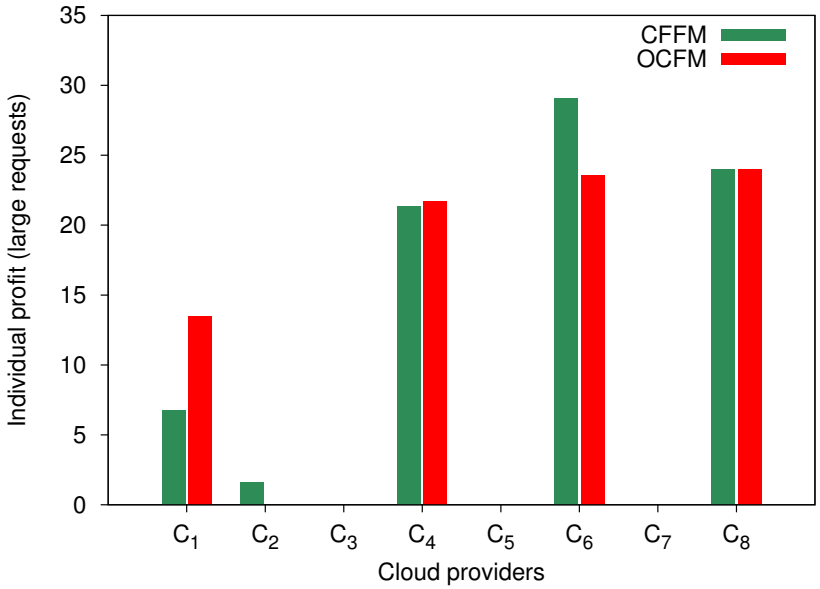


Figure 5.6: Profit of cloud providers (large requests)

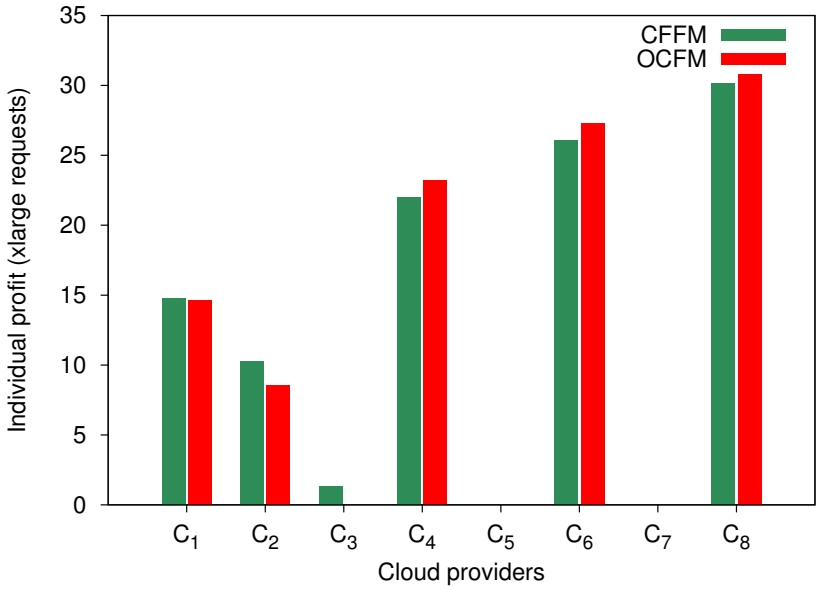


Figure 5.7: Profit of cloud providers (extralarge requests)

small, medium, large, and extra-large requests presented in Figs. 5.4 to 5.7, respectively. The reason that the difference between the average individual profit of CFFM and OCFM for the extra-large requests are much less than those for the other requests is that the number of federations that are checked during the process of merge-and-split is higher, leading

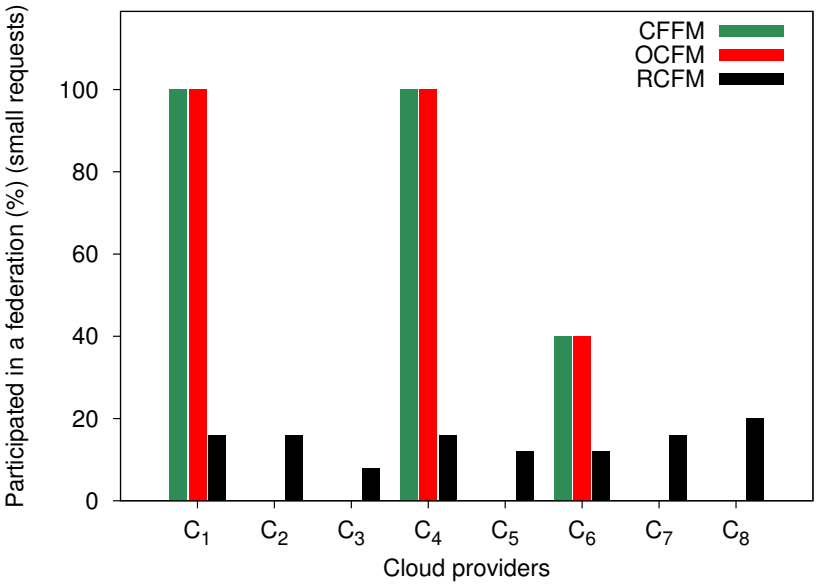


Figure 5.8: Percentage participation of cloud providers (small requests)

to a precise estimated normalized Banzhaf value. This is due to the fact that for larger requests the number of participating cloud providers in a federation increases resulting in more possibilities for merge-and-split operations. However, this comes at the cost of higher execution time.

We now investigate the performance of the proposed mechanism in more details. Figs. 5.8 to 5.11 show the percentage of participation of cloud providers in the federation, separately, for each class of requests. In all cases, the cloud provider with the lowest cost (i.e., $\mathcal{C}_1$) is a member of the formed federation. Fig. 5.8 shows that $\mathcal{C}_1$ and $\mathcal{C}_4$ participated in all formed federations obtained by CFFM and OCFM for small requests. For 40% of the requests a federation with size two does not have enough capacity to fulfill the requests, and thus, $\mathcal{C}_6$ participated in the formed federation on such cases. Figs. 5.9 shows that $\mathcal{C}_1$, $\mathcal{C}_4$, and $\mathcal{C}_6$ participated in all formed federations for the medium requests. This is due to the fact that their capacity is sufficient for the medium size requests. Note that all three cloud providers have to participate in the federations for the medium size requests unlike the small size requests case. In addition, as the size of the requests becomes larger, the cloud providers with higher costs are selected to participate in federations. For example, comparing Fig. 5.8 and

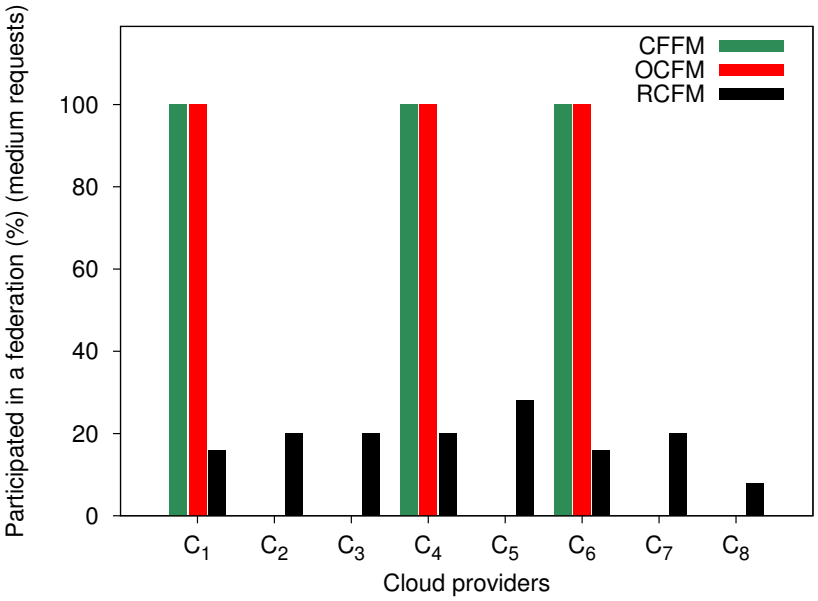


Figure 5.9: Percentage participation of cloud providers (medium requests)

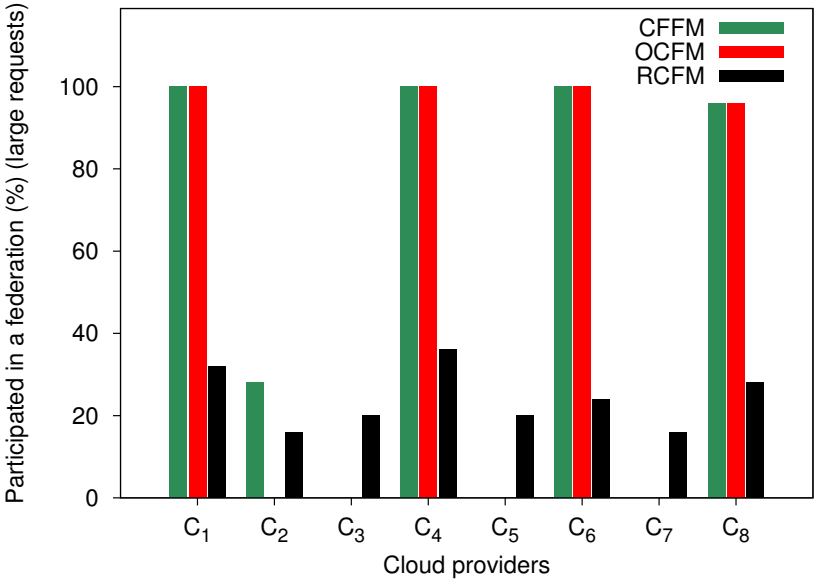


Figure 5.10: Percentage participation of cloud providers (large requests)

Fig. 5.10 we observe that $\mathcal{C}_1$, $\mathcal{C}_4$, and $\mathcal{C}_6$ are members of the formed federations obtained by CFFM and OCFM. Based on Table 5.5, these cloud providers have the lowest costs. By increasing the size of the requests from small to large, the available capacity of these cloud providers are not sufficient enough. To fulfill the large requests (Fig. 5.10) federations

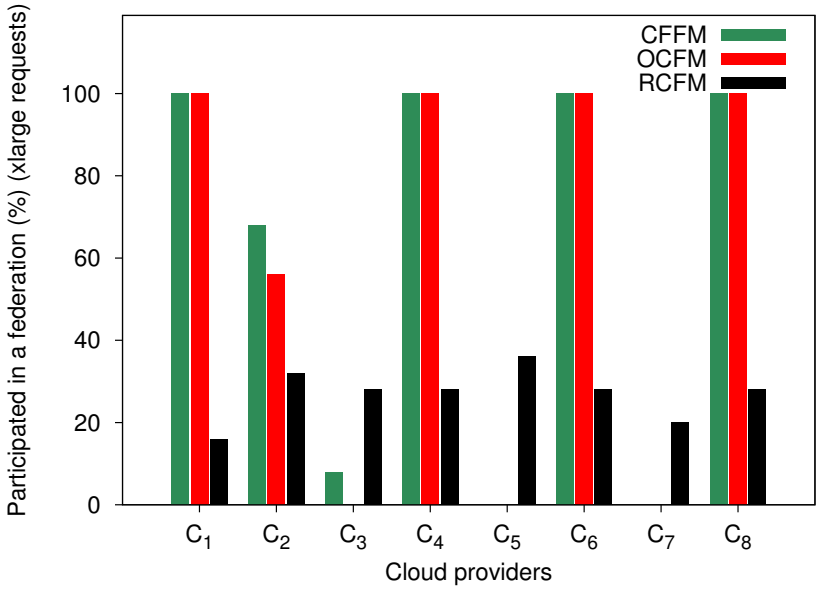


Figure 5.11: Percentage participation of cloud providers (extralarge requests)

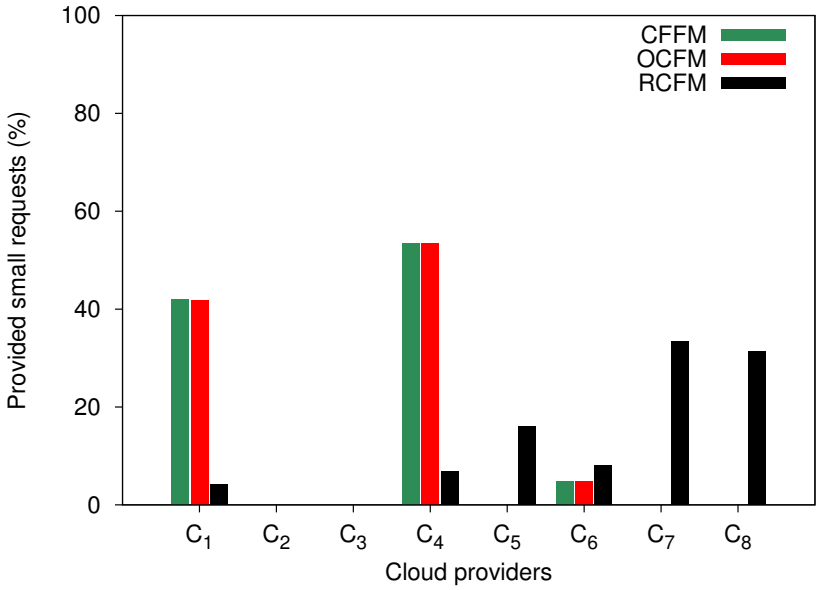


Figure 5.12: Percentage of request provided by cloud providers (small requests)

with larger capacity are needed. The next lowest cost cloud provider is $\mathcal{C}_8$, and thus, $\mathcal{C}_8$ is included in the federations formed by CFFM and OCFM to provide the requested VMs. Note that the percentage of participation of cloud providers in the federation obtained by RCFM is distributed over all cloud providers. This is due to the fact that RCFM selects

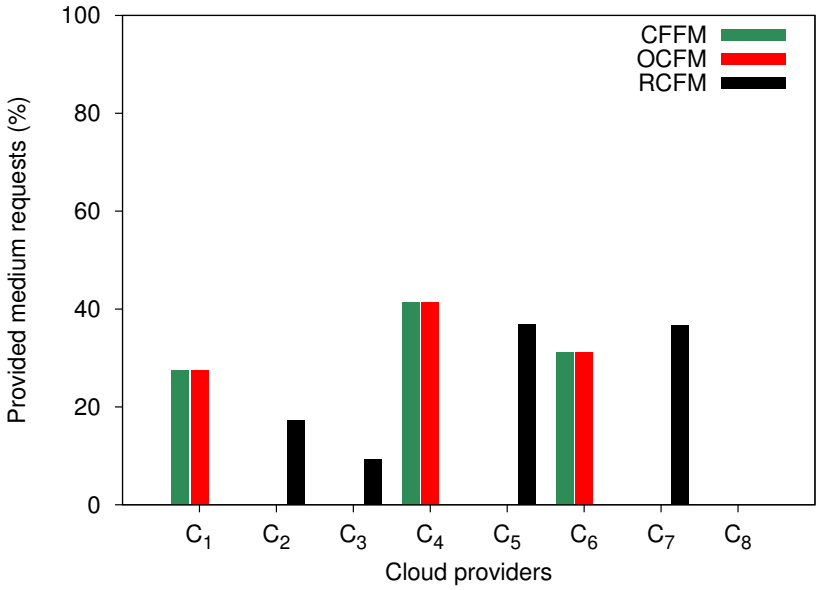


Figure 5.13: Percentage of request provided by cloud providers (medium requests)

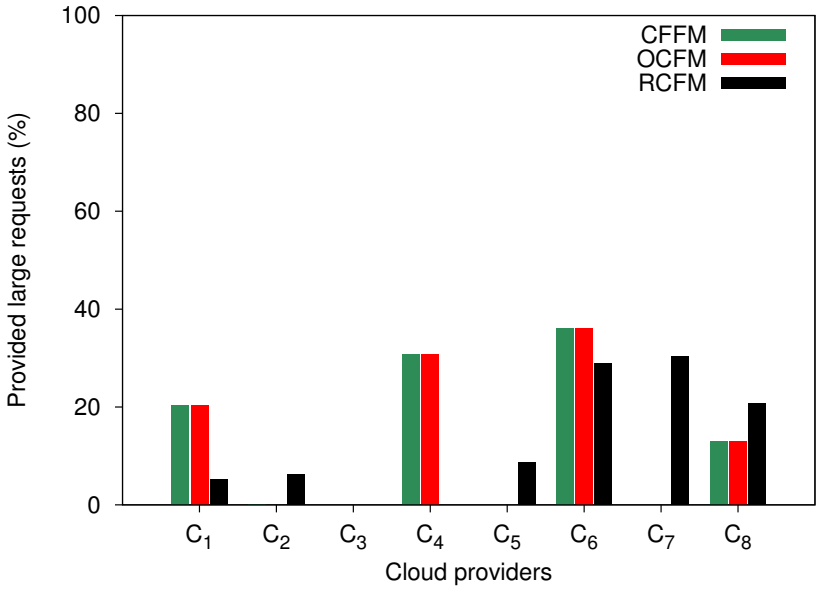


Figure 5.14: Percentage of request provided by cloud providers (large requests)

the participating cloud providers in a federation randomly without considering their cost.

Figs. 5.12 to 5.15 show the percentage of requested cores provided to the federation by each cloud provider. For example, Fig. 5.12 shows that using CFFM and OCFM, 41.8% of the small requests are provided by $\mathcal{C}_1$, 53.4% of are provided by $\mathcal{C}_4$, and 4.8% are provided

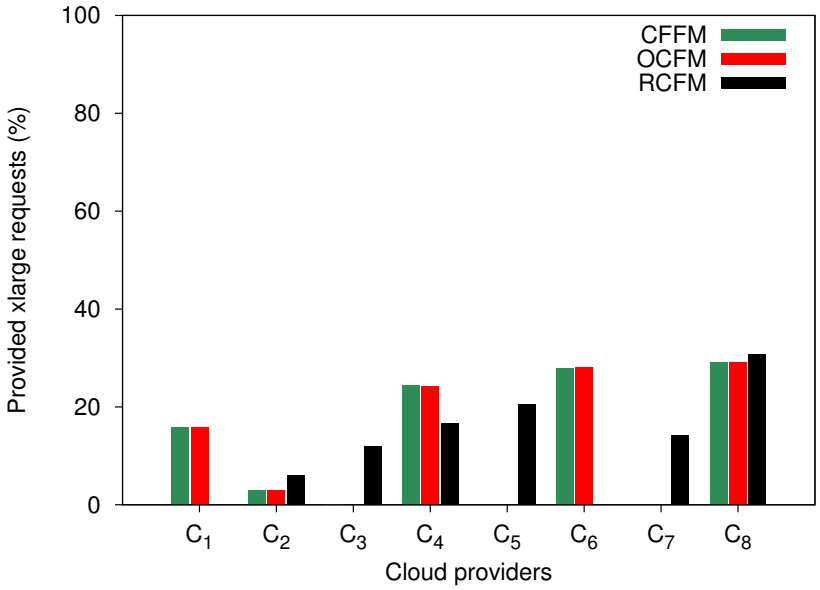


Figure 5.15: Percentage of request provided by cloud providers (extralarge requests)

by $\mathcal{C}_6$. However, RCFM selects the cloud providers randomly to provide VMs for the small requests, and thus, the percentage of provided requests by cloud providers is far from the optimal solution. As shown in Fig. 5.13, the percentage of provided requests by $\mathcal{C}_1$, $\mathcal{C}_4$, and $\mathcal{C}_6$ changes to 27.5%, 41.3%, and 31.2% , respectively. The results show that with the increase in the size of the requests, the percentage of requests provided by $\mathcal{C}_6$ increases. This is due to the fact that more resources are needed to fulfill the demand of medium requests, and thus, $\mathcal{C}_6$ needs to provide more resources in the federation. Note that the amount of provided resources depends on the available capacity of the cloud providers.

From the above results, we conclude that our proposed mechanism, CFFM, is able to form stable federations with total profit very close to the optimal profit. In addition, CFFM finds the results in reasonable amount of time making it suitable for real cloud settings.

5.5 Conclusion

In this chapter, we proposed a mechanism that improves the cloud providers' dynamic resource scaling capabilities to fulfill users' demands. We proposed a cloud federation formation game that characterizes the process of federation formation and then proposed

a novel cloud federation formation mechanism called CFFM. In the proposed mechanism, cloud providers dynamically cooperate to form a federation in order to provide the requested resources to a user. The resources are provisioned as VM instances of different types. The proposed mechanism forms cloud federations yielding the highest total profit. The mechanism also determines the individual profit of each participating cloud providers in the federation using the normalized estimated Banzhaf value. In addition, our proposed mechanism produces a stable cloud federation structure, that is, the participating cloud providers in the federation do not have incentives to break away from the federation. We performed extensive experiments to investigate the properties of our proposed mechanism. The results showed that our proposed mechanism is able to form stable federations with total profit very close to the optimal profit. In addition, our mechanism finds the stable cloud federation in a reasonable amount of time making it suitable for real cloud settings.

CHAPTER 6: ENERGY-AWARE SCHEDULING OF MAPREDUCE JOBS

6.1 Introduction

Several businesses and organizations are faced with an ever-growing need for analyzing the unprecedented amounts of available data. Such need challenges existing methods, and requires novel approaches and technologies in order to cope with the complexities of big data processing. One of the major challenges of processing data intensive applications is minimizing their energy costs. Electricity used in US data centers in 2010 accounted for about 2% of total electricity used nationwide [76]. In addition, the energy consumed by the data centers is growing at over 15% annually, and the energy costs make up about 42% of the data centers' operating costs [56]. Considering that server costs are consistently falling, it should be no surprise that in the near future a big percentage of the data centers' costs will be energy costs. Therefore, it is critical for the data centers to minimize their energy consumption when offering services to customers.

Big data applications run on large clusters within data centers, where their energy costs make energy efficiency of executing such applications a critical concern. MapReduce [36] and its open-source implementation, Hadoop [6], have emerged as the leading computing platforms for big data analytics. For scheduling multiple MapReduce jobs, Hadoop originally employed a FIFO scheduler. To overcome the issues with the waiting time in FIFO, Hadoop then employed the Fair Scheduler [190]. These two schedulers, however, do not consider improving the energy efficiency when executing MapReduce applications. Improving energy efficiency of MapReduce applications leads to a significant reduction of the overall cost of data centers. In this chapter, we design MapReduce scheduling algorithms that improve the energy efficiency of running each individual application, while satisfying the service level agreement (SLA). Our proposed scheduling algorithms can be easily incorporated and deployed within the existing Hadoop systems.

In most of the cases, processing big data involves running production jobs periodically. For example, Facebook processes terabytes of data for spam detection daily. Such produc-

tion jobs allow data centers to use job profiling techniques in order to get information about the resource consumption for each job. Job profiling extracts critical performance characteristics of map and reduce tasks for each underlying application. Data centers can use the knowledge of extracted job profiles to pre-compute new estimates of jobs' map and reduce stage durations, and then construct an optimized schedule for future executions. Furthermore, the energy consumption of each task on a machine can be profiled using automatic power-meter tools such as PDU Power Strip [4], which is currently a standard practice in data centers. Many researchers studied different profiling techniques [142, 170], and several MapReduce scheduling studies rely on such techniques [136, 171]. Our proposed algorithms schedule MapReduce production jobs having as the primary objective the minimization of energy consumption.

Most of the existing research on MapReduce scheduling focused on improving the makespan (i.e., minimizing the time between the arrival and the completion time of an application) of the MapReduce job's execution (e.g., [30, 33, 117, 200]). However, makespan minimization is not necessarily the best strategy for data centers. Data centers are obligated to deliver the services by their specified deadlines, and it is not in their best interests to execute the services as fast as they can in order to minimize the makespan. This strategy fails to incorporate significant optimization opportunities available for data centers to reduce their energy costs. The majority of production MapReduce workloads consists of a large number of jobs that do not require fast execution. By taking into account the energy consumed by the map and reduce tasks when making scheduling decisions, the data centers can utilize their resources efficiently and reduce the energy consumption. Our proposed energy-aware scheduling algorithms capture such opportunities and significantly reduce the MapReduce energy costs, while satisfying the SLA.

6.1.1 Our Contribution

To the best of our knowledge this is the first study that designs algorithms for *detailed task placement* of a MapReduce job to machines with the primary focus on minimizing the energy consumption. Our proposed algorithms can be incorporated into higher level energy

management policies in data centers. We first model the problem of scheduling MapReduce tasks for energy efficiency as an integer program. In the absence of computationally tractable optimal algorithms for solving this problem, we design two heuristic algorithms, called EMRSA-I and EMRSA-II, where EMRSA is an acronym for Energy-aware MapReduce Scheduling Algorithm. EMRSA-I and EMRSA-II provide very fast solutions making them suitable for deployment in real production MapReduce clusters. The time complexity of the proposed algorithms is polynomial in the number of map and reduce slots, the number of map tasks, and the number of reduce tasks. We perform experiments on a Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications such as TeraSort, Page Rank, and K-means Clustering. We use this data in an extensive simulation study to characterize the performance of the proposed algorithms. We show that the current practice scheduling methods, such as makespan minimization, produce schedules having energy consumption that is far from optimal. We compare the performance of EMRSA-I and EMRSA-II against the optimal solution for cases in which the optimal solution can be obtained in reasonable amount of time. The results show that EMRSA-I and EMRSA-II are capable of finding close to optimal solutions very fast. Due to the intractability of the problem, when the optimal results are not available, we show that the energy consumption for the schedules obtained by the proposed algorithms is very close to the lower bound solution obtained by the linear programming (LP) relaxation of the integer program.

6.1.2 Related Work

We summarize the related work from three perspectives: resource allocation and scheduling in data centers and clouds, MapReduce scheduling with different objectives, and energy savings in data centers.

Resource allocation and scheduling in data centers and clouds. Hacker and Mahadik [53] proposed scheduling polices for virtual high performance computing clusters. They presented a resource prediction model for each policy to estimate the resources needed within a cloud, the queue wait time for requests, and the size of the pool of spare resources needed.

Palanisamy et al. [135] proposed a new MapReduce cloud service model for production jobs. Their method creates cluster configurations for the jobs using MapReduce profiling and leverages deadline-awareness, allowing the cloud provider to optimize its global resource allocation and reduce the cost of resource provisioning. Ekanayake et al. [40] proposed a programming model and an architecture to enhance MapReduce runtime that supports iterative MapReduce computations efficiently. They showed how their proposed model can be extended to more classes of applications for MapReduce. Tian and Chen [165] proposed a cost function that models the relationship between the amount of input data, Map and Reduce slots, and the complexity of the Reduce function for the MapReduce job. Their proposed cost function can be used to minimize the cost with a time deadline or minimize the time under certain budget. Zhan et al. [194] proposed a cooperative resource provisioning solution using statistical multiplexing to save the server cost. Song et al. [162] proposed a two-tiered on-demand resource allocation mechanism consisting of the local and global resource allocation. In our previous studies [99, 104, 124], we proposed mechanisms for resource provisioning, allocation, and pricing in clouds considering several heterogeneous resources. However, none of the above mentioned studies consider the energy saving objectives.

MapReduce scheduling with different objectives. Zaharia et al. [191] studied the problem of speculative execution in MapReduce. They proposed a simple robust scheduling algorithm, Longest Approximate Time to End (LATE), which uses estimated finish times to speculatively execute the tasks that hurt the response time the most. Sandholm and Lai [155] designed a system for allocating resources in shared data and compute clusters that improves MapReduce job scheduling. Their approach is based on isolating MapReduce clusters in VMs with a continuously adjustable performance. Wang et al. [176] proposed a task scheduling technique for MapReduce that improves the system throughput in job-intensive environments without considering the energy consumption. Ren et al. [148] proposed a job scheduling algorithm to optimize the completion time of small MapReduce jobs. Their approach extends job priorities to guarantee the rapid response for small jobs. Chang et al. [30] proposed various online and offline algorithms for the MapReduce scheduling prob-

lem to minimize the overall job completion times. Their algorithms are based on solving a linear program (LP) relaxation. Moseley et al. [117] proposed a dynamic program for minimizing the makespan when all MapReduce jobs arrive at the same time. They modeled the problem as a two-stage flow shop problem, and proved that the dynamic program yields a PTAS if there is a fixed number of job-types. Pastorelli et al. [141] proposed a size-based approach to scheduling jobs in Hadoop to guarantee fairness and near-optimal system response times. Their scheduler requires a priori job size information, and thus, it builds such knowledge by estimating the sizes during job execution. Wolf et al. [182] proposed a flexible scheduling allocation scheme, called Flex, to optimize a variety of standard scheduling metrics such as response time and makespan, while ensuring the same minimum job slot guarantees as in the case of Fair scheduler. Sandholm and Lai [156] proposed a dynamic priority parallel task scheduler for Hadoop that prioritizes jobs and users and gives users the tool to optimize and customize their allocations to fit the importance and requirements of their jobs such as deadline and budget. Verma et al. [170] proposed a job scheduler for MapReduce environments that allocates the resources to production jobs. Their method can profile a job that runs routinely and then uses its profile in the designed MapReduce model to estimate the amount of resources required for meeting the deadline. Verma et al. [171] proposed a job scheduler that minimizes the makespan for MapReduce production jobs with no dependencies by utilizing the characteristics and properties of the jobs in a given workload. Nanduri et al. [120] proposed a heuristic scheduling algorithm to maintain a resource balance on a cluster, thereby reducing the overall runtime of the MapReduce jobs. Their job selection and assignment algorithms select the job that is best suitable on a particular node while avoiding node overloads. Ibrahim et al. [65] proposed a scheduling algorithm for map tasks to improve the overall performance of the MapReduce computation. Their approach leads to a higher locality in the execution of map tasks and to a more balanced intermediate data distribution. Kurazumi et al. [78] proposed dynamic processing slot scheduling for I/O intensive MapReduce jobs that use efficiently the CPU resources with low utilization caused by I/O wait related to task execution. However, these studies did not consider energy efficiency as their objectives.

Energy savings in data centers. Kaushik et al. [70] proposed an approach to partition the servers in a Hadoop cluster into hot and cold zones based on their performance, cost, and power characteristics, where hot zone servers are always powered on and cold zone servers are mostly idling. Cardosa et al. [26] proposed a spatio-temporal tradeoff that includes efficient spatial placement of tasks on nodes and temporal placement of nodes with tasks having similar runtimes in order to maximize utilization. Leverich and Kozyrakis [84] proposed a method for energy management of MapReduce jobs by selectively powering down nodes with low utilization. Their method uses a cover set strategy that exploits the replication to keep at least one copy of a data-block. As a result, in low utilization periods some of the nodes that are not in the cover set can be powered down. Chen et al. [34] proposed a method for reducing the energy consumption of MapReduce jobs without relying on replication. Their approach divides the jobs into time-sensitive and less time-sensitive jobs, where the former are assigned to a small pool of dedicated nodes, and the latter can run on the rest of the cluster. Maheshwari et al. [87] proposed an algorithm that dynamically reconfigures clusters by scaling up and down the number of nodes based on the cluster utilization. Lang and Patel [80] proposed a framework for energy management in MapReduce clusters by powering down all nodes in the cluster during a low utilization period. Wirtz and Ge [181] conducted an experimental study on the MapReduce efficiency. They analyzed the effects of changing the number of concurrent worker nodes, and the effects of adjusting the processor frequency based on workloads. Goiri et al. [50] proposed a MapReduce framework for a data center powered by renewable sources of energy such as solar or wind, and by the electrical grid for backups. Their proposed framework schedules jobs to maximize the green energy consumption by delaying many background computations within the jobs' bounded time. Salehi et al. [153] proposed an adaptive energy management policy employing a fuzzy reasoning engine to determine if the resources for a request have to be allocated through switching on resources, preemption, consolidation, or a combination of these. Shen and Wang [160] formulated several stochastic optimization models to investigate the trade-off between energy footprints and quality of service in cloud computing services. In their models, decisions include workload

scheduling and switching servers on/off based on loads. While the above frameworks can be used as data center-level energy minimization strategies, our focus is on minimizing the energy consumption by scheduling jobs, which can be considered as a cluster-level strategy in data centers. In addition, none of the above frameworks and systems exploit the job profiling information when making the decisions for task placement on the nodes to increase the energy efficiency of executing MapReduce jobs. Our proposed algorithms consider the significant energy consumption differences of different task placements on machines, and find an energy efficient assignment of tasks to machines.

6.1.3 Organization

The rest of the chapter is organized as follows. In Section 6.2, we describe the problem of scheduling MapReduce jobs for energy efficiency. In Section 6.3, we present our proposed algorithms. In Section 6.4, we evaluate the algorithms by extensive experiments. In Section 6.5, we summarize our results.

6.2 Energy-aware MapReduce Scheduling Problem

A MapReduce job comprising a specific number of map and reduce tasks is executed on a cluster composed of multiple machines. The job's computation consists of a map phase followed by a reduce phase. In the map phase, each map task is allocated to a map slot on a machine, and processes a portion of the input data producing key-value pairs. In the reduce phase, the key-value pairs with the same key are then processed by a reduce task allocated to a reduce slot. As a result, the reduce phase of the job cannot begin until the map phase ends. At the end, the output of the reduce phase is written back to the distributed file system. In Hadoop, job scheduling is performed by a master node running a job tracker process, which distributes jobs to a number of worker nodes in the cluster. Each worker runs a task tracker process, and it is configured with a fixed number of map and reduce slots. The task tracker periodically sends heartbeats to the job tracker to report the number of free slots and the progress of the running tasks.

We consider a big data application consisting of a set of M map and R reduce tasks that needs to be completed by deadline D. Tasks in each set can be run in parallel, but no reduce task can be started until all map tasks for the application are completed. Let $\mathcal{M}$ and $\mathcal{R}$ be the set of map and reduce tasks of the application, and $\mathcal{A}$ and $\mathcal{B}$ the set of slots on heterogeneous machines available for executing the map and the reduce tasks, respectively. The number of slots for each machine is decided by the system administrators when the Hadoop cluster is setup and each slot can handle only one map or reduce task at a time. Since we consider a heterogeneous cluster, the execution speed of a task on different slots from different machines may not be the same. Also, the energy required to execute a task on different slots may not be the same. We denote by e_{ij} the difference between energy consumption of slot $j \in \{\mathcal{A}, \mathcal{B}\}$ when executing task $i \in \{\mathcal{M}, \mathcal{R}\}$ and its idle energy consumption. In addition, we denote by p_{ij} the processing times of task $i \in \{\mathcal{M}, \mathcal{R}\}$ when executed on slot $j \in \{\mathcal{A}, \mathcal{B}\}$. We assume that the processing time of the tasks are known. In doing so, we use the knowledge of extracted job profiles to pre-compute the processing time of map and reduce tasks, along with their energy consumption. We define an indicator variable δ_{ti}, $\forall t, i \in \mathcal{M} \cup \mathcal{R}$, characterizing the dependencies of the map and reduce tasks as follows:

$$\delta_{ti} = \begin{cases} 1 & \text{if task } i \text{ should be assigned after task } t \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

We formulate the Energy-aware MapReduce Scheduling problem as an Integer Program

(called EMRS-IP), as follows:

$$\text{Minimize} \sum_{j\in\mathcal{A}}\sum_{i\in\mathcal{M}} e_{ij}X_{ij} + \sum_{j\in\mathcal{B}}\sum_{i\in\mathcal{R}}\sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti}e_{ij}Y_{ij} \tag{6.2}$$

Subject to:

$$\sum_{j\in\mathcal{A}} X_{ij} = 1, \forall i \in \mathcal{M} \tag{6.3}$$

$$\sum_{j\in\mathcal{B}}\sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti}Y_{ij} = 1, \forall i \in \mathcal{R} \tag{6.4}$$

$$\sum_{i\in\mathcal{M}} p_{ij}X_{ij} + \sum_{i\in\mathcal{R}}\sum_{t\in\mathcal{M}\cup\mathcal{R}} \delta_{ti}p_{ij'}Y_{ij'} \leq D, \quad \forall j \in \mathcal{A}, \forall j' \in \mathcal{B} \tag{6.5}$$

$$X_{ij} = \{0,1\}, \forall i \in \mathcal{M},\ \forall j \in \mathcal{A} \tag{6.6}$$

$$Y_{ij} = \{0,1\}, \forall i \in \mathcal{R},\ \forall j \in \mathcal{B} \tag{6.7}$$

where the decision variables X_{ij} and Y_{ij} are defined as follows:

$$X_{ij} = \begin{cases} 1 & \text{if map task } i \text{ is assigned to slot } j \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

$$Y_{ij} = \begin{cases} 1 & \text{if reduce task } i \text{ is assigned to slot } j \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

The objective function is to minimize the energy consumed when executing the MapReduce application considering the dependencies of reduce tasks on the map tasks. Constraints (6.3) ensure that each map task is assigned to a slot for execution. Constraints (6.4) ensure that each reduce task is assigned to a slot. Constraints (6.5) ensure that processing time of the application does not exceed its deadline. Constraints (6.6) and (6.7) represent the integrality requirements for the decision variables. The solution to EMRS-IP consists

of X and $\hat{Y}$, where $\hat{Y}_{ij} = \sum_{t \in \mathcal{M} \cup \mathcal{R}} \delta_{ti} Y_{ij}$, $i \in \mathcal{R}$, and $j \in \mathcal{B}$.

Note that based on constraints (6.5), the scheduler can assign all reduce tasks after finishing all map tasks without exceeding the deadline. This is due to the fact that these constraints can be interpreted as $\max_{\forall j \in \mathcal{A}} \sum_{i \in \mathcal{M}} p_{ij} X_{ij} + \max_{\forall j' \in \mathcal{B}} \sum_{i \in \mathcal{R}} p_{ij'} Y_{ij'} \leq D$. As a result, all reduce tasks can be assigned after time $\max_{\forall j \in \mathcal{A}} \sum_{i \in \mathcal{M}} p_{ij} X_{ij}$. In addition, the scheduler can assign multiple map tasks to a machine, as well as multiple reduce tasks. This is due to the fact that in bigdata applications the number of tasks is greater than the number of machines available in a cluster. The focus of this study is the detailed placement of map and reduce tasks of a job in order to reduce energy consumption. While it is important to consider data placement in an integrated framework for energy savings in data centers, data placement is beyond the scope of this study.

At the high level the problem we consider may appear as composed of two independent scheduling problems, one for the map tasks and one for the reduce tasks. This would be the case if the deadline for the map phase would be known. But since the deadline for map tasks is not known from the beginning, we cannot just simply divide the problem into two scheduling subproblems and solve them independently. Our proposed algorithms determine the map deadline as the tasks are allocated and schedule the map and reduce tasks to reduce the energy consumption of executing the job.

6.3 Energy-aware MapReduce Scheduling Algorithms

We design two heuristic algorithms called EMRSA-I and EMRSA-II for solving the energy-aware MapReduce scheduling problem. Our proposed algorithms, EMRSA-I and EMRSA-II, take the energy efficiency differences of different machines into account and determine a detailed task placement of a MapReduce job into slots while satisfying the user specified deadline. The two algorithms are presented as a single generic algorithm called EMRSA-X, in Algorithm 18.

The design of these algorithms require a metric that characterizes the energy consumption of each machine and induces an order relation among the machines. We define such a metric, called *energy consumption rate* of a slot j. EMRSA-I and EMRSA-II use different

energy consumption rate metrics as follows:

1) EMRSA-I uses energy consumption rate metrics based on the minimum ratio of energy consumption and processing time of tasks when executed on slot j, as follows:

$$ecr_j^m = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{A} \tag{6.10}$$

$$ecr_j^r = \min_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}, \forall j \in \mathcal{B} \tag{6.11}$$

where ecr_j^m and ecr_j^r represent the energy consumption rate of map slot j and reduce slot j, respectively.

2) EMRSA-II uses energy consumption rate metrics based on the average ratio of energy consumption and processing time of tasks when executed on slot j, as follows:

$$ecr_j^m = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}, \forall j \in \mathcal{A} \tag{6.12}$$

$$ecr_j^r = \frac{\sum_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}, \forall j \in \mathcal{B} \tag{6.13}$$

The ordering induced by these metrics on the set of slots determines the order in which the slots are assigned to tasks, that is, a lower ecr_j^m means that slot j has a higher priority to have a map task assigned to it. Similarly, a lower ecr_j^r means that slot j has a higher priority to have a reduce task assigned to it.

In addition, EMRSA-X uses the ratio of map and reduce processing times, denoted by f, in order to balance the assignment of map and reduce tasks. The ratio f is defined as follows:

$$f = \frac{\sum_{\forall i \in \mathcal{M}} p_{ij^m}}{\sum_{\forall i \in \mathcal{R}} p_{ij^r}} \tag{6.14}$$

This ratio is used in the task assignment process in each iteration of EMRSA-X. As we

already mentioned, we use job profiling of production jobs to estimate the processing time of map and reduce tasks. This information, extracted from job profiling (i.e., the values of p_{ij^m} and p_{ij^r}) is used by EMRSA-X to compute the ratio f.

A key challenge when designing the algorithms is that the user only specifies the deadline for the job and there is no information on the deadline for completing the map phase. However, since the reduce tasks are dependent on the map tasks, the algorithms have to determine a reasonable deadline for the map tasks with respect to the availability of the map slots in the cluster in order to utilize its resources efficiently. Our proposed algorithms find the assignments of map tasks to the map slots satisfying the determined map deadline, and then find the assignments of reduce tasks to the reduce slots satisfying the deadline D, where all the reduce tasks start after the map deadline.

First, EMRSA-X determines the assignment of large tasks in terms of their processing time, and the map deadline according to such tasks. The reason that EMRSA-X gives priority to large tasks is due to the hard deadline constraint, and the fact that there may not be many choices for large task placement configurations to avoid exceeding the deadline constraint. Then, EMRSA-X tries to close the optimality gap by filling with smaller tasks the leftover time of each slot based on the deadline. This leads to better utilization of each machine in the cluster.

EMRSA-X is given in Algorithm 18. EMRSA-X builds two priority queues $\mathcal{Q}^m$ and $\mathcal{Q}^r$ to keep the order of the map and reduce slots based on their energy consumption rates (lines 1-8). Then, it initializes the deadlines for map tasks, D^m, and reduce tasks, D^r, to infinity. In each iteration of the while loop, the algorithm chooses the slots with the lowest energy consumption rates (i.e., j^m and j^r) from the priority queues, and finds the task placement on the selected slots. For these slots, the ratio of processing time of map tasks to that of the reduce tasks, denoted by f, is calculated (line 13). Then, EMRSA-X sorts the unassigned map and reduce tasks, if there is any, based on their processing time on the selected slots (lines 14-15). Then, it determines the assignments of large tasks based on the metric f by calling ASSIGN-LARGE() (given in Algorithm 19). Then, it finds the assignments of small tasks by calling ASSIGN-SMALL() (given in Algorithm 20) if there is

Algorithm 18 EMRSA-X

1: Create an empty priority queue $\mathcal{Q}^m$
2: Create an empty priority queue $\mathcal{Q}^r$
3: **for all** $j \in \mathcal{A}$ **do**
4: $ecr_j^m = \min_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}$, for EMRSA-I; or
$ecr_j^m = \frac{\sum_{\forall i \in \mathcal{M}} \frac{e_{ij}}{p_{ij}}}{M}$, for EMRSA-II
5: $\mathcal{Q}^m$.enqueue(j, ecr_j^m)
6: **for all** $j \in \mathcal{B}$ **do**
7: $ecr_j^r = \min_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}$, for EMRSA-I; or
$ecr_j^r = \frac{\sum_{\forall i \in \mathcal{R}} \frac{e_{ij}}{p_{ij}}}{R}$, for EMRSA-II
8: $\mathcal{Q}^r$.enqueue(j, ecr_j^r)
9: $D^m \leftarrow \infty$; $D^r \leftarrow \infty$
10: **while** $\mathcal{Q}^m$ is not empty **and** $\mathcal{Q}^r$ is not empty **do**
11: $j^m = \mathcal{Q}^m$.extractMin()
12: $j^r = \mathcal{Q}^r$.extractMin()
13: $f = \frac{\sum_{\forall i \in \mathcal{M}} p_{ij^m}}{\sum_{\forall i \in \mathcal{R}} p_{ij^r}}$
14: $\mathcal{T}^m$: sorted unassigned map tasks $i \in \mathcal{M}$ based on p_{ij^m}
15: $\mathcal{T}^r$: sorted unassigned reduce tasks $i \in \mathcal{R}$ based on p_{ij^r}
16: **if** $\mathcal{T}^m = \emptyset$ **and** $\mathcal{T}^r = \emptyset$ **then break**
17: ASSIGN-LARGE()
18: ASSIGN-SMALL()
19: **if** $D^m = \infty$ **then**
20: $D^m = D - p^r$
21: $D^r = p^r$
22: **if** $\mathcal{T}^m \neq \emptyset$ **or** $\mathcal{T}^r \neq \emptyset$ **then**
23: No feasible schedule
24: **return**
25: **Output:** X, Y

any unallocated processing time on a slot. EMRSA-X assigns a new task to a slot whenever the slot becomes available. At the end of the first iteration, the algorithm sets the map and reduce deadlines based on the allocated tasks (lines 19-21).

We now describe the two procedures, ASSIGN-LARGE() and ASSIGN-SMALL() into more details. ASSIGN-LARGE() is given in Algorithm 19. ASSIGN-LARGE() selects the longest map task i^m and reduce task i^r from the sorted sets $\mathcal{T}^m$ and $\mathcal{T}^r$, respectively (lines 1-2). Then it checks the feasibility of allocating map task i^m to slot j^m and reduce task i^r to slot j^r by checking the total processing time of the tasks against the deadline D (line 4). If the assignment of map task i^m and reduce task i^r is feasible, the algorithm continues

Algorithm 19 ASSIGN-LARGE()

1: $i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
2: $i^r = \text{argmax}_{t \in \mathcal{T}^r} p_{tj^r}$
3: $p^m = 0$; $p^r = 0$
4: **if** $p_{i^m j^m} + p_{i^r j^r} \leq D$ **and** $p_{i^m j^m} \leq D^m$ **and** $p_{i^r j^r} \leq D^r$ **then**
5: $\quad \mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
6: $\quad \mathcal{T}^r = \mathcal{T}^r \setminus \{i^r\}$
7: $\quad p^m = p_{i^m j^m}$
8: $\quad p^r = p_{i^r j^r}$
9: $\quad X_{i^m j^m} = 1$
10: $\quad Y_{i^r j^r} = 1$
11: $\quad$ **do**
12: $\quad\quad i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
13: $\quad\quad i^r = \text{argmax}_{t \in \mathcal{T}^r} p_{tj^r}$
14: $\quad\quad$ **if** $f > 1$ **then**
15: $\quad\quad\quad$ **while** $\frac{p^m + p_{i^m j^m}}{p^r} < f$ **and** $p^m + p^r + p_{i^m j^m} \leq D$ **and** $p^m + p_{i^m j^m} \leq D^m$ **and** $\mathcal{T}^m \neq \emptyset$ **do**
16: $\quad\quad\quad\quad \mathcal{T}^m = \mathcal{T}^m \setminus \{i^m\}$
17: $\quad\quad\quad\quad p^m = p^m + p_{i^m j^m}$
18: $\quad\quad\quad\quad X_{i^m j^m} = 1$
19: $\quad\quad\quad\quad i^m = \text{argmax}_{t \in \mathcal{T}^m} p_{tj^m}$
20: $\quad\quad\quad$ Balance the assignment of reduce tasks (repeat lines 15-19 for reduce tasks).
21: $\quad\quad$ **else**
22: $\quad\quad\quad$ The code for $f < 1$ is similar to lines 15-20 and is not presented here.
23: $\quad$ **while** $p^m + p^r + p_{i^m j^m} + p_{i^r j^r} \leq D$ **and** $p^m + p_{i^m j^m} \leq D^m$ **and** $p^r + p_{i^r j^r} \leq D^r$ **and** ($\mathcal{T}^m \neq \emptyset$ **or** $\mathcal{T}^r \neq \emptyset$)

Algorithm 20 ASSIGN-SMALL()

1: {Assign small map tasks}
2: $i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
3: **while** $p^m + p^r + p_{ij^m} \leq D$ **and** $p^m + p_{ij^m} \leq D^m$ **and** $\mathcal{T}^m \neq \emptyset$ **do**
4: $\quad \mathcal{T}^m = \mathcal{T}^m \setminus \{i\}$
5: $\quad p^m = p^m + p_{ij^m}$
6: $\quad X_{ij^m} = 1$
7: $\quad i = \text{argmin}_{t \in \mathcal{T}^m} p_{tj^m}$
8: {Assign small reduce tasks}
9: $i = \text{argmin}_{t \in \mathcal{T}^r} p_{tj^r}$
10: **while** $p^m + p^r + p_{ij^r} \leq D$ **and** $p^m + p_{ij^m} \leq D^r$ **and** $\mathcal{T}^r \neq \emptyset$ **do**
11: $\quad \mathcal{T}^r = \mathcal{T}^r \setminus \{i\}$
12: $\quad p^r = p^r + p_{ij^r}$
13: $\quad Y_{ij^r} = 1$
14: $\quad i = \text{argmin}_{t \in \mathcal{T}^r} p_{tj^r}$

to select tasks from $\mathcal{T}^m$ and $\mathcal{T}^r$, and updates the variables accordingly (lines 5-23). To keep the assignments of the tasks in alignment with the ratio of processing time f, the procedure balances the assignment. In doing so, if $f > 1$ (i.e., the load of processing time of map tasks is greater than that of reduce tasks) and the ratio of the current assignment is less than f, then the algorithm assigns more map tasks to balance the allocated processing time close to f (lines 15-20). If the ratio of the current assignment is greater than f, the procedure assigns more reduce tasks to balance the allocated processing time (lines 22).

After allocating the map and reduce tasks with the largest processing time, EMRSA-X assigns small map and reduce tasks while satisfying the deadline by calling ASSIGN-SMALL() (given in Algorithm 20). ASSIGN-SMALL() selects the smallest map task i, and based on the already assigned tasks and the remaining processing time of the slot, it decides if allocating task i is feasible or not (line 3). Then, it selects the smallest reduce task i, and checks the feasibility of its assignment (line 10).

The time complexity of EMRSA-X is $O(A(M+\log A)+B(R+\log B)+\min(A,B)(M\log M+R\log R))$, where A, B, M, and R are the number of map slots, the number of reduce slots, the number of map tasks, and the number of reduce tasks, respectively. The first two terms correspond to the running time of the two for loops in lines 4-5 and 6-8, while the third term corresponds to the running time of the while loop in lines 10-21.

6.3.1 Example

We now describe how EMRSA-II algorithm works by considering an example. We consider a job with 2 map tasks $\{t_1^m, t_2^m\}$ and 2 reduce tasks $\{t_1^r, t_2^r\}$ with a deadline of 12, and a data center with 3 map slots $\{a_1, a_2, a_3\}$ and 2 reduce slots $\{b_1, b_2\}$. The processing time and energy consumption of the map and reduce tasks are presented in Table 6.1 and Table 6.2, respectively. For example, task t_1^m has 8 units of processing time and 8 units of energy consumption if it runs on map slot a_1 (i.e., $p_{11} = 8$ and $e_{11} = 8$). Then, we have $ecr^m = \{1, 2.5, 4.5\}$ and $ecr^r = \{3, 1.5\}$ for the map and reduce slots. EMRSA-II determines $\mathcal{Q}^m = \{a_1, a_2, a_3\}$ and $\mathcal{Q}^r = \{b_2, b_1\}$ (Algorithm 1, lines 1-8). Based on the priority queues, $\mathcal{Q}^m$ and $\mathcal{Q}^r$, the first map slot to take into account is a_1, and the first

Table 6.1: Example: Map tasks.

		Map tasks					
		Processing time			Energy consumption		
		a_1	a_2	a_3	a_1	a_2	a_3
Tasks	t_1^m	8	4	2	8	12	12
	t_2^m	3	2	1	3	4	3

Table 6.2: Example: Reduce tasks.

		Reduce tasks			
		Processing time		Energy consumption	
		b_1	b_2	b_1	b_2
Tasks	t_1^r	2	3	6	3
	t_2^r	2	2	6	4

reduce slot is b_2. For these slots, the longest tasks are t_1^m and t_1^r, respectively (i.e., $X_{11} = 1$ and $Y_{12} = 1$). Based on the deadline, the algorithm cannot assign more tasks to these slots. Therefore, the deadlines for the map and reduce tasks are $D^m = 8$ and $D^r = 12 - 8 = 4$, respectively. That means, map tasks can be assigned to the other slots with the deadline of 8, and the reduce tasks can be assigned to the other slots from time 8 by the deadline of 12.

The map tasks assignment is as follows. So far we have $X_{11} = 1$, the algorithm chooses the second map slot in $\mathcal{Q}^m$, and finds the longest task that has not been assigned to any slot yet. That means t_2^m is assigned to a_2 (i.e., $X_{22} = 1$). For the reduce tasks assignment, we already have $Y_{12} = 1$. The algorithm chooses the second reduce slot in $\mathcal{Q}^r$, and finds the longest task that has not been assigned to any slot yet. That means t_2^r is assigned to b_1 (i.e., $Y_{21} = 1$). This solution leads to a total energy consumption of 21 units, while satisfying the deadline constraint. However, the solution that minimizes the makespan will select $X_{13} = 1$ and $X_{22} = 1$ to obtain a map makespan of 2 units, and will select $Y_{11} = 1$ and $Y_{22} = 1$ to obtain a reduce makespan of 2 units. This solution leads to a total makespan of 4 units with a total energy consumption of 26. Both approaches obtain schedules that meet the deadline. However, our proposed algorithm reduces the energy consumption by 19%. Note

that the makespan for our approach is 11.

6.4 Experimental Results

We perform extensive experiments in order to investigate the properties of the proposed algorithms, EMRSA-I and EMRSA-II. We compare the performance of EMRSA-I and EMRSA-II with that of OPT, where OPT obtains the optimal solution minimizing the energy consumption. OPT is obtained by optimally solving the EMRS-IP problem (Equations (6.2) to (6.7)). Since OPT cannot find the optimal solutions in several cases due to the intractability of the problem, we present the results of the linear programming (LP) relaxation of EMRS-IP by changing the binary decision variables into continuous decision variables (constraints (6.6) and (6.7)). The LP relaxation of EMRS-IP, called EMRS-LP, transforms an NP-hard optimization problem (EMRS-IP) into a related problem that is solvable in polynomial time. EMRS-LP gives a lower bound on the optimal solution of EMRS-IP by allowing partial assignments of each task to machines. Therefore, $OPT_{EMRS-LP} \leq OPT_{EMRS-IP}$, where $OPT_{EMRS-LP}$ is the optimal solution to EMRS-LP, and $OPT_{EMRS-IP}$ is the optimal solution to EMRS-IP.

However, such partial assignment of each task (obtained by solving the relaxation of EMRS-IP) is not a solution for the problem and cannot be used in practice. We only use the solution of the EMRS-LP (the relaxation of EMRS-IP) as a lower bound for EMRS-IP and compare it with the solutions obtained by the other algorithms. We denote by L-BOUND the algorithm that solves EMRS-LP and produces the lower bound on the solutions.

In addition, we present the results of minimizing the makespan, MSPAN, to show how far the current practice in MapReduce scheduling is from the optimal solutions that consider energy savings objectives. MSPAN is obtained by optimally solving the IP corresponding to the MapReduce makespan minimization problem (the same constraints as in ERMSA-IP, but the objective is makespan minimization). Since MSPAN cannot find the optimal solutions in several cases due to the intractability of the problem, we implemented a greedy algorithm for makespan minimization, called G-MSPAN. G-MSPAN schedules the tasks on the machines such that the processing time of all machines are balanced. It assigns longer

Table 6.3: Selected HiBench workloads.

Category	Workload
Micro Benchmarks	TeraSort
Web Search	Page Rank
Machine Learning	K-means Clustering

tasks to faster machines to keep the balance.

To analyze the performance of EMRSA-I and EMRSA-II, we present two classes of experiments, small-scale and large-scale. In the small-scale experiments, we compare the performance of EMRSA-I, EMRSA-II, OPT, and MSPAN for small MapReduce jobs. For large jobs, however, we cannot obtain the optimal results for OPT and MSPAN even after 24 hours, thus, we compare the performance of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN.

EMRSA-I, EMRSA-II, OPT, L-BOUND, MSPAN, and G-MSPAN algorithms are implemented in C++. OPT, MSPAN, and L-BOUND are implemented using APIs provided by IBM ILOG CPLEX Optimization Studio Multiplatform Multilingual eAssembly [7]. In this section, we describe the experimental setup and analyze the experimental results.

6.4.1 Experimental Setup

We performed extensive experiments on a Hadoop cluster of 64 processors and measured the energy and execution time for several MapReduce HiBench benchmark workloads [64]. HiBench is a comprehensive benchmark suite for Hadoop provided by Intel to characterize the performance of MapReduce based data analysis running in data centers. HiBench contains ten workloads, classified into four categories: Micro Benchmarks, Web Search, Machine Learning, and Analytical Query. We select three workloads, TeraSort, Page Rank, and K-means Clustering, from different categories as shown in Table 6.3. The cluster is composed of four Intel nodes, with one node as a master. Two of the nodes have 24GB memory, 16 2.4GHz Intel processors, and a 1TB Hard Drive. The other two nodes have 16GB memory, 16 2.4GHz Intel processors, and a 1TB Hard Drive. The cluster has a total

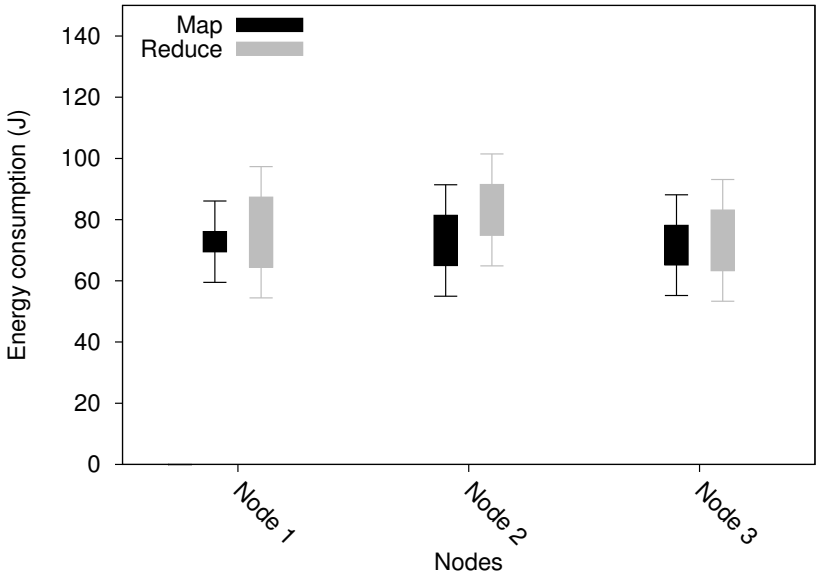


Figure 6.1: Energy needs of tasks for the actual and simulated architectures

of 80GB memory, 64 processors, 4TB of storage, and network speed of 1Gbps. We set one map slot and one reduce slot per processor. Energy measurements were taken using Wattsup? PRO ES.Net Power meter. The input voltage is 100-250 Volts at 60 HZ and the max wattage is 1800 Watts. The measurement accuracy is +/- 1.5% and the selected interval of time between records is one second.

We run and profiled several TeraSort, Page Rank, and K-means Clustering workloads from the HiBench benchmark set. Each workload contains both map and reduce tasks. For each workload, we collect its start time, finish time, the consumed power and other performance metrics. We used 240 workloads for job profiling. We run only one job at a time, and collect the energy measurements and execution times. Since the reduce tasks execute only after the execution of all map tasks is completed, we do not have overlaps between the map and reduce tasks. Based on the collected job profiles, we generated four small MapReduce jobs that we use in the small-scale experiments with the deadline of 250 seconds, and twenty four large MapReduce jobs, that we use in the large-scale experiments with the deadline of 1500 seconds. Since for production jobs the choice of the deadline is at the latitude of the users, we select the deadlines specifically to obtain feasible schedules. The execution time and the energy consumption of the map and reduce tasks composing these

Table 6.4: Terasort Workloads for the small scale experiments.

Workload	Map tasks	Reduce tasks
(48M, 48R)	48	48
(48M, 64R)	48	64
(64M, 48R)	64	48
(64M, 64R)	64	64

jobs were generated from uniform distributions having as the averages the average energy consumption and the average execution time of the map and reduce tasks extracted from the jobs profiled in our experiments. Fig. 6.1 shows the energy needs of map and reduce tasks for the actual and simulated architecture. The energy consumption range of each node is shown as a filled box, where the bottom and the top of the box represent the minimum and the maximum energy consumption, respectively. For the simulated architecture, the energy consumption of each node is generated within a range whose boundaries are represented in the figure as horizontal lines. The simulation experiments are conducted on AMD 2.93GHz hexa-core dual-processor systems with 90GB of RAM which are part of the Wayne State Grid System.

6.4.2 Analysis of Results

Small-scale experiments

We analyze the performance of EMRSA-I, EMRSA-II, OPT, and MSPAN for four small MapReduce TeraSort jobs with 10,737,418 records, where the number of map tasks and reduce tasks are presented in Table 6.4. For example, the smallest job represented by $(48M, 48R)$ has 48 map tasks and 48 reduce tasks. Fig. 6.2 presents the energy consumption of the jobs scheduled by the four algorithms we consider. The results show that EMRSA-I and EMRSA-II obtain the assignments of map and reduce tasks with energy consumption close to the optimal solution, obtained by OPT. OPT, EMRSA-I, and EMRSA-II are able to schedule the tasks with an average of 41.0%, 38.9%, and 39.2% less energy consumption than that of MSPAN, respectively. For example, the total energy consumptions for workload $(48M, 48R)$ obtained by EMRSA-I, EMRSA-II, OPT, and MSPAN are 5356, 5396,

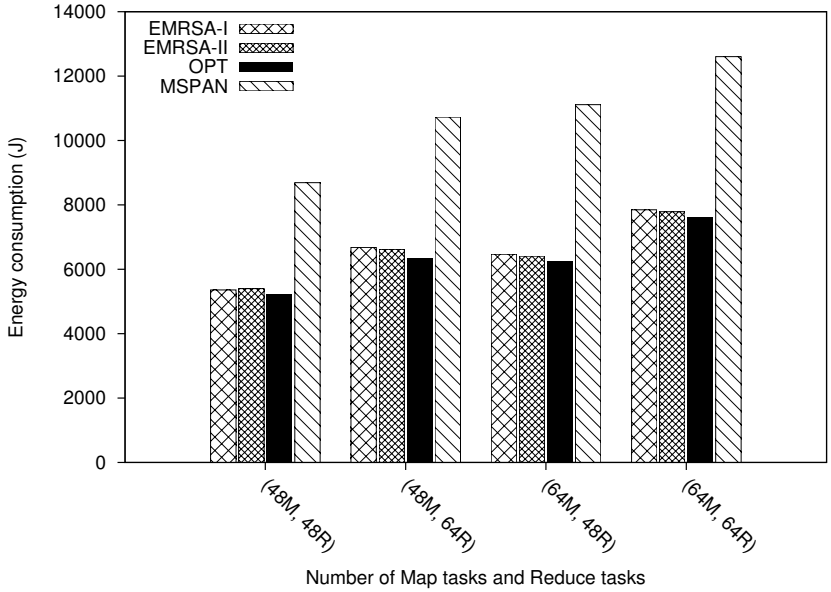


Figure 6.2: EMRSA-I and EMRSA-II performance on TeraSort: Energy consumption

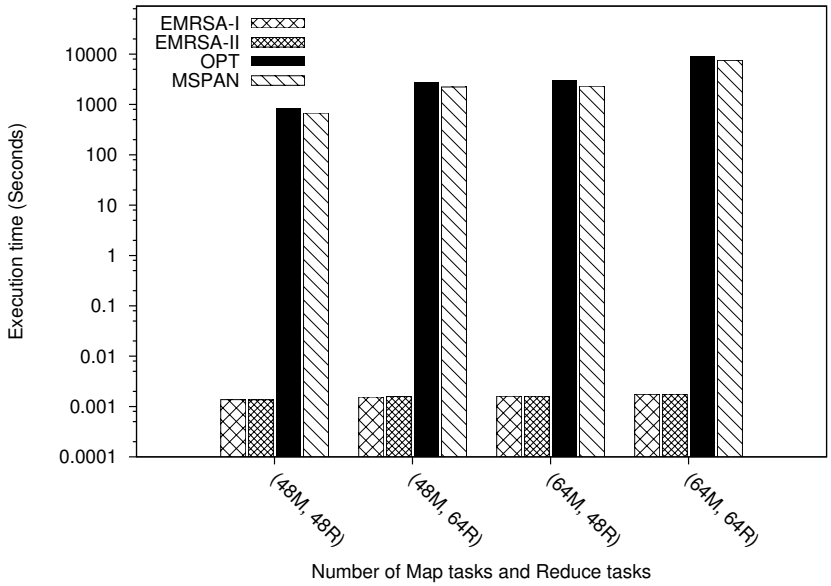


Figure 6.3: EMRSA-I and EMRSA-II performance on TeraSort: Execution time

5233, and 8687 J, respectively. While it is desirable to use OPT as a scheduler to reduce cost, the slow execution of OPT makes it prohibitive to use in practice. In addition, it is practically impossible to use OPT when it comes to scheduling big data jobs due to its prohibitive runtime. EMRSA-I and EMRSA-II are very fast and practical alternatives for scheduling big data jobs, leading to 39% reduction in energy consumption. However, the

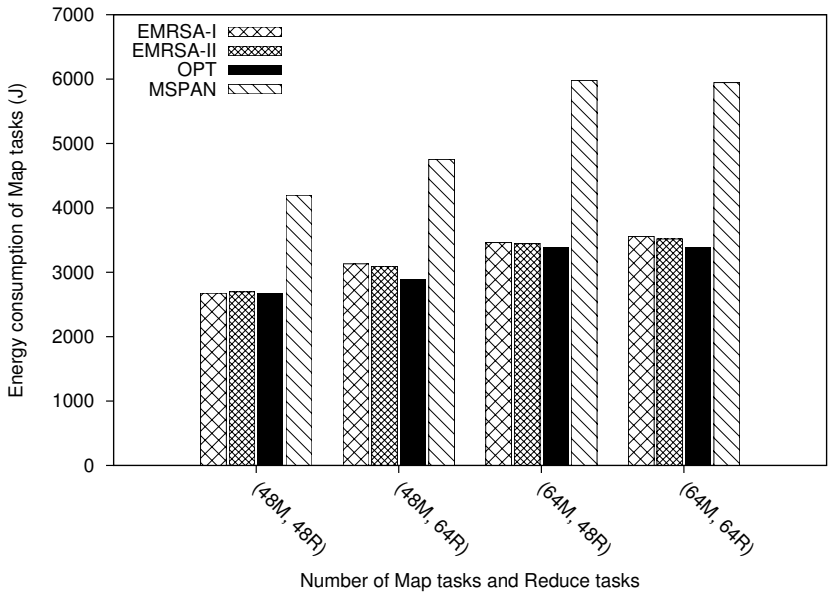


Figure 6.4: TeraSort energy consumption (small-scale experiments): Map tasks

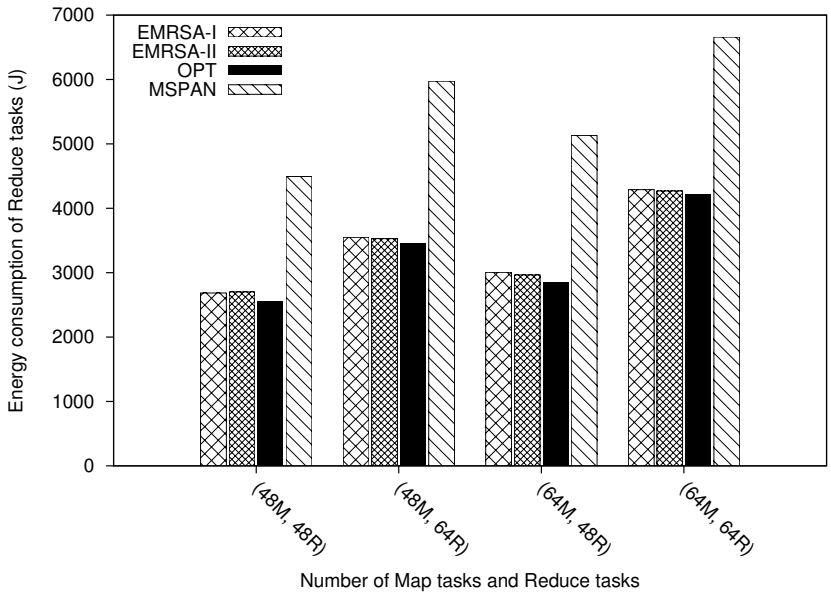


Figure 6.5: TeraSort energy consumption (small-scale experiments): Reduce tasks

energy consumption obtained by MSPAN is far from the optimal solution, making it not suitable for scheduling MapReduce jobs with the goal of minimizing the energy consumption.

Fig. 6.3 presents the execution time of the algorithms. The results show that EMRSA-I and EMRSA-II find the assignments in significantly less amount of time than OPT and

MSPAN. As shown in this figure, EMRSA-I and EMRSA-II obtain the solution in a time that is six orders of magnitude less than that of OPT. For example, the execution times of EMRSA-I, EMRSA-II, OPT, and MSPAN for the workload $(48M, 48R)$ are 0.001, 0.001, 673.7, and 839.3 seconds, respectively.

In Fig. 6.4 and Fig. 6.5, we present the energy consumption of map and reduce tasks in more details. When the number of reduce tasks is greater than the number of map tasks (e.g., workload $(48M, 64R)$), EMRSA-I and EMRSA-II capture more optimization opportunities for energy saving available for reduce tasks. In more detail, the energy consumptions of map tasks for workload $(48M, 64R)$ obtained by EMRSA-I, EMRSA-II, OPT, and MSPAN are 3130, 3090, 2897, and 4751 J, respectively, while the energy consumptions of reduce tasks for workload $(48M, 64R)$ are 3547, 3527, 3448, and 5972 J, respectively. However, when the workload has more map tasks than reduce tasks (e.g., workload $(64M, 48R)$), EMRSA-I and EMRSA-II save more energy for map tasks. The energy consumption for the map tasks of workload $(64M, 48R)$ obtained by employing EMRSA-X (shown in Fig. 6.4) is closer to the optimal than the energy consumption for the reduce tasks (shown in Fig. 6.5) for the same workload. This is due to the fact that for this workload, the load of the map tasks is greater than that of the reduce tasks, that is $f > 1$. For workload $(48M, 64R)$, where $f < 1$, EMRSA-X leads to an energy consumption closer to the optimal for the reduce tasks. This shows the effect of ratio f on the energy consumption.

Large-scale experiments

We analyze the performance of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN, for three types of benchmarks (TeraSort, Page Rank, and K-means Clustering) considering eight large MapReduce jobs for each, where the number of map tasks and reduce tasks are given in Table 6.5.

(i) Terasort: Fig. 6.6 shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. This figure shows that the energy consumption obtained by EMRSA-I and EMRSA-II is very close to the lower bound for all cases, which in turn im-

Table 6.5: Workloads for the large scale experiments.

Workload	Map tasks	Reduce tasks
(128M, 128R)	128	128
(128M, 256R)	128	256
(128M, 512R)	128	512
(256M, 128R)	256	128
(256M, 256R)	256	256
(256M, 512R)	256	512
(512M, 128R)	512	128
(512M, 512R)	512	512

plies that EMRSA-I and EMRSA-II solutions are even closer to the optimal solutions. This shows the near-optimality of solutions obtained by EMRSA-I and EMRSA-II. In some cases EMRSA-I obtains better results. However, the results show that EMRSA-I and EMRSA-II are able to find schedules requiring an average of 35.6% and 35.8% less energy than that of those obtained by G-MSPAN, respectively. Such reduction in energy consumption can be a great incentive for data centers to incorporate EMRSA-I and EMRSA-II for scheduling MapReduce jobs to reduce their costs. Note that the amount of energy savings obtained by EMRSA-I and EMRSA-II in the large-scale experiments is compared with that obtained by the G-MSPAN. However, in the small-scale experiments, we presented the amount of energy savings of EMRSA-I and EMRSA-II compared to the optimal makespan minimization algorithm. As the total number of map and reduce tasks increases (from 256 to 1024), the total amount of energy consumption of the workloads increases. In addition, this figure shows the sensitivity analysis on the number of tasks. By fixing the number of map tasks while increasing the number of reduce tasks, we observe an increase in the total energy consumption. For example, this behavior is shown for the first three workloads, where the number of map tasks is 128, and the number of reduce tasks is from 128 to 512.

Fig. 6.7 shows the execution time of the algorithms. EMRSA-I, EMRSA-II, and G-MSPAN find the results in less than a second for all selected MapReduce jobs. Note that the lower bound results are obtained by solving the LP relaxation, not the EMRSA-IP. The execution time of L-BOUND presenting the LP relaxation results is polynomial. In

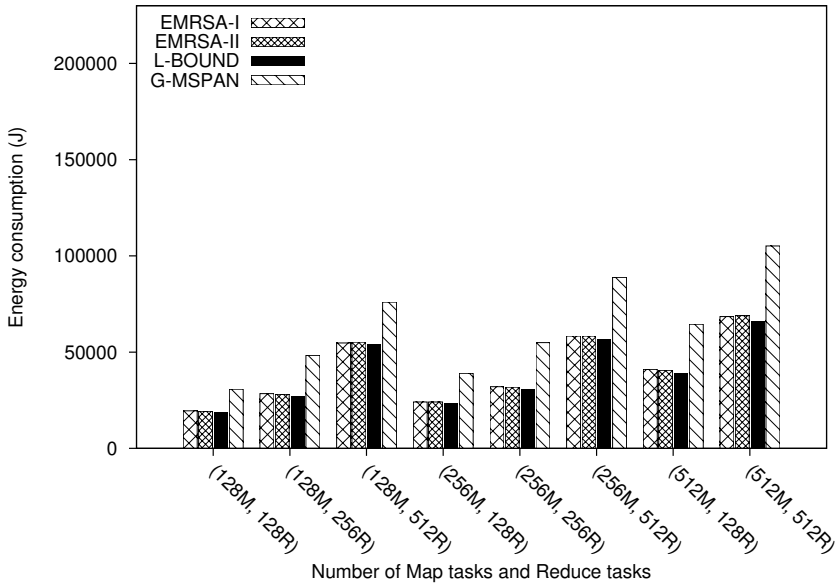


Figure 6.6: EMRSA-I and EMRSA-II performance on TeraSort: Energy consumption

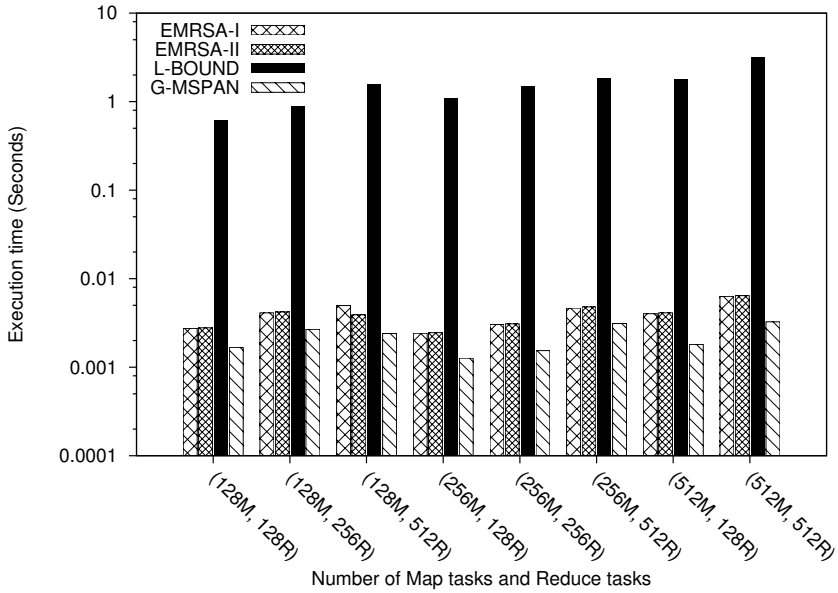


Figure 6.7: EMRSA-I and EMRSA-II performance on TeraSort: Execution time

addition, with the increase in the total number of map and reduce tasks, the execution time of L-BOUND increases. For example, the execution time of L-BOUND increases from workload (128M, 128R) to (128M, 512R). However, it decreases from workload (128M, 512R) to (256M,128R). The execution time of EMRSA-I and EMRSA-II follows the same behavior.

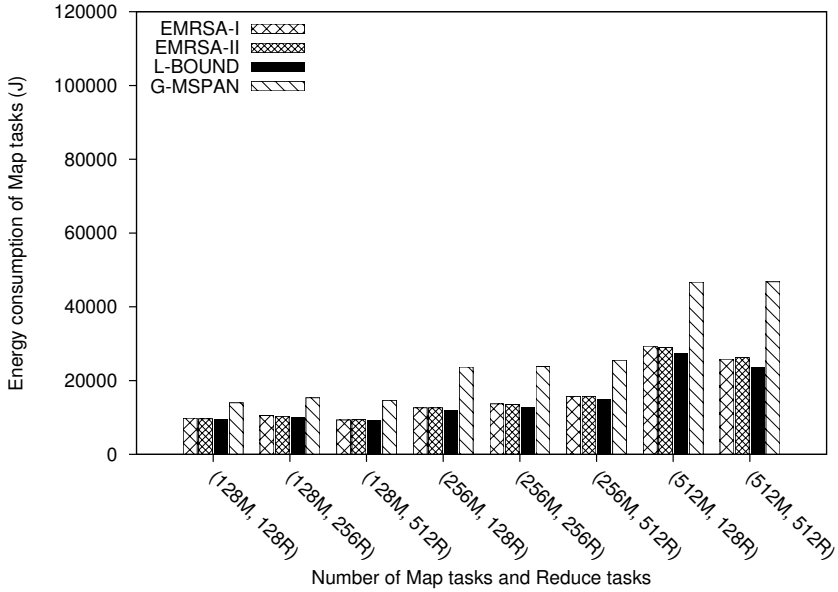


Figure 6.8: TeraSort energy consumption (large-scale experiments): Map tasks

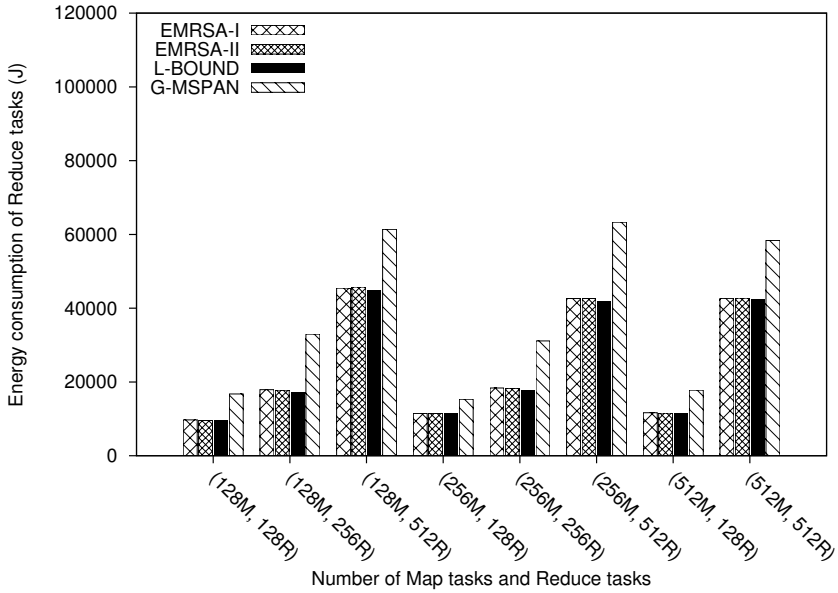


Figure 6.9: TeraSort energy consumption (large-scale experiments): Reduce tasks

In Fig. 6.8 and Fig. 6.9, we present the energy consumption of map and reduce tasks separately in more detail. The results show that for both map and reduce tasks, the solutions obtained by EMRSA-I and EMRSA-II are very close to the lower bounds. In addition, we perform sensitivity analysis with respect to the number of map and reduce tasks. Fig. 6.8 and Fig. 6.9 show how the energy consumption of map and reduce tasks

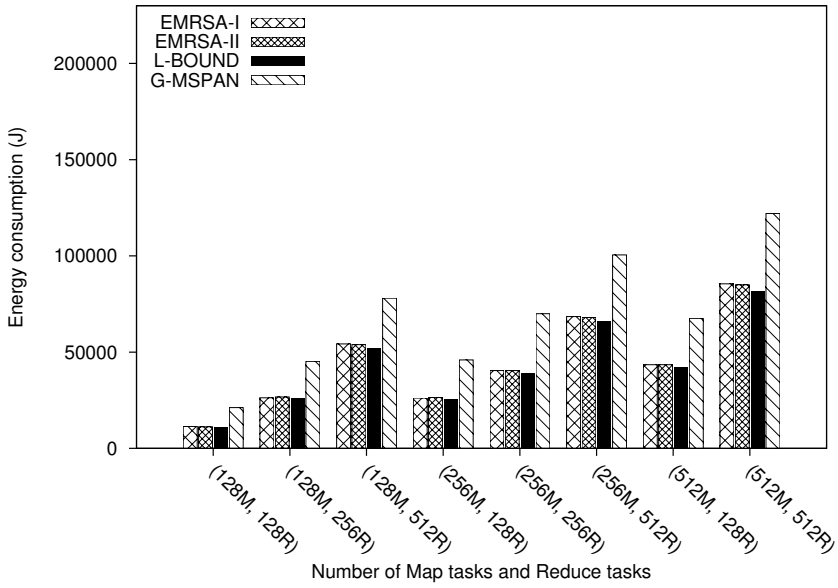


Figure 6.10: EMRSA-I and EMRSA-II performance on Page Rank: Energy consumption

changes by fixing the number of map tasks (e.g., to 128), and changing the number of reduce tasks (e.g., from 128 to 512). For example, for workloads (128M, 128R), (128M, 256R), and (128M, 512R), Fig. 6.8 shows that these workloads have almost the same energy consumption for their map tasks, which is expected as the number of map tasks are the same. However, Fig. 6.9 shows an exponential increase in the energy consumption of the reduce tasks for these workloads which is expected as the number of reduce tasks increases exponentially. For the sensitivity analysis of energy consumption with respect to number of reduce tasks, we analyze workloads (128M, 128R), (256M, 128R), and (512M, 128R). Fig. 6.9 shows that these workloads have almost the same energy consumption for their reduce tasks since the number of reduce tasks are the same. However, Fig. 6.8 shows that by increasing the number of map tasks, the energy consumption of the map tasks for these workload increases.

(ii) Page Rank: Fig. 6.10 shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. This figure shows that the obtained results by EMRSA-I and EMRSA-II are very close to the lower bounds obtained by L-BOUND. The results show that EMRSA-I and EMRSA-II are able to find schedules requiring an average of 35.3% and 35.5% less energy than that of those obtained by G-MSPAN, respectively. The sensitivity

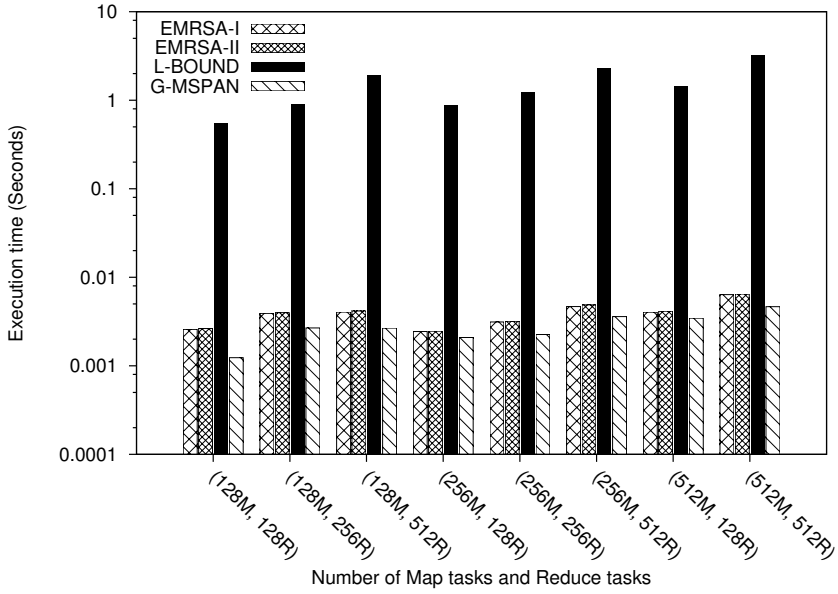


Figure 6.11: EMRSA-I and EMRSA-II performance on Page Rank: Execution time

analysis with respect to the number of tasks shows that by increasing the total number of map and reduce tasks, the energy consumption increases. For example, the total energy consumptions of $(256M, 128R)$ obtained by EMRSA-I, EMRSA-II, the L-BOUND, and G-MSPAN are 26,064, 26,364, 25,642.7, and 45,845 J, respectively, while the total energy consumptions of $(256M, 512R)$ are 68,388, 67,888, 65,832.3, and 100,426 J, respectively. For jobs with the same number of map tasks and the same number of reduce tasks, the energy consumption of Page Rank is similar to TeraSort energy consumption as shown in Fig. 6.6.

Fig. 6.11 shows the execution time of the algorithms. EMRSA-I, EMRSA-II, and G-MSPAN find the solutions very fast. With the increase in the total number of map and reduce tasks, the execution time of all algorithms increases. For example, the execution time of L-BOUND for workload $(512M, 128R)$ increases from 1.43 to 3.22 seconds for workload $(512M, 512R)$. In addition, the execution time of EMRSA-I and EMRSA-II for workload $(512M, 128R)$ increases from 0.004 to 0.006 seconds for workload $(512M, 512R)$.

Fig. 6.12 and Fig. 6.13, show the energy consumption of map and reduce tasks separately in more detail, and they are similar to the results for TeraSort workloads. These figures show the sensitivity analysis with respect to number of map and reduce tasks.

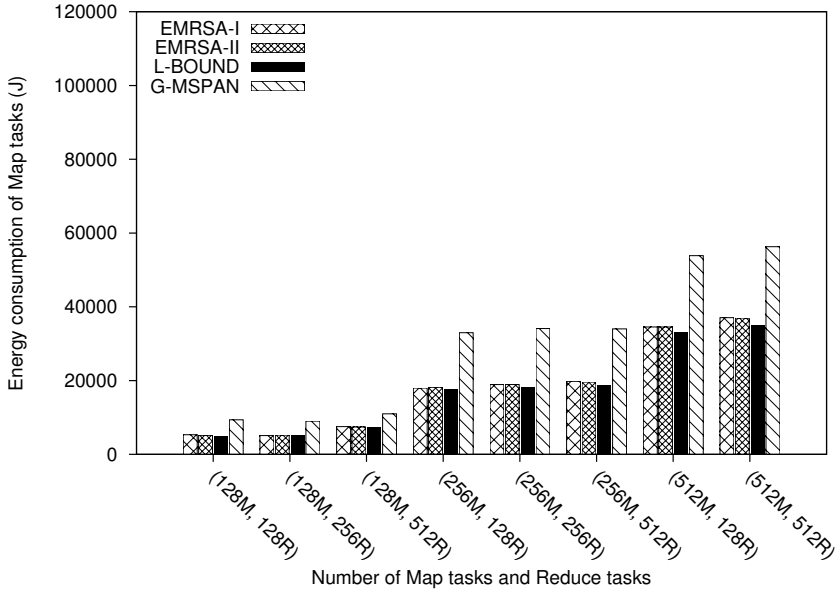


Figure 6.12: Page Rank energy consumption (large-scale experiments): Map tasks

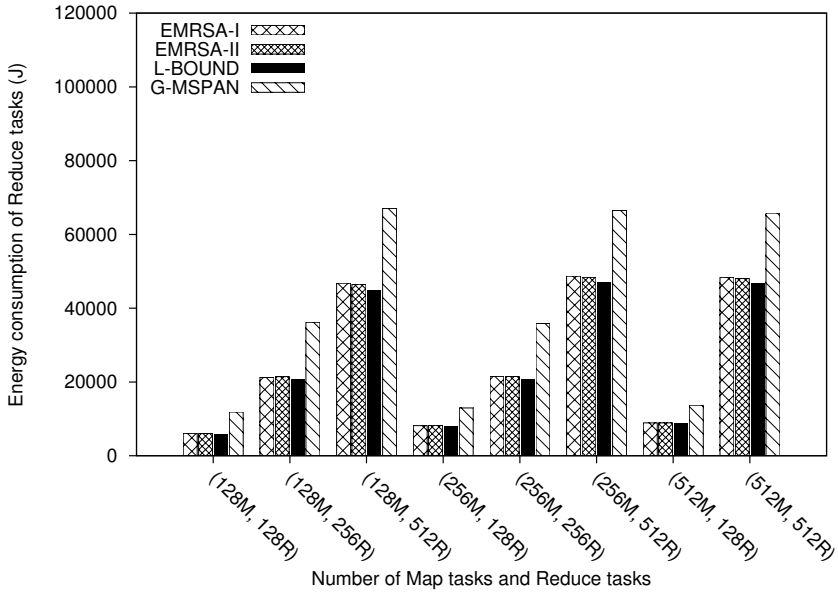


Figure 6.13: Page Rank energy consumption (large-scale experiments): Reduce tasks

(iii) K-means Clustering: Fig. 6.14 shows the energy consumption of EMRSA-I, EMRSA-II, L-BOUND, and G-MSPAN. The results show that the obtained solutions by EMRSA-I and EMRSA-II are very close to the lower bounds obtained by L-BOUND. This figure shows that EMRSA-I and EMRSA-II are able to save an average of 30.9% and 31.4% energy compared to G-MSPAN, respectively. This figure also shows the sensitivity analysis

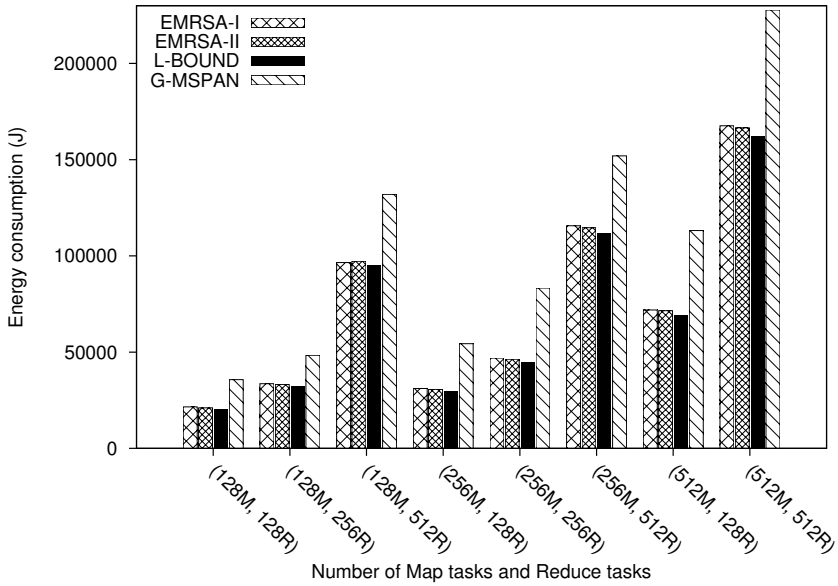


Figure 6.14: EMRSA-I and EMRSA-II performance on K-means: Energy consumption

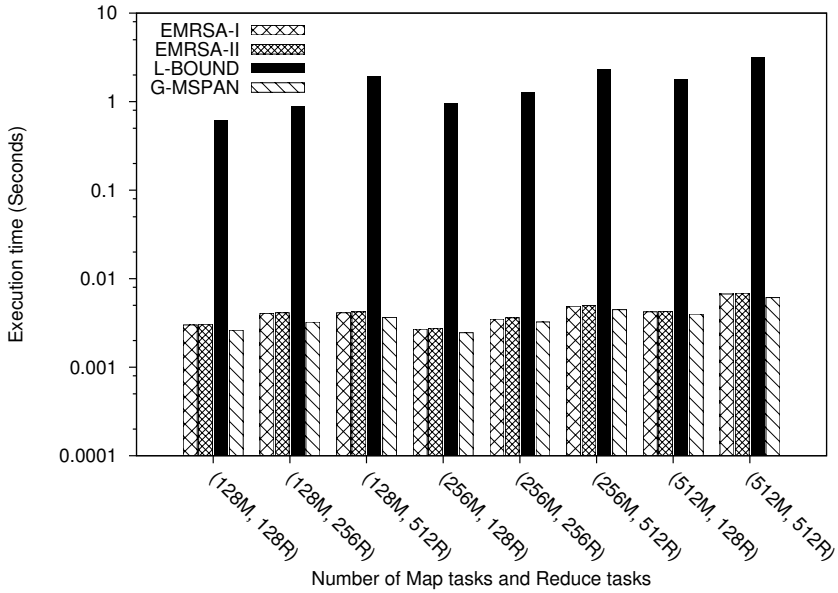


Figure 6.15: EMRSA-I and EMRSA-II performance on K-means: Execution time

on the total number of tasks, along with the detailed sensitivity analysis on the number of map and reduce separately. It confirms the above mentioned sensitivity analysis results for TeraSort and Page Rank. However, for similar workloads (i.e., having the same number of map tasks and the same number of reduce tasks) the energy consumption of K-means Clustering is almost twice the energy consumption of Page Rank and TeraSort. This shows

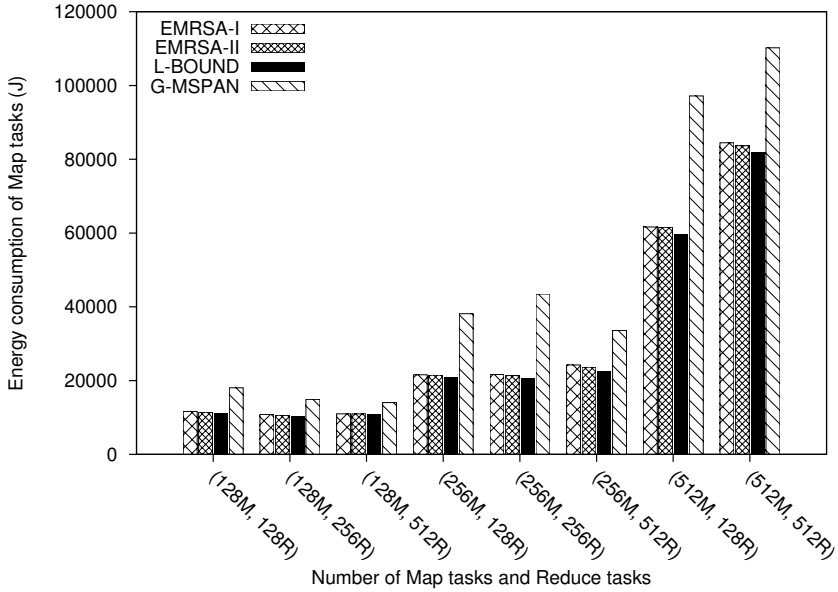


Figure 6.16: K-means Clustering energy consumption: Map tasks

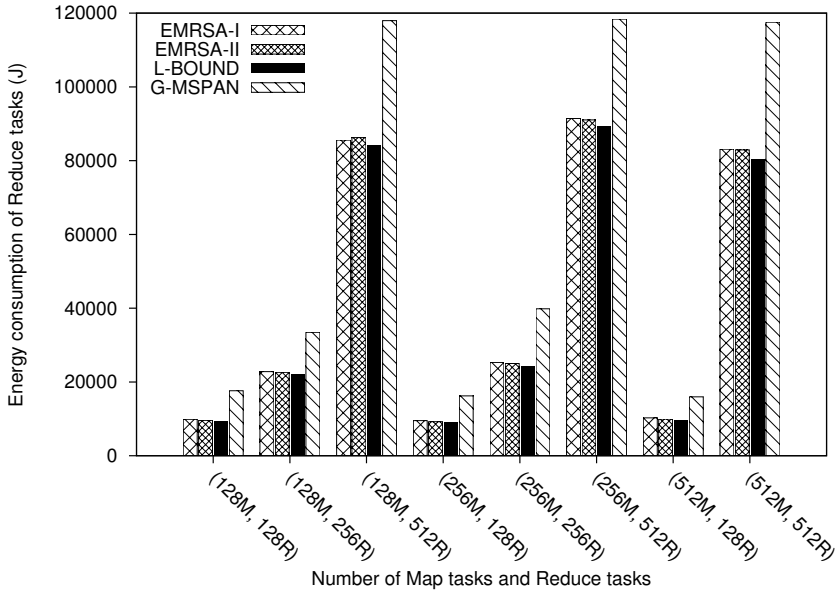


Figure 6.17: K-means Clustering energy consumption: Reduce tasks

that K-means Clustering tasks are more computationally complex than TeraSort and Page Rank, leading to consuming more energy.

In Fig. 6.15, we present the execution time of the algorithms. The results show that EMRSA-I, EMRSA-II, and G-MSPAN find the solutions very fast. In Fig. 6.16 and Fig. 6.17, we present the energy consumption of map and reduce tasks separately in more

details. The results show that for both map and reduce tasks, the obtained results by EMRSA-I and EMRSA-II are very close to the lower bounds. In addition, these figures show the sensitivity analysis with respect to the number of map and reduce tasks.

From all the above results, we conclude that EMRSA-I and EMRSA-II obtain MapReduce job schedules with significantly lower energy consumption, and require small execution times, making them suitable candidates for scheduling big data applications in data centers. In addition, the schedules obtained by EMRSA-I and EMRSA-II provide energy savings close to the optimal. The results show that makespan minimization is not necessarily the best strategy to consider when scheduling MapReduce jobs for energy efficiency in data centers. This is due to the fact that data centers are obligated to deliver the requested services according to the SLA, where such agreement may provide significant optimization opportunities to reduce energy costs. Such reduction in energy costs is a great incentive for data centers to adopt our proposed scheduling algorithms.

6.5 Conclusion

Due to the increasing need for big data processing and the widespread adoption of MapReduce and its open source implementation Hadoop for such processing, improving MapReduce performance with energy saving objectives can have a significant impact in reducing energy consumption in data centers. In this chapter, we show that there are significant optimization opportunities within the MapReduce framework in terms of reducing energy consumption. We proposed two energy-aware MapReduce scheduling algorithms, EMRSA-I and EMRSA-II, that schedule the individual tasks of a MapReduce job for energy efficiency while meeting the application deadline. Both proposed algorithms provide very fast solutions making them suitable for execution in real-time settings. We performed experiments on a Hadoop cluster to determine the energy consumption of several MapReduce benchmark applications such as TeraSort, Page Rank, and K-means Clustering. We then used this data in an extensive simulation study to analyze the performance of EMRSA-I and EMRSA-II. The results showed that the proposed algorithms are capable of obtaining near optimal solutions leading to significant energy savings.

CHAPTER 7: CONCLUSION

In this Ph.D. dissertation, we presented our research accomplishments in the field of resource management in cloud and big data systems. We presented the background knowledge and discussed a survey of relevant literature to lay the foundation of our work. We identified fundamental open problems in the field, formalized them as mathematical programs, and solved the problems by designing mechanisms and algorithms. We also evaluated our proposed mechanisms and algorithms, both theoretically and experimentally. In this chapter, we present a summary of our research, and we outline future directions of research that may stem from our work.

7.1 Summary

Although cloud computing has gained a great deal of attention, there are still some key impediments to large scale enterprise adoption. The ever-growing demand for cloud resources from businesses and individuals places the cloud resource management at the heart of the cloud providers' decision-making process. One of the major challenges faced by the cloud providers is cloud resource management. Cloud providers and cloud users, pursue different goals. Cloud providers aim to maximize revenue while achieving high resource utilization. On the other hand, users want to minimize their expenses while meeting their performance requirements. However, the challenge is how to allocate and price resources in a mutually optimal way despite the lack of information sharing among users and cloud providers. In addition, the cloud environment is highly variable and unpredictable. Cloud providers may oversubscribe users to a shared infrastructure to increase their resource utilization, while oversubscription will results in resource contention and interference. In addition, there are other factors that contribute to unpredictability of the environment such as heterogeneity of the VMs. These factors make these multi-criteria optimization problems very complex. In this dissertation, we addressed this fundamental challenge.

7.2 Future Research Directions

We believe that our research will encourage new research work in the area of resource management in large-scale distributed systems. The following are several important and promising directions of research that can be pursued following our work.

7.2.1 Resource Management in Geographically Distributed Clouds

Determining the locations that VMs should be placed in a cloud federation setting is a challenging problem. The objective is to minimize the cost while keeping the quality of service (e.g., response time). Existing solutions for the problem have either ignored the dynamics, or provided inadequate solutions that achieve both objectives at the same time. To address the dynamic resource management of resources in geographically distributed clouds, game-theoretic models can be applied.

7.2.2 Energy-Efficient Resource Management in Clouds

One of the main concerns for a cloud service provider is minimizing the operational cost, especially the cost of power consumption. It is possible to reduce power consumption of the hardware by means of deploying more Virtual Machines (VMs) onto fewer hosts.

7.2.3 Integrated System Design for Cloud Computing

When designing cloud computing systems, focusing on only one specific objective such as resource utilization without considering other objectives such as reducing data movement and data privacy can lead to inferior performance. One of the areas of interest is designing an integrated cloud system that support big data. Such a system will consider multiple objectives including location-aware big data computing, energy efficiency, and incentive-compatibility which bring about several challenges arising from the complexity of the problem and the fact that data centers need fast and autonomic decision making algorithms.

7.2.4 Double Auction Market Design for Trading Clusters

The commercial public cloud market is an emerging and evolving computing market where cloud providers and users can trade computing resources and services. The big data trend is generating massive data sets and requires unique services that are different from conventional computing services. Therefore, there is a need to fundamentally address such requirements by developing economics and market mechanisms for managing, trading, and pricing cloud services. Cloud-based technologies have a great potential to meet the requirements of new compute-intensive and data-intensive applications. Therefore, cloud providers can gain a part of the big data market share by facilitating and supporting big data processing as a pay-as-you-go model. However, gaining such market share comes at the cost of architectural changes to the current cloud frameworks along with designing economic mechanisms for shaping cloud computing into a big data pay-as-you-go paradigm. Currently, cloud providers offer IaaS to users in the form of virtual machines with limited computing resources. However, such services are not specifically designed for the big data applications. The unique characteristics of big data applications present a host of new challenges for the cloud providers. Big data applications require larger resources (e.g., entire clusters) compared to those required by conventional computing services. Auctions have proven to be effective market-based mechanisms for cloud services. Main stream cloud provider powerhouses such as Amazon and Microsoft have been offering cloud services in auction market for several years. We envision that two-sided auction markets for trading clusters of machines would be of interest. In such a markets, both users and providers will have an opportunity to host and take part in auctions.

7.2.5 Scalable Big Data Analytics Algorithms

The growing rate of big data is faster than Moore's law while the demand for more complex analytics is increasing. A key difficulty in scaling interactive analytical tools to support big data is maintaining sub-second latencies. Existing analytical tools with MapReduce-style frameworks cannot find solutions in reasonable amount of time which make them prohibitive

for interactive data exploration or online applications. This challenge necessitates the design of algorithms that provide reasonable decisions in the absence of perfect solutions. A line of research in this direction is designing distributed ranking algorithms for big data analysis. Such massively parallel ranking algorithms will obtain approximate solutions with well defined statistical error guarantees and sub-second latencies when processing big data.

7.3 Conclusion

In this Ph.D. dissertation, we presented our research on resource management in cloud and big data systems. We believe that this Ph.D. dissertation is a significant contribution to the existing literature on resource management in large-scale distributed systems. We proposed several mechanisms for VM allocation and pricing problem in clouds considering different settings. These mechanisms will provide the cloud providers the flexibility of dynamically determining the price of their resources and their cost share. The cloud providers will be free from building complex pricing models or generating user statistics for prediction of system usage. In addition, the cloud providers can use our proposed energy-aware schedulers in their data centers to lower their energy costs leading to lower service prices offered to their users. On the other hand, different types of users will be able to select their desired usage of cloud services.

List of Publications

Journal Articles (Published/Accepted)

J1. Energy-Aware Scheduling of MapReduce Jobs for Big Data Applications [105]
Lena Mashayekhy, Mahyar Nejad, Daniel Grosu, Quan Zhang, and Weisong Shi
IEEE Transactions on Parallel and Distributed Systems, Vol. 26, 2015 (accepted)

J2. An Online Mechanism for Resource Allocation and Pricing in Clouds [103]
Lena Mashayekhy, Mahyar Nejad, Daniel Grosu, and Athanasios Vasilakos
IEEE Transactions on Computers, Vol. 64, 2015 (accepted)

J3. A PTAS Mechanism for Provisioning and Allocation of Heterogeneous Cloud Resources [101]
Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu
IEEE Transactions on Parallel and Distributed Systems, Vol. 26, 2015 (accepted)

J4. Truthful Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds ***** [124]
Mahyar Nejad, Lena Mashayekhy, and Daniel Grosu
IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 2, pp. 594-603, 2015.

*** Best Paper Runner-Up Award - 2014 INFORMS Service Science**

J5. Cloud Federations in the Sky: Formation Game and Mechanism [99]
Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu
IEEE Transactions on Cloud Computing, Vol. 3, No. 1, pp. 14-27, 2015.

J6. Physical Machine Resource Management in Clouds: A Mechanism Design Approach [100]
Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu
IEEE Transactions on Cloud Computing, Special Issue on Economics and Market Mechanisms for Cloud Computing, 2014 (accepted).

J7. A Merge-and-Split Mechanism for Dynamic Virtual Organization Formation in Grids [93]
Lena Mashayekhy and Daniel Grosu
IEEE Transactions on Parallel and Distributed Systems, Vol. 25, No. 3, pp. 540-549, 2014.

J8. Computing Nash Equilibria in Bimatrix Games: GPU-based Parallel Support Enumeration ***** [147]

Safraz Rampersaud, Lena Mashayekhy, and Daniel Grosu

IEEE Transactions on Parallel and Distributed Systems, Vol.25, No.12, pp.3111-3123, 2014.

*** Selected as the Featured Article for the IEEE TPDS December 2014 Issue**

J9. Hierarchical Time-Dependent Shortest Path for Routing on Dynamic Road Networks under ITS [122]

Mahyar Nejad, Lena Mashayekhy, Ratna Chinnam, and Anthony Phillips

IIE Transactions, Vol. 47, 2015. (accepted)

Journal Articles Under Review

J10. Optimal Routing for Plug-in Hybrid Electric Vehicles ***** [125]

Mahyar Nejad, Lena Mashayekhy, Daniel Grosu, and Ratna Chinnam

INFORMS Transportation Science, 2014. (under second round review)

*** Best Student Paper Award, INFORMS Energy, Natural Resources, and the Environment Section (ENRE), 2014.**

*** Runner-Up Award for the Best Student Paper, Production and Operations Management Society (POMS), College of Sustainable Operations, 2014.**

J11. A Reputation-Based Mechanism for Dynamic Virtual Organization Formation in Grids [95]

Lena Mashayekhy and Daniel Grosu

IEEE Transactions on Computers, 2014. (under second round review)

J12. Truthful Mechanisms for Competitive Reward-Based Scheduling [88]

Lena Mashayekhy, Nathan Fisher, and Daniel Grosu

IEEE Transactions on Computers, 2014. (under second round review)

J13. Online Scheduling and Pricing for Electric Vehicle Charging [121]

Mahyar Nejad, Lena Mashayekhy, Ratna Chinnam, and Daniel Grosu

IIE Transactions, 2014. (under second round review)

Refereed Conference Papers

C14. A Two-Sided Market Mechanism for Trading Big Data Computing Commodities [98]

Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu

Proc. of the IEEE International Conference on Big Data (IEEE BigData 2014), Washington DC, USA, October 2014.

C15. A Framework for Data Protection in Cloud Federations [97]
Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu
Proc. of the 43rd International Conference on Parallel Processing (ICPP'14), pp. 283-290, Minneapolis, USA, September 2014.

C16. Energy-aware Scheduling of MapReduce Jobs [102]
Lena Mashayekhy, Mahyar Nejad, Daniel Grosu, Dajun Lu, and Weisong Shi
Proc. of the 3rd IEEE International Congress on Big Data (BigData'14)-Research Track, pp. 32-39, Anchorage, USA, June 2014. (Acceptance rate: 19%)

C17. Incentive-Compatible Online Mechanisms for Resource Provisioning and Allocation in Clouds [104]
Lena Mashayekhy, Mahyar Nejad, Daniel Grosu, and Athanasios Vasilakos
Proc. of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)-Research Track, pp. 312-319, Anchorage, USA, June 2014. (Acceptance rate: 19%)

C18. Strategy-proof Mechanisms for Resource Management in Clouds [94]
Lena Mashayekhy and Daniel Grosu
Proc. of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'14), pp. 554-557, Chicago, USA, May 2014.

C19. A Truthful Approximation Mechanism for Autonomic Virtual Machine Provisioning and Allocation in Clouds [96]
Lena Mashayekhy, Mahyar Nejad, and Daniel Grosu
Proc. of the ACM Cloud and Autonomic Computing Conference (CAC'13), pp. 1-10, Miami, USA, August 2013.

C20. A Family of Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds[123]
Mahyar Nejad, Lena Mashayekhy, and Daniel Grosu
Proc. of the 6th IEEE International Conference on Cloud Computing (CLOUD'13)-Research Track, pp. 188-195, Santa Clara, USA, July 2013. (Acceptance rate: 18%)

C21. Computing Nash Equilibria in Bimatrix Games: GPU-based Parallel Support Enumeration [146]
Safraz Rampersaud, Lena Mashayekhy, and Daniel Grosu
Proc. of the 31st IEEE International Performance Computing and Communications Conference (IPCCC'12), pp. 332-341, Austin, USA, December 2012. (Acceptance

rate: 27.8%)

C22. A Coalitional Game-Based Mechanism for Forming Cloud Federations [90]
Lena Mashayekhy and Daniel Grosu
Proc. of the 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'12), pp. 223-227, Chicago, USA, November 2012. (Acceptance rate: 27.2%)

C23. A Reputation-Based Mechanism for Dynamic Virtual Organization Formation in Grids [92]
Lena Mashayekhy and Daniel Grosu
Proc. of the 41st International Conference on Parallel Processing (ICPP'12), pp. 108-117, Pittsburgh, USA, September 2012. (Acceptance rate: 28%)

C24. Effects of Traffic Network Dynamics on Hierarchical Community-based Representations of Large Road Networks [127]
Mahyar Nejad, Lena Mashayekhy, and Ratna Chinnam
Proc. of the 15th IEEE International Intelligent Transportation Systems Conference (ITSC'12), pp. 1900-1905, Anchorage, USA, September 2012.

C25. A Distributed Merge-and-Split Mechanism for Dynamic Virtual Organization Formation in Grids [91]
Lena Mashayekhy and Daniel Grosu
Proc. of the 11th IEEE International Symposium on Network Computing and Applications (IEEE NCA'12), pp. 36-43, Cambridge, USA, August 2012. (Acceptance rate: 29%)

C26. A Merge-and-Split Mechanism for Dynamic Virtual Organization Formation in Grids [89]
Lena Mashayekhy and Daniel Grosu
Proc. of the 30th IEEE International Performance Computing and Communications Conference (IPCCC'11), pp. 1-8, Orlando, USA, November 2011. (Acceptance rate: 27%)

C27. State Space Reduction in Modeling Traffic Network Dynamics for Dynamic Routing under ITS [126]
Mahyar Nejad, Lena Mashayekhy, Ali Taghavi, and Ratna Chinnam
Proc. of the 14th IEEE International Intelligent Transportation Systems Conference (ITSC'11), pp. 277-282, Washington DC, USA, October 2011.

C28. Measuring of Strategies' Similarity in Automated Negotiation [107]
Lena Mashayekhy, Mohamad Nematbakhsh, and Behroz Ladani
Proc. of the International Conference on Data Mining and Applications (ICDMA'07), pp. 859-863, IAENG, Hong Kong, March 2007. (Nominated for the Best Paper Award)

C29. E-Negotiation Model based on Data Mining [106]
Lena Mashayekhy, Mohamad Nematbakhsh, and Behroz Ladani
Proc. of the IADIS International Conference on e-Commerce 2006, pp. 369-373, Barcelona, Spain, December 2006.

Book Chapters

B30. Comparing Negotiation Strategies Based on Offers [109]
Lena Mashayekhy, Mohamad Nematbakhsh, and Behroz Ladani
Frontiers in Artificial Intelligence and Applications, Applications of Data Mining in E-Business and Finance, Vol. 177, pp. 87-98, IOS Press, The Netherlands, 2008.

B31. Detecting Similar Negotiation Strategies [108]
Lena Mashayekhy, Mohamad Nematbakhsh, and Behroz Ladani
Trends in Intelligent Systems and Computer Engineering, Vol. 6, pp. 297-309, Springer, 2008.

REFERENCES

[1] Amazon EC2 Instance Types. [Online]. Available: http://aws.amazon.com/ec2/instance-types/.

[2] Amazon EC2 Spot Instance Curriculum. [Online]. Available: http://aws.amazon.com/ec2/spot-tutorials/.

[3] Amazon EC2 Spot Instances. [Online]. Available: http://aws.amazon.com/ec2/purchasing-options/spot-instances/.

[4] APC. [Online]. Available: http://www.apc.com/.

[5] Grid Workloads Archive. [Online]. Available: http://gwa.ewi.tudelft.nl.

[6] Hadoop. [Online]. Available: http://hadoop.apache.org/.

[7] IBM ILOG CPLEX V12.1 user's manual. [Online]. Available: ftp://ftp.software.ibm.com/software/websphere/ilog/docs/.

[8] Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/.

[9] Titan. [Online]. Available: http://www.olcf.ornl.gov/titan/.

[10] Top 500 Supercomputers. [Online]. Available: http://www.top500.org.

[11] Windows Azure. [Online]. Available: http://www.windowsazure.com/en-us/pricing/calculator/.

[12] Z. Abbasi, M. Pore, and S. K. Gupta. Online server and workload management for joint optimization of electricity cost and carbon footprint across data centers. In *Proc. 28th IEEE Intl. Symp. on Parallel & Dist. Proc.*, 2014.

[13] K. Apt and A. Witzel. A generic approach to coalition formation. *International Game Theory Review*, 11(3):347–367, 2009.

[14] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Comput. Syst.*, 28(5):755–768, 2012.

[15] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[16] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

[17] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *Proc. of the 31st IEEE Intl. Conf. on Dist. Comp. Syst.*, pages 700–709, 2011.

[18] M. Bjorkqvist, L. Chen, and W. Binder. Opportunistic service provisioning in the cloud. In *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, pages 237–244, 2012.

[19] A. Bogomolnaia and M. Jackson. The stability of hedonic coalition structures. *Games & Econ. Behavior*, 38(2):201–230, 2002.

[20] P. Briest, P. Krysta, and B. Vöcking. Approximation techniques for utilitarian mechanism design. *SIAM Journal on Computing*, 40(6):1587–1622, 2011.

[21] D. Bruneo. A stochastic model to investigate data center performance and qos in iaas cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):560–569, 2014.

[22] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.

[23] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[24] R. Buyya, C. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proc. of the 10th IEEE Intl. Conf. on High Performance Computing & Communications*, pages 5–13, 2008.

[25] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya. Optimal multiserver configuration for profit maximization in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1087–1096, 2013.

[26] M. Cardosa, A. Singh, H. Pucha, and A. Chandra. Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud. *IEEE Transactions on Computers*, pages 1737–1751, 2012.

[27] E. Casalicchio, D. A. Menascé, and A. Aldhalaan. Autonomic resource provisioning in cloud systems with availability goals. In *Proc. of the 2013 ACM Cloud & Autonomic Computing Conf.*, page 1. ACM, 2013.

[28] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. In *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing*, pages 337–345, 2010.

[29] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing*, 5(2):164–177, 2012.

[30] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *Proc. of the 30th IEEE Intl. Conf. on Computer Communications*, pages 3074–3082, 2011.

[31] C. Chekuri and I. Gamzu. Truthful mechanisms via greedy iterative packing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 56–69. Springer, 2009.

[32] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[33] F. Chen, M. S. Kodialam, and T. V. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. In *Proc. of the IEEE INFOCOM*, pages 1143–1151, 2012.

[34] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proc. of the 7th ACM European Conf. on Computer Systems*, pages 43–56, 2012.

[35] E. Clarke. Multipart pricing of public goods. *Public choice*, 11(1):17–33, 1971.

[36] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. of the 6th USENIX Symp. on Operating System Design and Implementation*, pages 137–150, 2004.

[37] S. Dobzinski and S. Dughmi. On the power of randomization in algorithmic mechanism design. In *Proc. of the 50th Annual IEEE Symp. on Foundations of Computer Science*, pages 505–514. IEEE, 2009.

[38] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng. Energy-saving virtual machine placement in cloud data centers. In *Proc. 13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Comput.*, pages 618–624, 2013.

[39] J. Doyle, R. Shorten, and D. O'Mahony. Stratus: Load balancing the cloud for carbon emissions control. *IEEE Transactions on Cloud Computing*, 1(1):116–128, 2013.

[40] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proc. 19th ACM Int'l Symp. High Performance Distr. Comp.*, pages 810–818, 2010.

[41] J. Feigenbaum, M. Schapira, and S. Shenker. Distributed algorithmic mechanism design. *Algorithmic Game Theory*, pages 363–384, 2007.

[42] Y. Feng, B. Li, and B. Li. Price competition in an oligopoly market with multiple iaas cloud providers. *IEEE Transactions on Computers*, 63(1):59–73, 2014.

[43] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[44] E. J. Friedman and D. C. Parkes. Pricing wifi at starbucks: issues in online mechanism design. In *Proc. of the 4th ACM Conf. on Electronic commerce*, pages 240–241, 2003.

[45] H. Fu, Z. Li, C. Wu, and X. Chu. Core-selecting auctions for dynamically allocating heterogeneous vms in cloud computing. In *Proc. 7th IEEE Intl. Conf. Cloud Comput.*, pages 152–159, 2014.

[46] A. Gershkov and B. Moldovanu. Efficient sequential assignment with incomplete information. *Games and Economic Behavior*, 68(1):144–154, 2010.

[47] T. Ghazar and N. Samaan. Pricing utility-based virtual networks. *IEEE Transactions on Network and Service Management*, 10(2):119–132, June 2013.

[48] C. Ghribi, M. Hadji, and D. Zeghlache. Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms. In *Proc. 13th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Comput.*, pages 671–678, 2013.

[49] I. Goiri, J. Guitart, and J. Torres. Characterizing cloud federation for enhancing providers' profit. In *Proc. IEEE Intl. Conf. on Cloud Computing*, pages 123–130, 2010.

[50] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *Proc. of the 7th ACM European Conf. on Computer Systems*, pages 57–70, 2012.

[51] T. Groves. Incentives in teams. *Econometrica: Journal of the Econometric Society*, 41(4):617–631, 1973.

[52] M. Guazzone, C. Anglano, and M. Canonico. Energy-efficient resource management for cloud computing infrastructures. In *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 424–431, 2011.

[53] T. J. Hacker and K. Mahadik. Flexible resource allocation for reliable virtual cluster computing systems. In *Proc. ACM Conf. High Perf. Comp., Networking, Storage and Analysis*, page 48, 2011.

[54] M. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. C. Parkes. Online auctions with re-usable goods. In *Proc. 6th ACM Conf. on Electronic Commerce*, pages 165–174, 2005.

[55] M. T. Hajiaghayi, R. Kleinberg, and D. C. Parkes. Adaptive limited-supply online auctions. In *Proc. of the 5th ACM Conf. on Electronic commerce*, pages 71–80, 2004.

[56] J. Hamilton. Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services. In *Proc. of the Conf. on Innovative Data Systems Research*, 2009.

[57] J. D. Hartline and B. Lucier. Bayesian algorithmic mechanism design. In *Proc. of the 42nd ACM Symp. on Theory of computing*, pages 301–310, 2010.

[58] M. Hassan, B. Song, and E. Huh. Distributed resource allocation games in horizontal dynamic cloud federation platform. In *Proc. of the 13th IEEE Intl. Conf. on High Performance Computing & Communications*, pages 822–827, 2011.

[59] D. Hilley. Cloud computing: A taxonomy of platform and infrastructure-level offerings. *CERCS Report. College of Computing, George Institute of Technology*, 2009.

[60] M. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya. Randomized approximation scheme for resource allocation in hybrid-cloud environment. *The Journal of Supercomputing*, pages 1–17, 2014.

[61] M. Hu, J. Luo, and B. Veeravalli. Optimal provisioning for scheduling divisible loads with reserved cloud resources. In *Proc. 18th IEEE Int. Conf. on Networks*, pages 204–209, 2012.

[62] Y. Hua, X. Liu, and H. Jiang. Antelope: A semantic-aware data cube scheme for cloud data center networks. *IEEE Transactions on Computers*, 2014.

[63] H. Huang, L. Wang, B. C. Tak, L. Wang, and C. Tang. Cap 3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, pages 228–235, 2013.

[64] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Proc. of the IEEE 26th Conf. on Data Engineering Workshops*, pages 41–51, 2010.

[65] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu. Maestro: Replica-aware map scheduling for mapreduce. In *Proc. 12th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Comp.*, pages 435–442, 2012.

[66] G. N. Iyer and B. Veeravalli. On the resource allocation and pricing strategies in compute clouds using bargaining approaches. In *Proc. 17th IEEE Int. Conf. on Networks*, pages 147–152, 2011.

[67] V. Jalaparti and G. Nguyen. Cloud resource allocation games. *Technical report, University of Illinois at Urbana-Champaign (2010), http://hdl.handle.net/2142/17427*, 2010.

[68] I. Jangjaimon and N.-F. Tzeng. Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. *IEEE Transactions on Computers*, page 1, 2014.

[69] Z. Kang and H. Wang. A novel approach to allocate cloud resource with different performance traits. In *Proc. 10th IEEE Intl. Conf. on Services Computing*, pages 128–135, 2013.

[70] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In *Proc. 2nd IEEE Int'l Conf. on Cloud Computing Technology and Science*, pages 274–287, 2010.

[71] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[72] M. Kesavan, R. Soundararajan, A. Gavrilovska, I. Ahmad, O. Krieger, and K. Schwan. Practical compute capacity management for virtualized datacenters. *IEEE Trans. Cloud Comput.*, 1(1):88–100, 2013.

[73] A. Khosravi, S. K. Garg, and R. Buyya. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In *Proc. Int. European Conf. Parallel and Distrib. Comput.*, pages 317–328, 2013.

[74] D. Knuth. *The Art of Computer Programming, Volume 4, Combinatorial Algorithms, Part 1*. Addison-Wesley, 2011.

[75] P. Kokkinos, T. Varvarigou, A. Kretsis, P. Soumplis, and E. Varvarigos. Cost and utilization optimization of amazon ec2 instances. In *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, pages 518–525, 2013.

[76] J. Koomey. Growth in data center electricity use 2005 to 2010. *Oakland, CA: Analytics Press. August*, 1, 2011.

[77] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai. Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems. In *Proc. of IEEE INFOCOM*, 2014.

[78] S. Kurazumi, T. Tsumura, S. Saito, and H. Matsuo. Dynamic processing slots scheduling for i/o intensive jobs of hadoop mapreduce. In *Proc. of the 3rd IEEE Intl. Conf. on Networking and Computing*, pages 288–292, 2012.

[79] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz. Maximizing cloud provider profit from equilibrium price auctions. In *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, pages 83–90, 2012.

[80] W. Lang and J. M. Patel. Energy management for mapreduce clusters. *Proc. of the VLDB Endowment*, 3(1-2):129–139, 2010.

[81] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[82] D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial auctions*. MIT Press, Cambridge, MA, 2006.

[83] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya. Exploiting performance and cost diversity in the cloud. In *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, pages 107–114, 2013.

[84] J. Leverich and C. Kozyrakis. On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[85] H. Li, C. Wu, Z. Li, and F. Lau. Profit-maximizing virtual machine trading in a federation of selfish clouds. In *Proc. of the IEEE INFOCOM*, pages 25–29, 2013.

[86] B. Lucier and A. Borodin. Price of anarchy for greedy auctions. In *Proc. 21st ACM-SIAM Symp. on Discrete Algo.*, pages 537–553, 2010.

[87] N. Maheshwari, R. Nanduri, and V. Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Generation Computer Systems*, 28(1):119–127, 2012.

[88] L. Mashayekhy, N. Fisher, and D. Grosu. Truthful mechanisms for competitive reward-based scheduling. *IEEE Transactions on Computers (under second review)*, 2015.

[89] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. In *Proc. IEEE Intl. Perf. Comp. and Comm. Conf.*, pages 1–8, 2011.

[90] L. Mashayekhy and D. Grosu. A coalitional game-based mechanism for forming cloud federations. In *Proc. of the 5th IEEE Intl. Conf. on Utility and Cloud Computing*, pages 223–227, 2012.

[91] L. Mashayekhy and D. Grosu. A distributed merge-and-split mechanism for dynamic virtual organization formation in grids. In *Proc. 11th IEEE Intl. Conf. on Network Computing and Applications*, pages 36–43, 2012.

[92] L. Mashayekhy and D. Grosu. A reputation-based mechanism for dynamic virtual organization formation in grids. In *Proc. 41st IEEE Intl. Conf. on Parallel Processing*, pages 108–117, 2012.

[93] L. Mashayekhy and D. Grosu. A merge-and-split mechanism for dynamic virtual organization formation in grids. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):540–549, 2014.

[94] L. Mashayekhy and D. Grosu. Strategy-proof mechanisms for resource management in clouds. In *Proc. of the 14th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing*, pages 554–557, 2014.

[95] L. Mashayekhy and D. Grosu. A reputation-based mechanism for dynamic virtual organization formation in grids. *IEEE Transactions on Computers (under second review)*, 2015.

[96] L. Mashayekhy, M. Nejad, and D. Grosu. A truthful approximation mechanism for autonomic virtual machine provisioning and allocation in clouds. In *Proc. of the ACM Cloud and Autonomic Computing Conf.*, pages 1–10, 2013.

[97] L. Mashayekhy, M. Nejad, and D. Grosu. A framework for data protection in cloud federations. In *Proc. 43rd IEEE Intl. Conf. on Parallel Processing*, 2014.

[98] L. Mashayekhy, M. Nejad, and D. Grosu. A two-sided market mechanism for trading big data computing commodities. In *Proc. of the IEEE Intl. Conf. on Big Data*, 2014.

[99] L. Mashayekhy, M. Nejad, and D. Grosu. Cloud federations in the sky: Formation game and mechanism. *IEEE Transactions on Cloud Computing*, 3(1):14–27, 2015.

[100] L. Mashayekhy, M. Nejad, and D. Grosu. Physical machine resource management in clouds: A mechanism design approach. *IEEE Transactions on Cloud Computing*, PrePrints, 2014.

[101] L. Mashayekhy, M. Nejad, and D. Grosu. A PTAS mechanism for provisioning and allocation of heterogeneous cloud resources. *IEEE Transactions on Parallel and Distributed Systems*, PrePrints, 2014.

[102] L. Mashayekhy, M. Nejad, D. Grosu, D. Lu, and W. Shi. Energy-aware scheduling of mapreduce jobs. In *Proc. of the 3rd IEEE Intl. Congress on Big Data*, pages 32–39, 2014.

[103] L. Mashayekhy, M. Nejad, D. Grosu, and A. Vasilakos. An online mechanism for resource allocation and pricing in clouds. *IEEE Transactions on Computers (accepted)*, PrePrints, 2015.

[104] L. Mashayekhy, M. Nejad, D. Grosu, and A. V. Vasilakos. Incentive-compatible online mechanisms for resource provisioning and allocation in clouds. In *Proc. of the 7th IEEE Intl. Conf. on Cloud Computing*, pages 312–319, 2014.

[105] L. Mashayekhy, M. Nejad, D. Grosu, Q. Zhang, and W. Shi. Energy-aware scheduling of mapreduce jobs for big data applications. *IEEE Transactions on Parallel and Distributed Systems*, PrePrints, 2014.

[106] L. Mashayekhy, M. A. Nematbakhsh, and B. T. Ladani. E-negotiation model based on data mining. In *Proc. of the IADIS e-Commerce 2006 Intl. Conf.*, pages 369–373, 2006.

[107] L. Mashayekhy, M. A. Nematbakhsh, and B. T. Ladani. Measuring of strategies' similarity in automated negotiation. In *Proc. of the Intl. Conf. on Data Mining and Applications*, pages 859–863, 2007.

[108] L. Mashayekhy, M. A. Nematbakhsh, and B. T. Ladani. Detecting similar negotiation strategies. *Trends in Intelligent Systems and Computer Engineering*, pages 297–307, 2008.

[109] L. Mashayekhy, M. A. Nematbakhsh, and B. Tork. Comparing negotiation strategies based on offers. *Frontiers in Artificial Intelligence and Applications, Applications of Data Mining in E-Business and Finance*, page 87, 2008.

[110] C. Mastroianni, M. Meo, and G. Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Trans. Cloud Comput.*, 1(2):215–228, 2013.

[111] Y. Matsui and T. Matsui. Np-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1-2):305–310, 2001.

[112] M. Maurer, I. Brandic, and R. Sakellariou. Adaptive resource configuration for cloud infrastructure management. *Future Generation Comput. Syst.*, 29(2):472–487, 2013.

[113] M. Mazzucco, D. Dyachuk, and R. Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *Proc. 3rd IEEE Int. Conf. on Cloud Comput.*, pages 131–138, 2010.

[114] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[115] M. Mihailescu and Y. M. Teo. Dynamic resource pricing on federated clouds. In *Proc. of the 10th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing*, pages 513–517, 2010.

[116] M. Mihailescu and Y. M. Teo. The impact of user rationality in federated clouds. In *Proc. of the 12th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing*, pages 620–627, 2012.

[117] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. In *Proc. 23rd Annual ACM Symp. on Parallelism in Algorithms and Architectures*, pages 289–298, 2011.

[118] A. Mu'alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proc. of the 18th National Conf. on Artificial intelligence*, pages 379–384. American Association for Artificial Intelligence, 2002.

[119] A. Mu'Alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. *Games and Economic Behavior*, 64(2):612–631, 2008.

[120] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma. Job aware scheduling algorithm for mapreduce framework. In *Proc. of the IEEE 3rd Intl. Conf. on Cloud Computing Technology and Science*, pages 724–729, 2011.

[121] M. Nejad, L. Mashayekhy, R. Chinnam, and D. Grosu. Online scheduling and pricing for electric vehicle charging. *IIE Transactions (under second review)*, 2015.

[122] M. Nejad, L. Mashayekhy, R. Chinnam, and A. Phillips. Hierarchical Time-Dependent Shortest Path for Routing on Dynamic Road Networks under ITS. *IIE Transactions*, PrePrints, 2015.

[123] M. Nejad, L. Mashayekhy, and D. Grosu. A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. In *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, pages 188–195, 2013.

[124] M. Nejad, L. Mashayekhy, and D. Grosu. Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):594 – 603, 2015.

[125] M. Nejad, L. Mashayekhy, D. Grosu, and R. Chinnam. Optimal routing for plug-in hybrid electric vehicles. *INFORMS Transportation Science (under second review)*, 2014.

[126] M. Nejad, L. Mashayekhy, A. Taghavi, and R. Chinnam. State space reduction in modeling traffic network dynamics for dynamic routing under its. In *Proc. of the 14th Intl. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, pages 277–282, 2011.

[127] M. M. Nejad, L. Mashayekhy, and R. B. Chinnam. Effects of traffic network dynamics on hierarchical community-based representations of large road networks. In *Proc. of the 15th Intl. IEEE Conf. on Intelligent Transportation Systems*, pages 1900–1905, 2012.

[128] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. of the 31st annual ACM Symp. on Theory of computing*, pages 129–140, 1999.

[129] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic game theory.* Cambridge University Press, 2007.

[130] D. Niyato, A. Vasilakos, and Z. Kun. Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach. In *Proc. IEEE/ACM Int. Symp. Cluster, Cloud & Grid Comput.*, pages 215–224, 2011.

[131] A. Nordal, A. Kvalnes, J. Hurley, and D. Johansen. Balava: Federating private and public clouds. In *Proc. of the IEEE World Congress on Services*, pages 569–577, 2011.

[132] G. Owen. Multilinear extensions and the banzhaf value. *Naval Research Logistics Quarterly*, 22(4):741–750, 1975.

[133] G. Owen. *Game Theory.* Academic Press, New York, NY, USA, 3rd edition, 1995.

[134] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, 41(3):289–302, 2007.

[135] B. Palanisamy, A. Singh, and L. Liu. Cost-effective resource provisioning for mapreduce in a cloud. *IEEE Transactions on Parallel and Distributed Systems (forthcoming)*, 2014.

[136] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: locality-aware resource allocation for mapreduce in a cloud. In *Proc. Conf. High Performance Comp., Networking, Storage and Analysis*, 2011.

[137] C. Papagianni, V. Maglaris, C. Cervell, A. Leivadeas, S. Papavassiliou, et al. On the optimal allocation of virtual resources in cloud computing networks. *IEEE Transactions on Computers*, 62(6):1060–1071, 2013.

[138] M. Parashar, M. AbdelBaky, I. Rodero, and A. Devarakonda. Cloud paradigms and practices for computational and data-enabled science and engineering. *Computing in Science & Engineering*, 15(4):10–18, 2013.

[139] D. C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[140] D. C. Parkes and S. Singh. An MDP-based approach to online mechanism design. In *Proc. 17th Annual Conf. on Neural Information Processing Systems*, 2003.

[141] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, and P. Michiardi. Hfsp: size-based scheduling for hadoop. In *Proc. of IEEE Intl. Conf. on Big Data*, pages 51–59, 2013.

[142] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé. Resource-aware adaptive scheduling for mapreduce clusters. In *Proc. of the 12th ACM/IFIP/USENIX Intl. Middleware Conf.*, pages 187–207, 2011.

[143] M. Polverini, A. Cianfrani, S. Ren, and A. Vasilakos. Thermal-aware scheduling of batch jobs in geographically distributed data centers. *IEEE Trans. Cloud Comput.*, (PrePrints):1, 2013.

[144] R. Porter. Mechanism design for online real-time scheduling. In *Proc. 5th ACM Conf. on Electronic Commerce*, pages 61–70, 2004.

[145] A. Prasad and S. Rao. A mechanism design approach to resource procurement in cloud computing. *IEEE Transactions on Computers*, 2014.

[146] S. Rampersaud, L. Mashayekhy, and D. Grosu. Computing nash equilibria in bimatrix games: Gpu-based parallel support enumeration. In *Proc. the 31st IEEE Intl. Perf. Comp. and Comm. Conf.*, pages 332–341, 2012.

[147] S. Rampersaud, L. Mashayekhy, and D. Grosu. Computing nash equilibria in bimatrix games: Gpu-based parallel support enumeration. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3111–3123, 2014.

[148] Z. Ren, X. Xu, M. Zhou, J. Wan, and W. Shi. Workload analysis, implications and optimization on a production hadoop cluster: A case study on taobao. *IEEE Transactions on Services Computing*, 7(2):307–321, 2014.

[149] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, et al. Reservoir-when one cloud is not enough. *Computer*, 44(3):44–51, 2011.

[150] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, et al. The reservoir model and architecture for open federated cloud computing. *IBM J. of Res. and Dev.*, 53(4):4–1, 2009.

[151] B. Rochwerger, C. Vázquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, et al. An architecture for federated cloud computing. *Cloud Computing*, pages 391–411, 2010.

[152] M. Rodriguez and R. Buyya. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans. Cloud Comput.*, 99(PrePrints):1, 2014.

[153] M. A. Salehi, P. Radha Krishna, K. S. Deepak, and R. Buyya. Preemption-aware energy management in virtualized data centers. In *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, pages 844–851, 2012.

[154] N. Samaan. A novel economic sharing model in a federation of selfish cloud providers. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):12–21, 2014.

[155] T. Sandholm and K. Lai. Mapreduce optimization using regulated dynamic prioritization. In *Proc. 11th ACM Int'l Conf. on Measurement and Modeling of Computer Syst.*, pages 299–310, 2009.

[156] T. Sandholm and K. Lai. Dynamic proportional share scheduling in hadoop. In *Job scheduling strategies for parallel processing*, pages 110–131. Springer, 2010.

[157] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111:209–238, 1999.

[158] L. S. Shapley. A value for n-person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Ann. Math. Studies*, pages 307–317. Princeton University Press, Princeton, New Jersey, USA, 1953.

[159] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Proc. of the 31st Intl. Conf. on Distributed Computing Systems*, pages 559–570, 2011.

[160] S. Shen and J. Wang. Stochastic modeling and approaches for managing energy footprints in cloud computing service. *Service Science*, 6(1):15–33, 2014.

[161] W. Song, Z. Xiao, Q. Chen, and H. Luo. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Transactions on Computers*, 2014.

[162] Y. Song, Y. Sun, and W. Shi. A two-tiered on-demand resource allocation mechanism for vm-based data centers. *IEEE Transactions on Services Computing*, 6(1):116–129, 2013.

[163] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proc. USENIX Workshop on Power Aware Comput. and Syst.*, volume 10, 2008.

[164] F. Teng and F. Magoules. Resource pricing and equilibrium allocation policy in cloud computing. In *Proc. 10th IEEE Int. Conf. Computer and Inf. Tech.*, pages 195–202, 2010.

[165] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *Proc. IEEE Int'l Conf. on Cloud Computing*, pages 155–162, 2011.

[166] A. Toosi, R. Calheiros, R. Thulasiram, and R. Buyya. Resource provisioning policies to increase IaaS provider's profit in a federated cloud environment. In *Proc. of the 13th IEEE Intl. Conf. on High Performance Computing & Communications*, pages 279–287, 2011.

[167] C.-W. Tsai, W.-C. Huang, M.-C. Chiang, C.-S. Yang, and M.-H. Chiang. A hyper-heuristic scheduling algorithm for cloud. *IEEE Trans. Cloud Comput.*, 99(PrePrints):1, 2014.

[168] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing*, pages 228–235, 2010.

[169] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[170] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In *Proc. 8th ACM Int'l Conf. on Autonomic Comp.*, pages 235–244, 2011.

[171] A. Verma, L. Cherkasova, and R. H. Campbell. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In *Proc. 20th IEEE Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecom. Syst.*, pages 11–18, 2012.

[172] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[173] L. Wang and S. U. Khan. Review of performance metrics for green data centers: a taxonomy study. *The Journal of Supercomputing*, 63(3):639–656, 2013.

[174] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu. Joint virtual machine assignment and traffic engineering for green data center networks. *SIGMETRICS Performance Evaluation Review*, 41(3):107–112, 2013.

[175] W. Wang, B. Li, and B. Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *Proc. of IEEE INFOCOM*, 2014.

[176] X. Wang, D. Shen, G. Yu, T. Nie, and Y. Kou. A throughput driven task scheduler for improving mapreduce performance in job-intensive environments. In *Proc. of the 2nd IEEE Intl. Congress on Big Data*, pages 211–218, 2013.

[177] Y. Wang, A. Nakao, and A. V. Vasilakos. Heterogeneity playing key role: Modeling and analyzing the dynamics of incentive mechanisms in autonomous networks. *ACM Transactions on Autonomous & Adaptive Systems*, 7(3):31, 2012.

[178] J. Watzl. *A framework for exchange-based trading of cloud computing commodities*. PhD thesis, Ludwig Maximilian University of Munich, 2014.

[179] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2):252–269, 2010.

[180] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2):252–269, 2010.

[181] T. Wirtz and R. Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *Proc. of the IEEE Intl. Green Computing Conf. and Workshops*, pages 1–8, 2011.

[182] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In *Proc. of the ACM/IFIP/USENIX 11th Int'l Conf. on Middleware*, pages 1–20, 2010.

[183] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of the 4th USENIX conf. on Networked systems design & implementation*, volume 7, pages 11–13, 2007.

[184] Z. Xiao, Q. Chen, and H. Luo. Automatic scaling of internet applications for cloud computing services. *IEEE Transactions on Computers*, 2014.

[185] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu. Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach. In *Proc. of the 31st Intl. Conf. on Distributed Computing Systems*, pages 571–580, 2011.

[186] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proc. of the IEEE*, 102(1):11–31, 2014.

[187] H. Xu and B. Li. Dynamic cloud pricing for revenue maximization. *IEEE Trans. Cloud Comput.*, 1(2):158–171, 2013.

[188] X. Yang, B. Nasser, M. Surridge, and S. Middleton. A business-oriented cloud federation model for real-time applications. *Future Generation Computer Systems*, 28(8):1158–1167, 2012.

[189] S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Proc. 3rd IEEE Int. Conf. Cloud Comput.*, pages 236–243, 2010.

[190] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, UC Berkeley, April 2009.

[191] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proc. of the 8th USENIX Conf. on Operating systems design and implementation*, pages 29–42, 2008.

[192] S. Zaman and D. Grosu. An online mechanism for dynamic VM provisioning and allocation in clouds. In *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, pages 253–260, 2012.

[193] S. Zaman and D. Grosu. Combinatorial auction-based allocation of virtual machine instances in clouds. *Journal of Parallel & Distributed Computing*, 73(4):495–508, 2013.

[194] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang. Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers. *IEEE Transactions on Computers*, 62(11):2155–2168, 2013.

[195] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proc. of IEEE INFOCOM*, 2013.

[196] L. Zhang, Z. Li, and C. Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proc. of IEEE INFOCOM*, pages 433–441, 2014.

[197] L. Zhang, Z. Li, C. Wu, and M. Chen. Online algorithms for uploading deferrable big data to the cloud. In *Proc. of IEEE INFOCOM*, 2014.

[198] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang. Optimal resource rental planning for elastic applications in cloud market. In *Proc. of the IEEE 26th Intl. Symp. on Parallel & Distributed Processing*, pages 808–819, 2012.

[199] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. Lau. Dynamic pricing and profit maximization for clouds with geo-distributed datacenters. In *Proc. of IEEE INFOCOM*, 2014.

[200] Y. Zheng, N. B. Shroff, and P. Sinha. A new analytical technique for designing provably efficient mapreduce schedulers. In *Proc. of the IEEE INFOCOM*, pages 1600–1608, 2013.

ABSTRACT

RESOURCE MANAGEMENT IN CLOUD AND BIG DATA SYSTEMS

by

LENA MASHAYEKHY

August 2015

Advisor: Dr. Daniel Grosu

Major: Computer Science

Degree: Doctor of Philosophy

Cloud computing is a paradigm shift in computing, where services are offered and acquired on demand in a cost-effective way. These services are often virtualized, and they can handle the computing needs of big data analytics. The ever-growing demand for cloud services arises in many areas including healthcare, transportation, energy systems, and manufacturing. However, cloud resources such as computing power, storage, energy, dollars for infrastructure, and dollars for operations, are limited. Effective use of the existing resources raises several fundamental challenges that place the cloud resource management at the heart of the cloud providers' decision-making process. One of these challenges faced by the cloud providers is to provision, allocate, and price the resources such that their profit is maximized and the resources are utilized efficiently. In addition, executing large-scale applications in clouds may require resources from several cloud providers. Another challenge when processing data intensive applications is minimizing their energy costs. Electricity used in US data centers in 2010 accounted for about 2% of total electricity used nationwide. In addition, the energy consumed by the data centers is growing at over 15% annually, and the energy costs make up about 42% of the data centers' operating costs. Therefore, it is critical for the data centers to minimize their energy consumption when offering services to customers. In this Ph.D. dissertation, we address these challenges by designing, developing, and analyzing mechanisms for resource management in cloud computing systems and data centers. The goal is to allocate resources efficiently while optimizing a global performance

objective of the system (e.g., maximizing revenue, maximizing social welfare, or minimizing energy). We improve the state-of-the-art in both methodologies and applications. As for methodologies, we introduce novel resource management mechanisms based on mechanism design, approximation algorithms, cooperative game theory, and hedonic games. These mechanisms can be applied in cloud virtual machine (VM) allocation and pricing, cloud federation formation, and energy-efficient computing. In this dissertation, we outline our contributions and possible directions for future research in this field.

AUTOBIOGRAPHICAL STATEMENT

Lena Mashayekhy received her BSc degree in computer science from Iran University of Science and Technology, and her MSc degree from the University of Isfahan. She is currently a PhD candidate in computer science at Wayne State University, Detroit, Michigan. She has published more than thirty peer-reviewed papers in venues such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE BigData, IEEE CLOUD, and ICPP. She received several awards including: the 2014 INFORMS ENRE Best Student Paper Award, the 2014 INFORMS Service Science Best Paper Runner-Up Award, the 2014 POMS College of Sustainable Operations Best Student Paper Runner-Up Award, and the Ralph Kummler H. Award for Distinguished Achievement in Graduate Student Research. Her research interests include Cloud Computing, Big Data, Parallel Algorithms, Sustainable Computing, Electric Vehicles, and Game Theory. She is a student member of the ACM, the IEEE, and the IEEE Computer Society.

Following graduation, Lena will join the University of Delaware as an assistant professor.