Performance Modeling of Cloud Computing Centers

by

Hamzeh Khazaei

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Doctor of Philosophy

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
January 2013

Thesis advisor

Jelena Mišić and Raşit Eskicioğlu

Author

Hamzeh Khazaei

Performance Modeling of Cloud Computing Centers

Abstract

Cloud computing is a general term for system architectures that involves delivering hosted services over the Internet, made possible by significant innovations in virtualization and distributed computing, as well as improved access to high-speed Internet. A cloud service differs from traditional hosting in three principal aspects. First, it is provided on demand, typically by the minute or the hour; second, it is elastic since the user can have as much or as little of a service as they want at any given time; and third, the service is fully managed by the provider – user needs little more than computer and Internet access. Typically a contract is negotiated and agreed between a customer and a service provider; the service provider is required to execute service requests from a customer within negotiated quality of service (QoS) requirements for a given price.

Due to dynamic nature of cloud environments, diversity of user's requests, resource virtualization, and time dependency of load, providing expected quality of service while avoiding over-provisioning is not a simple task. To this end, cloud provider must have efficient and accurate techniques for performance evaluation of cloud computing centers. The development of such techniques is the focus of this thesis.

This thesis has two parts. In first part, Chapters 2, 3 and 4, monolithic performance models are developed for cloud computing performance analysis. We begin with Poisson

task arrivals, generally distributed service times, and a large number of physical servers. Later on, we extend our model to include finite buffer capacity, batch task arrivals, and virtualized servers with a large number of virtual machines in each physical machine.

However, a monolithic model may suffer from intractability and poor scalability due to large number of parameters. Therefore, in the second part of the thesis (Chapters 5 and 6) we develop and evaluate tractable functional performance sub-models for different servicing steps in a complex cloud center and the overall solution obtain by iteration over individual sub-model solutions. We also extend the proposed interacting analytical sub-models to capture other important aspects including pool management, power consumption, resource assigning process and virtual machine deployment of nowadays cloud centers. Finally, a performance model suitable for cloud computing centers with heterogeneous requests and resources using interacting stochastic models is proposed and evaluated.

Contents

List of Figures

List of Tables

Acknowledgments

It would not have been possible to write this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

First and foremost, my utmost gratitude to Prof. Jelena Mišić, whose guidance, support and encouragement I will never forget. Prof. J. Mišić has been my inspiration as I hurdle all the obstacles in the completion this research work.

Next, I would like to thank my co-supervisor Dr. Raşit Eskicioğlu for his help and contribution on this work. My special thank goes to Prof. Vojislav B. Mišić for all his great contribution, support and friendship that have been invaluable on both academic and personal level, not to mention his advices and unsurpassed knowledge of cloud computing.

Amongst my fellow postgraduate students in the department of computer science, I wish to thank my friends Saeed Rashwand and Subir Biswas for their consults and friendly advices. Last, but by no means least, I am most grateful to my dear wife, Nasim Beigi-Mohammadi, for her everlasting support and encouragement.

Any errors or inaccuracies that remain in this work, of course, are entirely my own and I am responsible for them.

This thesis is dedicated to my wonderful wife for all her kindness and unbounded support.

Chapter 1

Introduction

In this Chapter, we first describe *Cloud Computing* as a new paradigm of computing. The main challenge and obstacle of cloud computing, *Quality of Service* (QoS), is defined and the general aspects of QoS are explored. Later the performance evaluation in cloud computing, the related work and the challenges are highlighted. We wrap up the Chapter with outline and organization of the thesis proposal.

1.1 Cloud Computing

With the advancement of human society, basic essential services are commonly provided such that everyone can easily obtain access to them. Today, utility services such as water, electricity, gas, and telephony are deemed necessary for fulfilling daily life routines. These utility services are accessed so frequently that they need to be available at any time. Consumers are then able to pay service providers based on their usage of these utility services. In 1969, Leonard Kleinrock said: “As of now, computer networks are still in their

infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country." [58] This vision of the computing utility based on a service provisioning model anticipates the massive transformation of the entire computing industry in the 21st century, whereby computing services will be readily available on demand, like other utility services. As a result, there will be no need for the consumers to invest heavily in building and maintaining complex IT infrastructure [11]; instead, they will pay only when they access computing services. Nowadays, significant innovations in virtualization and distributed computing, as well as improved access to high-speed Internet, have accelerated interest in cloud computing [94]. It is quickly gaining acceptance: According to IDC, 17 billion dollars was spent on cloud-related technologies, hardware and software in 2009, and spending is expected to grow to 45 billion by 2013 [74]. Cloud computing has a service oriented architecture in which services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), where equipment such as hardware, storage, servers, and networking components are made accessible over the Internet); Platform-as-a-Service (PaaS), which includes computing platforms—hardware with operating systems, virtualized servers, and the like; and Software-as-a-Service (SaaS), which includes software applications and other hosted services [23].

A cloud service differs from traditional hosting in three principal aspects. First, it is provided on demand, typically by the minute or the hour; second, it is elastic since the user can have as much or as little of a service as they want at any given time; and third, the service is fully managed by the provider [40].

There is no unique definition for cloud computing. In Table 1.1, we present some com-

Table 1.1: Cloud Computing: Some Definitions.

Author/Reference	**Year**	**Definition**
R. Buyya [11]	2008	A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.
E. Hand [31]	2007	Cloud computing is a new computing paradigm, in which not just our data but even our software resides in there and we access everything not only through our PCs but also Cloud-friendly devices, such as smart phones and PDAs.
L. Wang [99]	2010	A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way.
I. Foster [21]	2008	A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.
L. Vaquero [94]	2008	Cloud Computing is a novel paradigm for the provision of computing infrastructure, which aims to shift the location of the computing infrastructure to the network in order to reduce the costs associated to management and maintenance of hardware and software resources.

mon definitions. we also state our definition of cloud computing as follows:

"Cloud computing is a new computing paradigm, whereby shared resources such as infrastructure, hardware platform, and software applications are provided to users on-demand over the Internet (Cloud) as services."

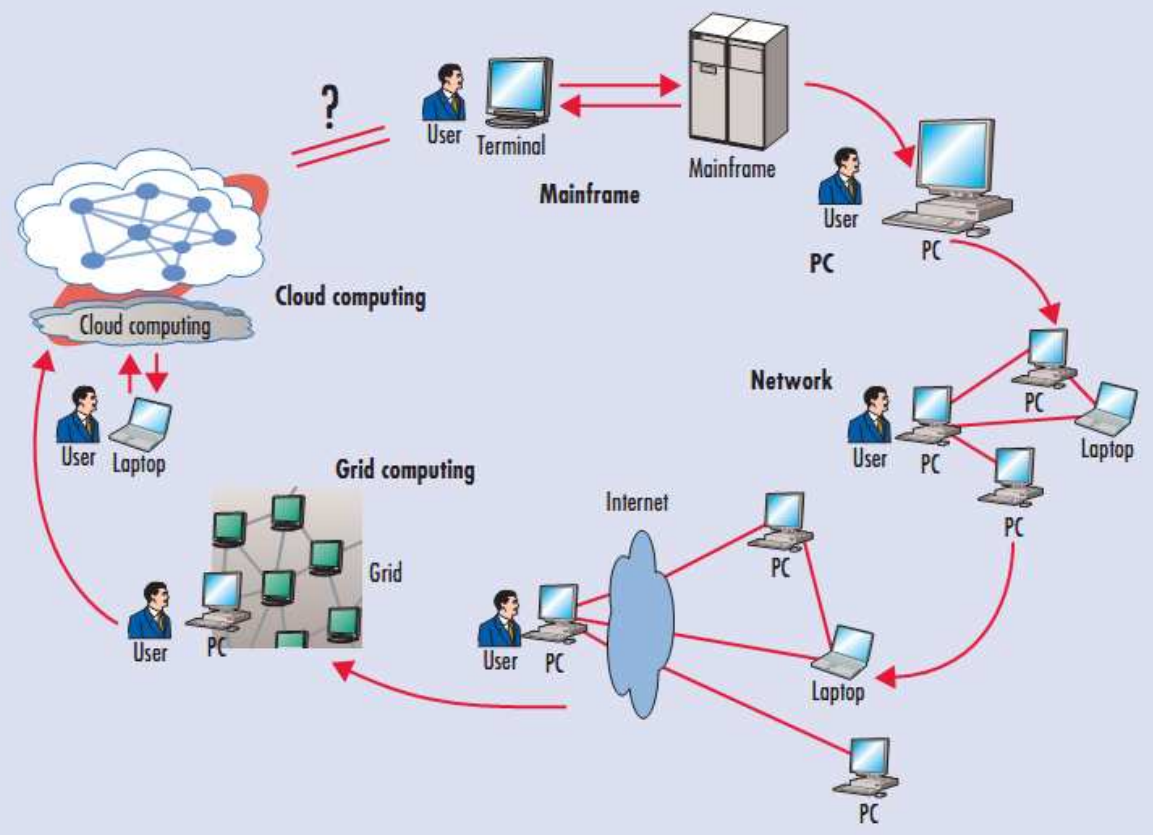


Figure 1.1: Computing paradigm shift. Over six distinct phases, computers have evolved from dummy terminals to grids and clouds [97].

Fig. 1.1 illustrates the computing paradigm shift of the last half century [97]. Specifically, the figure identifies six distinct phases. In Phase 1, people used terminals to connect to powerful mainframes shared by many users. Back then, terminals were basically little more than keyboards and monitors. In Phase 2, stand-alone personal computers (PCs) became powerful enough to satisfy users' daily work; One didn't have to share a mainframe with anyone else. Phase 3 ushered in computer networks that allowed multiple computers to connect to each other. One could work on a PC and connect to other computers through local networks to share resources. Phase 4 saw the advent of local networks that could

connect to other local networks to establish a more global network users could now connect to the Internet to utilize remote applications and resources. Phase 5 brought us the concept of an electronic grid to facilitate shared computing power and storage resources (distributed computing). People used PCs to access a grid of computers in a transparent manner. Now, in Phase 6, cloud computing lets us exploit all available resources on the Internet in a scalable and simple way.

Armbrust et al. [5] have listed 10 top obstacles and opportunities for growth of Cloud Computing (Table 1.2); the first three concern adoption, the next five affect growth, and the last two are policy and business obstacles. Each obstacle is paired with an opportunity, ranging from product development to research projects, which can overcome that obstacle.

Table 1.2: Top 10 Obstacles to and Opportunities for Growth of Cloud Computing [5].

	Obstacle	**Opportunity**
1	Availability of Service	Use Multiple Cloud Providers; Use Elasticity to Prevent DDOS
2	Data Lock-In	Standardize APIs; Compatible SW to enable Surge Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls; Geographical Data Storage
4	Data Transfer Bottlenecks	FedExing Disks; Data Backup/Archival; Higher BW Switches
5	Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on ML; Snapshots for Conservation
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses; Bulk use sales

Although benefits and opportunities that cloud computing has been bringing about are tremendous, its challenges and problems that require huge amount of effort to be addressed by researchers. Since 2007 that cloud computing has gotten high attention until now, the

trend of search volume index has been increasing sharply. Fig. 1.2 shows the trend of search volume index for Cloud Computing versus Grid Computing using Google trend service during past years.

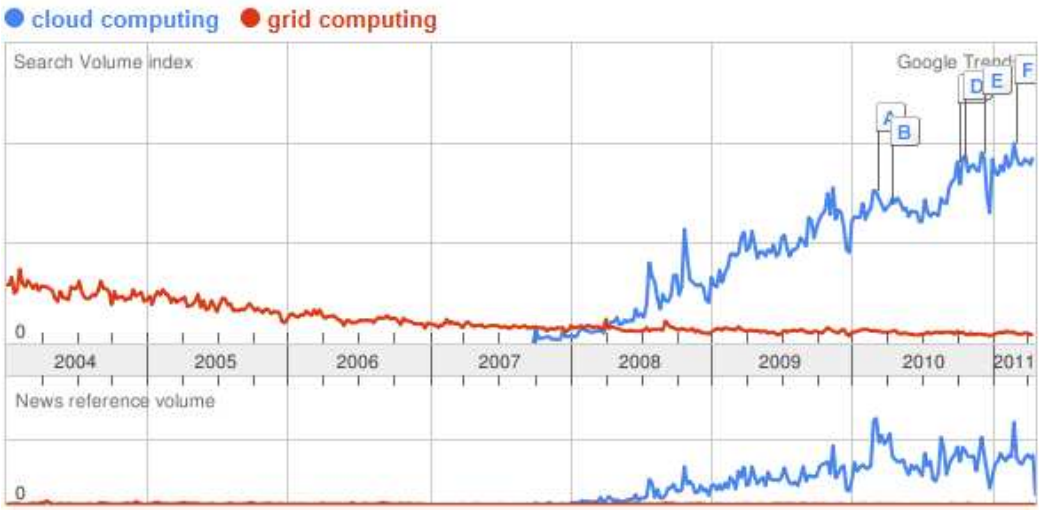


Figure 1.2: Search Volume index for Cloud Computing vs Grid Computing; provided by Google trends on April 2011.

1.2 Quality of Service

The success of next-generation cloud computing infrastructures will depend on how effectively these infrastructures will be able to instantiate and dynamically maintain computing platforms, constructed out of cloud resources and services, that meet varying resource and service requirements of cloud costumer applications, which are characterized by Quality of Service (QoS) requirements, such as timeliness, scalability, high availability, trust, security, specified in the so-called Service Level Agreements (SLAs). In essence, an SLA is a legally binding contract which states the QoS guarantees that an execution environ-

ment, in this case a cloud based computing platform, has to provide its hosted applications with [20].

Because everything is delivered to cloud users as services, QoS has a great impact on growth and acceptability of cloud computing paradigm. Both industrial and research communities are showing increases of interest into QoS assurance within the context of cloud computing. In order that cloud service providers can provide cloud users with the appropriate quality of service, they must be guaranteed a determined QoS from the provider server.

Providing quality services require a solid model that provides detailed insights of computing centers. As our discussion of existing research on performance modeling (section 1.4) shows, there is no proper model in literature that adequately addresses the scale, diversity and dynamic of today cloud computing centers. The first intent of this work is to propose an analytical model for performance modeling of cloud computing centers. Provided that offering quality services may be more attainable for cloud services providers.

1.3 Performance Analysis of Cloud Computing Centers

Due to dynamic and virtualized nature of cloud environments, diversity of user's requests and time dependency of load, providing expected quality of service while avoiding over-provisioning is not a simple task [101]. To ensure that the QoS perceived by end clients is acceptable, the providers must exploit techniques and mechanisms that guarantee a minimum level of QoS. Although QoS has multiple aspects such as response time, throughput, availability, reliability, and security, the primary aspect of QoS considered in this work is related to response time [25].

Cloud computing has been the focus of much research in both academia and industry, however, implementation-related issues have received much more attention than performance-related ones; in this work we describe an analytical model for evaluating the performance of cloud server farms under vast variety of configurations and assumptions and verify its accuracy with numerical calculations and simulations.

SLA outlines all aspects of cloud service usage and the obligations of both service providers and clients, including various performance descriptors collectively referred to as QoS, which may include response time, availability, throughput, reliability, security, and many others. An important subset of performance indicators , which this work deals with, consists of response time, task blocking probability, probability of immediate service, and mean number of tasks in the system [99]. These indicators may be determined using the tools of queuing theory [57]; however, cloud centers differ from traditional queuing systems in a number of important aspects:

- A cloud center can have a large number of facility (server) nodes, typically of the order of hundreds or thousands [2]; traditional queuing analysis rarely considers systems of this size.

- Task service times must be modeled by a general, rather than the more convenient exponential, probability distribution. Consider, for example, the Amazon Elastic Compute Cloud (EC2) which offers three types of services [2]: *Reserved* services are reserved and paid for in advance, so they are guaranteed by the provider and thus experience virtually no queuing; *Spot* services are also allocated in advance, but they go to the customer who offer a higher bid than Spot price set by Amazon; Spot Prices fluctuate periodically depending on the supply of and demand for Spot Instance ca-

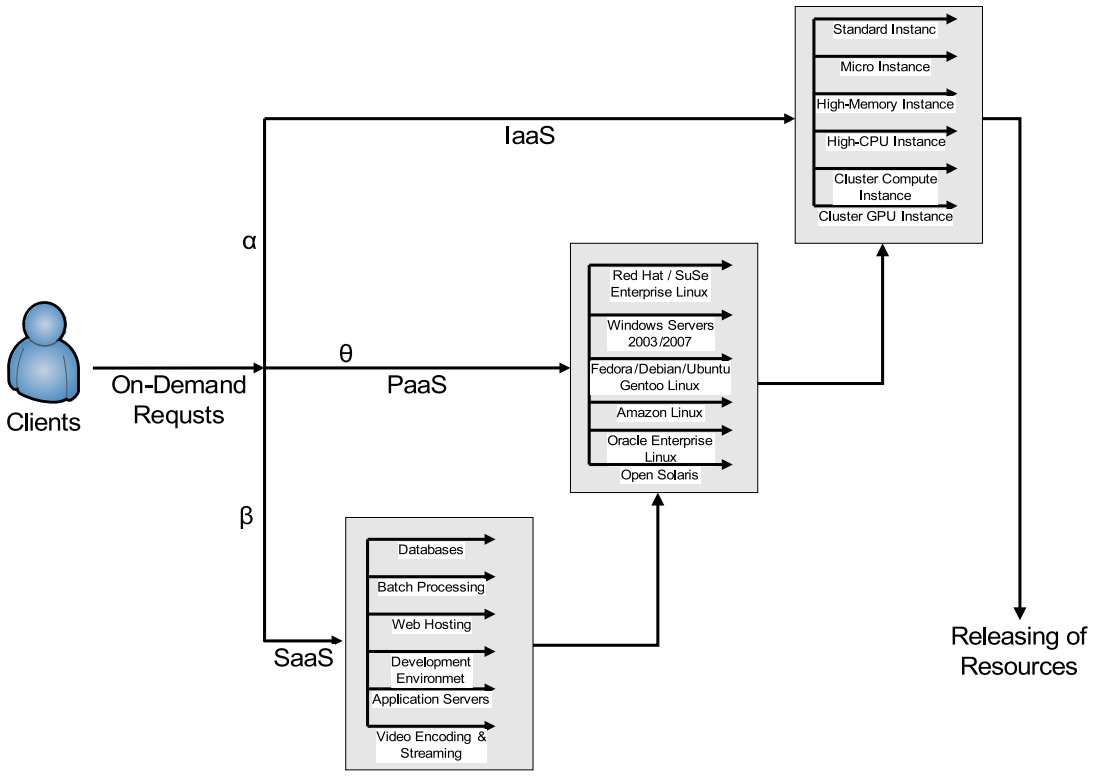


Figure 1.3: Amazon EC2 structure, adapted from [2].

pacity. Finally, *On-Demand* services provide no advance reservations and no long term commitment, which is why the clients' tasks may experience non-negligible queuing delays and, possibly, blocking. Within the on-demand category, a request may target a specific infrastructure instance, a platform, or a software application, with probability of α, θ, and β, respectively, as shown in Fig. 1.3; the same story is held within each tuple. Because we assumed that task requests arrive one at a time, all the probabilities, α, θ, and β, as well as branching probabilities within tuples are independent of each other. Assuming that the service time for each tuple follows a simple exponential or Erlang distribution, the aggregate service time of the cloud

center would follow a hyper-exponential or hyper-Erlang distribution – one in which the coefficient of variation, CoV, defined as the ratio of standard deviation and mean value, exceeds one [13]. Consequently, even in the above-mentioned scenario, which is a rather simple one, only a general assumption of distribution can substantiate the actual task service time; in practical calculations, we will have to choose a specific probability distribution that allows widely varying values for the coefficient of variation CoV.

- Finally, due to the dynamic nature of cloud environments, diversity of user's requests and time dependency of load, cloud centers must provide expected quality of service at widely varying loads [101, 7].

1.4 Performance Modeling: a Review

Cloud computing has attracted considerable research attention, but only a small portion of the work done so far has addressed performance issues, and the rigorous analytical approach has been adopted by only a handful among these. Xiong et al. [101] modeled a cloud center as the classic open network, from which the distribution of response time is obtained, assuming that both inter-arrival and service times are exponential. Using the distribution of response time, the relationship among the maximal number of tasks, the minimal service resources and the highest level of services was found.

Yang et al. [102] modeled the cloud center as an $M/M/m/m+r$ queuing system from which the distribution of response time was determined. Inter-arrival and service times were both assumed to be exponentially distributed, and the system had a finite buffer of

size $m + r$. The response time was broken down into waiting, service, and execution periods, assuming that all three periods are independent (which is unrealistic, according to authors' own argument).

Yigitbasi et al. [107] studied the response time in terms of various metrics, such as the overhead of acquiring and realizing the virtual computing resources, and other virtualization and network communication overhead. To address these issues, they have designed and implemented C-Meter, a portable, extensible, and easy-to-use framework for generating and submitting test workloads to computing clouds.

1.5 Performance Modeling Based on Queuing Theory

As we mentioned in section 1.3, a large queuing system (number of servers is large, e.g., more than 100) that is assumed to have Poisson arrival process and generally distributed service time can be a proper abstract model for performance evaluation of a cloud center. However analyzing queuing system with generally distributed service time is not an easy task; the only generally distributed service time queuing systems that have exact and closed-form solution are the $M/G/1$ and $M/G/m/m$, which are not quite suitable for cloud computing performance modeling. Cloud centers have large number of servers as well as extra capacity for holding upcoming requests.

In general, analysis in the case where either inter-arrival or service times (or both) are not exponential is very complex. Theoretical analyses have mostly relied on extensive research in performance evaluation of $M/G/m$ queuing systems, as outlined in [35, 64, 68, 71, 72, 103]. As solutions for mean response time and queue length in $M/G/m$ systems cannot be obtained in closed form, suitable approximations were sought. However, most

of these provide reasonably accurate estimates of mean response time only when number of servers is comparatively small, (say, less than ten or so), but fail for large number of servers [10, 53, 87, 92]. Approximation errors are particularly pronounced when the offered load ρ is small, and/or when both the number of servers m and the coefficient of variation of service time distribution, *CoV*, are large.

An approximate solution for steady-state queue length distribution in an $M/G/m$ system with finite waiting space was described in [55]. As the approximation was given in an explicit form, its numerical computation is easier than when using earlier approximations [35, 91]. The proposed approach is exact for $M/G/m/m + r$ when $r = 0$, and reasonably accurate in the general case when $r \neq 0$, but only when the number of servers m is small, below 10 or so.

A similar approach in the context of $M/G/m$ queues, but extended so as to approximate the blocking probability and, thus, to determine the smallest buffer capacity such that the rate of lost tasks remains under predefined level, was described in [54]. An interesting finding is that the optimal buffer size depends on the order of convexity for the service time; the higher this order is, the larger the buffer size should be.

An approximation for the average queuing delay in an $M/G/m/m + r$ queue, based on the relationship of joint distribution of remaining service time to the equilibrium service distribution, was proposed in [71]. Another approximation for the blocking probability in $M/G/m/m + r$ queues, based on the exact solution for finite capacity $M/M/m/m + r$ queues, was proposed in citesmith. Again, the estimate of the blocking probability is used to guide the allocation of buffers so that the loss/delay blocking probability remains below a specific threshold.

It is worth noting that most of these results rely on some approximation(s) to obtain a closed-form solution; as a result, they are applicable in a limited range of values of the relevant system parameters:

- Approximations are reasonably accurate only when the number of servers is comparatively small, typically below 10 or so, which makes them unsuitable for performance analysis of cloud computing data centers.

- Approximations are very sensitive to the probability distribution of task service times, and they become increasingly inaccurate when the coefficient of variation of the service time, CoV, increases toward and above the value of one.

- Finally, approximation errors are particularly pronounced when the traffic intensity ρ is small, and/or when both the number of servers m and the CoV of the service time, are large [10, 53, 92].

Therefore, the results mentioned above are not directly applicable to performance analysis of cloud computing server farms where the number of servers is huge, the distribution of service times is unknown and does not, in general, follow any of the 'well-behaved' probability distributions such as exponential distribution, and where the offered load can vary from very low values to those close to one. Consequently, we developed the model for performance evaluation of cloud centers that will be described and evaluated in the next Chapters.

1.6 Heterogeneity in Cloud Computing

One of the prime challenges in cloud computing performance modeling is the innate heterogeneity in such environments. We plan to cover this issue in this thesis partially and leave the rest as future work. The main contributions in this area have been listed in followings.

Rosenberg et al. [77] dealt with heterogeneity in computing. They propose a framework and obtain three results: (1) if one can replace only one computer in a cluster by a faster one, then it is (almost) always most advantageous to replace the fastest computer;(2) if the computers in two n-computer clusters have the same mean speed, then the cluster with the larger variance in computers speeds is (almost) always the faster one; (3) heterogeneity can actually lend power to a cluster.

Rosenberg et al. [77] considered the impact of server heterogeneity on the performance of server systems. They show later heterogeneity is not always detrimental using a simple analytical model for a system with two clusters of two servers each; larger system are dealt with through simulation.

Yeo et al. [106] tried to answer this question: *how slow a physical node can be for a given task to maintain its optimal computing quality in terms of execution time and energy cost*? To tackle the issue, a probabilistic model for heterogeneous cloud environment is established. Using proposed model, the authors evaluate the trade-off of task execution time and energy consumed by task execution. Their finding suggest that computing nodes that are 3x or slower that the fastest node should be discarded from the cloud for achieving an optimal energy delay product. The authors do not discuss whether such a strategy is still optimal in terms of other performance indicators or not. Moreover, in cloud computing

centers, their suggestion might not be completely true where virtualization may help to make the most of slow servers.

Yeo et al. [106] discussed the limiting values of node (servers) speed needed to maintain prescribed values of executing time and energy cost. Using a probabilistic model for a heterogeneous cloud environment, they however evaluated the trade-of of task execution time and energy consumed by task execution. The authors do not discuss whether such a strategy is optimal in terms of other performance indicators or not. Moreover, in cloud computing centers, their suggestion might not be completely true where virtualization may help to make the most of slow servers.

1.7 Cloud-Specific Task Assignment Policy

Assuming heterogeneous environment brings another challenge which is known as decision making of task assignment policy. Although this issue has been extensively studied in the area of cluster and grid computing, proposed techniques are not quite applicable/useful to cloud computing paradigm due to different nature of cloud architecture and user requests. One direction of future research can be proposing an efficient task assignment strategy whereby the utilization of cloud resources as well as the performance indicators (e.g., waiting time, response time, blocking probability and probability of immediate service) are improved simultaneously. Now, we survey existing literature in task assignment policies in clouds.

Moon et al. [69] presented the analysis of resource scheduling to achieve SLA-aware profit optimization in cloud services. The priority of jobs in the queue is individually evaluated and then the job with the highest priority is selected. Priority is computed based

on a task cost function and the penalty cost associated with tasks while they are waiting in the queue.

A green scheduling algorithm integrating a neural network predictor for optimizing server power consumption in cloud computing is proposed in [16]. The authors employ the predictor to predict future load demand based on historical demand. According to the prediction, the algorithm turns off unused servers and restarts them to minimize the number of running servers, thus minimizing the energy use at the points of consumption to benefit all other levels. In this work, task rejection rate is the only criterion for comparing the proposed algorithm with the others.

A number of green task scheduling algorithms are implemented in [109]. They have two main steps namely: assigning as many tasks as possible to a cloud server with lowest energy, and setting the same optimal speed for all tasks assigned to each cloud server. These green algorithms are developed for heterogeneous cloud servers with adjustable speed and other parameters to effectively reduce energy consumption and finish all tasks before a deadline. Using simulation they show that three of their algorithms outperform random algorithm and they also identify the shortest task first (STF) as the best algorithm among six proposed algorithms.

Load distribution and balancing in general parallel and distributed computing systems have been extensively studied and a large body of literature exists (see the excellent collection [83]). In particular, the problems of optimal load distribution have been investigated by using queuing models [39, 89], with various performance metrics such as weighted mean waiting time [9], arithmetic average response time [59, 82], probability of load imbalance [60], probability of load balancing success [76], mean response ratio [88], and mean

miss rate [32]. Optimal load distribution in a heterogeneous distributed computer system with both generic and dedicated applications was studied in [8, 78, 61].

Efrosinin et al. [17] analyzed a multi-server heterogeneous exponential queue. They demonstrate the methods for the calculation of the steady-state probabilities and deriving the waiting and sojourn time (response time) distributions. Some performance characteristics of such a system under the optimal control policy are calculated and compared with the same characteristics for the model under other heuristic control policies, e.g., the usage of the Fastest Free Server (FFS) or Random Server Selection (RSS). However, the analytical model is a multidimensional Markov process w.r.t. number of servers. If the number of servers gets larger, which is the case in cloud computing area, the analytical model will be more complex correspondingly.

The problem of optimal load distribution of generic tasks on multiple heterogeneous servers preloaded with special tasks in a cloud computing environment is proposed in [62]. The problem is formulated as a multi-variable optimization problem based on a queuing model. The authors develop algorithms to find the numerical solution of an optimal load distribution and the minimum average response time of generic tasks.

1.8 Organization of the Thesis

We started off with developing monolithic analytical models for performance evaluation of cloud computing centers. Performance models presented in Chapter 2 are incorporating the main concepts of nowadays cloud computing centers: having Poisson arrival of task requests, generally distributed service time and large number of servers. Then, we extend the performance model to incorporate the user requests with hyper-exponentially distributed

task service time. Later on the assumption of finite capacity, which makes the model closer to real cloud centers, is taking into account. Incorporating batch task arrivals is the next improvement of the performance model (Chapter 3). More specifically, we assume generally distributed batch size, Poisson arrival process, generally distributed service time, large number of servers and a finite capacity of input buffer for task requests. In Chapter 4, we extend the analytical model for performance evaluation of highly virtualized cloud computing centers. Each server is permitted to run up to 200 VMs, just like state-of-the-art servers, and the effects of virtualization on servers performance based on the real benchmark from cloud providers is considered.

However, a monolithic model may suffer from intractability and poor scalability due to large number of parameters. Therefore, we develop and evaluate tractable functional performance sub-models to address the mentioned restrictions of monolithic models (Chapters 5 and 6). We construct separate sub-models for different servicing steps in a complex cloud center and the overall solution obtain by iteration over individual sub-model solutions. We also extend the proposed interacting analytical sub-models to capture other important aspects including pool management, power consumption, resource assigning process and virtual machine deployment of nowadays cloud centers. Finally, a performance model suitable for cloud computing centers with heterogeneous requests and resources using interacting stochastic models is proposed and evaluated.

In the last part of this thesis, Chapter 7, the future research is planned. In a nutshell, future research activities will embrace two parts: firstly, we plan to extend the heterogeneity from different angle in the performance model, namely VMs, PMs and user requests. Secondly, we plan to propose and evaluate a cloud-specific task assignment policy that

improves both performance and energy consumption of cloud centers.

The main contribution of this thesis is developing detailed performance models for cloud computing centers. More specifically, we incorporate generally distributed service time, large number of servers, batch arrivals and a highly virtualized infrastructure as the new features to our proposed monolithic performance models. Next, we break down the monolithic performance models into sub-models to maintain tractability while extend the granularity of models. Using interacting performance sub-models, for the first time, we propose and evaluate a pool management schema as well as a heterogeneous performance model suitable for large IaaS clouds.

Chapter 2

Performance Modeling: Single Task Arrivals

In this Chapter, we present our work on performance evaluation of cloud computing centers under single task arrivals. We investigate the performance of a Cloud center considering all important parameters; we model a cloud server farms analytically and also simulate with the object-oriented Petri net-based discrete event simulation engine Artifex by RSoftDesign, Inc. [79]. We begin with the evaluation of the performance of a typical cloud center with large number of servers, generally distributed service time and variable offered load [40].

We assume that any request goes through a *facility node* and then leaves the cloud center. A facility node may contain different computing resources such as web servers, database servers, and others, as shown in Fig. 2.1. We consider the time that a request spends in one of those facility node as the response time; response time does not follow any specific distribution. This model is flexible in terms of cloud center size and service

time of customer requests;

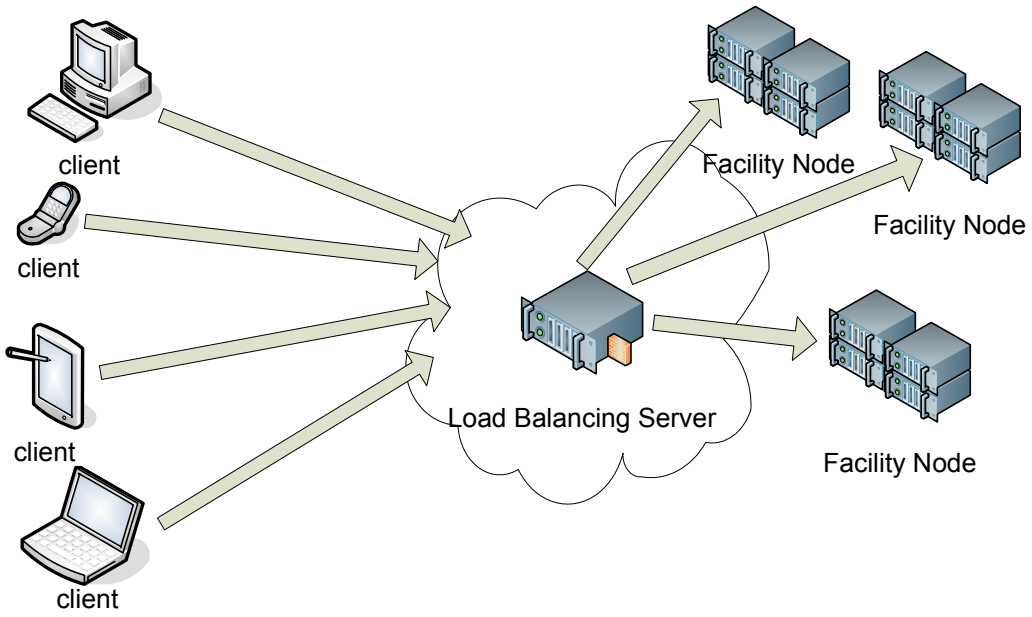


Figure 2.1: Cloud clients and service provider

We developed the analytical model based on queuing theory. We adopted $M/G/m$ queuing system as the abstract model for performance evaluation due to the characteristics of cloud computing centers: having Poisson arrival of task requests, generally distributed service time and large number of servers. Since there is no exact solution for $M/G/m$ queuing system known in the literature, we extended our model to incorporate the cloud centers which have hyper-exponentially family (those distributions that have coefficient of variation more than one) distribution of service time [41, 42]. Later on we added an assumptions of finite capacity for cloud center, which made the model closer to a real cloud centers [45].

This Chapter is organized as follows: Section 2.1 is an introduction to the basic char-

acteristics of the model. Section 2.2 introduces analytical model for single arrival with finite/infinite capacity. Section 2.5 concludes the Chapter providing main findings and results during this Chapter. Finally, In Section 2.4, we explain our simulation approach and parameters.

2.1 Introduction

In this Chapter, we model cloud centers by two queuing systems in each of which we will focus on the specific aspects of the cloud centers.

First, we model the cloud environment as an $M/G/m$ queuing system which indicates that inter-arrival times of requests are exponentially distributed, the service time is generally distributed and the number of facility nodes is m. Also, due to the the nature of cloud environment (i.e., it is a service provider with potentially many customers), we pose no restrictions on the number of facility nodes. Here we are mainly focused on the scale (large number of servers) and generally distributed service time for tasks in a cloud center. The relevant performance indicators are obtained by solving the model for various values of variable parameters.

Second, we model the cloud center as an $M/G/m/m+r$ queuing system with the input buffer of finite capacity. Due to the introduction of finite capacity, the system may experience blocking of task requests. In this model we investigate the system under different buffer space with fixed load values; we also examine the performance of general purpose clouds (cloud centers that accept various types of requests) versus specific purpose clouds under the same conditions (i.e., same load and probability distribution of service time).

Therefore, we incorporate necessary assumptions that are required for having a real

performance model of cloud centers:

(i) Random arrival process (Poisson process)

(ii) Single arrival of tasks

(iii) Generally distributed task service time

(iv) Homogeneous vs. heterogeneous type of requests (Hyper/Hypo exponential service time)

(v) Scale (Large number of servers)

These aspects have not been adequately addressed in existing research, as explained in Chapter 1.

2.2 Single Task Arrivals and Infinite Buffer Capacity

In this Section, we explain our model for a cloud center with single task arrivals and infinite capacity of input arrival buffer. We model a cloud center as a $M/G/m$ queuing system which indicates that the inter-arrival time of requests is exponentially distributed, the service times of customers' requests are independent and identically distributed random variables with a general distribution whose service rate is μ; both μ and CoV, the coefficient of variation defined as standard deviation divided by the mean, are finite and less then one [40, 41]. We concentrate on hypo-exponential distributed service time by assuming the CoV less than one. Later we will incorporate hyper-exponential distributions for task service time as well.

2.2.1 Approximate Analytical Model

An $M/G/m$ queuing system may be considered as a non-Markov process which can be analyzed by applying the embedded Markov chain technique. Embedded Markov Chain technique requires selection of Markov points in which the state of the system is observed. We monitor the number of the tasks in the system (both in service and queued) at the moments immediately before the task request arrival. However, such points (task's arrivals) are not Markov points since the distribution of service times are not Markovian. So we are trying to approximately characterize the state of the system at the arrival points. Therefore our analytical model is an approximation of the real system.

If we consider the system at arrival moments (Observation Points) and number these instances 0, 1, 2, ..., then we get a Markov chain [57]. Here, the system under consideration contains m servers, which render service in order of task request arrivals. Task request inter-arrival time A is exponentially distributed with rate to $\frac{1}{\lambda}$. We will denote its Cumulative Distribution Function (CDF) as $A(x) = Prob[A < x]$ and its probability density function (pdf) as $a(x) = \lambda e^{-\lambda x}$. Laplace Stieltjes Transform (LST) of interarrival time is

$$A^*(s) = \int_0^\infty e^{-sx} a(x) dx = \frac{\lambda}{\lambda + s} \tag{2.1}$$

Task service times are identically and independently distributed according to a general distribution B, with a mean service time equal to $\overline{b} = \frac{1}{\mu}$. The CDF of the service time is $B(x) = Prob\,[B < x]$, and its pdf is $b(x)$. The LST of service time is

$$B^*(s) = \int_0^\infty e^{-sx} b(x) dx S$$

.

Residual task service time is the time from a random point in task execution until task

completion. We will denote it as B_+. This time is necessary for our model since it represents time distribution between task arrival z and departure of the task which was in service when task arrival z occurred. It can be shown as well that probability distribution of elapsed service time (between start of the task execution and next arrival of task request B_-) has the same probability distribution [85]. The LST of residual and elapsed task service times can be calculated in [85] as

$$B_+^*(s) = B_-^*(s) = \frac{1 - B^*(s)}{s\bar{b}} \tag{2.2}$$

The offered load may be defined as

$$\rho \triangleq \frac{\lambda}{m\mu} \tag{2.3}$$

For practical reasons, we assume that the system never enters saturation, which means that any request submitted to the center will get access to the required facility node after a finite queuing time. Furthermore, we also assume each task is serviced by a single server (i.e., there are no batch arrivals), and we do not distinguish between installation (setup), actual task execution, and finalization components of the service time.

2.2.2 Approximate Markov Chain

We are looking at the system at the moments of task request arrivals – these points are treated as Markov points. A given Markov chain has a steady-state solution if it is ergodic. Based on conditions for ergodicity [57] and the above-mentioned assumptions, it is easy to prove that our Markov Chain is ergodic (we provide the proof in Section 2.3.10). Then, using the steady-state solution, we can extract the distribution of number of tasks in the system as well as the mean response time.

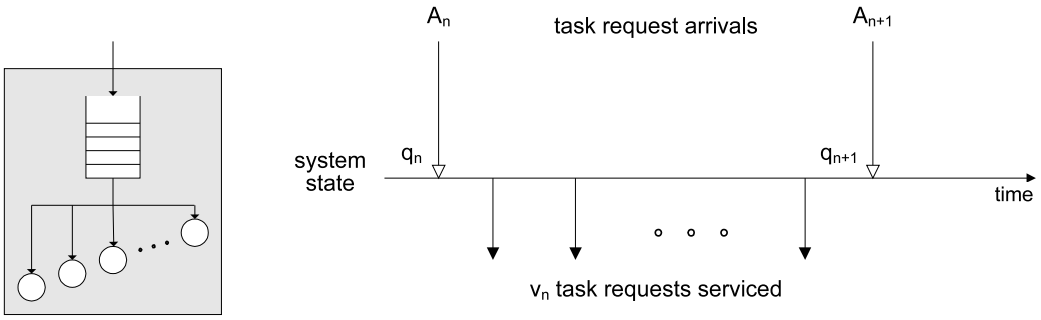


Figure 2.2: Embedded observing points

Let A_n and A_{n+1} denote the moment of n^{th} and $(n+1)^{th}$ arrivals to the system, respectively, while q_n and q_{n+1} denote the number of tasks found in the system immediately before these arrivals; this is schematically shown in Fig. 2.2. If v_{n+1} denotes the number of tasks which are serviced and depart from the system between A_n and A_{n+1}, the following holds:

$$q_{n+1} = q_n - v_{n+1} + 1 \tag{2.4}$$

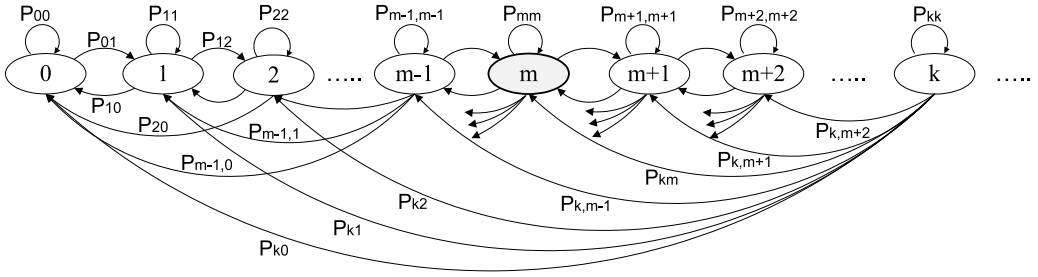


Figure 2.3: State-transition-probability diagram for the $M/G/m$ approximate embedded Markov chain

We need to calculate the transition probabilities associated with this Markov chain, defined as

$$p_{ij} \triangleq Prob\left[q_{n+1} = j | q_n = i\right] \tag{2.5}$$

i.e., the probability that $i + 1 - j$ customers are served during the interval between two successive task request arrivals. Obviously for $j > i + 1$

$$p_{ij} = 0 \tag{2.6}$$

since there are at most $i+1$ tasks present between the arrival of A_n and A_{n+1}. The Markov state-transition-probability diagram as in Fig. 2.3, where states are numbered according to the number of tasks currently in the system (i.e those in service and those awaiting service). For clarity, some transitions are not fully drawn. We have also highlighted the state m because the transition probabilities are different for states on the left and right hand side of this state (i.e., below and above m).

2.2.3 Departure Probabilities

Due to ergodicity of the Markov chain, an equilibrium probability distribution exist for the number of tasks present at the arrival instants; so we define

$$\pi_k = \lim_{n \to +\infty} Prob\left[q_n = k\right] \tag{2.7}$$

As shown by [85], the direct method of solution for this equilibrium distribution requires that we solve the following system of linear equations:

$$\pi = \pi \mathbf{P} \tag{2.8}$$

where $\pi = [\pi_0, \pi_1, \pi_2, \ldots]$, and $\mathbf{P}$ is the matrix whose elements are one-step transition probabilities p_{ij}.

To find the elements of the transition probability matrix, we need to count the number of tasks departing from the system between two successive arrivals. Consider the behaviour

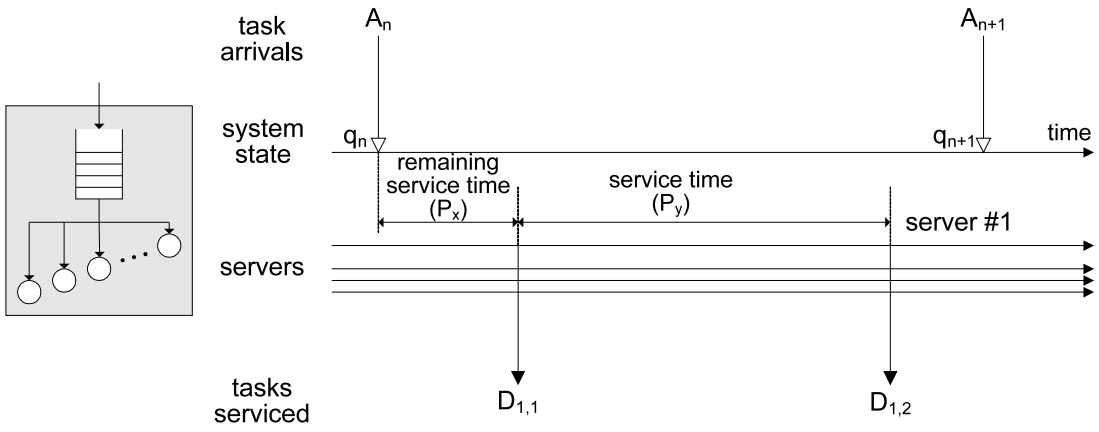


Figure 2.4: System behavior between two arrivals.

of the system, as shown in Fig. 2.4. Each server has zero or more departures during the time between two successive task request arrivals (the inter-arrival time). Let us focus on an arbitrary server, which (without loss of generality) could be the server number 1. For a task to finish and depart from the system during the inter-arrival time, its remaining duration (residual service time defined in (2.2)) must be shorter than the task inter-arrival time. This probability will be denoted as P_x, and it can be calculated as

$$\begin{aligned} P_x &= Prob\,[A > B_+] = \int_{x=0}^{\infty} P\{A > B_+ | B_+ = x\,\} P\{B_+ = x\} \\ &= \int_{x=0}^{\infty} P\{A > B_+ | B_+ = x\,\} dB_+(x) = \int_{x=0}^{\infty} \left(\int_{y=x}^{\infty} \lambda e^{-\lambda y} dy \right) dB_+(x) \\ &= \int_{x=0}^{\infty} \left| -e^{-\lambda y} \right|_{y=x}^{\infty} dB_+(x) = \int_{0}^{\infty} e^{-\lambda x} dB_+(x) = B_+^*(\lambda) \end{aligned} \tag{2.9}$$

Physically this result presents probability of no task arrivals during residual task service time. In the case when arriving task can be accommodated immediately by an idle server (and therefore queue length is zero) we have to evaluate the probability that such task will

depart before next task arrival. We will denote this probability as P_y and calculate it as:

$$\begin{aligned} P_y &= Prob\,[A > B] = \int_{x=0}^{\infty} P\{A > B | B = x\}P\{B = x\} \\ &= \int_{x=0}^{\infty} P\{A > B | B = x\}dB(x) = \int_{x=0}^{\infty} \left(\int_{y=x}^{\infty} \lambda e^{-\lambda y} dy \right) dB(x) \\ &= \int_{x=0}^{\infty} \left| -e^{-\lambda y} \right|_{y=0}^{\infty} dB(x) = \int_{0}^{\infty} e^{-\lambda x} dB(x) = B^*(\lambda) \end{aligned} \tag{2.10}$$

However, if queue is non-empty upon task arrival, the following situation may happen. If between two successive new task arrivals a completed task departs from a server, that server will take a new task from the non-empty queue. That task may be completed as well before the next task arrival and if the queue is still non-empty new task may be executed, and so on until either queue gets empty or new task arrives. Therefore probability of $k > 0$ job departures from a single server, given that there are enough jobs in the queue, can be derived from expressions (2.9) and (2.10) as:

$$P_{z,k} = B_+^*(\lambda)(B^*(\lambda))^{k-1} \tag{2.11}$$

Note that $P_{z,1} = P_x$. Using these values we are able to compute the transition probabilities matrix.

2.2.4 Transition Matrix

Based on our Markov chain, we may identify four different regions of operation for which different conditions hold; these regions are schematically shown in Fig. 2.5, where the numbers on horizontal and vertical axes correspond to the number of tasks in the system immediately before a task request arrival (i) and immediately upon the next task request arrival (j), respectively.

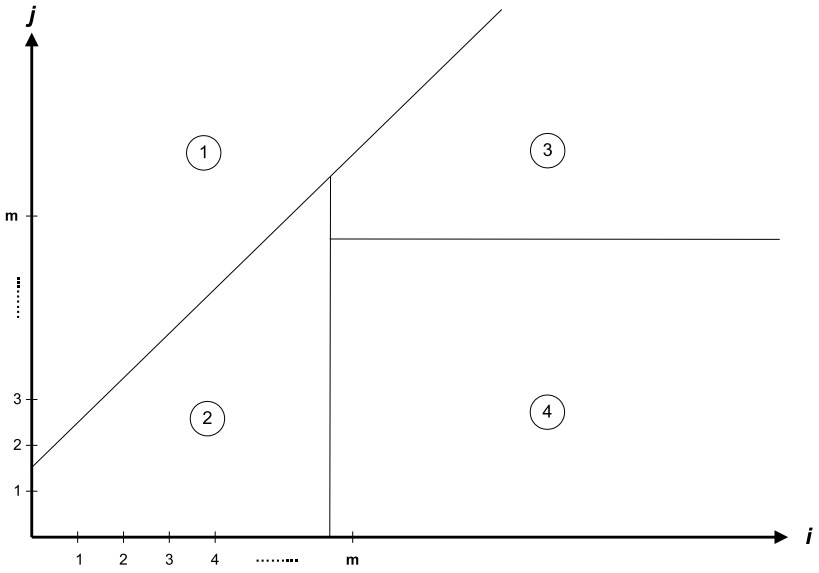


Figure 2.5: Range of validity for p_{ij} equations

- Regarding the region labeled 1, we already know from Eq. 2.6 that $p_{ij} = 0$ for $i + 1 < j$.

- In region 2, no tasks are waiting in the queue, hence $i < m$ and $j \leq m$. Between the two successive request arrivals, $i + 1 - j$ tasks will complete their service. For all transitions located on the left side of state m in Fig. 2.3, the probability of having $i + 1 - j$ departures is

$$p_{ij} = \binom{i}{i-j} P_x^{i-j}(1-P_x)^j P_y + \binom{i}{i+1-j} P_x^{i+1-j}(1-P_x)^{j-1}(1-P_y) \tag{2.12}$$

- Region 3 corresponds to the case where all servers are busy throughout the inter-arrival time, i.e., $i, j \geq m$. In this case all transitions remain to the right of state m

in Fig. 2.3, and state transition probabilities can be calculated as

$$p_{ij} = \sum_{s=\phi}^{\sigma} \binom{m}{s} P_x^s (1-P_x)^{m-s} P_{z,2}^{i+1-j-s} (1-P_{z,2})^s \tag{2.13}$$

In the last expression, the summation bounds are

$$\sigma = min\left[i+1-j, m\right]$$

$$\phi = min\left[i+1-j, 1\right]$$

- Finally, region 4, in which $i \geq m$ and $j \leq m$, describes the situation where the first arrival (A_n) finds all servers busy and a total of $i-m$ tasks waiting in the queue, which it joins; while at the time of the next arrival (A_{n+1}) there are exactly j tasks in the system, all of which are in service. The transition probabilities for this region are

$$p_{ij} = \sum_{s=1}^{\sigma} \binom{m}{s} P_x^s (1-P_x)^{m-s} \binom{\eta}{\alpha} P_{z,2}^{\psi} (1-P_{z,2})^{\zeta} \beta \tag{2.14}$$

where we used the following notation:

$$\begin{aligned}
\sigma &= \; min\left[m, i+1-j\right] \\
\eta &= \; min\left[s, i+1-m\right] \\
\alpha &= \; min\left[s, i+1-j-s\right] \\
\psi &= \; max\left[0, i+1-j-s\right] \\
\zeta &= \; max\left[0, j-m+s\right] \\
\beta &= \begin{cases} 1 & \text{if } \psi \leq i+1-m \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{2.15}$$

2.2.5 Numerical Validation

The steady-state balance equations outlined above can't be solved in closed form, hence we must resort to a numerical solution. To obtain the steady-state probabilities

$\pi = [\pi_0, \pi_1, \pi_2, ...]$, as well as the mean number of tasks in the system (in service and in the queue) and the mean response time, we have used the probability generating functions (PGFs) for the number of tasks in the system:

$$P(z) = \sum_{k=0}^{\infty} \pi_z z^k \tag{2.16}$$

and solved the resulting system of equations using Maple 13 from Maplesoft, Inc. [66]. Since the PGF is an infinite series, it must be truncated for numerical solution; we have set the number of equations to twice the number of servers, which allows us to achieve satisfactory accuracy (as will be explained below), plus the necessary balance equation

$$\sum_{i=0}^{i=2m} \pi_i = 1. \tag{2.17}$$

the mean number of tasks in the system is, then, obtained as

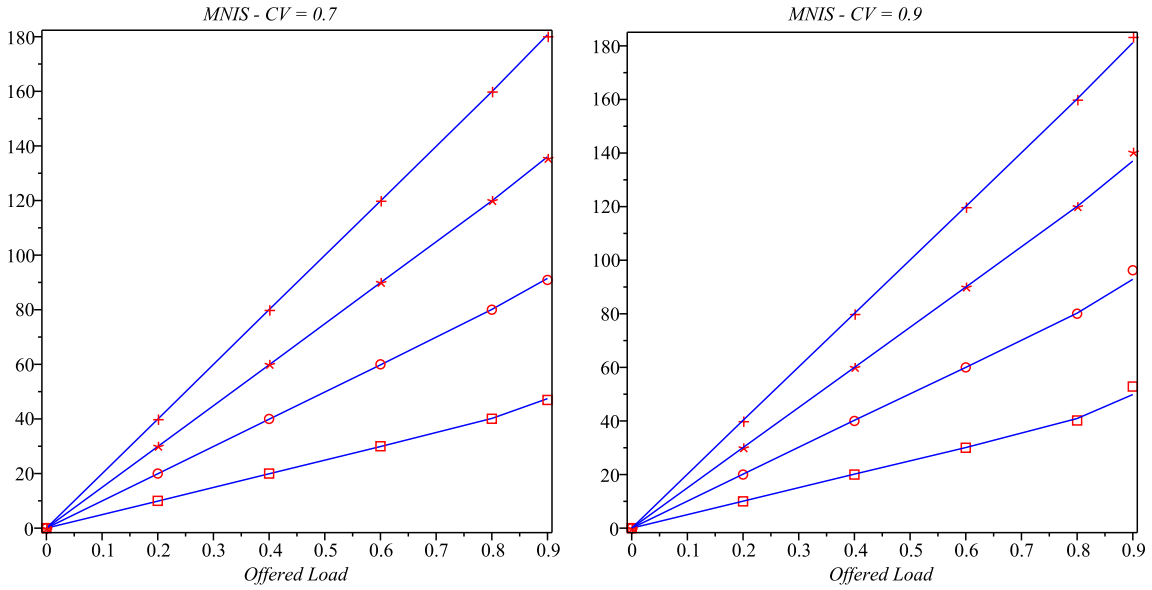
$$E[QS] = P^{'}(1) \tag{2.18}$$

while the mean response time is obtained using Little's law as

$$E[RT] = E[QS]/\lambda \tag{2.19}$$

We have assumed that the task service time follow the Gamma distribution with different values for shape and scale parameters; however, our model may accommodate other distributions without any changes. Then, we have performed two experiments with variable task request arrival rate and coefficient of variation CoV (which can be adjusted in the Gamma distribution independently of the arrival rate).

To validate the analytical solutions we have also built a discrete even simulator of the cloud server farm using object-oriented Petri net-based simulation engine Artifex by RSoft-Design, Inc. [79].

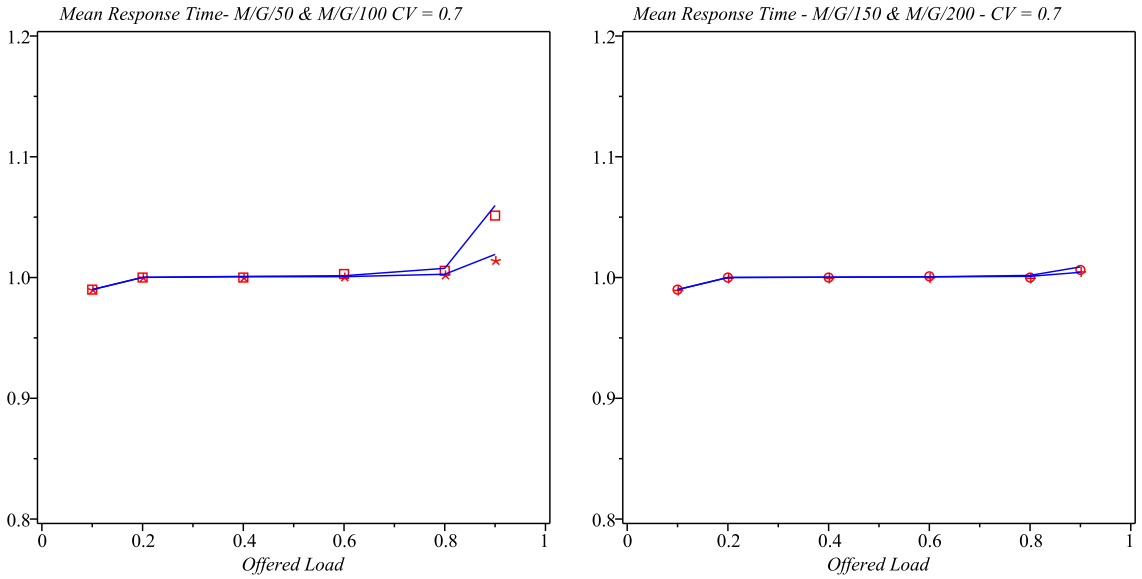


(a) $CoV = 0.7$. (b) $CoV = 0.9$.

Figure 2.6: Mean number of tasks in the system: $m = 50$ (denoted with squares), 100 (circles), 150 (asterisks), and 200 (crosses).

The diagrams in Fig. 2.6 show analytical and simulation results (shown as lines and symbols, respectively) for mean number of tasks in the system as functions of the offered load ρ, under different number of servers. Two different values of the coefficient of variation, $CoV = 0.7$ and 0.9, were used; the corresponding results are shown in Figs. 2.6(a) and 2.6(b). As can be seen, the results obtained by solving the analytical model agree very well with those obtained by simulation.

The diagrams in Fig. 2.7, 2.8 show the mean response time, again for the same range of input variables and for the same values of the coefficient of variation. As above, solid lines correspond to analytical solutions, while different symbols correspond to different number of servers. As could be expected, the response time is fairly steady up to the offered load of around $\rho = 0.8$, when it begins to increase rapidly. However, the agreement between the



(a) Results for 100 servers.

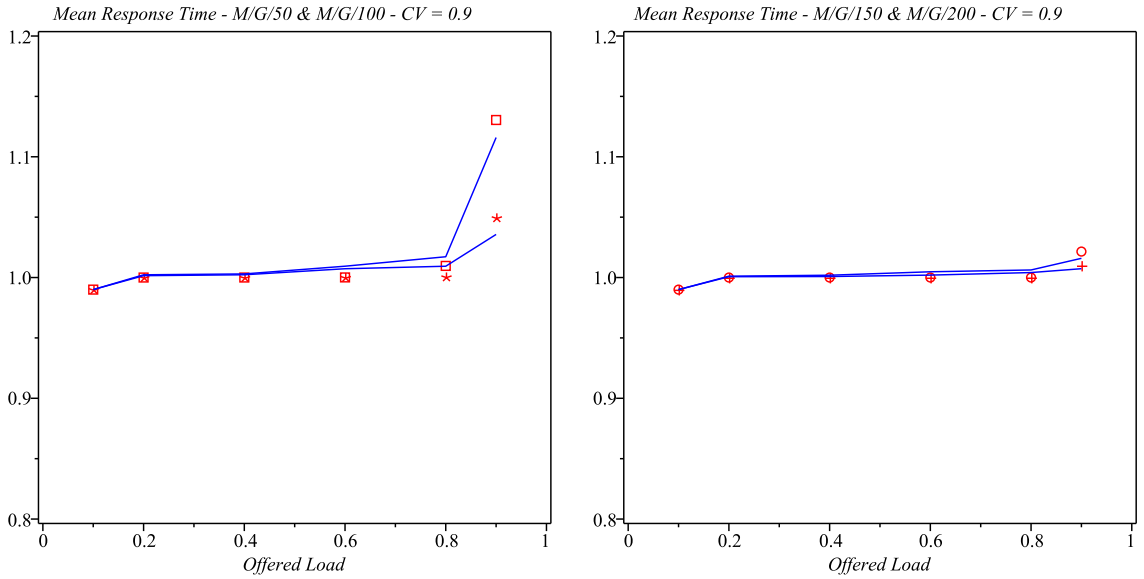
(b) Results for 200 servers.

Figure 2.7: Mean response time $CoV = 0.7$, $m = 50$ (denoted with squares), 100 (asterisks), 150 (circles), and 200 (crosses).

analytical solutions and simulation results is still very good, which confirms the validity of our modeling approach. As can be seen in this work we considered a cloud center with CoV less than one that means the type of task requests are the same (homogeneous). In next Section, we will deal with cloud centers that accept different type of task requests, in which CoV is more than one.

2.2.6 Hyper-exponential Distribution for Service Time

Here we also model a cloud server farm based on the $M/G/m$ queuing system. However in this work [41], we allow the CoV to be more than one; in other words, we consider cloud centers with hyper-exponentially distributed task service time. As we mentioned in Chapter 1, most of the approaches in literature fail to provide accurate results when the



(a) Results for 100 servers.

(b) Results for 200 servers.

Figure 2.8: Mean response time for $CoV = 0.9$, $m = 50$ (denoted with squares), 100 (asterisks), 150 (circles), and 200 (crosses).

CoV was larger than one. In this work we extend our model to accurately model cloud centers that accept various types of requests.

Similar to the case of hypo-exponential distributed task service time, we adopt approximate Embedded Markov chain technique; however we extract the transition probabilities in such a way that can accommodate cloud centers with more dispersed service time distribution (CoV larger than one) as well. The approximate Markov chain is presented in Fig. 2.9.

Departure probabilities are followed the values defined in Section 2.2.3. However transition probabilities in region 3 and 4 are different from those which were defined in Section 2.2.4. As can be seen in Fig. 2.10 the system may have couple of departures between two arrivals with a combination of probabilities (2.9) and (2.10).

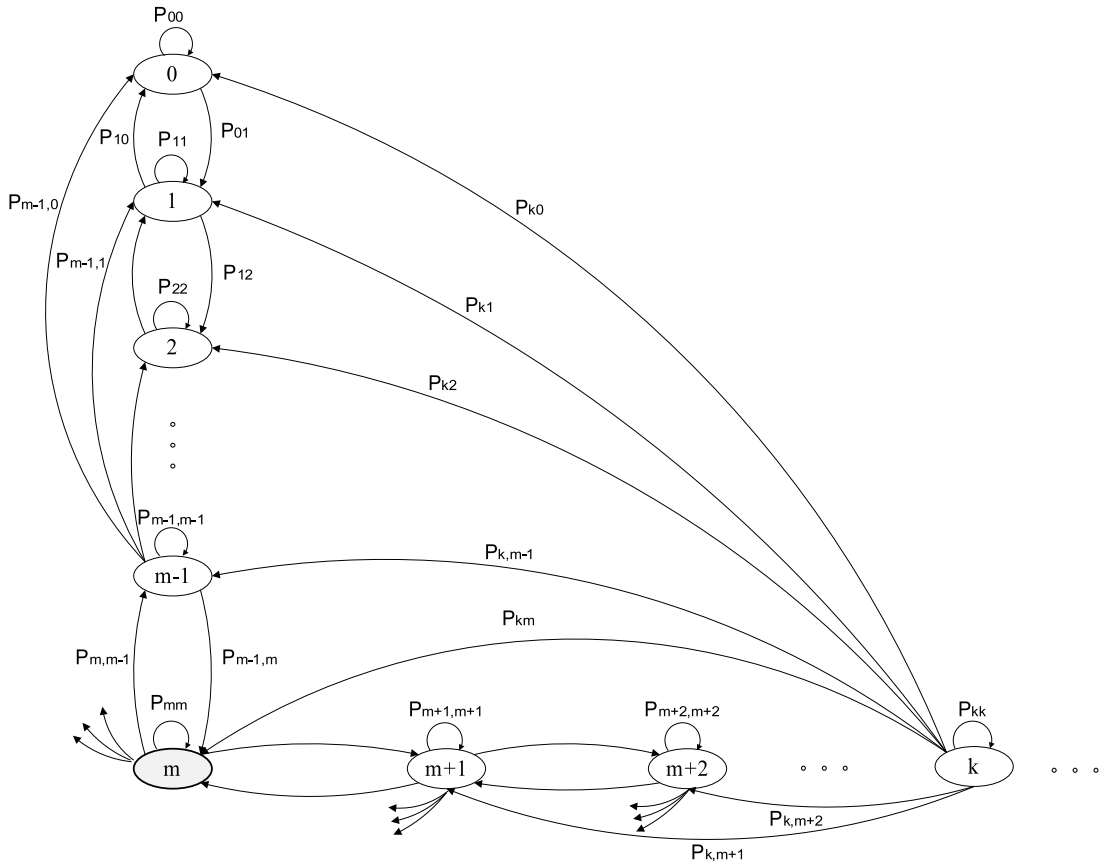


Figure 2.9: State-transition-probability diagram for the system.

we define the transition probabilities in region 3 in Fig. 2.5 as

$$p_{ij} = \sum_{s_1=\min(w,1)}^{\min w,m} \binom{m}{s_1} P_x^{s_1}(1-P_x)^{m-s_1} \cdot \sum_{s_2=\min(w-s_1,1)}^{\min w-s_1,s_1} \binom{s_1}{s_2} P_{z,2}^{s_2}(1-P_{z,2})^{s_1-s_2} \cdot \binom{s_2}{w-s_1-s_2} P_{z,3}^{w-s_1-s_2}(1-P_{z,3})^{s_2} \quad (2.20)$$

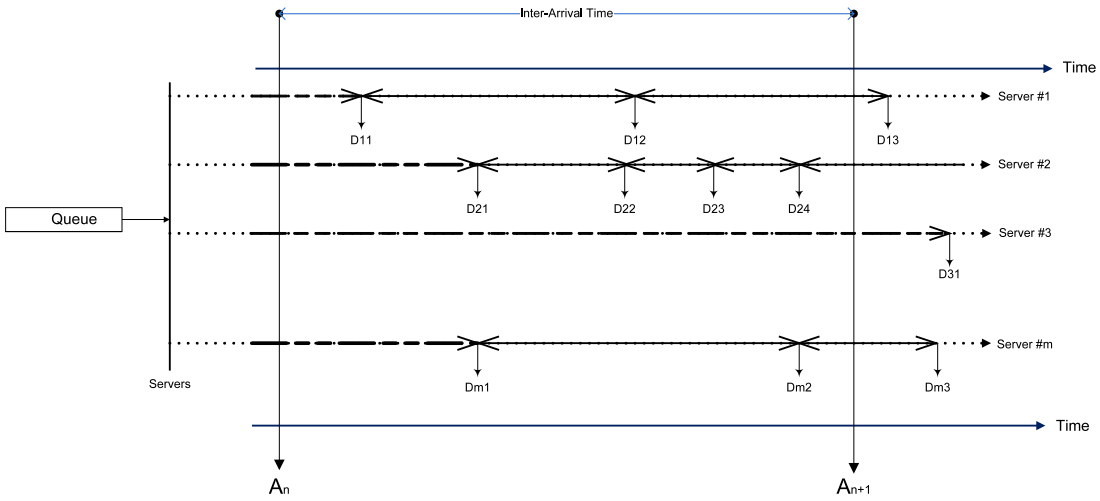


Figure 2.10: System behavior between two arrivals.

and in region 4 as follows:

$$p_{ij} = \sum_{s_1=(m-j)}^{\min w,m} \binom{m}{s_1} P_x^{s_1}(1-P_x)^{m-s_1}\cdot \sum_{s_2=\min(w-s_1,m-j)}^{\min w-s_1,s_1} \binom{s_1}{s_2} P_{z,2}^{s_2}(1-P_{z,2})^{s_1-s_2} \cdot \binom{s_2}{w-s_1-s_2} P_{z,3}^{w-s_1-s_2}(1-P_{z,3})^{s_2} \tag{2.21}$$

We solve equilibrium balance equations as we did in Section 2.2.5.

2.2.7 Distribution of Waiting and Response Time

Let W denote the waiting time in the steady state, and similarly let $W(x)$ and $W^*(s)$ be the CDF of W and its LST, respectively. For the $M/G/m$ systems, Kimura [53] showed that the queue length, Q, has the same distribution as the number of tasks which arrive during the waiting time, W,

$$Q(z) = W^*(\lambda(1-z)) \tag{2.22}$$

The left hand side of (2.22) in our system can be calculated as:

$$Q(z) = \sum_{k=0}^{k=m-1} \pi_k + \sum_{k=m}^{k=2m} \pi_k z^{k-m} \tag{2.23}$$

Hence, we have

$$W^*(s) = Q(z)|_{z=1-(s/\lambda)} = Q(1 - s/\lambda) \tag{2.24}$$

Moreover, the LST of response time is

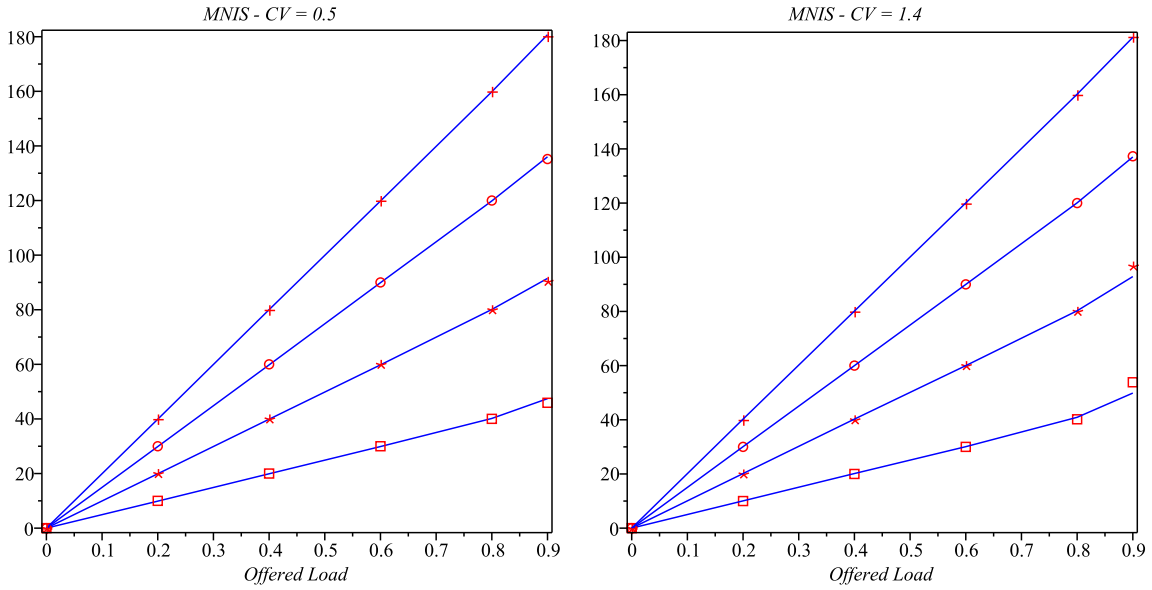
$$T^*(s) = W^*(s)\ B^*(s) \tag{2.25}$$

in which the $B^*(s)$ is the LST of service time. The i th moment $t^{(i)}$ of the response time distribution is given by

$$\begin{aligned} t^{(i)} &= \int_0^\infty x^i dT(x) \ = \ i \int_0^\infty x^{i-1}[1 - T(x)]dx \\ &= (-1)^i T^{*(i)}(0) \qquad i = 2, 3, 4, \ldots \end{aligned} \tag{2.26}$$

2.2.8 Experimental Results

Since we assumed generally distributed service time, our model can accommodate any probability distributions. We assumed Gamma distribution for the service time of tasks because the CoV of this distribution can be set independently of mean. We have performed two experiments with variable task request arrival rates and two values of CoV for task's service time. To validate the analytical solutions we have also extended our simulation model accordingly. The diagrams in Fig. 2.11 show analytical and simulation results (shown as symbols and lines, respectively) for mean number of tasks in the system as functions of the offered load ρ, under different number of servers. Two different values of the coefficient of variation, $CoV = 0.5$ and 1.4, were used; the corresponding results



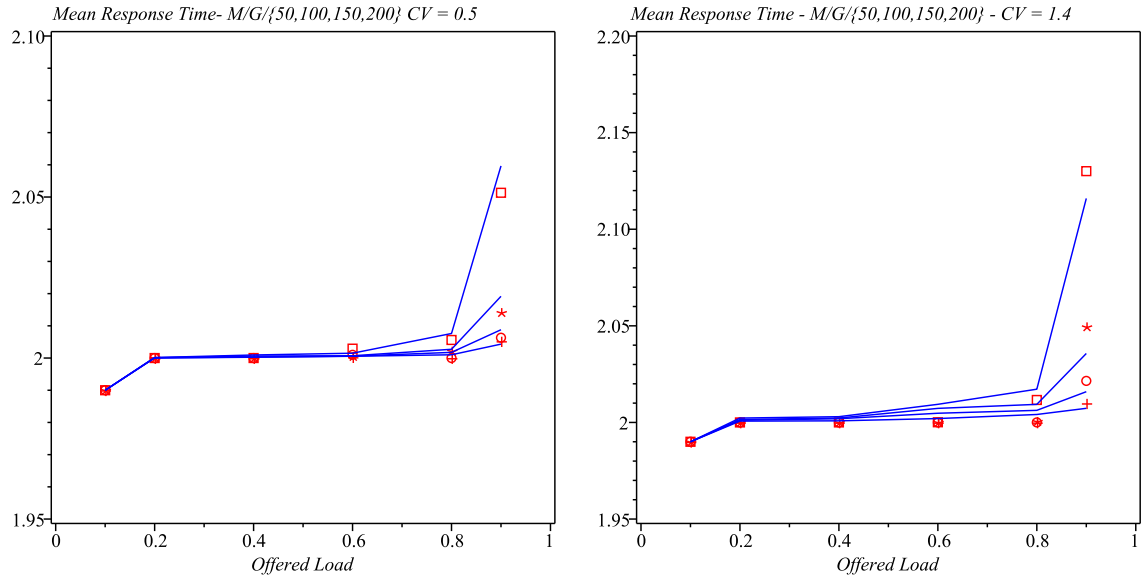
(a) $CoV = 0.5$ (b) $CoV = 1.4$

Figure 2.11: Mean number of tasks in the system: $m = 50$ (denoted with squares), 100 (circles), 150 (asterisks), and 200 (crosses).

are shown in Figs. 2.11(a) and 2.11(b). As can be seen, the results obtained by solving the analytical model agree very well with those obtained by simulation.

The diagrams in Fig. 2.12(a) and Fig. 2.12(b) show the mean response time, again for the same range of input variables and for the same values of the coefficient of variation. As above, solid lines correspond to the results of simulation, while different symbols correspond to different number of servers for analytical model results. As could be expected, the response time is fairly steady up to the offered load of around $\rho = 0.8$, when it begins to increase rapidly. However, the agreement between the analytical solutions and simulation results is still very good, which confirms the validity of our modeling approach.

In addition, we have calculated the higher moments of response time which can be used for measuring some distribution attributes such as *variance*, *skewness* and *kurtosis*.



(a) Results for $CoV = 0.5$

(b) Results for $CoV = 1.4$

Figure 2.12: Mean response time $m = 50$ (denoted with squares), 100 (asterisks), 150 (circles), and 200 (crosses).

The results of simulation and analytical model for higher moments of response time are presented in Tables 2.1 and 2.2, which reinforce the accuracy of the analytical model.

2.3 Single Arrival and Finite Capacity

We add another realistic assumption to our model which is the finite capacity of input buffer space. In such a system if there was no room in the input buffer for new arrival task, it would be blocked and get lost [45]. In this work, besides usual performance metrics (i.e., mean number of tasks in system, mean response and waiting time) we also obtain blocking probability, probability of immediate service as well as higher moments related performance indicators (i.e., standard deviation, skewness and kurtosis). In this work we obtain

Table 2.1: Moments of response time, CoV = 0.5

M/G/m	Model	2th Moment	3th Moment	4th Moment
M/G/50	Simulation	5.00278	15.01101	52.53985
	Analytical	5.01102	15.04245	52.67288
M/G/100	Simulation	5.00873	15.03138	52.61369
	Analytical	5.00035	15.00136	52.50549
M/G/150	Simulation	5.01352	15.05314	52.70024
	Analytical	5.00001	15.00006	52.50024
M/G/200	Simulation	5.10002	15.00094	52.80003
	Analytical	5.00000	15.00001	52.50007

Table 2.2: Moments of response time, CoV = 1.4

M/G/m	Model	2th Moment	3th Moment	4th Moment
M/G/50	Simulation	12.26235	123.23339	1735.6139
	Analytical	12.05379	120.47252	1680.25010
M/G/100	Simulation	12.00365	120.02421	1680.00451
	Analytical	12.00116	120.01035	1680.13771
M/G/150	Simulation	12.00093	120.00723	1680.0008
	Analytical	12.00004	120.00037	1680.00502
M/G/200	Simulation	12.00011	120.00051	1680.00001
	Analytical	11.99999	119.99995	1679.99944

the probability distribution of response and waiting time that provide in depth insights into behaviour of cloud centers under different parameter settings.

2.3.1 Approximate Analytical Model

We model an cloud server farm as a $M/G/m/m+r$ queuing system which indicates that the inter-arrival time of requests is exponentially distributed, the task service times are independent and identically distributed random variables with a general distribution whose service rate is μ. The system under consideration contains m servers, which render service in order of task request arrivals (FCFS). The capacity of system is $m+r$ which means the buffer size for incoming request is equal to r. We assume arrival process is Markovian because first the population size of a typical cloud center is relatively high, second the probability by which every single user request for cloud services is relatively small and finally users' requests are independent of each other; theoretically, if the user population size of a queuing system is relatively high and the probability by which users arrive at system is relatively small and independent, the arrival process can be adequately modeled as a Markovian process. For detailed discussion see [30].

An $M/G/m/m+r$ queuing system may be considered as a non-Markovian process which can be approximately analyzed by applying the embedded Markov chain technique. Embedded Markov Chain technique requires selection of Markov points in which the state of the system is observed. Therefore, we model the number of the tasks in the system (both in service and queued) at the moments immediately before task request arrivals. However, as we mentioned before the arrival moments are not Markovian since the service time of tasks are generally distributed; using embedded Markov chain technique, we approximately characterize the state of system at such instances. If we enumerate these instances as 0, 1, 2, $\ldots$, $m+r$, we obtain a homogeneous Markov chain [57].

Task request arrivals follow a Poisson process so task request inter-arrival time A is

exponentially distributed with rate of $\frac{1}{\lambda}$. We denote its CDF as $A(x) = Prob[A < x]$ and its pdf as $a(x) = \lambda e^{-\lambda x}$. LST of inter-arrival time is

$$A^*(s) = \int_0^\infty e^{-sx} a(x) dx = \frac{\lambda}{\lambda + s}$$

Task service times are identically and independently distributed according to a general distribution B, with a mean service time equal to $\bar{b} = \frac{1}{\mu}$. The CDF of the service time is $B(x) = Prob\,[B < x]$, and its pdf is $b(x)$. The LST of service time is

$$B^*(s) = \int_0^\infty e^{-sx} b(x) dx$$

Residual task service time is the time interval from an arbitrary point (an arrival point in a Poisson process) during a service time to the end of the service time; we denote it as B_+. *Elapsed task service time* is the time interval from the beginning of a service time to an arbitrary point of the service time; we denote it as B_-. It can be shown that probability distribution of residual and elapsed task service times has the same probability distribution and LST of them can be calculated as [85]

$$B_+^*(s) = B_-^*(s) = \frac{1 - B^*(s)}{s\bar{b}} \tag{2.27}$$

The *traffic intensity* may be defined as

$$\rho \triangleq \frac{\lambda}{m\mu} \tag{2.28}$$

where for practical reason we assume that $\rho < 1$. At the moment of task arrival if the system is full, the task would be lost. We assume that each task is serviced by a single server and we do not distinguish between installation (setup), actual task execution, and finalization components of the service time.

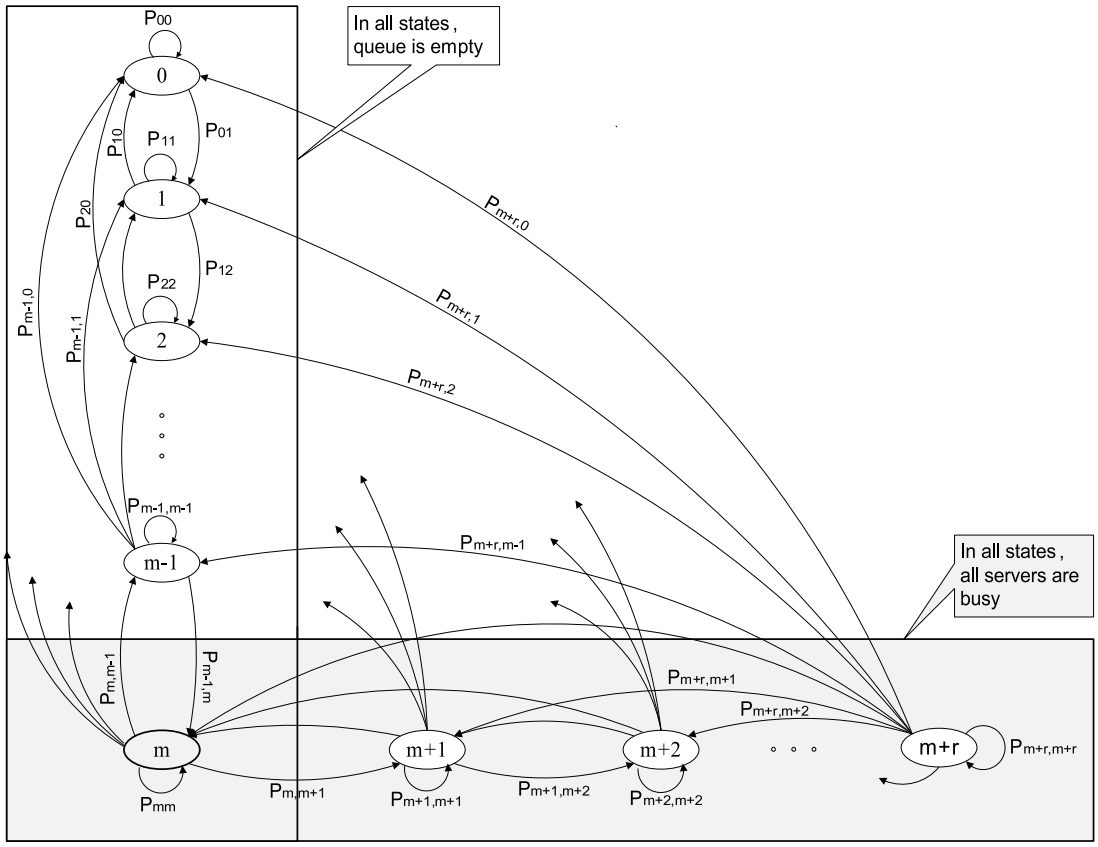


Figure 2.13: Approximate embedded Markov chain

2.3.2 Approximate Embedded Markov Chain

We are looking at the system at the moments of task request arrivals – we treat these points as Markov points. A given homogeneous Markov chain has a steady state solution if it is ergodic, according to the definitions in [85]. Since the system has the finite capacity, the Markov chain is finite (See Fig. 2.13). Departure probabilities are identical to those we obtained in Section 2.2.3. However the transition probability matrix or range of validity is different (Fig. 2.14).

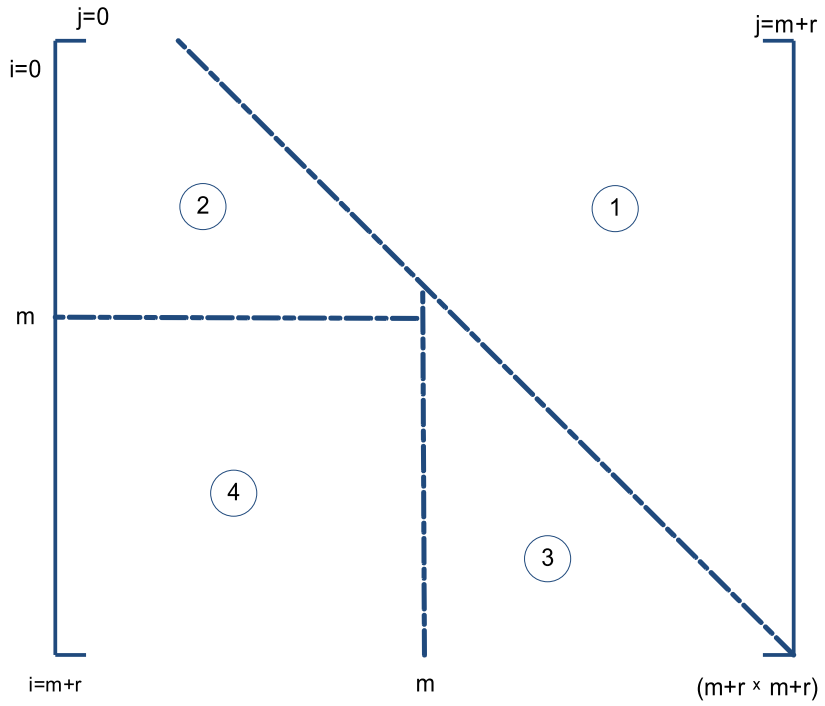


Figure 2.14: One-step transition probability matrix: Range of validity for p_{ij} equations.

2.3.3 Equilibrium Balance Equations

After finding matrix $\mathbf{P}$, we can establish the balance equations. Such balance equations will have a unique steady state solution if corresponding Markov chine is *ergodic*. A given Markov chain is ergodic if it is *irreducible* (all states are reachable from all other states), *aperiodic* (for each states, the probability of returning to that state is positive for all steps) and *recurrent non-null* (each state is revisited at finite time with probability 1) [85]. For convenience in notation, we call our Markov chain *MGM-MChain* afterwards.

Theorem 2.3.1 *MGM-MChain is ergodic.*

Proof See Section 2.3.10.

It is known from [57] that for an ergodic Markov chain, there exist a unique steady state distribution. In order to obtain the steady state distribution we need to solve the balance equations. The balance equations balance the leaving and entering a state in equilibrium. Here, the steady state balance equations can't be solved in closed form, hence we must resort to a numerical solution. The balance equations are:

$$\pi_i \ = \ \sum_{j=0}^{m+r} \pi_j p_{ji} \ , \quad 0 \leq i \leq m+r \tag{2.29}$$

and augmented by the normalization equation

$$\sum_{i=0}^{m+r} \pi_i = 1. \tag{2.30}$$

So far we have $m+r+2$ equations which includes $m+r+1$ linearly independent equations from (2.29) and one normalization equation from (2.30); however we have $m + r + 1$ variables $[\pi_0, \pi_1, \pi_2, \ldots, \pi_{m+r}]$; so in order to obtain the unique equilibrium solution we need to remove one of the equations; the wise choice is the last equation in (2.29) due to minimum information this equation holds about the system compared to the others.

2.3.4 Distribution of Number of Tasks in the System

Once we obtain the steady state probabilities, we are able to establish the PGFs for the number of tasks in the system at the time of a task arrivals:

$$\Pi(z) = \sum_{k=0}^{m+r} \pi_z z^k \tag{2.31}$$

For any Poisson arrivals system, Poisson Arrival See Time Average, PASTA property holds [85]; thus, the PGF $\Pi(z)$ for the distribution of number of tasks in system at an arrival time is identical to the PGF $P(z)$ for the distribution of number of tasks in system

at an arbitrary time:

$$P(z) = \Pi(z) \tag{2.32}$$

Mean number of tasks in the system, then, obtained as

$$\overline{p} = P^{'}(1) \tag{2.33}$$

By Little's law, the mean response time is obtained as

$$\bar{t} = \frac{\overline{p}}{\lambda} \tag{2.34}$$

2.3.5 Distribution of Waiting and Response Time

Let W denote the waiting time in the steady state, and similarly let $W(x)$ and $W^*(s)$ be the CDF, of W and its LST, respectively. For the $M/G/m$ systems, Kimura [53] showed that Q, the queue length, has the same distribution as W; so the number of tasks which arrive during the waiting time is:

$$Q(z) = W^*(\lambda(1-z)) \tag{2.35}$$

The left hand side of (2.35) in our system can be calculated as:

$$Q(z) = \sum_{k=0}^{m-1} \pi_k + \sum_{k=m}^{m+r} \pi_k z^{k-m} \tag{2.36}$$

As we have a finite capacity system (i.e., there may exist blocking), we shall use *effective arrival rate* as:

$$\lambda_e = \lambda(1 - \pi_{m+r})$$

Hence, we have

$$W^*(s) = Q(z)|_{z=1-(s/\lambda_e)} = Q(1 - s/\lambda_e) \tag{2.37}$$

Moreover, the LST of response time is

$$T^*(s) = W^*(s)\ B^*(s) \tag{2.38}$$

in which the $B^*(s)$ is the LST of service time. The i th central moment, $t^{(i)}$, of the response time distribution is given by

$$t^{(i)} = \int_0^\infty x^i dT(x)\ =\ i\int_0^\infty x^{i-1}[1-T(x)]dx = (-1)^i T^{*(i)}(0) \qquad i = 2, 3, 4, \ldots \tag{2.39}$$

Using the first and second moments we can calculate *standard deviation* as [38]:

$$\sigma_T = \sqrt{t^{(2)} - \bar{t}^2} \tag{2.40}$$

skewness as:

$$sk = \frac{t^{(3)} - 3\bar{t}\sigma_T^2 - \bar{t}^3}{\sigma_T^3} \tag{2.41}$$

and *kurtosis* as:

$$ku = \frac{t^{(4)} - 4t^{(3)}\bar{t} - 6t^{(2)}\bar{t}^2 - 3\bar{t}^4}{\sigma_T^4} - 3 \tag{2.42}$$

2.3.6 Probability of Immediate Service

Here we are interested in the probability with that tasks will get into service immediately upon arrival, without any queuing:

$$P_{nq}\ =\ \sum_{i=0}^{m-1} \pi_i \tag{2.43}$$

For such tasks, the response time would be equal to the service time.

2.3.7 Blocking Probability and Buffer Size

Since arrivals are independent of buffer state and the distribution of number of tasks in the system was obtained, we are able to directly calculate the blocking probability of a system with buffer size of r:

$$Pb_r \;=\; \pi_{m+r} \tag{2.44}$$

The appropriate buffer size, r_ϵ, in order to have the blocking probability below the certain value, ϵ, is:

$$r_\epsilon \;=\; min\{r \geq 0 \mid Pb_r \leq \epsilon \;\&\; Pb_{r-1} > \epsilon\} \tag{2.45}$$

2.3.8 Numerical Validation

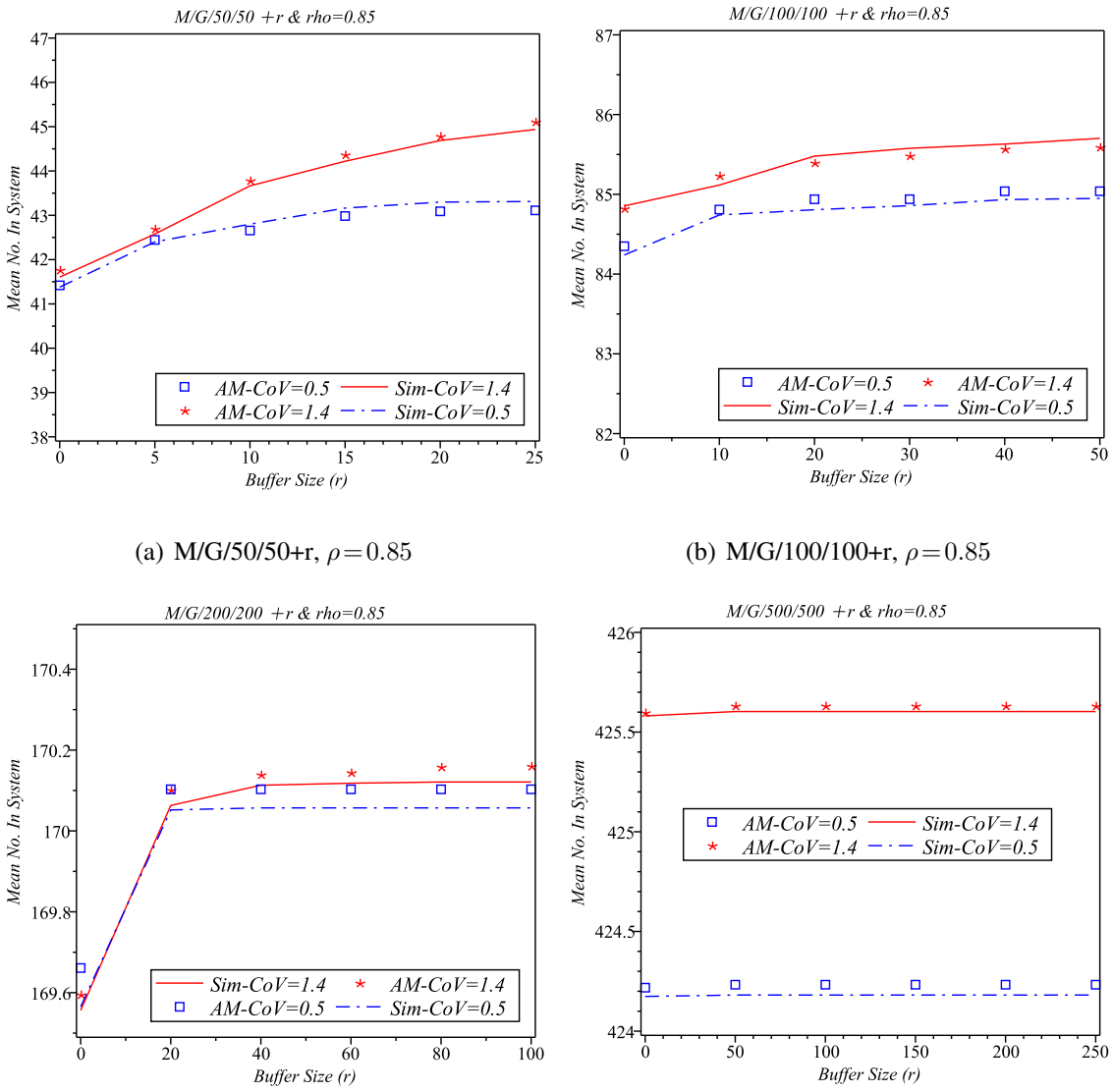
The resulting balance equations of analytical model have been solved using Maple 13 from Maplesoft, Inc. [66]. As can be seen in Eqs. (2.20) and (2.21), in analytical model we modeled *v=3* potential task departures between two consecutive task arrivals. To validate the analytical solution, we have extended our simulation model according to the analytical model.

We perform experiments for four cloud centers under variable capacities:
the number of servers are assumed to be 50, 100, 200 and 500; traffic intensity of $\rho = 0.85$ and $CoV = 0.5,\ 1.4$ are set for experiments. The size of input buffer varies from $r = 0$ to $m/2$ in five steps. First couple of configurations, 50 and 100 server, may not be that realistic compared to today cloud center dimension at first glance, though, they provide a clear picture of system behavior with different input buffer capacity; moreover, at the end of this Section we will discuss the advantages of having some small separated parallel queues (small centers) instead of having one large center with single queue.

There is no precise statistic or empirical results on percentage of different types of instances for a real cloud provider. For instance, Amazon does not publish any information regarding average traffic intensity, buffer space and the percentage of reserved, on-demand or spot instances in their various centers; therefore, we assumed a cloud center which has around 1000 server in which up to 50% percent of them supposed to be in on-demand category. The input buffer size is also assumed to be comparatively small, at most $m/2$, because as can be seen in our discussion, at the end of this Section, providing larger input buffer space has no effect on the system's behavior; usually the system will get the equilibrium state by lower buffer size than $m/2$.

The traffic intensity of cloud center is also assumed to be rather high, $\rho = 0.85$, because we suppose that a cloud provider tries to keep the traffic intensity up as much as possible in order to make the most of prepared installed infrastructure. Lower traffic intensity may be reasonable for large providers such as Google and Microsoft.

First we present mean number of tasks in the system and the results are presented in Fig. 2.15. As can be seen mean number of tasks in the system increases smoothly while the buffer size is getting larger values. Moreover, the larger the number of server, will resulted in the lower the impact of increasing the buffer size. For instance, even with traffic intensity of 0.85%, for a system with 500 server having 50 or infinite buffer rooms is identical.
We also have computed the blocking probability and the probability of immediate service for four experiments (see Fig. 2.16). Results confirm that if the buffer size increased linearly the blocking probability would decrease exponentially. Moreover the appropriate buffer size for having blocking rate less than a certain value can be estimated from the plot. For instance, in case of $M/G/50/50 + r$, having tasks loss rate below two percent,



(a) M/G/50/50+r, $\rho = 0.85$

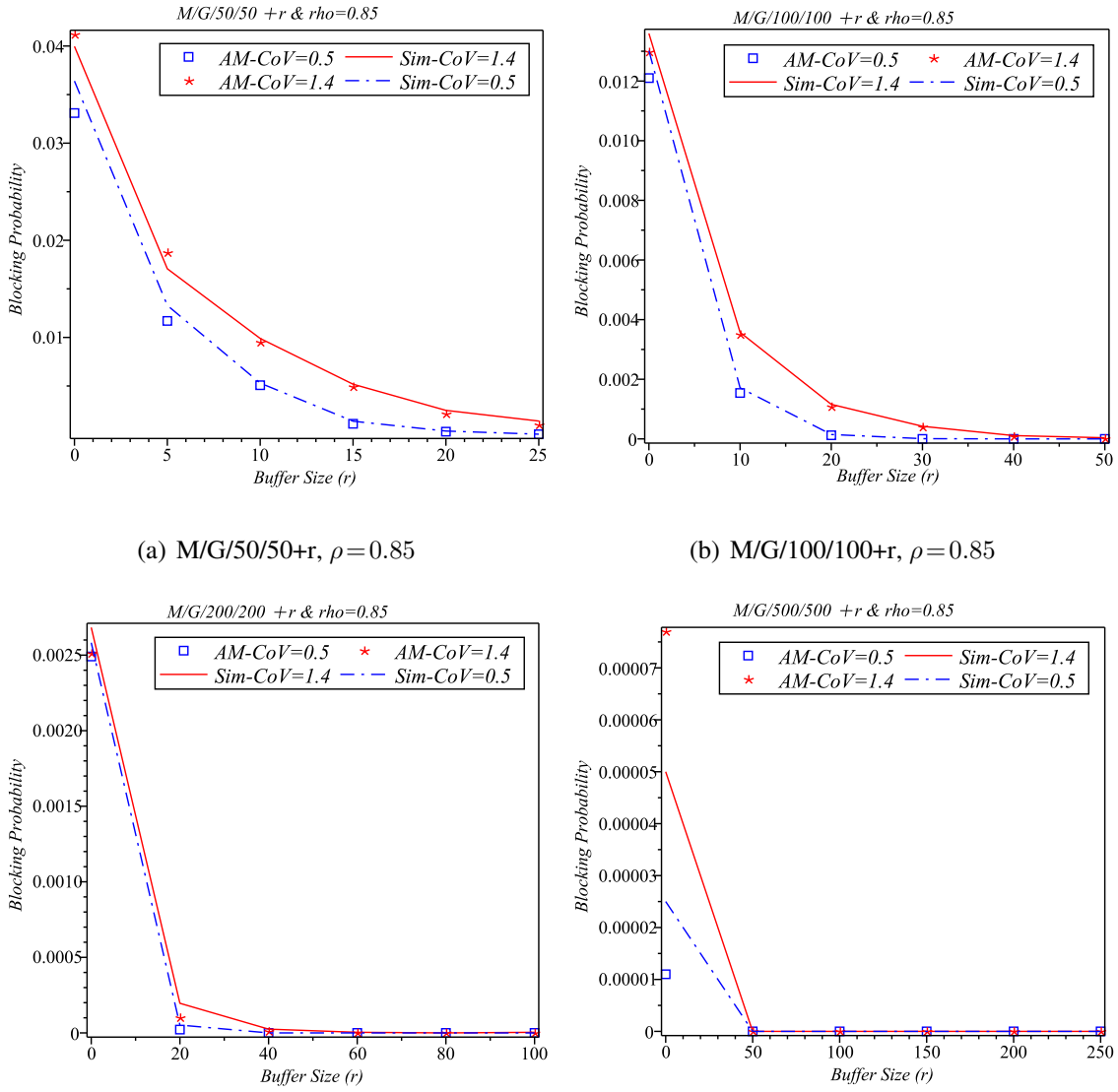
(b) M/G/100/100+r, $\rho = 0.85$

(c) M/G/200/200+r, $\rho = 0.85$

(d) M/G/500/500+r, $\rho = 0.85$

Figure 2.15: Mean number of tasks in the system

is demanding the buffer size of at least five. As can be seen in Fig. 2.16(c) and (d),in case of large system, adding small amount of input buffer space will result in no blocking probability at all.



(a) M/G/50/50+r, $\rho = 0.85$

(b) M/G/100/100+r, $\rho = 0.85$

(c) M/G/200/200+r, $\rho = 0.85$

(d) M/G/500/500+r, $\rho = 0.85$

Figure 2.16: Blocking probability

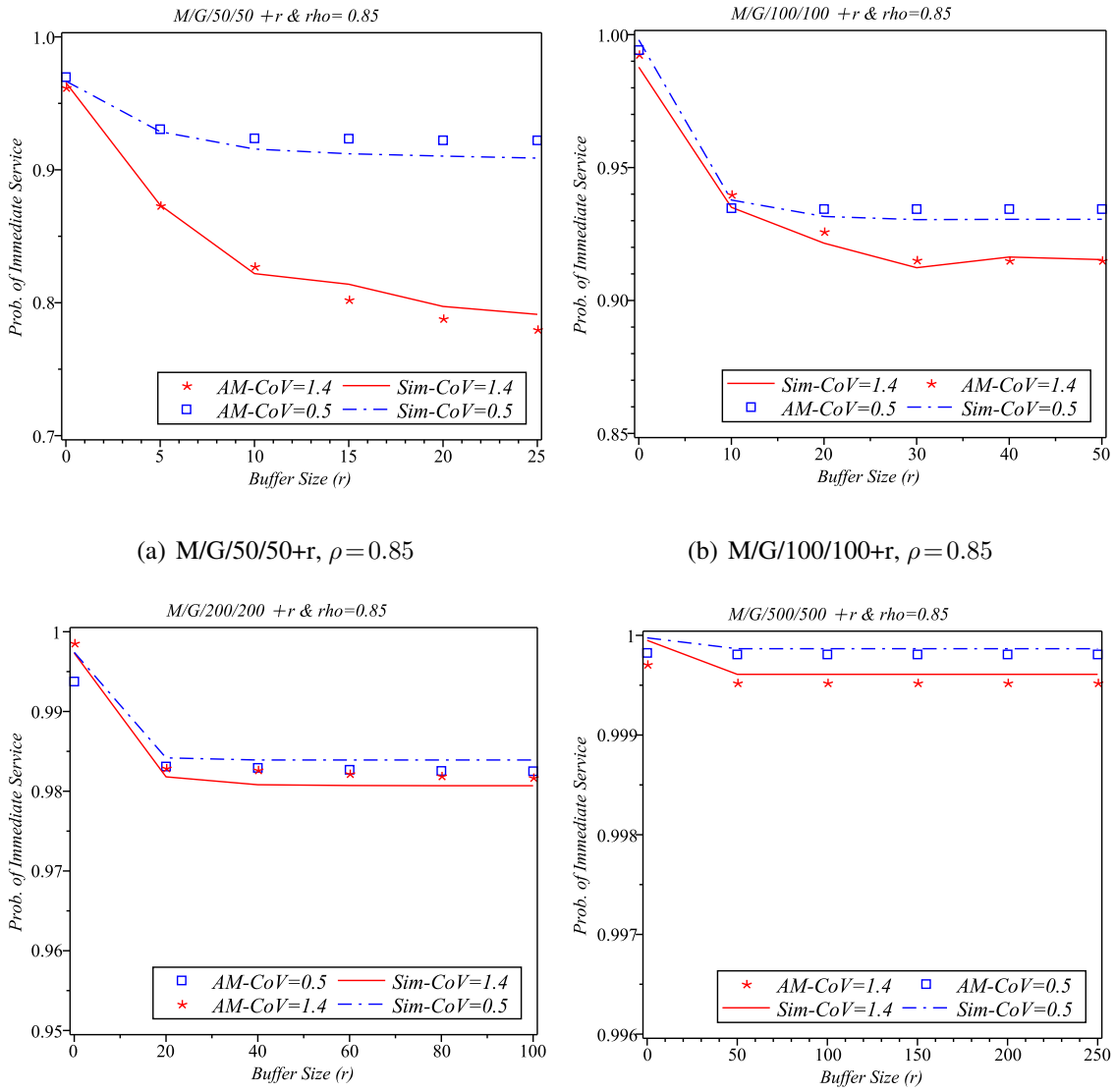
Since the percentage of tasks which can get immediately into the service is an important non-functional service property in SLA, we also demonstrate the probability of immediate service. Intuitively the ratio of tasks which can immediately get into service has no relation

to the capacity of system since getting blocked or serviced is determined by existence of at least one idle server; however as can be seen in Fig. 2.17, particularly for small buffer size the probability of immediate service has sharply decreased. In case of having no buffer space, tasks either would get through service or get blocked at once. On the other hand, input buffer space will result in either immediate entry into service, queuing or blocking. In other words, queuing decreases the probability of blocking as well as the probability of getting immediately into service; however based on *traffic intensity* after adding some buffer space the trend of immediate service probability will not be affected by extra capacity any more.

We have also characterized the probability distribution of response time in Tables 2.3, 2.4, 2.5 and 2.6; here, we only show the results for cloud centers with 50 and 100 servers; because the discussion is aimed to highlight the influences of CoV of service time on response time. The results state that mean response time is slowly increasing as system capacity is growing. Another fact is that the larger the number of servers will result in the shorter the queue time for the same *traffic intensity*; as can be seen beyond a certain value of r adding more buffer space no longer has effect on system's mean response time.

In addition, we have calculated other characteristics of response time distribution; *standard deviation*, *skewness* and *kurtosis* have been obtained in order to have a better understanding of the distribution. Standard deviation in cases of $CoV = 1.4$ is clearly higher than systems with $CoV = 0.5$ which resulted in longer queue time in the system with higher coefficient of variation.

Regarding skewness, having positive skew indicates that the tail on the right hand side is longer than the left hand side and the bulk of the values lie to the left of the mean. In



(a) M/G/50/50+r, $\rho = 0.85$

(b) M/G/100/100+r, $\rho = 0.85$

(c) M/G/200/200+r, $\rho = 0.85$

(d) M/G/500/500+r, $\rho = 0.85$

Figure 2.17: Probability of immediate service

case of $CoV = 1.4$ we have even larger value for skewness which states that the higher the value of CoV of service time distribution will result in the longer the tail of response time distribution. In case of $CoV = 0.5$ the skewness has a relatively low value which

indicates that the distribution of response time is relatively symmetric; however when the $CoV = 1.4$ the distribution tends to be more asymmetric by degree of three. In other words, in cloud centers with diverse services, there may exist some tasks which have a relatively long response time than others; consequently in those general purpose cloud centers, some customers may unexpectedly experience long delay or even get blocked.

The last indicator is kurtosis which indicates whether date set are peaked or flat relative to normal distribution with the same mean and standard deviation. As can be seen in Tables 2.3, 2.4, 2.5 and 2.6, for $CoV = 0.5$ kurtosis is around 1.5 showing that the values of response time are relatively peaked around the mean in comparison to normal distribution with the same mean and standard deviation.

However in case of $CoV = 1.4$ the kurtosis has the value around 12 which means that data set (i.e., response time values) tend to have a distinct peak near the mean, decline rather rapidly, and have heavy tails on the right side of mean. Having heavy tail on the right hand side of response time distribution implies abrupt high traffic in the cloud center which may lead to long delay or even blocking of incoming tasks. Like skewness, the kurtosis values for two experiments show that in heterogeneous cloud centers it is more difficult to maintain SLA compared to homogeneous centers.

Distinguishing between tasks and allocating dedicate buffer space to different classes of tasks can be a way to avoid sudden long delay. In other words, establishing a few parallel distinct homogeneous cloud centers instead of having one central heterogeneous center could be a solution which will decrease the waiting time; this is the another reason for us to provide performance results for cloud centers with 50 and 100 servers. As can be inferred having many stand-by servers may help to maintain the SLA easily, though, providing such

amount of servers, which mostly are idle, is not economic for most cloud provider; however for some mass providers e.g. Google, due to various resources of income and long term business provisioning, it might be reasonable to even provide lots of computing and storage capacity, temporarily for free.

The agreement of simulation and analytical results for mentioned attributes of response time distribution in Tables 2.3, 2.4, 2.5 and 2.6 validates the analytical model and shows that the assumption of having *v=3* (Eqs. (2.20) and (2.21)) departures is sufficient to obtain accurate results.

2.3.9 Ergodicity: Basic Definitions, Lemmas and Theorems

Let $S = [E_0, E_1, E_2, \ldots, E_{m+r}]$ be the set of all states in a Markov chain and $p_{ij}(n)$ indicates the probability of transition from state i to j with n steps. Now we state definitions, lemmas and theorems from [30] which may be used for classifying the states in a Markov chain:

$f_i^{(n)} \triangleq$ P [first return to i occurs in n steps after leaving i]

Then, the probability of ever returning to state i is:

$$f_i = \sum_{n=1}^{\infty} f_i^{(n)} \tag{2.46}$$

Definition *(Recurrence)* State i is called **recurrent** (or persistent) if $f_i = 1$. if $f_i < 1$, i is called **transient**.

Considering states for which $f_i = 1$, we may then define the *mean recurrence time* of i as

Table 2.3: Characterization of response time distribution, $CoV = 0.5$, $\rho = 0.85$

M/G/m/m+r	Model	Mean	Std. Deviation	Skewness	Kurtosis
M/G/50/50	Simulation	1.99902	1.00076	1.00894	1.53956
	Analytical	2.00146	0.99999	1.00000	1.50000
M/G/50/55	Simulation	2.01092	1.00389	1.00344	1.49820
	Analytical	2.00861	1.00048	0.99863	1.49707
M/G/50/60	Simulation	2.02427	1.00612	1.00393	1.51463
	Analytical	2.01155	1.00101	0.99739	1.49401
M/G/50/65	Simulation	2.03227	1.00433	0.99227	1.49429
	Analytical	2.01239	1.00126	0.99691	1.49267
M/G/50/70	Simulation	2.03192	1.00733	1.00042	1.51530
	Analytical	2.01260	1.00134	0.99679	1.49225
M/G/50/75	Simulation	2.03456	1.00856	1.00501	1.53650
	Analytical	2.01264	1.00137	0.99677	1.49215

$$\overline{M_i} \triangleq \sum_{n=1}^{\infty} n f_i^{(n)} \tag{2.47}$$

This is merely the average time to return to i. With this we may then classify states even further.

Definition *(non-null)* The recurrent state i is called $\begin{cases} \textbf{null} & \text{if } \overline{M_i} = \infty \\ \textbf{non-null} \text{ or } (positive) & \text{if } \overline{M_i} < \infty \end{cases}$

There is a simple criterion for nullity in terms of the transition probabilities.

Definition *(Aperiodicity)* The period $d(i)$ of a state i is defined by $d(i) = \gcd\{n : p_{ii}(n) >$

Table 2.4: Characterization of Response Time Distribution, $CoV = 1.4$, $\rho = 0.85$

M/G/m/m+r	Model	Mean	Std. Deviation	Skewness	Kurtosis
M/G/50/50	Simulation	1.99833	2.82871	2.81420	11.87201
	Analytical	1.97940	2.84335	2.79543	11.78114
M/G/50/55	Simulation	2.01377	2.83533	2.81978	11.92889
	Analytical	2.02784	2.81267	2.86457	12.23715
M/G/50/60	Simulation	2.04367	2.84625	2.81567	11.93122
	Analytical	2.06498	2.78835	2.92330	12.61659
M/G/50/65	Simulation	2.05187	2.84596	2.82824	12.03993
	Analytical	2.09241	2.76935	2.97165	12.92445
M/G/50/70	Simulation	2.06683	2.84839	2.80565	11.78003
	Analytical	2.11210	2.75509	3.00941	13.16255
M/G/50/75	Simulation	2.07016	2.84512	2.80438	11.78610
	Analytical	2.12597	2.74472	3.03766	13.33945

$0\}$, the greatest common divisor of the epochs at which return is possible. We call i **periodic** if $d(i) > 1$ and **aperiodic** if $d(i) = 1$.

Definition *(Communicability)*: We say i communicate with j, written $i \to j$, if the chain may ever visit state j with positive probability, starting form i. That is, $i \to j$ if $p_{ij}(n) > 0$ for some $n \geq 0$. We say i and j **intercommunicate** if $i \to j$ and $j \to i$, in which case we write $i \leftrightarrow j$.

It can be seen that $\leftrightarrow$ is an equivalence relation. The state space S can be partitioned into the equivalence classes of $\leftrightarrow$. Within each equivalence class all states are of the same type.

Table 2.5: Characterization of Response Time Distribution, $CoV = 0.5$, $\rho = 0.85$

M/G/m/m+r	Model	Mean	Std. Deviation	Skewness	Kurtosis
M/G/100/100	Simulation	2.00001	0.99982	1.00085	1.50497
	Analytical	2.00013	0.99999	1.00000	1.50000
M/G/100/110	Simulation	2.00607	1.00229	1.00464	1.52855
	Analytical	2.00116	1.00005	0.99985	1.49967
M/G/100/120	Simulation	2.00630	0.99997	0.99494	1.47472
	Analytical	2.00127	1.00007	0.99980	1.49957
M/G/100/130	Simulation	2.00813	1.00149	0.99871	1.50931
	Analytical	2.00128	1.00007	0.99980	1.49956
M/G/100/140	Simulation	2.00811	1.00148	0.99865	1.50930
	Analytical	2.00128	1.00007	0.99980	1.49956
M/G/100/150	Simulation	2.00811	1.00148	0.99865	1.50930
	Analytical	2.00128	1.00007	0.99980	1.49956

Definition A set C of states is called

(a) **closed** if $p_{ij} = 0$ for all $i \in C$, $j \notin C$.

(b) **irreducible** if $i \leftrightarrow j$ for all i, $j \in C$.

Definition *(Ergodicity)* A Markov chain is called **ergodic** if it is irreducible, recurrent non-null, and aperiodic.

Table 2.6: Characterization of Response Time Distribution, $CoV = 1.4$, $\rho = 0.85$

M/G/m/m+r	Model	Mean	Std. Deviation	Skewness	Kurtosis
M/G/100/100	Simulation	2.00017	2.82583	2.82017	11.93192
	Analytical	1.99578	2.83142	2.82171	11.95564
M/G/100/110	Simulation	2.00797	2.83127	2.82095	11.87528
	Analytical	2.00641	2.82405	2.83830	12.06510
M/G/100/120	Simulation	2.01185	2.83181	2.82681	11.91145
	Analytical	2.00924	2.82207	2.84283	12.09488
M/G/100/130	Simulation	2.02090	2.83783	2.81866	11.80995
	Analytical	2.00992	2.82159	2.84393	12.10210
M/G/100/140	Simulation	2.01651	2.83547	2.82114	11.92060
	Analytical	2.01008	2.82147	2.84419	12.10381
M/G/100/150	Simulation	2.01835	2.83600	2.82775	11.99053
	Analytical	2.01011	2.82145	2.84424	12.10413

2.3.10 Proof of Ergodicity

Based on theorems, lemmas and definitions in Section 2.3.9 we show our Markov chain, which we will denote as *MGM-MChain* for brevity, is ergodic.

Lemma 2.3.2 *MGM-MChain is irreducible.*

Proof *MGM-MChain* is a finite Markov chain with state space of S. All we need to show is that S is closed and does not include any proper close subset. Without loss of generality we examine the state $m + r$ in *MGM-MChain* (Fig. 2.13). We are going to establish the

communicability class of the $m+r$. Considering Eqs. (2.20) and (2.21), we have:

$$\forall k \in S \ , \ m+r \rightarrow k$$

since there is a direct communications between $m+r$ and all other states with a non-zero probability. Now we consider $m+r-1$; this state can communicate with $m+r$ with the probability of $p_{m+r-1,m+r} > 0$; so we can write $m+r-1 \rightarrow m+r$. Therefore $m+r-1 \leftrightarrow m+r$ is held. Consequently with the same reasoning we have:

$$\begin{aligned} &\{m+r-2 \leftrightarrow m+r-1\}, \{m+r-3 \leftrightarrow m+r-2\}, \\ &\{m+r-4 \leftrightarrow m+r-3\}, \ \ldots \ , \{0 \leftrightarrow 1\} \end{aligned} \tag{2.48}$$

thus

$$\forall k \in S, m+r \leftrightarrow k \tag{2.49}$$

As a result, we just showed that S is closed and does not include any proper closed subset. So *MGM-MChain* is irreducible.

Lemma 2.3.3 *MGM-MChain is recurrent non-null.*

Proof At first we show that *MGM-MChain* is recurrent and then later we will deal with the nullity. Based on recurrence definition we need to show

$$\forall k \in S, \quad f_k = 1$$

By induction on system capacity, we prove above statement. For the basis of induction, $k=1$, *MGM-MChain*, Fig. 2.18, has the minimum capacity and the following holds:

$$p_{00} + p_{01} = 1 \ \text{ and } \ p_{11} + p_{10} = 1$$

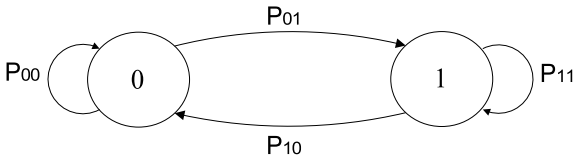


Figure 2.18: MGM-MChain for minimal capacity (k=1)

Now we examine f_1:

$$\begin{aligned} f_1 &= p_{11} + p_{10}\sum_{i=0}^{\infty} p_{00}^i p_{01} = p_{11} + p_{10}\sum_{i=0}^{\infty} p_{00}^i(1-p_{00}) \\ &= p_{11} + p_{10}\sum_{i=0}^{\infty} p_{00}^i - p_{00}^{i+1} = p_{11} + p_{10} \Rightarrow f_1 = 1 \end{aligned} \tag{2.50}$$

With the same reasoning we can show that $f_0 = 1$ as well. Now we assume that for $k = z$ we have

$$f_0 = f_1 = \ldots = f_z = 1$$

Then we need to show that for $k = z + 1$ following is held:

$$f_0 = f_1 = \ldots = f_z = f_{z+1} = 1$$

For this configuration the *MGM-MChain* is exactly the same with Fig. 2.13 but here the last state is $z + 1$. We can merge all the states $\{0, 1, \ldots, z\}$ to one state named z^*; then accordingly *MGM-MChain* would be composed of two states, Fig. 2.19, in which:

$$\begin{cases} p_{z+1,z^*} = \sum_{i=0}^{z} p_{z+1,i} \\ p_{z^*,z^*} = \sum_{i=0}^{z} p_{ii} \end{cases}$$

And

$$\begin{cases} p_{z+1,z+1} + p_{z+1,z^*} = 1 \\ p_{z,z+1} + p_{z^*,z^*} = 1 \end{cases}$$

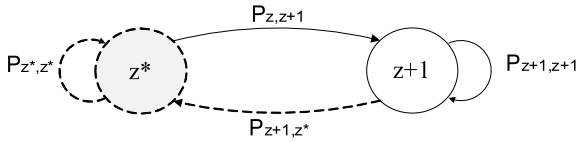


Figure 2.19: Merged MGM-MChain in two states

Like the basis of induction, $k=1$, we can show that:

$f_{z+1}=1$ and $f_{z^*}=1$.

So far, we proved that *MGM-MChain* is recurrent. In order to show that it is non-null we need to have:

$$\overline{M_i} = \sum_{n=1}^{\infty} n f_i^{(n)} < \infty$$

In other words, the mean return time for any states should be a finite quantity.

Lemma 2.3.4 (Ratio Test) *Series* $\sum_{n=0}^{\infty} a_n$ *is absolutely convergence if [4]* $\lim_{n\to\infty} |\frac{a_{n+1}}{a_n}| < 1$

.

Having lemma 2.3.4 in mind, we can proceed:

$$\begin{aligned} &\lim_{n\to\infty} |\frac{(n+1)f_i^{(n+1)}}{(n)f_i^{(n)}}| = \lim_{n\to\infty} |\frac{n+1}{n}| \cdot \lim_{n\to\infty} |\frac{f_i^{(n+1)}}{f_i^{(n)}}| \\ &= 1 \cdot \lim_{n\to\infty} |\frac{f_i^{(n+1)}}{f_i^{(n)}}| = \lim_{n\to\infty} |\frac{f_i^{(n+1)}}{f_i^{(n)}}| < 1 \end{aligned} \tag{2.51}$$

The last step is straight forward since

$$p_{ii}(n+1) < p_{ii}(n)$$

Therefore for each state, i, $\overline{M_i} < \infty$. As a result *MGM-MChain* is both recurrent and non-null.

Lemma 2.3.5 *MGM-MChain is aperiodic.*

Proof All states in *MGM-MChain* have a self loop with a non-zero probability; that is

$$\forall k \in S,\ p_{kk} > 0 \Rightarrow d(k) = 1$$

This means all the states are aperiodic. Consequently *MGM-MChain* is aperiodic.

Theorem 2.3.6 *MGM-MChain is ergodic.*

Proof Based on Lemmas 2.3.2, 2.3.3, 2.3.5 and ergodicity definition.

2.4 Simulation

In order to validate the developed analytical model, we have built a discrete event simulator of the cloud server farm using object-oriented Petri net-based simulation engine Artifex by RSoftDesign, Inc. [79]. Artifex is a development platform for discrete event simulation. It uses a graphical modeling and simulation environment to model, design, simulate, and analyze discrete event systems. Artifex is widely used to simulate optical network protocols, control plane architecture, and switching mechanisms.

2.5 Summary of the Chapter

In this Chapter, we analyzed the performance of cloud computing center under various assumptions, configurations and parameter settings. We developed an analytical model for the basis of our investigation into cloud centers performance evaluation. Moreover we simulated the cloud center to validate our analytical finding. We performed modeling using different queuing systems based on configurations and parameters that we were focused on.

Using Artifex we simulated correspond simulation model and the results were examined against each other.

First, we started off with a cloud center under basic assumptions: the task request arrival process was Markovian; single arrival; service time was assumed to be generally distributed (with CoV less than one); large number of server was assumed; we also supposed that every task will go through a facility node (servers) and then leaves the system (i.e., there is no any feedback among servers). For the first step we consider a cloud center with infinite input buffer capacity for task requests.

Second, we added the assumption of having finite capacity of input buffer as well as service time with CoV more than one. We have obtained the most important performance characteristics: mean number of tasks in system, mean response time, mean waiting time, blocking probability and probability of immediate service; we also extract the probability distribution of response time. By leveraging probability distribution of response time, higher moment related performance indicators such as standard deviation, skewness and kurtosis has also been obtained.

Numerical and simulation results show that the proposed analytical model provides an accurate results for all above-mentioned performance indicators. Our results also indicate that a cloud center that accommodates heterogeneous services may impose longer waiting time for its clients compared to its homogeneous equivalent with the same traffic intensity. As a result, fulfilling the terms of SLA may be satisfied with lower over-provisioning of resources in a cloud centers that accept a certain type of task request as opposed to general purpose cloud center (heterogeneous services). On the other hand, our finding suggested that cloud consumers had better to submit their requests in the cloud centers that deal with

their type of request.

Chapter 3

Performance Modeling: Batch (super-task) Arrivals

In Chapter 2, we have investigated the performance modeling of cloud centers under single arrivals. In this Chapter, we extend our model to accommodate batch arrivals [43]. A preliminary version of this analysis, including the comprehensive characterization of response time, was proposed in [46]. This Chapter is organized as follows: we survey related work for batch arrivals in Section 3.1. Our analytical model for batch arrivals is described in Section 3.2. In Section 3.2 the approximate Markov chain is presented. We present numerical validation in Section 3.4. The summery of the Chapter is outlined in Section 3.6.

3.1 Related Work

To the best of our knowledge, there is no published work in which the performance evaluation of a cloud center has been investigated under batch task arrivals. We employ $M^{[x]}/G/m/m+r$ queuing system in order to deal with the performance evaluation of a cloud center under batch arrivals of tasks. We assume generally distributed batch size, Markovian arrival process, generally distributed service time, large number of servers and a finite capacity of input buffer for task requests. There are only a handful of research works in literature that investigated a queuing system with batch arrival as well as generally distributed service time:

An upper bound for the mean queue length and lower bounds for the delay probabilities (that of an arrival batch and that of an arbitrary task in the arrival batch) was described in [104]. An approximate formula is also developed for the general batch-arrival queue $GI^{[x]}/G/m$. In spite of the simplicity, the approach is accurate enough for small batch sizes, up to 3, as well as small number of servers (less than 10).

Federgruen et al. [18] proposed an approximation method for the computation of the steady-state distribution of the number of tasks in queue as well as the moments of the waiting time distribution. They examined both hypo-exponential and hyper-exponential distribution family for service time, which is necessary for modeling a dynamic system such as cloud farms; however they just performed numerical results for a system with up to seven server and there is no result or indication about the efficiency of the method in case of larger number of servers or a system with finite capacity.

Another approximate formula was proposed in [14]. The authors presented an approximate formula for the steady-state average number of tasks in the $M^{[x]}/G/m$ queuing sys-

tem. The derivation of the formula is based on a heuristic argument whereby a reformulation of the number of tasks in $M^{[x]}/G/1$ is extended to the multi-server queue. From a computational viewpoint, the approach is simple to apply, though, the relative percentage error incurred seem to be unavoidable when the number of server is large, the mean batch size is small or the coefficient of variation of service time is larger than one.

A diffusion approximation for an $M^{[x]}/G/m$ queue was developed in [56]. The authors derived an approximate formula for the steady-state distribution of the number of tasks in the system, delay probability and mean queue length. However, the diffusion approach gives unacceptably large errors when batch size is larger than 3.

Overall, existing methods are not well suited to the analysis of cloud center where the number of servers may potentially be huge, the distribution of service times is unknown, and batch arrival of requests is allowed.

3.2 Approximate Analytical Model

A system with above mentioned description can be modeled as an $M^{[x]}/G/m/m+r$ queuing system. In this system, super-task (i.e., a batch request) arrivals follow a Poisson process so that super-task inter-arrival time A is exponentially distributed with a rate of $\frac{1}{\lambda}$. We denote its CDF with $A(x) = Prob[A < x]$ and its pdf with $a(x) = \lambda e^{-\lambda x}$; the LST of this pdf is

$$A^*(s) = \int_0^\infty e^{-sx} a(x) dx = \frac{\lambda}{\lambda + s}.$$

Let g_k be the probability that the super-task size is equal to k, $k = 1, 2, ..., \textit{MBS}$, in which *MBS* is the maximum batch size which, for obvious reasons, must be finite; both g_k and *MBS* depend on user application. Let $\overline{g}$ and $\Pi_g(z)$ be the mean value and the PGF of

the task burst size, respectively.

$$\begin{aligned} \Pi_g(z) &= \sum_{k=1}^{MBS} g_k z^k \\ \overline{g} &= \Pi_g^{(1)}(1) \end{aligned} \tag{3.1}$$

Tasks within a super-task have service times which are identically and independently distributed according to a general distribution B, with a mean service time of $\overline{b} = \mu^{-1}$. The CDF of the service time is $B(x) = Prob\,[B < x]$ and its pdf is $b(x)$, while the corresponding LST is

$$B^*(s) = \int_0^\infty e^{-sx} b(x) dx.$$

Residual task service time, denoted with B_+, is the time interval from an arbitrary point during a service interval to the end of that interval. *Elapsed task service time*, denoted with B_-, is the time interval from the beginning of a service interval to an arbitrary point within that interval. Residual and elapsed task service times have the same probability distribution [85], the LST of which can be calculated as

$$B_+^*(s) = B_-^*(s) = \frac{1 - B^*(s)}{s\overline{b}} \tag{3.2}$$

The *traffic intensity* may be defined as

$$\rho \triangleq \frac{\lambda \overline{g}}{m\mu} \tag{3.3}$$

For practical reasons, we assume that $\rho < 1$.

3.3 Approximation by Embedded Processes

To solve this model, we adopt a technique similar to embedded Markov chain in [85, 57]. The problem stems from the fact that our $M^{[x]}/G/m/m+r$ system is non-Markovian,

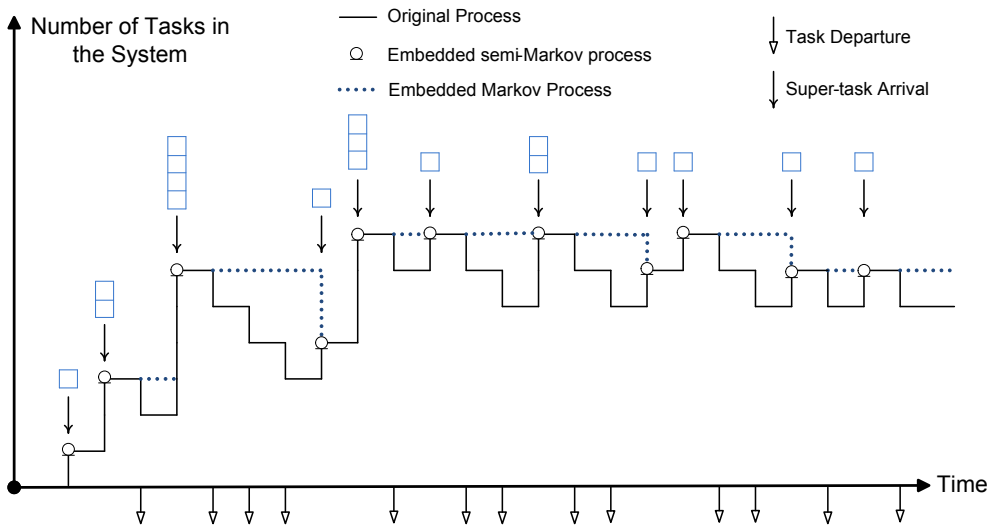


Figure 3.1: A sample path of the original process, embedded semi-Markov process, and embedded Markov process.

so we propose an approximate solution with two steps.

First, we identify a semi-Markov process, hereafter referred to as *ESMP*, embedded in the original process between two arrivals. To determine the behavior of the embedded semi-Markov process, we need to determine the state residence times.

Second, we introduce an approximate embedded Markov process, hereafter referred to as aEMP, that models the original process in a discrete manner. In the aEMP, task departures that occur between two successive super-task arrivals (of which there can be zero, one, or several) are considered to occur at the times of super-task arrivals.

Therefore, the aEMP models the the original process only at the moments of super-task arrivals. Since we can't determine the exact moments of task departures, the counting process is only approximate, but the approximation error is rather low and decreases with an increase in load, as will be discussed below.

A sample path of the three processes: the original one, the embedded semi-Markov, and

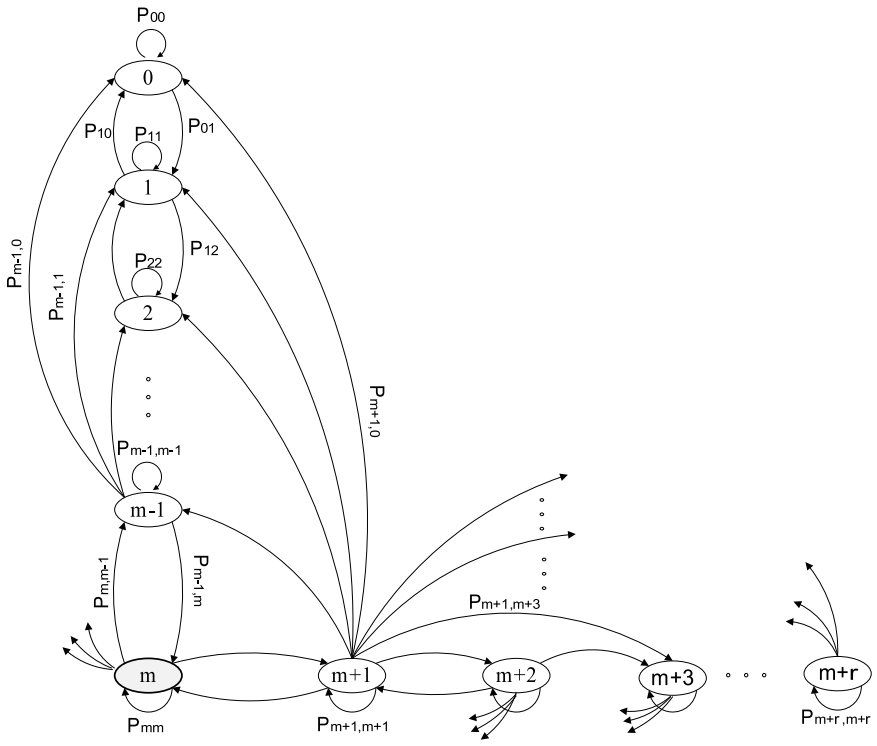


Figure 3.2: The approximate embedded Markov chain (aEMC) associated with the approximate embedded Markov process (aEMP).

the approximate embedded Markov one, is shown schematically in Fig. 3.1.

The approximate embedded Markov chain, hereafter referred to as *aEMC*, that corresponds to aEMP is shown in Fig. 3.2. States are numbered according to the number of tasks currently in the system (which includes both those in service and those awaiting service). For clarity, some transitions are not fully drawn.

Let A_n and A_{n+1} indicate the moment of n^{th} and $(n+1)^{th}$ super-task arrivals to the system, respectively, while q_n and q_{n+1} indicate the number of tasks found in the system immediately before these arrivals; this is schematically shown in Fig. 3.3. If k is the size of super-task and v_{n+1} indicates the number of tasks which depart from the system between

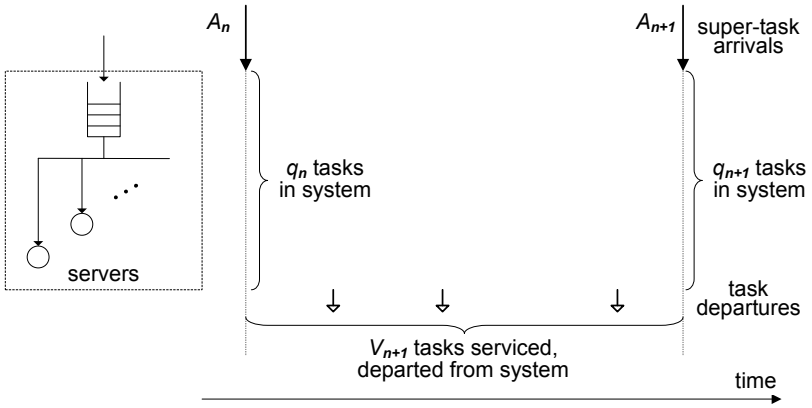


Figure 3.3: Observation points.

A_n and A_{n+1}, then

$$q_{n+1} = q_n - v_{n+1} + k \tag{3.4}$$

To solve the aEMC, we need to calculate the corresponding transition probabilities, defined as

$$P(i,j,k) \triangleq Prob\left[q_{n+1} = j | q_n = i \text{ and } X_g = k\right].$$

In other words, we need to find the probability that $i + k - j$ customers are served during the interval between two successive super-task arrivals. Such counting process requires the exact knowledge of system behavior between two super-task arrivals. Obviously for $j > i + k$,

$$P(i,j,k) = 0 \tag{3.5}$$

since there are at most $i+k$ tasks present between the arrival of A_n and A_{n+1}. For calculating other transition probabilities in the aEMC, we need to identify the distribution function of residence time for each state in ESMP. Let us now describe these residence times in

more detail.

- Case 1: The state residence time for the first departure is remaining service time, $B_+(x)$, since the last arrival is a random point in the service time of the current task.
- Case 2: If there is a second departure from the same physical machine (PM), then clearly the state residence time is the service time ($B(x)$).
- Case 3: No departure between two super-task arrivals as well as the last departure before the next arrival make the state residence time exponentially distributed with the mean value of $\frac{1}{\lambda}$.
- Case 4: If i^{th} departure is from another PM, then the CDF of state residence time is $B_{i+}(x)$. Consider the departure D_{21} in Fig. 3.4, which takes place after departure D_{11}. Therefore the moment of D_{11} could be considered as an arbitrary point in the remaining service time of the task in PM #2, so the CDF of residence time for the second departure is $B_{2+}(x)$. As a result, the LST of $B_{2+}(x)$ is the same as $B^*_+(s)$, similar to (3.2) but with an extra recursion step. Generally, the LSTs of residence times between subsequent departures from different PMs may be recursively defined as follow

$$B^*_{i+}(s) = \frac{1 - B^*_{(i-1)+}(s)}{s \cdot \overline{b}_{(i-1)+}}, \quad i = 1, 2, 3, \cdots \tag{3.6}$$

in which $\overline{b}_{(i-1)+} = [-\frac{d}{ds} B^*_{(i-1)+}(s)]_{s=0}$. To maintain the consistency in notation we also define

$$\overline{b}_{0+} = \overline{b}$$

$$B^*_{0+}(s) = B^*(s)$$

$$B^*_{1+}(s) = B^*_+(s)$$

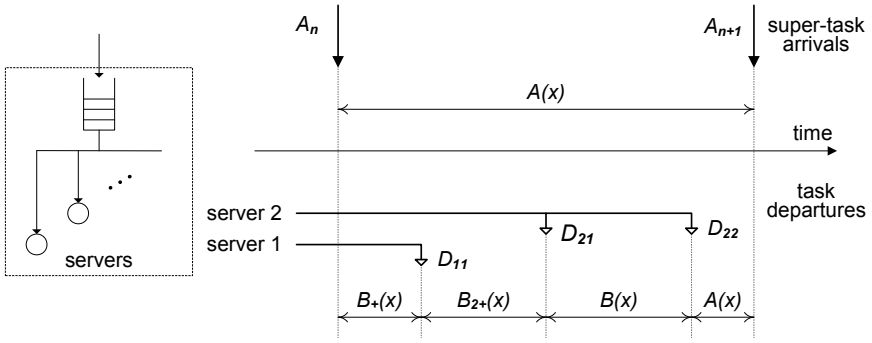


Figure 3.4: System behavior between two observation points.

Let $R_k(x)$ denotes the CDF of residence times at state k in ESMP:

$$R_k(x) = \begin{cases} B_+(x), & \text{Case 1} \\ B(x), & \text{Case 2} \\ A(x), & \text{Case 3} \\ B_{i+}(x), & \text{Case 4} \quad i = 2, 3, \cdots \end{cases} \tag{3.7}$$

3.3.1 Departure Probabilities

To find the elements of the transition probability matrix, we need to count the number of tasks departing from the system in the time interval between two successive super-task arrivals. Therefore at first step, we need to calculate the probability of having k arrivals during the residence time of each state. Let $\mathcal{N}(B_+)$, $\mathcal{N}(B)$ and $\mathcal{N}(B_{i+})$, where $i = 2, 3, \ldots$, indicate the number of arrivals during time periods $B_+(x)$, $B(x)$ and $B_{i+}(x)$, respectively.

Due to Poisson distribution of arrivals, we may define the following probabilities:

$$\begin{aligned}
\alpha_k &\triangleq \text{Prob}[\mathcal{N}(B_+) = k] = \int_0^\infty \frac{(\lambda x)^k}{k!} e^{-\lambda x} dB_+(x) \\
\beta_k &\triangleq \text{Prob}[\mathcal{N}(B) = k] = \int_0^\infty \frac{(\lambda x)^k}{k!} e^{-\lambda x} dB(x) \\
\delta_{ik} &\triangleq \text{Prob}[\mathcal{N}(B_{i+}) = k] = \int_0^\infty \frac{(\lambda x)^k}{k!} e^{-\lambda x} dB_{i+}(x)
\end{aligned} \tag{3.8}$$

We are also interested in the probability of having no arrivals; this will help us calculate the transition probabilities in the aEMC.

$$\begin{aligned}
P_x = \alpha_0 &\triangleq \text{Prob}[\mathcal{N}(B_+) = 0] = \int_0^\infty e^{-\lambda x} dB_+(x) = B_+^*(\lambda) \\
P_y = \beta_0 &\triangleq \text{Prob}[\mathcal{N}(B) = 0] = \int_0^\infty e^{-\lambda x} dB(x) = B^*(\lambda) \\
P_{ix} = \delta_0 &\triangleq \text{Prob}[\mathcal{N}(B_{2+}) = 0] = \int_0^\infty e^{-\lambda x} dB_{i+}(x) = B_{i+}^*(\lambda) \\
P_{xy} &= P_x P_y
\end{aligned} \tag{3.9}$$

Note that $P_{1x} = P_x$. we may also define the probability of having no departure between two super-task arrivals. Let A be an exponential random variable with the parameter of λ, and let B_+ be a random variable which is distributed according to $B_+(x)$ (remaining service time). The probability of having no departures is

$$\begin{aligned}
P_z = &\; Prob\,[A < B_+] \\
= &\int_{x=0}^\infty P\{A < B_+ | B_+ = x\} dB_+(x) \\
= &\int_{x=0}^\infty P\{A < x\} dB_+(x) \\
= &\int_{x=0}^\infty \left(\int_{y=0}^x \lambda e^{-\lambda y} dy \right) dB_+(x) \\
= &\int_{x=0}^\infty \left[1 - e^{-\lambda y}\right]_{y=0}^x dB_+(x) \\
= &\int_0^\infty (1 - e^{-\lambda x}) dB_+(x) \\
= &\; 1 - B_+^*(\lambda) \\
= &\; 1 - P_x
\end{aligned} \tag{3.10}$$

Using probabilities P_x, P_y, P_{xy}, P_{ix} and P_z we may define the transition probabilities for aEMC. Note that due to tractability we use only P_x, P_{2x} and P_{3x} that is the computation of Equ. (3.6) up to 3 recursions. However under moderate high load ($\rho < 0.9$) it is unlikely to have more than 3 departures from a single PM between two arrivals.

3.3.2 Transition Matrix

The transition probabilities may be depicted in the form of a one-step transition matrix, as shown Fig. 3.5 where rows and columns correspond to the number of tasks in the system immediately before a super-task arrival (i) and immediately after the next super-task arrival (j), respectively. As can be seen, four operating regions may be distinguished, depending on whether the input queue is empty or non-empty before and after successive super-task arrivals. Each region has a distinct transition probability equation that depends on current state i, next state j, super-task size k, and number of departures between two super-task arrivals. Note that, in all regions, $P(i, j, k) = P_z$ if $i + k = j$.

Region 1

In this region, the input queue is empty and remains empty until next arrival, hence the transitions originate and terminate on the states labeled m or lower. Let us denote the number of tasks which depart from the system between two super-task arrivals as $w(k) = i + k - j$. For $i, j \leq m$, the transition probability is

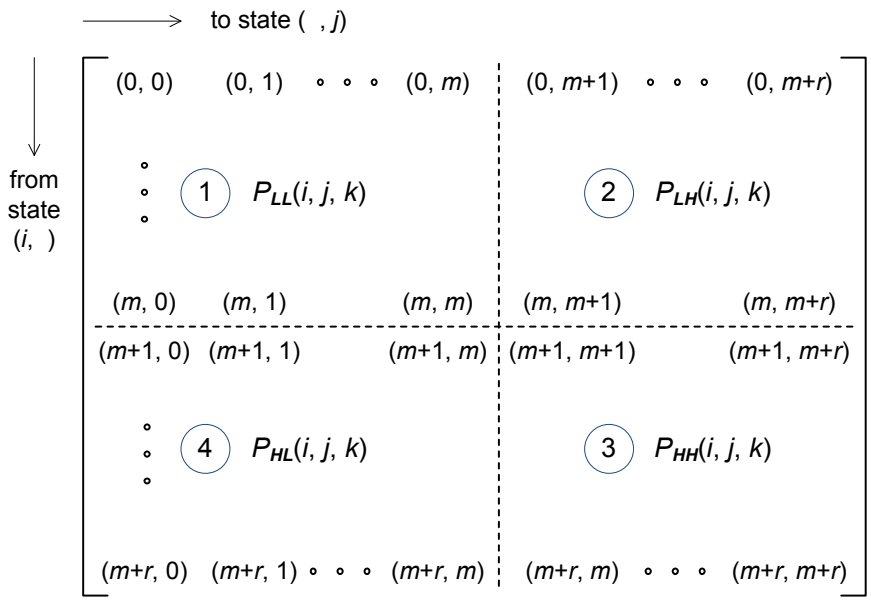


Figure 3.5: One-step transition probability matrix: range of validity for $P(i, j, k)$ equations.

$$P_{LL}(i,j,k) = \begin{cases} \sum_{z=0}^{min(i,w(k))} \binom{i+k}{w(k)} P_x^{w(k)} (1-P_x)^j, & \text{if } i+k \le m \\ \sum_{z=i+k-m}^{min(i,w(k))} \binom{i}{z} P_x^z (1-P_x)^{i-z} \cdot \\ \binom{k}{w(k)-z} P_{xy}^{w(k)-z} (1-P_{xy})^{z-i+j}, & \text{if } i+k > m \end{cases} \tag{3.11}$$

Region 2

In this region, the queue is empty before the transition but non-empty afterward, hence $i \le m, j > m$. This means that the arriving super-task was queued because it could not be serviced immediately due to the insufficient number of idle PMs. The corresponding

transition probabilities are

$$P_{LH}(i,j,k) = \Pi_{s=1}^{w(k)} \left[(i-s+1)P_{sx}\right] \cdot (1-P_x)^{i-w(k)} \tag{3.12}$$

Number of idle PMs

To calculate the transition probabilities for regions 3 and 4, we need to know the probability of having n idle PMs out of m. Namely, due to the total rejection policy, a super-task may have to wait in the queue even when there are idle PMs: this happens when the number of idle PMs is smaller than the number of tasks in the super-task at the head of the queue. To calculate this probability, let us consider an d Poisson batch arrival process in which the batch size is a generally distributed random variable, and each arriving batch is stored in a finite queue. Storing the arrival batches in the queue will be continued until either the queue gets full or the last arrival batch can't fit in. If the queue size is t, the mean batch size is $\overline{g}$, and the maximum batch size is equal to *MBS*, what is the probability, $P_i(n)$, of having n unoccupied spaces in the queue? Unfortunately, this probability can't be computed exactly, and an approximate solution is needed. To this end, we have simulated the queue size for different mean batch sizes, using the object-oriented Petri net-based simulation engine Artifex by RSoftDesign, Inc. [79]. We have fixed the queue size at $m = 200$, and ran the simulation experiment one million times; the resulting probability distribution is shown in Fig. 3.6.

The shape indicates an exponential dependency so we have empirically found the parameters that give the best fit, as shown in Table 3.1, for different values of mean batch size. In all cases, the approximation error remains below 0.18%.

This allows us to calculate the transition probabilities for regions 3 and 4 in the transi-

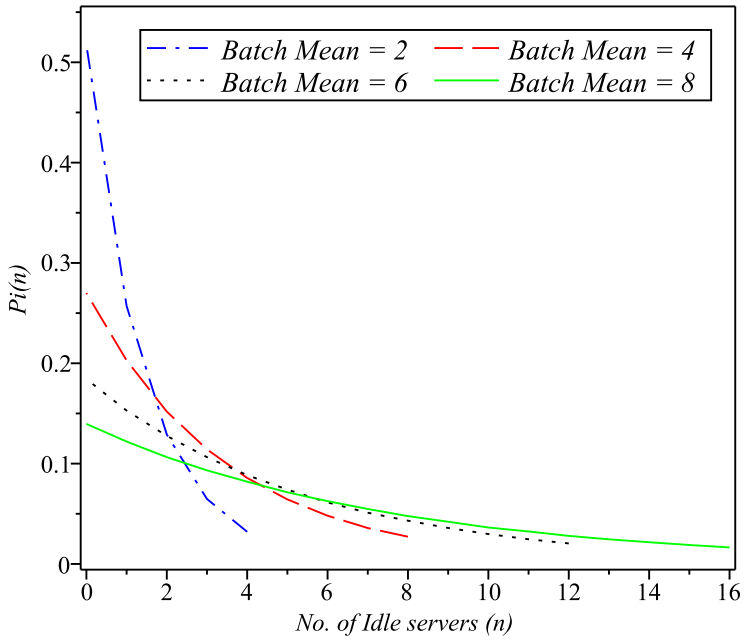


Figure 3.6: Probability of having n idle PMs ($P_i(n)$)s for different mean batch sizes.

Table 3.1: Parameters for optimum exponential curves ae^{bx}.

	batch size			
parameter	2	4	6	8
a	0.5154051	0.2725053	0.1839051	0.1393975
b	-0.6918772	-0.2913967	-0.1825455	-0.1334660

tion probability matrix.

Region 3

Region 3 corresponds to the case where the queue is not empty before and after the arrivals, i.e., $i, j > m$. In this case, transitions start and terminate at states above m in Fig. 3.2, and the state transition probabilities can be computed as

$$P_{HH}(i,j,k) \simeq \sum_{\psi=(m-MBS+1)}^{m} \sum_{s_1=\min(w(k),1)}^{\min(w(k),\psi)} \binom{\psi}{s_1} P_x^{s_1}(1-P_x)^{\psi-s_1} \cdot P_i(m-\psi)\cdot$$

$$\sum_{\delta=max(0,m-\psi+s_1-MBS+1)}^{m-\psi+s_1} \sum_{s_2=\min(w(k)-s_1,1)}^{\min(\delta,w(k)-s_1)} \binom{\delta}{s_2} P_{2x}^{s_2}(1-P_{2x})^{\delta-s_2} \cdot P_i(m-\psi+s_1-\delta)\cdot$$

$$\sum_{\phi=max(0,m-\psi+s_1-\delta+s_2-MBS+1)}^{m-\psi+s_1-\delta+s_2} \binom{\phi}{w(k)-s_1-s_2} P_{3x}^{w(k)-s_1-s_2}(1-P_{3x})^{\phi-w(k)+s_1+s_2}\cdot$$

$$P_i(m-\psi+s_1-\delta+s_2-\phi) \tag{3.13}$$

Region 4

Finally, region 4 corresponds to the situation where the queue is non-empty at the time of the first arrival, but it is empty at the time of the next arrival: $i > m, j \leq m$. The transition probabilities for this region are

$$P_{HL}(i,j,k) \simeq \sum_{\psi=(m-MBS+1)}^{m} \sum_{s_1=\min(0,\psi-j)}^{\min(w(k),\psi)} \binom{\psi}{s_1} P_x^{s_1}(1-P_x)^{\psi-s_1} \cdot P_i(m-\psi)\cdot$$

$$\sum_{\delta=max(0,m-\psi+s_1-MBS+1)}^{m-\psi+s_1} \sum_{s_2=\min(w(k)-s_1,\psi-j)}^{\min(\delta,w(k)-s_1)} \binom{\delta}{s_2} P_{2x}^{s_2}(1-P_{2x})^{\delta-s_2} \cdot P_i(m-\psi+s_1-\delta)\cdot$$

$$\sum_{\phi=max(0,m-\psi+s_1-\delta+s_2-MBS+1)}^{m-\psi+s_1-\delta+s_2} \binom{\phi}{w(k)-s_1-s_2} P_{3x}^{w(k)-s_1-s_2}(1-P_{3x})^{\phi-w(k)+s_1+s_2}\cdot$$

$$P_i(m-\psi+s_1-\delta+s_2-\phi) \tag{3.14}$$

3.3.3 Equilibrium Balance Equations

We are now ready to set the balance equations for the transition matrix:

$$\pi_i \;=\; \sum_{j=\max[0,i-MBS]}^{m+r} \pi_j p_{ji} \;, \quad 0 \le i \le m+r \tag{3.15}$$

augmented by the normalization equation

$$\sum_{i=0}^{m+r} \pi_i = 1. \tag{3.16}$$

The total number of equations is $m + r + 2$, but there are only $m + r + 1$ variables $[\pi_0, \pi_1, \pi_2, \ldots, \pi_{m+r}]$. Therefore, we need to remove one of the equations in order to obtain the unique equilibrium solution (which exists if the corresponding Markov chain were *ergodic*). A wise choice would be the last equation in (3.15) which holds the minimum amount of information about the system in comparison with the others. Here, the steady state balance equations can't be solved in closed form, hence we must resort to a numerical solution.

3.3.4 Distribution of the Number of Tasks in the System

Once we obtain the steady state probabilities we are able to establish the PGF for the number of tasks in the system at the time of a super-task arrival:

$$\Pi(z) = \sum_{k=0}^{m+r} \pi_z z^k \tag{3.17}$$

Due to batch arrivals, the PASTA property doesn't hold; thus, the PGF $\Pi(z)$ for the distribution of the number of tasks in the system at an arbitrary time is not the same as the PGF $P(z)$ for the distribution of the number of tasks in the system at the time of a super-task arrival. To obtain the former, we employ a two-step approximation technique

with an embedded semi-Markov process and an approximate embedded Markov process, as explained above. The aEMP imitates the original process but it will be updated only at the moments of super-task arrivals. Let $H_k(x)$ be the CDF of the residence time that aEMP stays in the state k:

$$H_k(x) \triangleq Prob[t_{n+1} - t_n \leq x \mid q_n = k] = 1 - e^{\lambda x}, \quad k = 0, 1, 2, ..., m + r \tag{3.18}$$

which in our system does not depend on n. The mean residence time in state k is

$$\overline{h}_k = \int_0^\infty [1 - H_k(x)]dx = \frac{1}{\lambda}, \quad k = 0, 1, 2, ..., m + r \tag{3.19}$$

and the steady-state distribution in aEMP is given by [85]

$$p_k^{sm} = \frac{\pi_k \overline{h}_k}{\sum_{j=0}^{m+r} \pi_j \overline{h}_j} = \frac{\pi_k}{\lambda \sum_{j=0}^{m+r} \pi_j 1/\lambda} = \pi_k \tag{3.20}$$

where $\{\pi_k;\ k = 0, 1, ..., m+r\}$ is the distribution probability at aEMC. So the steady-state probability of aEMP is identical with the embedded aEMC. We now define the CDF of the elapsed time from the most recent observing point looking form an arbitrary time by

$$H_k^-(y) = \frac{1}{\overline{h}_k} \int_0^y [1 - H_k(x)]dx, k = 0...m + r \tag{3.21}$$

The arbitrary-time distribution is given by

$$\begin{aligned} p_i = \sum_{j=i}^{m+r} p_j^{sm} \int_0^\infty \text{Prob[changes in } y \text{ that bring the state from } j \text{ to } i \text{] } dH_j^-(y) \\ = \sum_{j=i}^{m+r} \pi_j P(j, i, 0) \end{aligned} \tag{3.22}$$

The PGF of the number of tasks in system is given by

$$P(z) = \sum_{i=0}^{m+r} p_i z^i \tag{3.23}$$

Mean number of tasks in the system, then, obtained as

$$\overline{p} = P^{'}(1) \tag{3.24}$$

3.3.5 Blocking Probability

Since arrivals are independent of buffer state and the distribution of number of tasks in the system was obtained, we are able to directly calculate the blocking probability of a super-task in the system with buffer size of r:

$$P_b(r) \;=\; \sum_{k=0}^{MBS-1} \left[\sum_{i=0}^{MBS} p_{m+r-i-k}(1 - G(i)) \right] \cdot P_i(k) \tag{3.25}$$

The buffer size, r_ϵ, required to keep the blocking probability below the certain value, ϵ, is:

$$r_\epsilon \;=\; \{r \geq 0 \mid P_b(r) \leq \epsilon \;\&\; P_b(r-1) > \epsilon\} \tag{3.26}$$

3.3.6 Probability of Immediate Service

We are also interested in the probability that a super-task will get service immediately upon arrival, without any queuing. In this case, the response time would be equal to the service time:

$$P_{nq} \;=\; \sum_{k=0}^{MBS-1} \left[\sum_{j=0}^{m-k-MBS} p_j + \sum_{i=m-k-MBS+1}^{m-k-1} p_i G(m-k-i) \right] \cdot P_i(k) \tag{3.27}$$

3.3.7 Distribution of Response and Waiting Time in Queue

Let W denote the waiting time in the steady state, and similarly let $W(x)$ and $W^*(s)$ be the CDF, of W and its LST, respectively. For the $M^{[x]}/G/m/m+r$ systems the queue length has the same distribution as W, the number of tasks which arrive during the waiting time:

$$Q(z) = W^*(\lambda_e(1-z)) \tag{3.28}$$

in which $\lambda_e = \lambda(1 - P_b)$.

The left hand side of (3.28) in our system can be calculated as

$$Q(z) = \sum_{\psi=(m-MBS+1)}^{m} \left[\sum_{k=0}^{\psi-1} p_k + \sum_{k=\psi}^{m+r} p_k z^{\psi-k} \right] \cdot P_i(m-\psi) \tag{3.29}$$

Hence, we have

$$W^*(s) = Q(z)|_{z=1-(s/\lambda_e)} = Q(1-s/\lambda_e) \tag{3.30}$$

Moreover, the LST of response time is

$$T^*(s) = W^*(s)\ B^*(s) \tag{3.31}$$

in which the $B^*(s)$ is the LST of service time. The i-th moment, $t^{(i)}$, of the response time distribution is given by

$$t^{(i)} = \int_0^\infty x^i dT(x) \ = \ i \int_0^\infty x^{i-1}[1-T(x)]dx = (-1)^i T^{*(i)}(0) \qquad i = 2, 3, 4, \ldots \tag{3.32}$$

Using the moments we can calculate *standard deviation* as [38]:

$$\sigma_T = \sqrt{t^{(2)} - \bar{t}^2} \tag{3.33}$$

skewness as:

$$sk = \frac{t^{(3)} - 3\bar{t}\sigma_T^2 - \bar{t}^3}{\sigma_T^3} \tag{3.34}$$

and *kurtosis* as:

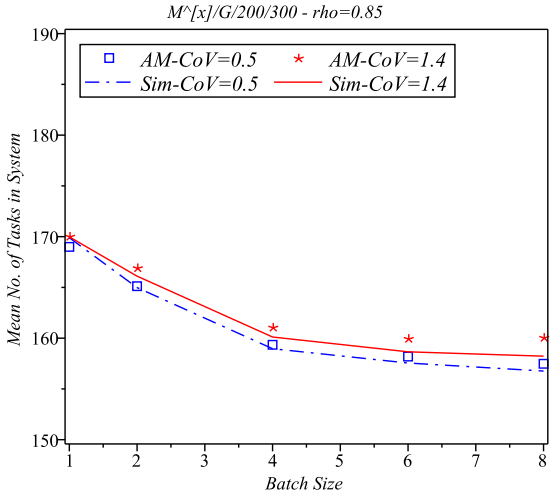
$$ku = \frac{t^{(4)} - 4t^{(3)}\bar{t} - 6t^{(2)}\bar{t}^2 - 3\bar{t}^4}{\sigma_T^4} - 3 \tag{3.35}$$

3.4 Numerical Validation

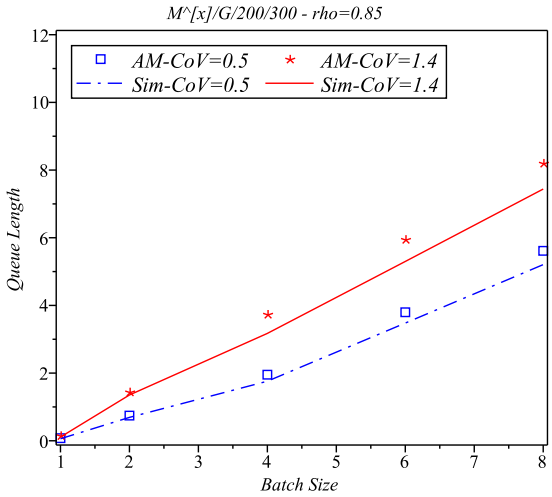
The resulting balance equations of analytical model have been solved using Maple 13 from Maplesoft, Inc. [66]. To validate the analytical solution, we have also built a discrete

event simulator of the cloud center using the Artifex engine [79]. We have configured a cloud center with $m = 200$ PMs and an input queue with $r = 100$ task slots. Traffic intensity was set to $\rho = 0.85$, which may seem too high but could easily be reached in private cloud centers or public centers of small/medium size providers. Task service time is assumed to have Gamma distribution, while the distribution of batch size is assumed to be Geometric. We set the maximum batch size as $MBS = \frac{3}{g} + 1$ in which g is the probability of success in the Geometric distribution. Note that in real cloud centers there is an upper limit for the number of PMs requested by a customer. Gamma distribution is chosen because it allows the coefficient of variation to be set independently of the mean value. Two values are used for the coefficient of variation: $CoV = 0.5$, which results in hypo-exponentially distributed service times, and $CoV = 1.4$, which results in hyper-exponentially distributed service times. In all plots, the simulation and analytical results are labeled with *Sim* and *AM*, respectively.

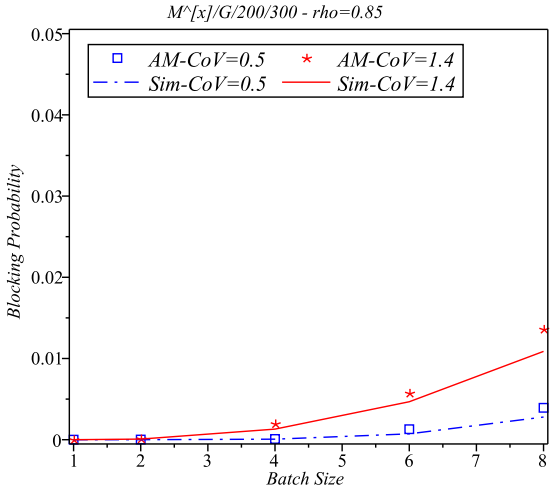
Diagrams in Fig. 3.7 show main performance indicators of this cloud system: mean number of tasks in the system, mean queue size, blocking probability, and the probability that a super-task will receive immediate service (i.e., that there will be a sufficient number of idle PMs to accommodate the super-task immediately upon arrival). As can be seen, the queue size increases with mean batch size whereas the mean number of tasks in the system decreases with it. This may be attributed to the total rejection policy, which lets some PMs to remain idle even though there exist some super-tasks in the queue. As might be expected, the blocking probability increases as the size of batches increases, while the probability of immediate service (which might be termed availability) decreases slowly. Nevertheless, the probability of immediate service is above 0.75 in the observed range of mean batch



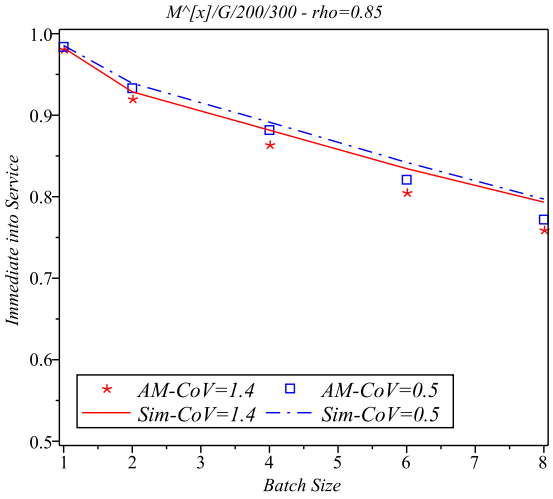
(a) Mean number of tasks in the system.



(b) Mean queue length.



(c) Blocking probability.

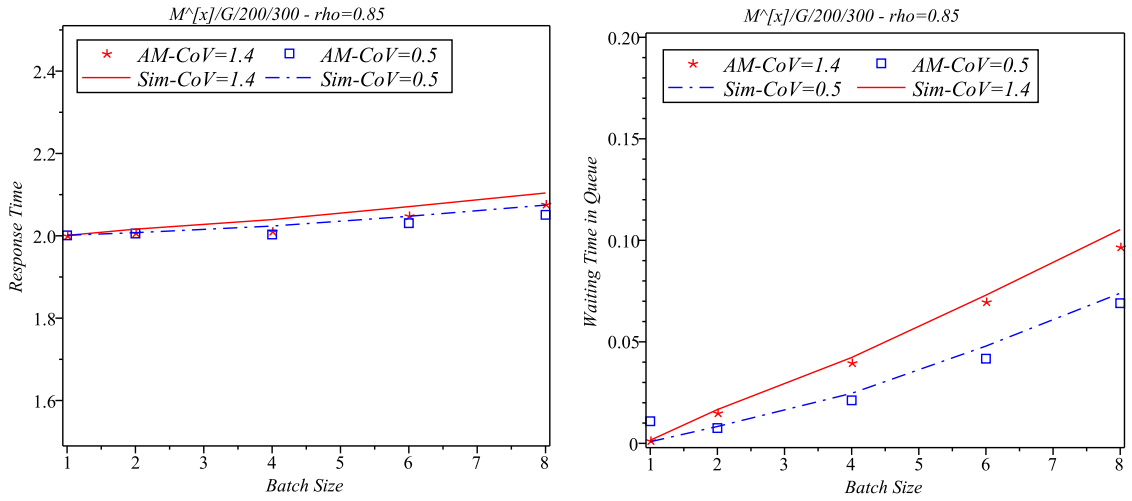


(d) Probability of getting immediate service.

Figure 3.7: Performance measures as function of mean batch size of super-tasks.

sizes, which means that more than 75% of user requests will be serviced immediately upon arrival, even though the total traffic is $\rho = 0.85$, which is rather high.

We have also computed the system response time and queue waiting time for super-



(a) Response time

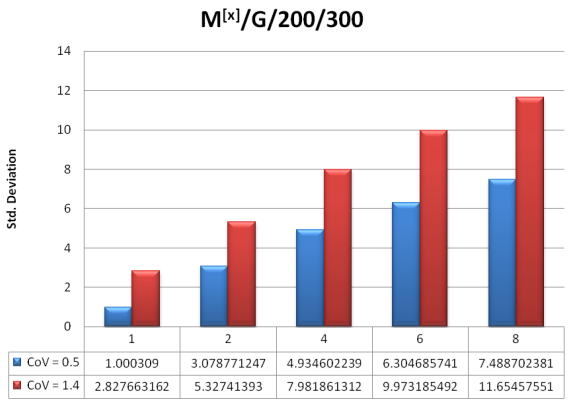
(b) Waiting time in queue

Figure 3.8: Response time and waiting time in queue.

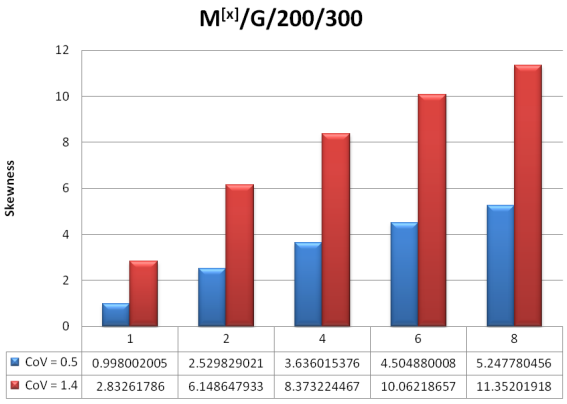
tasks. Note that, because of the total rejection policy explained above, the response and waiting times for super-tasks are identical for all individual tasks within the super-task. As depicted in Fig. 3.8, response time—which is the sum of waiting time and service time— and waiting time slowly increase with an increase in mean batch size. This may be attributed to the fact that larger mean batch size leads to larger super-tasks, which are more likely to remain longer in the queue before the number of idle PMs becomes sufficiently high to accommodate all of its individual tasks.

In all of the experiments described above, the agreement between analytical and simulation results is very good, which confirms the validity of the proposed approximation procedure using eMPS and aEMP/aEMC.

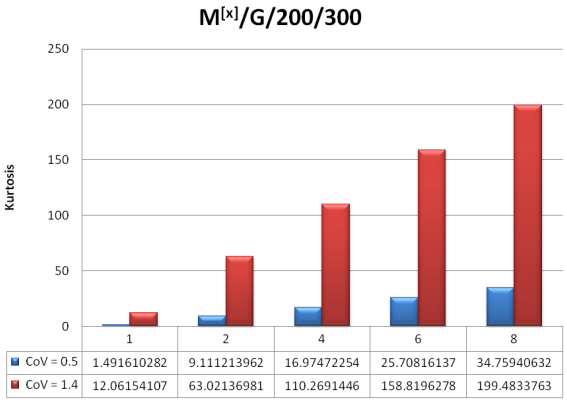
Not unexpectedly, performance at hyper-exponentially distributed service times ($CoV = 1.4$) is worse in general than that at hypo-exponential ones ($CoV = 0.5$). To obtain fur-



(a) Standard Deviation.



(b) Skewness.



(c) Kurtosis.

Figure 3.9: Higher moment related performance metrics.

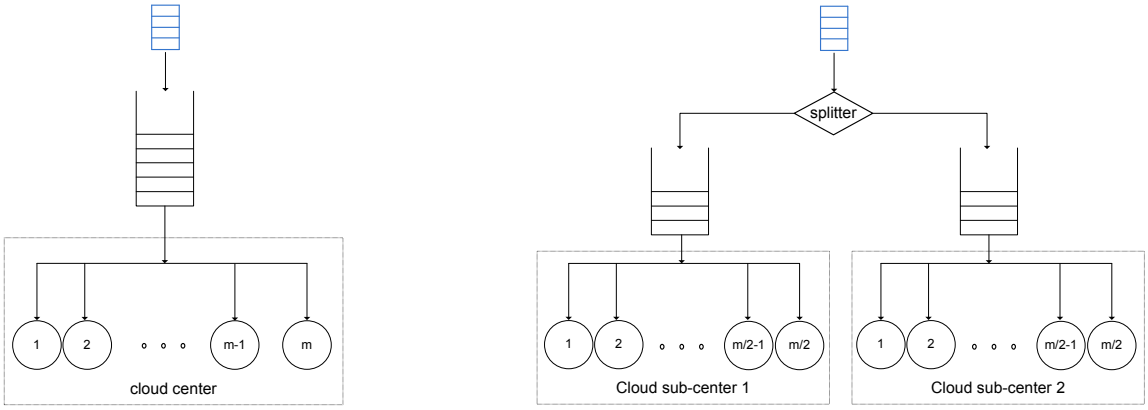
ther insight into cloud center performance, we have also calculated the higher moments of response time which are shown in Fig. 3.9. As can be seen, standard deviation, shown in Fig 3.9(a), increases as the mean batch size increases, and the response time is more dispersed when the service time of tasks is hyper-exponentially distributed.

With respect to the higher moment-based measure, let us note that skewness is a measure of symmetry of a distribution around the mean (a value of zero indicates a fully sym-

metric distribution, while positive/negative values indicate that the tails are longer on the left/right hand side of the mean, respectively), while kurtosis indicates whether the distribution is more 'peaked' or 'flat' with respect to the normal distribution with the same mean and standard deviation. As can be seen, the skewness, shown in Fig. 3.9(b), is rather high, and increases with batch size and/or CoV of service time. This indicates that, the higher the CoV of service time distribution, the longer the tail of the response time distribution will be. As can be seen from Fig. 3.9(c), kurtosis is high which indicates the response time distribution is relatively peaked around the mean. In addition, the kurtosis values obtained for $CoV = 1.4$ is noticeably higher than the corresponding values for $CoV = 0.5$. These results imply that, in practice, the distribution of response time will exhibit a rapid increase with a very pronounced mean value, and then decrease slowly with a rather long tail. A long tail means that some super-tasks may experience much longer response time than the others, esp. when the input traffic consists of super-tasks with widely varying service times and/or large number of tasks. Consequently, a non-negligible portion of super-tasks will experience extremely long delays, or even blocking, which may be unacceptable for cloud operators, in private as well as in public clouds, and ways to improve performance must be sought.

3.5 The Impact of Homogenization

The results presented above indicate that the response time is very sensitive to the coefficient of variation of task service times and the number of tasks in a super-task. Therefore, some way to keep these values within certain bounds may help improve performance of the cloud center. Since users can't be forced to submit requests with prescribed characteristics,



(a) Single, heterogeneous cloud center.

(b) Two homogeneous cloud sub-centers.

Figure 3.10: Two configurations of the cloud computing center.

the cloud provider may try to do so internally. One way of 'taming' the requests is to separate the input super-task stream into two or more sub-streams that are better behaved, and then feed those sub-streams into separate sub-clouds. In this manner, a single 'homogeneous' cloud center, shown in Fig. 3.10(a), is to be partitioned into two or more sub-clouds which deal with more 'homogeneous' request streams, Fig. 3.10(b). Partitioning should be made on the basis of the coefficient of variation of task service times and/or on the basis of the size of (i.e., the number of tasks within) the super-task.

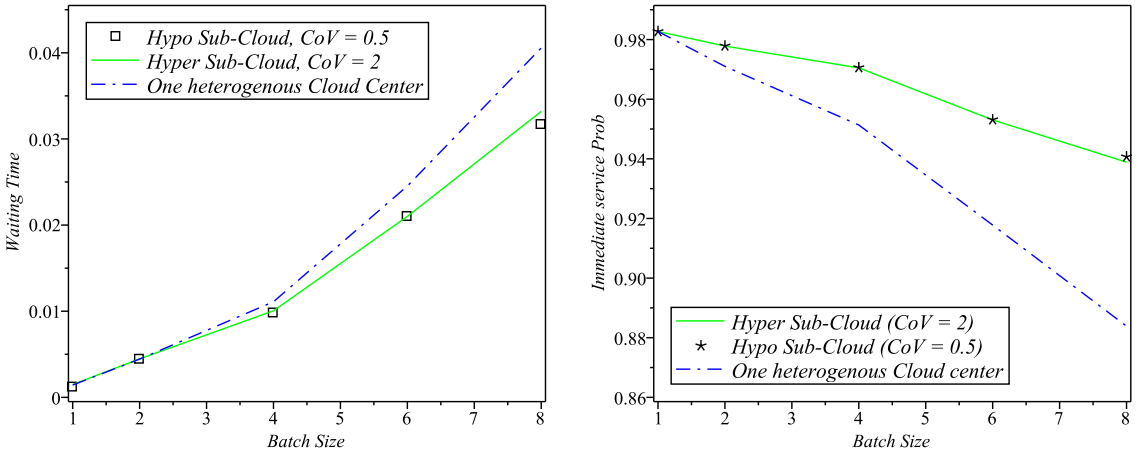
To validate this approach, we carried out two experiments with a cloud center with $m = 200$ PMs and $r = 100$ slots in the input task buffer. In both cases, the incoming super-tasks have mean number of tasks of 2 and Gamma-distributed task service times.

First, super-tasks are partitioned according to the CoV of their task service times. We assume two types of super-tasks arriving at the cloud center: super-tasks in which task service times are hypo-exponentially distributed with $CoV = 0.5$ and super-tasks having tasks

with hyper-exponentially distributed service time with $CoV = 2$. The primary configuration kept both incoming super-task streams in the same queue and fed them to the original cloud center. Then, the incoming streams were split, and the resulting sub-streams were fed to sub-clouds with $m' = 100$ PMs and an input buffer with $r' = 50$ task slots. However, the separation was not complete, and partial server sharing was implemented: namely, if one of the sub-clouds could not accommodate the super-task at the head of the queue but the other sub-cloud could, the super-task was routed to the other sub-cloud. The results, shown in Fig. 3.11(a), confirm that the aggregate waiting time is indeed lower in the case of homogeneous sub-centers than in the case of one heterogeneous center, despite its larger capacity. This is further confirmed by the diagrams in Fig. 3.11(b) that show the probability of immediate service; again, homogeneous sub-clouds provide better performance than a single but heterogeneous cloud. Similar results, albeit in a different setting, were reported in [9, 82].

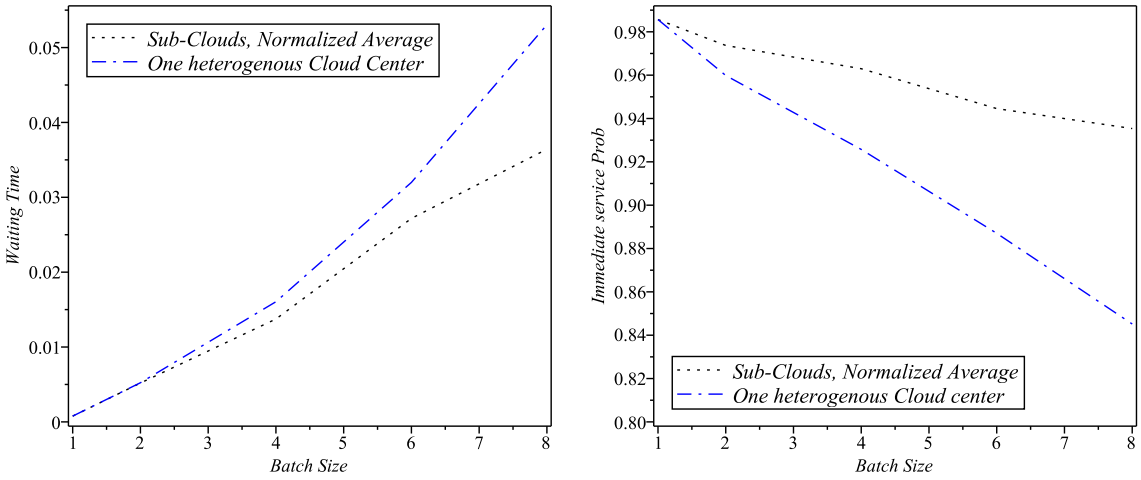
In our second experiment, incoming super-tasks were partitioned according to their size, as follows. Super-tasks with size of $\overline{g}+2$ (where $\overline{g}$ denotes mean super-task size) were sent to one of the sub-clouds, while those with larger size were sent to another. As above, partial sharing of sub-clouds was allowed. As can be seen from the diagrams Fig. 3.12, homogenization of this type also leads to better performance in terms of both mean waiting time and probability of immediate service.

From the results of these experiments, we may conclude that a simple partitioning of input requests may help cloud providers improve customer-perceived performance. Reduced waiting times also indicate that the utilization of cloud PMs is better, which is advantageous for the cloud provider. Moreover, economic incentives (such as appropriate pricing



(a) Waiting time. (b) Probability of immediate service.

Figure 3.11: One heterogeneous center versus two homogeneous sub-clouds; homogenization is based on CoV of tasks' service time.



(a) Waiting time. (b) Probability of immediate service.

Figure 3.12: One heterogeneous center versus two homogeneous sub-clouds; homogenization is based on super-tasks' size.

policies) should be considered to encourage customers to break down super-tasks (batches) into the smallest possible size; these issues will be the topic of our future work.

3.6 Summary of the Chapter

In this Chapter, we have described the first analytical model for performance evaluation of a cloud computing center under Poisson batch arrivals with generally distributed task sizes and generally distributed task service times, under a total acceptance/rejection policy. The model is based on a two-stage approximation technique where the original non-Markovian process is first modeled with an embedded semi-Markov process, which is then modeled by an approximate embedded Markov process but only at the time instants of super-task arrivals; the number of departures between two successive super-task arrivals is counted approximately. This technique provides quite accurate computation of important performance indicators such as the mean number of tasks in the system, queue length, mean response and waiting time, blocking probability and the probability of immediate service. While the model is approximate, its accuracy is more than sufficient under a wide range of input parameter values; in particular, it allows accurate modeling of cloud centers with a large number of PMs.

Our findings also show that cloud centers which allow requests with widely varying service times may experience longer waiting time and lower probability of getting immediate service for its clients. A simple partitioning-based homogenization in which super-tasks with similar characteristics are grouped into separate streams and serviced by separate subclouds, is shown to provide superior performance with respect to the case where all requests are serviced by a single cloud center. Partitioning may be done on the basis of super-task

size and/or coefficient of variation of task service times. In this manner, cloud customers may obtain better service from cloud centers specifically geared to handling their type of services, and they may expect generally better performance if their request are submitted in super-tasks with the minimum possible number of required PMs.

Chapter 4

Performance Modeling: Virtualized Environment

In chapter 3, we have incorporated the batch arrivals to our performance model which means a super-task may request multiple physical machines (PM) at once. Therefore, the proposed performance model did not support single PM sharing among super-tasks. In this chapter, however, we introduce the concept of virtualized PM to our performance model to increase the fidelity of the model to the real Infrastructure-as-a-Service (IaaS) cloud centers [47]. This chapter is organized as follows: in section 4.1, a virtualized environment and VMware VMmark performance score are elaborated. Our analytical model is described in section 4.2. We present numerical validation in section 4.3. The summery of the chapter is outlined in section 3.6.

4.1 Virtualized Physical Machines

Performance evaluation is particularly challenging in scenarios where virtualization is used to provide a well defined set of computing resources to the users [22], and even more so when the degree of virtualization, i.e., the number of virtual machines (VMs) running on a single physical machine (PM) is high, at the current state of technology, means well over hundred of VMs. For example, the recent VMmark benchmark results by VMware citevmmark-res, a single physical machine can run as many as 35 tiles or about 200 VMs with satisfactory performance. Table 4.1 shows the workloads and applications being run with each VMmark tile. A tile consists of six different VMs that imitate a typical data center environment. Note that the standby server virtual machine does not run an application; however, it does run an operating system and is configured as one CPU with a specified amount of memory and disk space [95].

Table 4.1: VMmark workload summary per tile (adapted from [33]).

VM	**Benchmark**	**Operating System**	**VM Specification**	**Metric**
Database Server	MySQL	SUSE Linux ES x64	2 vCPUs 2GB RAM	Commit/min
Mail Server	LoadSIM	MS Win. 2003 x86	2 vCPUs 1GB RAM	Action/min
Java Server	SPECjbb2005	MS Win. 2003 x64	2 vCPUs 1GB RAM	Order/min
File Server	Dbench	SUSE Linux ES x86	1 vCPUs 256MB RAM	MB/sec
Web Server	SPECjbb2005	SUSE Linux ES x64	2 vCPUs 512MB RAM	Access/min
Standby	N/A	MS Win. 2003 x86	1 vCPUs 256MB RAM	N/A

We assume that the cloud center consists of many physical machines, each of which can host a number of virtual machines, as shown in Fig. 4.1. Incoming requests are routed

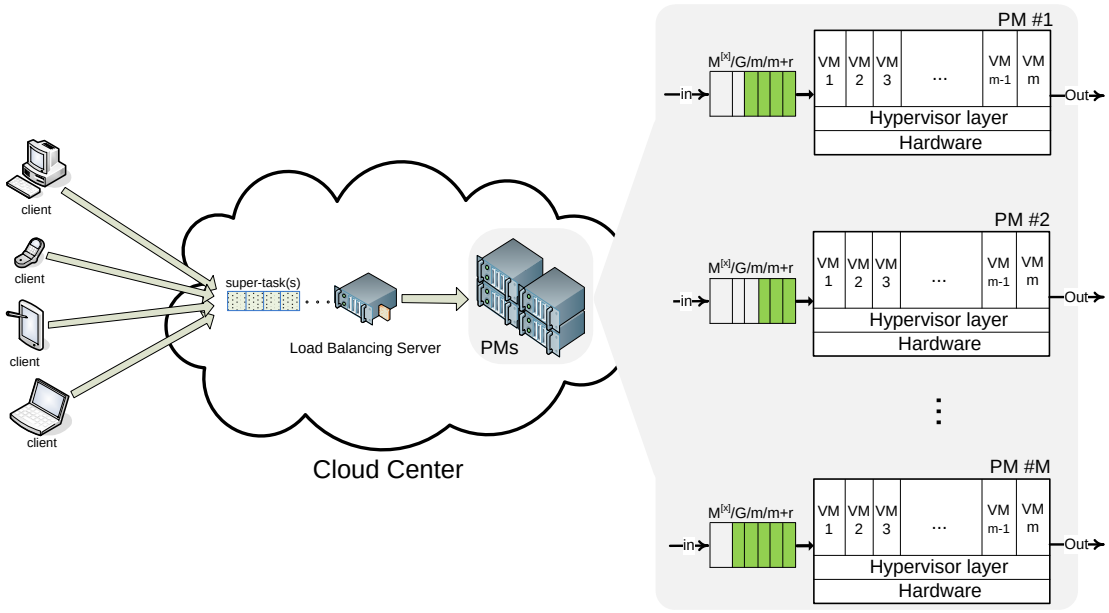


Figure 4.1: The architecture of the cloud center.

through a load balancing server to one of the PMs. Users can request one or more VMs at a time, i.e., we allow batch (or super-) task arrivals, which is consistent with the so-called *On-Demand* services in the Amazon Elastic Compute Cloud (EC2) [2]; as these services provide no advance reservation and no long-term commitment, clients may experience delays in fulfillment of requests. While the number of potential users is high, each user typically submits single batch request at a time with low probability. Therefore, the super-task arrival can be adequately modeled as a Poisson process [30].

When a super-task arrives, the load balancing server attempts to provision it – i.e., allocate it to a single PM with the necessary capacity. As each PM has a finite input queue for individual tasks, this means that the incoming super-tasks are processed as follows:

- If a PM with sufficient spare capacity is found, the super-task is provisioned imme-

diately.

- Else, if a PM with sufficient space in the input queue can be found, the tasks within the super-task are queued for execution.

- Otherwise, the super-task is rejected.

In this manner, all tasks within a super-task are processed together, whether they are accepted or rejected. This policy, known as total acceptance/rejection, benefits the users that typically would not accept partial fulfillment of their service requests. It also benefits the cloud providers since provisioning of all tasks within a super-task on a single PM reduces inter-task communication and, thus, improves performance.

As statistical properties of task service times are not well known and cloud providers don't publish relevant data, it is safe to assume a general distribution for task service times, preferably one that allows the coefficient of variation (CoV) to be adjusted independently of the mean value [13]. However, those service times apply to a VM running on a non-loaded PM. When the total workload of a PM increases, so does the overhead required to run several VMs simultaneously, and the actual service times will increase. Our model incorporates this adjustment as well, as will be seen below.

In summary, our work advances the performance analysis of cloud computing centers by incorporating a high degree of virtualization, batch arrival of tasks and generally distributed service time. These key aspects of cloud centers have not been addressed in a single performance model previously.

4.2 Analytical Model

We assume that the cloud center consists of M identical PMs with up to m VMs each, as shown in Fig. 4.1. Super-task arrivals follow a Poisson process, as explained above, and the inter-arrival time A is exponentially distributed with a rate of $\frac{1}{\lambda_i}$. We denote its cumulative distribution function (CDF) with $A(x) = Prob[A < x]$ and its probability density function (pdf) with $a(x) = \lambda_i e^{-\lambda x}$; the LST of this pdf is

$$A^*(s) = \int_0^\infty e^{-sx} a(x) dx = \frac{\lambda_i}{\lambda_i + s}.$$

Note that the total arrival rate to the cloud center is λ so that

$$\lambda = \sum_{i=1}^{M} \lambda_i.$$

Let g_k be the probability that the super-task size is $k = 1, 2, \ldots$ ***MBS***, in which ***MBS*** is the maximum batch size which, for obvious reasons, must be finite; both g_k and ***MBS*** depend on user application. Let $\overline{g}$ and $\Pi_g(z)$ be the mean value and probability generating function (PGF) of the task burst size, respectively:

$$\begin{aligned} \Pi_g(z) &= \sum_{k=1}^{MBS} g_k z^k \\ \overline{g} &= \Pi_g^{(1)}(1) \end{aligned} \tag{4.1}$$

Therefore, we model each PM in the cloud center as an $M^{[x]}/G/m/m+r$ queuing system. Each PM may run up to m VMs and has a queue size of r tasks.

Service times of tasks within a super-task are identically and independently distributed according to a general distribution. However, the parameters of the general distribution (e.g., mean value, variance and etc.) are dependent on the virtualization degree of PMs. The degradation of mean service time due to workload can be approximated using data from

recent cloud benchmarks [6]; unfortunately, nothing can be inferred about the behavior of higher moments of the service time citecao:2012. For example, Fig. 4.2 depicts the VMmark performance results for a family of same-generation PMs under different number of deployed tiles [96]. The VMmark score is an overall performance of a server under virtualized environment; a detailed description of VMmark score can be found in [95]. From date in Fig. 4.2, we can extract the dependency of mean service time on the number of deployed VMs, normalized to the service time obtained when the PM runs a single tile, as shown in Fig. 4.3. Note that mean service time represents both computation and communication time for a task within super-tasks.

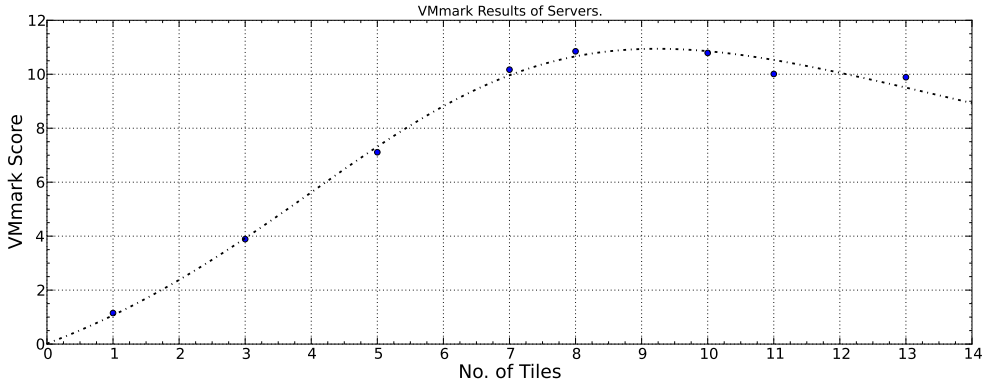


Figure 4.2: Performance vs. no. of tiles.

The dependency shown in Fig. 4.3 may be approximated to give the normalized mean service time, $\overline{b}_n(y)$, per tile as a function of virtualization degree:

$$\overline{b}_n(y) = \frac{\overline{b}(y)}{\overline{b}(1)} = 1/(0.996 - (8.159 * 10^{-6}) * y^{4.143}) \tag{4.2}$$

where y denotes the number of tiles deployed on a single PM. Therefore, the CDF of the service time will be $B_y(x) = Prob\,[B_y < x]$ and its pdf is $b_y(x)$, while the corresponding

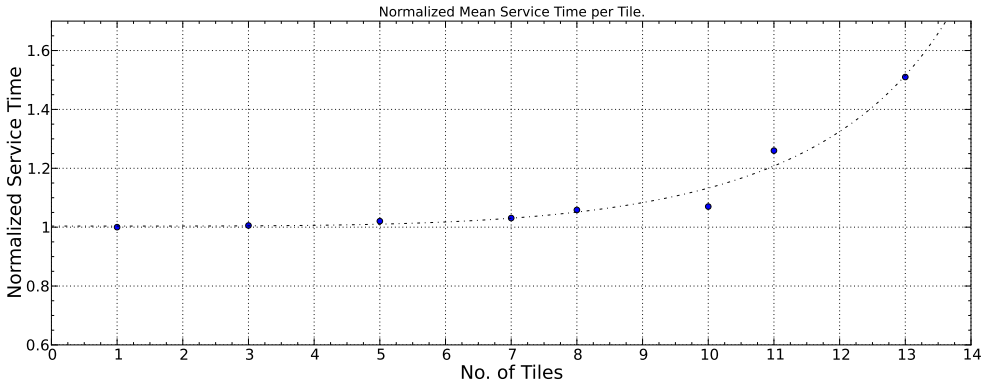


Figure 4.3: Normalized mean service time vs. no. of tiles.

LST is

$$B_y^*(s) = \int_0^\infty e^{-sx} b_y(x) dx.$$

The mean service time is then:

$$\overline{b}(y) = -B_y^{'*}(0).$$

If the maximum number of allowed tiles on each PM is Ω, then the aggregate LST will be

$$B^*(s) = \sum_{y=1}^{\Omega} p_y B_y^*(s).$$

where p_y is the probability of having y tiles on a single PM. Then the aggregate mean service time is

$$\overline{b} = -B^{'*}(0) = \sum_{y=1}^{\Omega} p_y \overline{b}(y).$$

The mass probabilities $\{p_y,\ y = 1 \,..\, \Omega\}$ are dependent on the *traffic intensity* and obligations stemming from service level agreement between the provider and the user. A cloud provider may decide to deploy more tiles on PMs to reduce the waiting time – which, according to (4.2), will lead to deterioration of service time. We will examine the dependency between y and the traffic intensity in our numerical validation.

As service times are not exponentially distributed, the corresponding process for the PM performance model is not Markovian and direct analysis of this system is not possible. To arrive at a tractable solution, we use a staged approximation-based procedure, as follows. First, we define an embedded semi-Markov process that coincides with the original process at the moments of super-task arrivals and task departures. Second, we define an embedded Markov process that approximates the semi-Markov process at the time of super-task arrivals. In this process, task departures between two successive super-task arrivals are counted, but they are assumed to occur at the time of next super-task arrival. The approximate embedded Markov chain corresponding to the embedded Markov process is then solved to obtain the relevant performance indicators.

Hereafter, the analytical model steps are the same as the analytical model in chapter 3. Note that in this chapter we model each PM as a $M^{[x]}/G/m/m+r$ queuing system as opposed to the whole cloud center. Due to different parameter setting and configuration, the number of idle PMs (i.e., Fig. 4.4 and Table 4.2) is calculated in a different manner.

Table 4.2: Parameters for optimum exponential curves ab^x.

	mean super-task size				
parameter	5	10	15	20	25
a	1.99E-01	1.00E-01	6.87E-02	5.40E-02	4.59E-02
b	8.00E-01	9.00E-01	9.33E-01	9.50E-01	9.60E-01

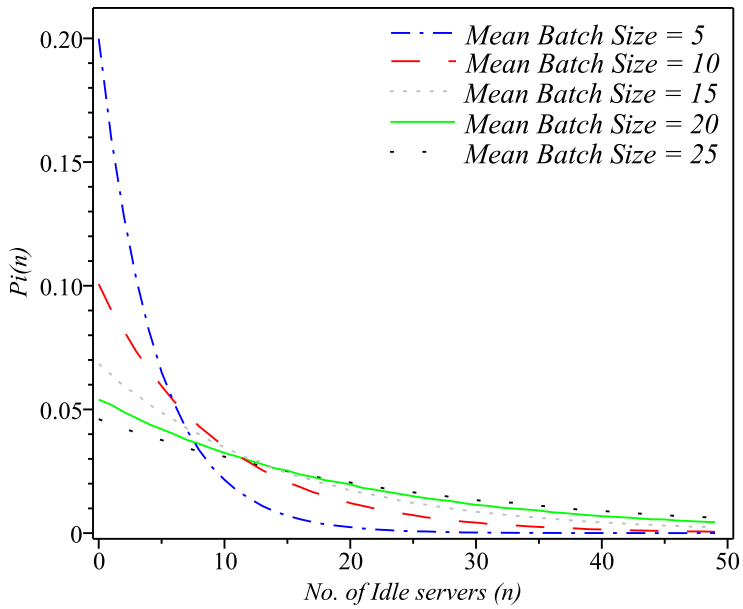


Figure 4.4: Probability of having n idle PMs ($P_i(n)$)s for different mean batch sizes.

4.3 Numerical Validation

We have configured a cloud center with $M = 1000$ PMs under two degrees of virtualization: 100 and 200 VMs per PM. We refer them as *moderate* and *heavy* configuration respectively. Latest VMmark results [96], show that only a few types of current state-of-the-art PMs can run up to 200 VMs with an acceptable performance while most of them are optimized to run up to 100 VMs.

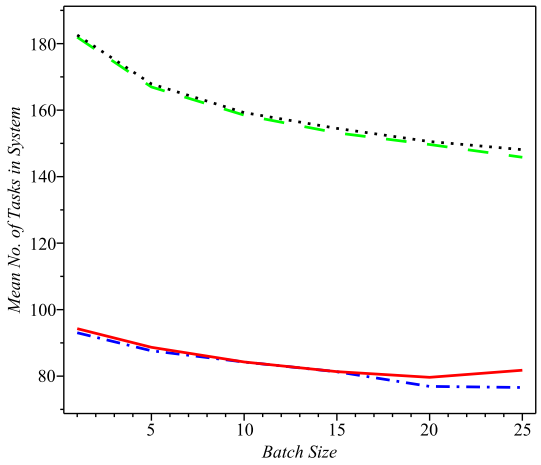
The input queue size is set to $r = 300$ tasks. Traffic intensity is set to $\rho = 0.85$, which may seem too high but could easily be reached in private cloud centers or public centers of small/medium size providers. The distribution of batch size is assumed to be truncated Geometric, with maximum batch size set to $MBS = \frac{3}{g} + 1$ in which g is the probability of success in the Geometric distribution. Note that real cloud centers may impose an upper

limit for the number of servers requested by a customer. Task service time is assumed to have Gamma distribution, which is chosen because it allows the coefficient of variation to be set independently of the mean value. The mean value depends on the degree of virtualization: on account of (4.2), we set the task service time as 120 and 150 minute for moderate and heavy virtualization, respectively. Two values are used for the coefficient of variation: $CoV = 0.5$ and 1.4, which give rise to hypo- and hyper-exponentially distributed service times, respectively.

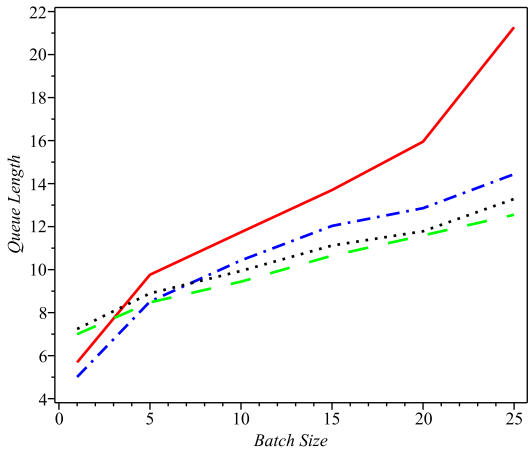
4.3.1 Analytical Model Results

Diagrams in Fig. 4.5 show the analytical results for various performance indicators vs. super-task size. As can be seen, the queue size increases with mean batch size whereas the mean number of tasks in the system decreases with it. The moderate configuration always outperforms the heavy configuration for single request arrivals (i.e., super-tasks with size one), as opposed to bigger super-tasks. This may be attributed to the total rejection policy, which lets some PMs to remain idle even though there exist some super-tasks (potentially big super-tasks) in the queue. As might be expected, the blocking probability increases as the size of batches increases, while the probability of immediate service (which might be termed availability) decreases almost linearly. Nevertheless, the probability of immediate service is above 0.5 in the observed range of mean batch sizes, which means that more than 50% of user requests will be serviced immediately upon arrival, even though the total traffic is $\rho = 0.85$, which is rather high.

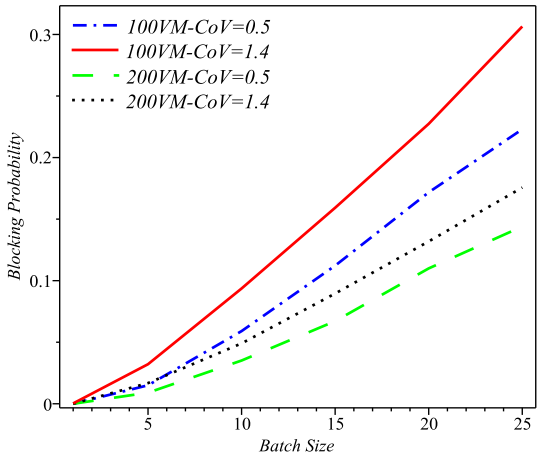
We have also computed the system response time and queue waiting time for super-tasks. Note that, because of the total rejection policy explained above, the response and



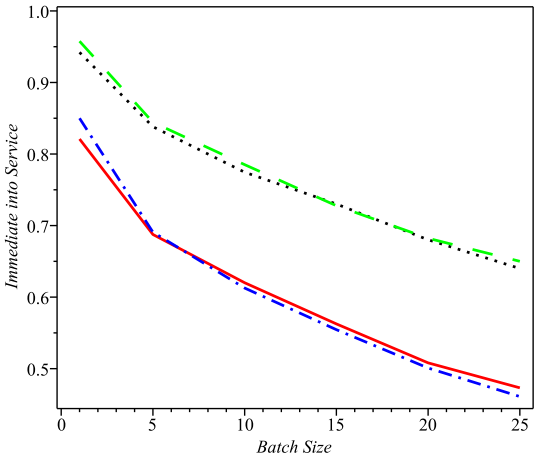
(a) Mean number of tasks in the system.



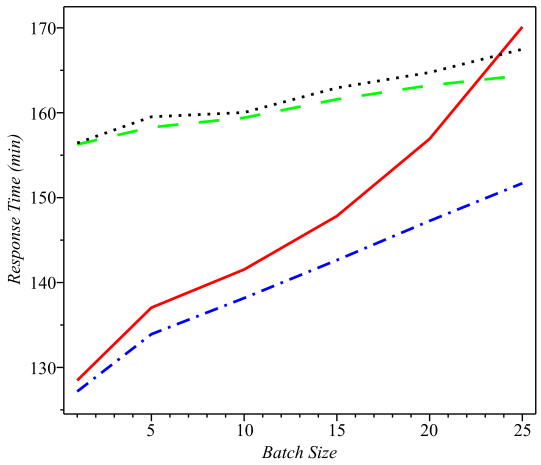
(b) Mean queue length.



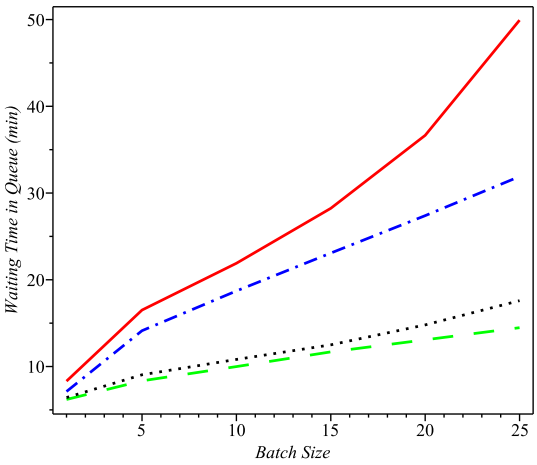
(c) Blocking probability.



(d) Probability of getting immediate service.



(e) Response time.



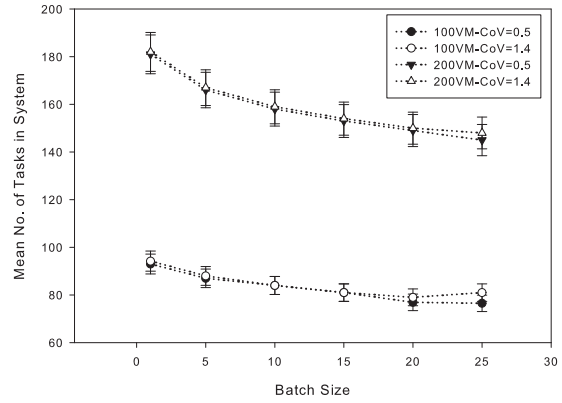
(f) Waiting time in the queue.

Figure 4.5: Performance measures as function of mean batch size of super-tasks.

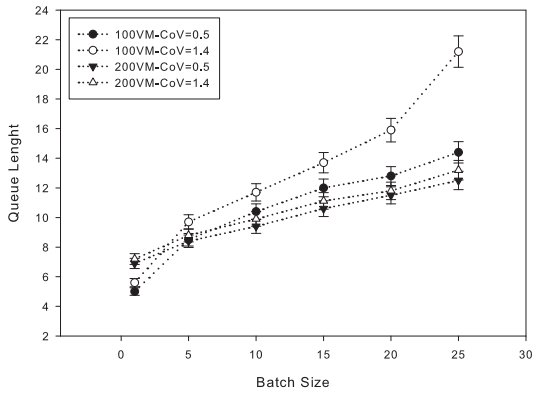
waiting times for super-tasks are identical for all individual tasks within the super-task. As depicted in Fig. 4.5(e), response time—which is the sum of waiting time and service time—and waiting time, Fig. 4.5(f), slowly increase with an increase in mean batch size. This may be attributed to the fact that larger mean batch size leads to larger super-tasks, which are more likely to remain longer in the queue before the number of idle PMs becomes sufficiently high to accommodate all of its individual tasks. Also, heavy configuration imposes shorter waiting time in queue on super-tasks. In other words, waiting time in queue is heavily influenced by admission policy rather than overhead of virtualization. However, response time is more sensitive to the degree of virtualization. It can also be seen that CoV of service time plays an important role on total response time for large super-tasks (i.e., size 25). In all of the experiments described above, the agreement between analytical and simulation results is very good, which confirms the validity of the proposed approximation procedure using eSMP and aEMP/aEMC.

4.3.2 Simulation Results

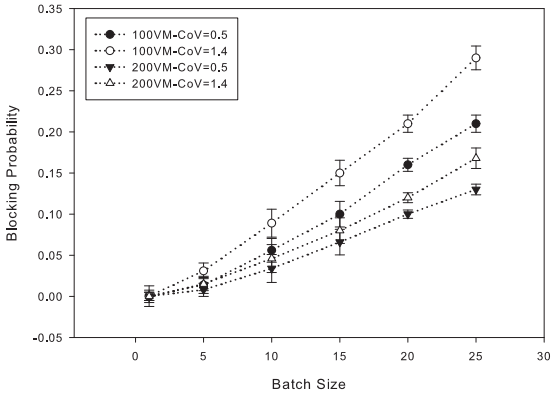
Using the object-oriented Petri net-based simulation engine Artifex by RSoftDesign Inc. [79], we developed an independent simulation model. Fig. 4.6 presents corresponding simulation results, with 95% confidence interval, for Fig. 4.5 in Section 4.3.1. As can be seen, simulation results are in a good match with results from analytical model.



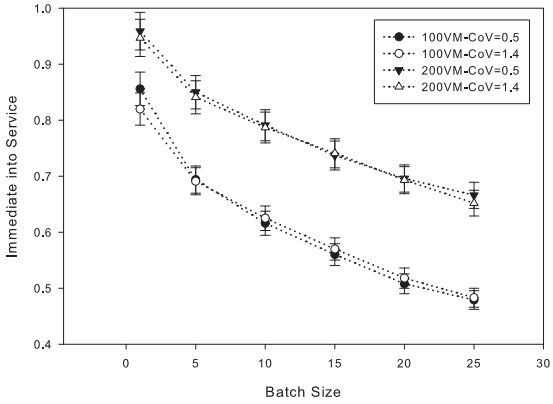
(a) Mean number of tasks in the system.



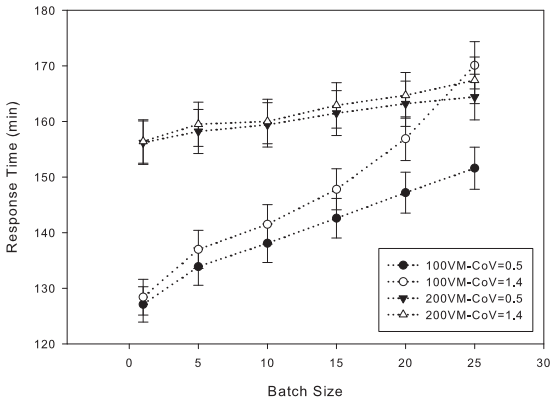
(b) Mean queue length.



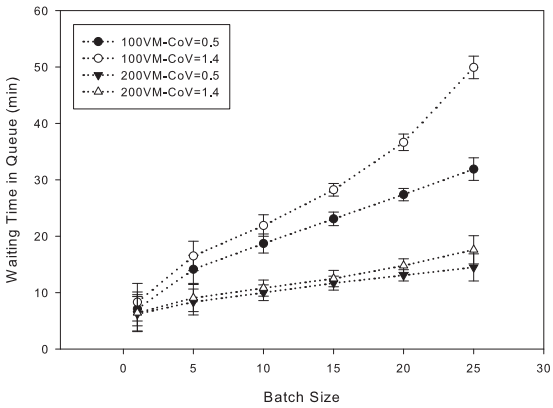
(c) Blocking probability.



(d) Probability of getting immediate service.



(e) Response time.



(f) Waiting time in the queue.

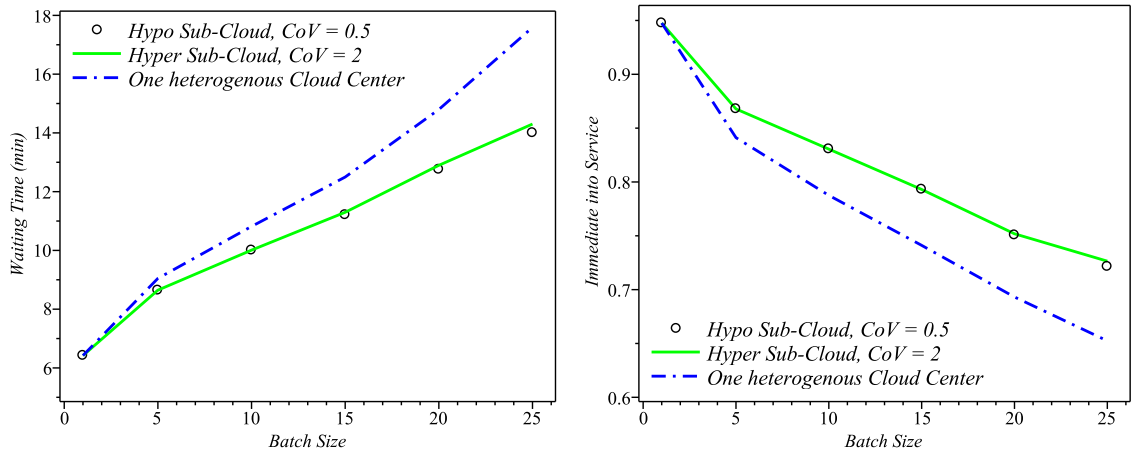
Figure 4.6: Performance measures as function of mean batch size of super-tasks (Simulation Model).

4.4 The Impact of Homogenization

The results presented above indicate that the response time is very sensitive to the coefficient of variation of task service times and the number of tasks in a super-task. Therefore, some way to keep these values within certain bounds may help improve performance of the cloud center. Since users cannot be forced to submit requests with prescribed characteristics, the cloud provider may try to do so internally. One way of 'taming' the requests is to separate the input super-task stream into two or more sub-streams that are better behaved, and then feed those sub-streams into separate sub-clouds. In this manner, a single 'homogeneous' cloud center is to be partitioned into two or more sub-clouds which deal with more 'homogeneous' request streams. Partitioning should be made on the basis of the coefficient of variation of task service times and/or on the basis of the size of (i.e., the number of tasks within) the super-task.

To validate this approach, we carried out two experiments with a cloud center consisting of $M = 1000$ PMs, each of which runs up to 100 VMs and has an input buffer of capacity $r = 250$ slots. In both cases, the incoming super-tasks have mean number of tasks of 2 and Gamma-distributed task service times.

First, super-tasks are partitioned according to the CoV of their task service times. We assume two types of super-tasks arriving at the cloud center: super-tasks in which task service times are hypo-exponentially distributed with $CoV = 0.5$ and super-tasks having tasks with hyper-exponentially distributed service time with $CoV = 2$. The primary configuration kept both incoming super-task streams in the same pool of PMs in the original cloud center. Then, the incoming streams were split, and the resulting sub-streams were fed to sub-clouds with $M' = 500$ PMs and an input buffer with $r' = 250$ task slots. However,



(a) Waiting time.

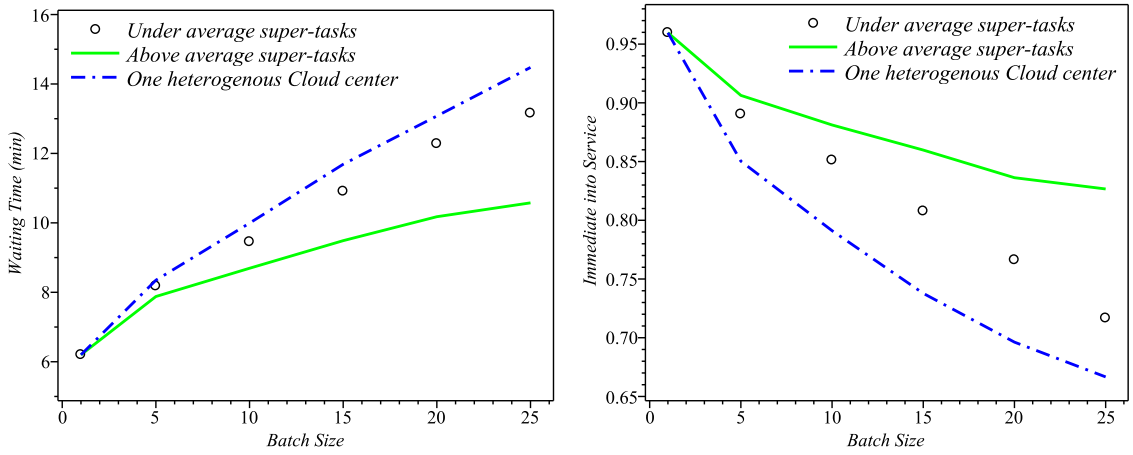
(b) Probability of immediate service.

Figure 4.7: One heterogeneous center versus two homogeneous sub-clouds; homogenization is based on CoV of tasks' service time.

the separation was not complete, and partial server sharing was implemented: namely, if one of the sub-clouds could not accommodate the super-task at the head of the queue but the other sub-cloud could, the super-task was routed to the other sub-cloud. The results, shown in Fig. 4.7(a), confirm that the aggregate waiting time is indeed lower in the case of homogeneous sub-centers than in the case of one heterogeneous center, despite its larger capacity. This is further confirmed by the diagrams in Fig. 4.7(b) that show the probability of immediate service; again, homogeneous sub-clouds provide better performance than a single but heterogeneous cloud. Similar results, albeit in a different setting, were reported in [9, 82].

In our second experiment, incoming super-tasks were partitioned according to their size, as follows. Super-tasks with size up to $\overline{g}$ (where $\overline{g}$ denotes mean super-task size) were sent to one of the sub-clouds, while those with larger size were sent to another. As above, partial sharing of sub-clouds was allowed. As can be seen from the diagrams Fig. 4.8,

homogenization of this type also leads to better performance in terms of both mean waiting time and probability of immediate service.



(a) Waiting time.

(b) Probability of immediate service.

Figure 4.8: One heterogeneous center versus two homogeneous sub-clouds; homogenization is based on super-tasks' size.

From the results of these experiments, we may conclude that a simple partitioning of input requests may help cloud providers improve customer-perceived performance. Reduced waiting times also indicate that the utilization of cloud servers is better, which is advantageous for the cloud provider. Moreover, economic incentives (such as appropriate pricing policies) should be considered to encourage customers to break down super-tasks (batches) into the smallest possible size; these issues will be the topic of our future work.

4.5 Summary of the Chapter

In this chapter, we have described an analytical model for performance evaluation of a highly virtualized cloud computing center with additional correction for performance

deterioration under heavy workload. More specifically, we permitted each PM to run up to 200 VMs, just like state-of-the-art PMs, and considered the effects of virtualization on PMs performance based on the real data.

Our findings show that in virtualized cloud centers which allow requests with widely varying service times may experience longer waiting time and lower probability of getting immediate service for its clients. Through a simple partitioning-based homogenization in which super-tasks with similar characteristics are grouped into separate streams and serviced by separate sub-clouds, is shown to provide superior performance with respect to the case where all requests are serviced by a single cloud center. Partitioning may be done on the basis of super-task size and/or coefficient of variation of task service times. In this manner, cloud customers may obtain better service from cloud centers specifically geared to handling their type of services, and they may expect generally better performance if their request are submitted in super-tasks with the minimum possible number of required PMs.

Chapter 5

Interacting Fine-Grained Performance Model

In Chapter 1, we stated the reasons for assuming general distribution for task service time. We have developed the monolithic abstract model (Chapters 2, 3 and 4) in which generally distributed service time of tasks was assumed and desired performance indicators have been obtained. Although service time and waiting time in queue are the dominant components of response time, other delays attributed to provisioning and servicing steps may not be negligible. Therefore, we are going to take into account other types of delays imposed on user's tasks by cloud centers. In other words, we are going to break down a *facility node* (described in Chapter 1) in order to increase the scope of our analytical model. By exploring the facility nodes (i.e., breaking down the response time) one monolithic model may not capture the whole cloud behavior due to large scale and high complexity associated with the cloud computing architecture. Therefore we construct a fine-grained performance model that includes distinct sub-models. Each sub-model is designed for dif-

ferent provisioning and servicing steps of a complex cloud service; the overall solution is obtained by integrating individual sub-model solutions [44]. Later on we establish an availability model on top of the fine-grained performance model to calculate effective performance metrics [48].

This Chapter is organized as follows: in Section 5.1, an introduction to the virtualized cloud center is presented. Section 5.2 highlights the related works in which an analytical model has been proposed for virtualized cloud center. The integrated pure performance model has been elaborated in Section 5.4. In Section 5.5, we present the numerical validation results of performance model. The availability model and analytical results of end-to-end performance model have been presented in Sections 5.7 and 5.8 respectively. Section 5.9 summarizes the Chapter.

5.1 Introduction

IaaS cloud providers, such as Amazon EC2 [3], IBM Cloud [36], GoGrid [29], NephoScale [70], Rackspace [75] and others, deliver, on-demand, operating system (OS) instances provisioning computational resources in the form of virtual machines deployed in the cloud provider data center.

Quantifying and characterizing performance measures in a virtualized environment requires appropriate modeling; the model ought to cover vast parameter space while it is still tractable. A monolithic model may suffer from intractability and poor scalability due to large number of parameters [63].

Instead, in this Chapter we develop and evaluate tractable functional sub-models and their interaction model while iteratively solve them. We construct separate sub-models for

different servicing steps in a complex cloud center and the overall solution obtain by iteration over individual sub-model solutions [28]. We assume that the cloud center consists of a number of Physical Machines (PM) that are allocated to users in the order of task arrivals. More specifically, user requests may share a PM using virtualization technique. Since many of the large cloud centers employ virtualization to provide the required resources such as PMs [22], we consider PMs with a high degree of virtualization. Real cloud providers offer complex requests for their users. For instance, in Amazon EC2, the user is allowed to run up to 20 On-Demand or Reserved Instances, and up to 100 Spot Instances per region [3]. Such complex requests are equal to *super-tasks* in our context. Our proposed model embraces realistic aspects and provides deep insight into performance analysis of today's cloud centers. The main features of our analytical model can be listed as follows:

- Assumes Poisson arrival of user requests;
- Incorporates complex user requests by introducing super-tasks;
- Supports high degree of virtualization;
- Captures different delays imposed by cloud centers on user requests;
- Characterizes the service availability at cloud center;
- Provides information for capacity planning;
- Offers tractability and scalability;

In the following sections, we survey existing literature in above mentioned areas and elaborate our future work that we are envisioning at this point of time.

5.2 Related Work

Virtualization technology has been adopted to cope with several problems in current cloud centers such a hardware under-utilization, data center space shortage, and high system administration costs. However, such virtualization techniques complicated the performance evaluation of cloud centers. Recently a few performance models have been developed and they differ in:

(a) which aspects of a real system are or are not captured in the model (e.g. distribution of delays (e.g. service time, inter-arrival time, installation, start up and etc.), communication, failure and etc).

(b) how these aspects are modeled.

In the following text we describe the existing models:

Many research works carried out a measurement-based performance evaluation of the Amazon EC2 [3], IBM Blue Cloud [36] or other cloud providers in the context of scientific computing [37, 108, 15, 73, 98, 100, 80, 1]. Here we survey those that proposed a general analytical model for performance evaluation of virtualized cloud centers.

The performance of cloud computing services for scientific computing workloads was examined in [37]. The authors quantified the presence of Many-Task Computing (MTC) users in real scientific computing workloads. Then, they performed an empirical evaluation of the performance of four commercial virtualized cloud computing services including Amazon EC2. Also, the authors compared the performance characteristics and cost models of clouds and other scientific computing platforms, for general and MTC-based scientific computing workloads.

Kim et al. [52] presented an analytic approach for availability model for virtualized and non-virtualized systems using a hierarchical analytic model in which a fault tree is used in the upper level and CTMC are used in the lower level. They incorporate hardware failures, virtual machine monitor (VMM) failures, VM failures, and application failures. Authors evaluate system availability, downtime in minutes per year, and capacity oriented availability using some experimental data and guesstimates.

A technique that can increase availability of application servers through the use of virtualization and clustering is proposed in [90]. Using analytical modeling via CTMCs authors analyze multiple design choices when a single physical server and dual physical servers are used to host multiple virtual machines.

Ghosh et al. [28] proposed a general analytic model based approach for an end-to-end performance analysis of a cloud service. They illustrated their approach using IaaS cloud, where service availability and provisioning response delays are two key QoS metrics. The proposed approach reduces the complexity of performance analysis of cloud centers by dividing the overall model into sub-models and then obtaining the overall solution by iteration over individual sub-model solutions. The model is limited to single request arrivals, which is not quite realistic; cloud users may ask for multiple PMs or VMs by submitting in a single request. In addition, the effect of virtualization has not been reflected explicitly in their results.

Ghosh et al. [26] quantified the resiliency of IaaS cloud with respect to changes in demand and available capacity. Using a stochastic reward net based model for provisioning and servicing requests in a IaaS cloud, they quantified the resiliency of IaaS cloud with respect to two key performance measures, namely, task rejection rate and total response

delay. In this work, only, the effect of abrupt changes in arrival process and system capacity (i.e., physical machines) have been studied on mentioned performance metrics.

An analytical model for availability analysis of large scale IaaS cloud is proposed in [63]. To reduce the complexity of analysis, the authors employ the same decomposition technique as [28] using stochastic reward net (SRN), a high level Petri net, in order to facilitate construction and simulation-based solution of Markov chains. Dependencies among the sub-models are resolved using fixed-point iteration technique. The authors also develop a monolithic model and show that the error introduced by interactive sub-models is negligible.

5.3 Analytical Model

In IaaS cloud, when a request is processed, a pre-built or customized disk image is used to create one or more VM instances [26]. In this work, we assume that pre-built images fulfill all user requests. We assume that Physical Machines (PMs) are categorized into three server pools: hot (i.e., with running VMs), warm (i.e., turned on but without running VM) and cold (i.e., turned off) [28]. Such categorization reduces operational costs and offers the capability for disaster recovery [3, 51]. Each PM has up to two instantiation units that configure and deploy VMs on a PM. Instantiation of a VM from an image and provisioning it on hot PMs has the minimum delay compared to warm and cold PMs. Warm PMs require more time to be ready for provisioning and cold PMs require additional time to be started up before a VM instantiation. In this work we allow users to request more than one VM by submitting a *super-task* [43]. So, a super-task may contain more than one task, each of which requires one VM. The size of super-task is assumed to be geometrically distributed.

However, for practical reason we set a maximum size of super-tasks (MSS) to conduct the numerical analysis (i.e., truncated geometric distribution). Therefore the probability of having a super-task with size i is

$$p_i = \begin{cases} (1-p)^{i-1}p, & \text{if } i < MSS \\ \\ (1-p)^{MSS-1}, & \text{if } i = MSS \end{cases} \tag{5.1}$$

in which p is the probability of success in geometric distribution.

Due to high interaction among tasks within a super-task, each super-task will be provisioned on the same PM. However, the number of tasks (VMs) requested by a single super-task cannot exceed the maximum number of VMs that can run on a single PM. In this manner the inter-tasks communication overhead among tasks within a given super-task will be decreased significantly. In this work, we assume that all PMs and VMs are homogeneous and adopt *total acceptance policy* [86]. Under this policy the system tries to service all tasks within a super-task at the same time; provided that partial admission of a super-task is not considered. User requests (super-tasks) are submitted to a global finite queue and then processed on the first-in, first-out basis (FIFO).

Resource Assigning Module (RAM) processes the super-task at the head of global queue as follows: first, it tries to provision the super-task on a PM machine in the hot pool. If the process is not successful then RAM tries the warm pool and finally the cold pool. Ultimately, RAM either assigns a PM in one of the pools to a super-task or the super-task gets rejected. As a result, user requests (super-tasks) may get rejected either due to lack of space in global input queue or insufficient resource at the cloud center. When a running task finishes, the capacity used by that VM is released and becomes available for

servicing the next task.

Table 5.1 describes the symbols and acronyms that we use in next sections.

Table 5.1: Symbols and corresponding descriptions

Symbol	**Description**
RAM	Resource Assigning Module
VMPM	Virtual Machine Provisioning Module
RASM	Resource Allocation Sub-Model
VMPSM	Virtual Machine Provisioning Sub-Model
MSS	Maximum Super-task Size
p_i	Probability of having a super-task with size i
λ_{st}	Mean arrival rate of super-tasks
L_q	Size of global queue
P_h, P_w, P_c	Prob. of success in hot, warm and cold pool
N_h, N_w, N_c	Number of PMs in hot, warm and cold pool
ϕ_h, ϕ_w, ϕ_c	Instantiation rate of VMs in hot, warm or cold pool
$\lambda_h, \lambda_w, \lambda_c$	Arrival rate to a PM in hot, warm and cold pool
$\alpha_h, \alpha_w, \alpha_c$	Look up rate in hot, warm and cold pool
BP_q	Prob. of blocking due to lack of room in global queue
BP_r	Prob. of blocking due to lack of capacity
P_{reject}	Total probability of blocking ($BP_q + BP_r$)
$\overline{wt}$	Mean waiting time in global queue
$\overline{lut}$	Mean look-up delay among pools
$\overline{wt_{PM}}$	Mean waiting time in a PM queue
$\overline{pt}$	Mean VM provisioning time
$\overline{td}$	Mean total delay for a task before getting into service

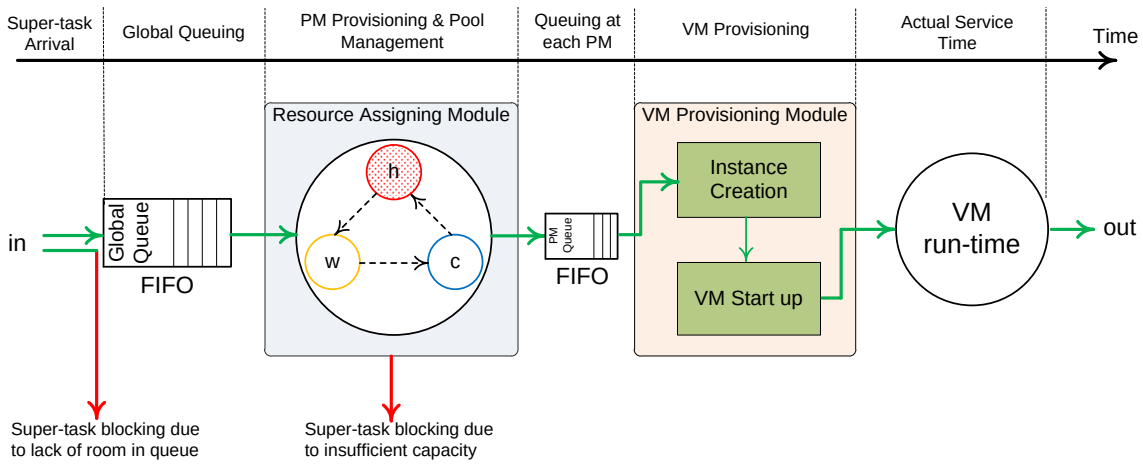


Figure 5.1: The steps of servicing and corresponding delays.

We identify four main delays imposed on user requests (Fig. 5.1): queuing delay in the global queue; look-up time at the RAM; queuing delay at the chosen PM; and VM provisioning delay. The response time is the sum of the sum of these delays and the actual service time. In order to model each module precisely, we design two stochastic sub-models, captures the details of the RAM and VMPM modules. We then connect these sub-models into an overall model to compute the cloud performance metrics: task rejection probability and mean response delay. Finally, we solve these sub-models and show how performance metrics are affected by variations in workload (super-task arrival rate, super-task size, task mean service time, number of VMs per PM) and system capacity (i.e., number of PMs in each pool). We describe our analysis in the following sections.

5.4 Integrated Stochastic Models

As mentioned in Section 5.2, due to high complexity of cloud centers, the monolithic models were hard to analyze and extend. By introducing interacting analytical sub models, the complexity of system is divided into sub-models so that they can be analyzed accurately in order of processing of task components. Moreover, the sub-models are scalable enough to capture the flexibility (on-demand services) of the cloud computing paradigm. In this Chapter, we implement the sub-models using interactive Continuous Time Markov Chain (CTMC). The sub-models are interactive such that the output of one sub-model is input to the other ones and vice versa.

5.4.1 Resource Allocation Sub-Model

The resource allocation process is described in the resource allocation sub-model (RASM) shown in Fig. 5.2. RASM is a two dimensional CTMC in which we take care of number of super-task in the queue as well as the current pool on which provisioning is taking place. Since the number of potential users is high, but each user submits super-tasks one at a time with low probability, the super-task arrival can be modeled as a Poisson process [30] with rate λ_{st}. Super-tasks are lined up in the global finite queue to be processed in a first-in, first-out (FIFO) basis. Each state of Markov chain is labeled as (i, j), where i indicates the number of super-tasks in queue and j denotes the pool on which the leading super-task is under provisioning. State $(0, 0)$ indicates that system is empty which means there is no request under provisioning or in the queue. Index j can be h, w or c that indicate current super-task is undergoing provisioning on hot, warm or cold pool respectively. The global queue size is L_q and one more super-task can be at the deployment unit for provisioning so

the total system capacity is $L_q + 1$.

Let P_h, P_w and P_c be the success probabilities of finding a PM that can accept the current super-task in hot, warm and cold pool respectively. We assume that $1/\alpha_h$, $1/\alpha_w$ and $1/\alpha_c$ are the mean look-up delays for finding an appropriate PM in hot, warm and cold pool respectively. Upon arrival of first super-task, system moves to state $(0, h)$ which means the super-task will be provisioned immediately in the hot pool. Afterwards, depending on the upcoming event, three possible transitions can occur:

(a) Another super-task has arrived and system transits to state $(1, h)$ with rate λ_{st}.

(b) A PM in hot pool accepts the super-task so that system moves back to state $(0, 0)$ with rate $P_h\alpha_h$.

(c) None of PMs in hot pool has enough capacity to accept the super-task, so the system will examine the warm pool (i.e., transit to state $(0, w)$) with rate $(1 - P_h)\alpha_h$.

On state $(0, w)$, RASM tries to provision the super-task on warm pool; if one of the PMs in warm pool can accommodate the super-task, the system will get back to $(0, 0)$, otherwise, RASM examines the cold pool (i.e., transition from state $(0, w)$ to $(0, c)$). If none of the PMs in cold pool can provision the super-task as well, the system moves back from $(0, c)$ to $(0, 0)$ with rate $(1 - P_c)\alpha_c$ which means the user request (super-task) will get rejected due to insufficient resources in the cloud center. During the provisioning in cold pool, another super-task may arrive and takes the system to state $(1, c)$ which means there is one super-task in deployment unit and one awaiting in global queue. Finally, the super-task under provisioning decision leaves the deployment unit, once it receives a decision from RASM and the next super-task at the head of global queue will go under provisioning. In

this sub-model, arrival rate of super-task (λ_{st}) and look-up delays ($1/\alpha_h, 1/\alpha_w$ and $1/\alpha_c$) are exogenous parameters and success probabilities (P_h, P_w and P_c) are calculated from the VM provisioning sub-model. The VM provisioning sub-model will be discussed in the next Section.

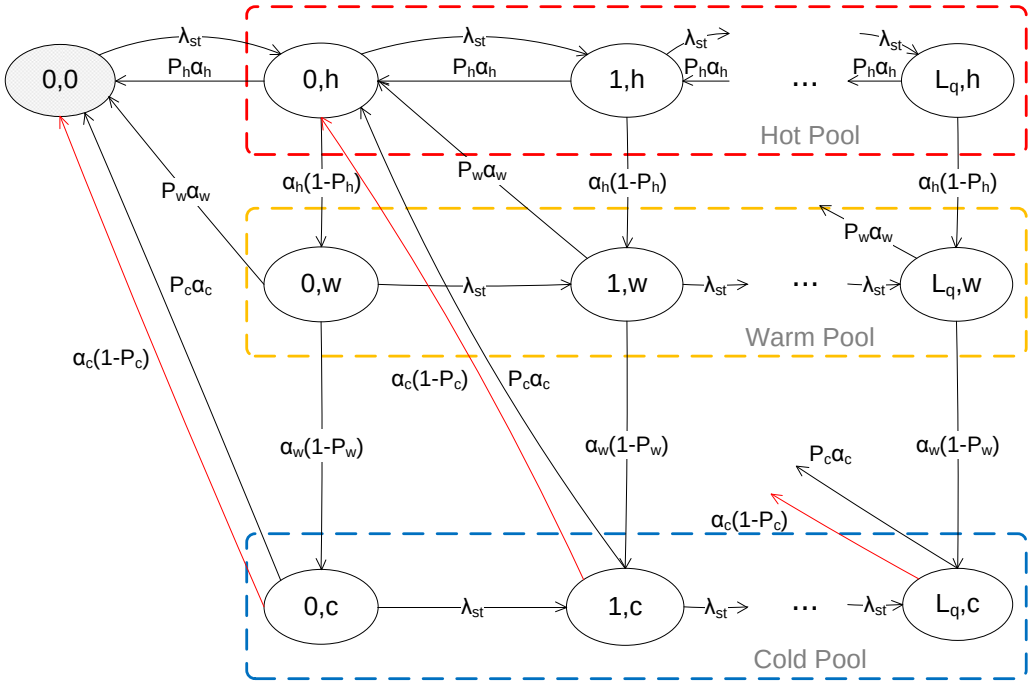


Figure 5.2: Resource allocation sub-model.

Using steady-state probabilities $\pi_{(i,j)}$, some performance metrics such as blocking probability and probability of immediate service can be calculated. Two types of blocking may happen to a given super-task:

(a) Blocking due to a full global queue occurs with the probability of

$$BP_q = \pi_{(L_q,h)} + \pi_{(L_q,w)} + \pi_{(L_q,c)} \tag{5.2}$$

(b) Blocking due to insufficient resources (PMs) at pools [34], with the probability of

$$BP_r = \sum_{i=0}^{L_q} \frac{\alpha_c(1-P_c)}{\alpha_c + \lambda_{st}} \pi_{(i,c)} \tag{5.3}$$

The probability of reject (P_{reject}) is, then, $P_{reject} = BP_q + BP_r$. In order to calculate the mean waiting time in queue, we first establish the probability generating function (PGF) for the number of super-tasks in the queue [85],

$$Q(z) = \pi_{(0,0)} + \sum_{i=0}^{L_q} (\pi_{(i,h)} + \pi_{(i,w)} + \pi_{(i,c)}) z^i \tag{5.4}$$

The mean number of super-tasks in queue ($\overline{q}$) is the first derivative of $Q(z)$ at $z = 1$ [86],

$$\overline{q} = Q'(1) \tag{5.5}$$

Applying Little's law [57], the mean waiting time in queue ($\overline{wt}$) is given by:

$$\overline{wt} = \frac{\overline{q}}{\lambda_{st}(1 - P_{reject})} \tag{5.6}$$

Look-up time among pools can be considered as a Coxian distribution with 3 steps (Fig. 5.3). So it can be calculated as [57, 93],

$$\overline{lut} = \frac{1/\alpha_h + (1-P_h)((1/\alpha_w) + (1-P_w)(1/\alpha_c))}{1 - BP_q} \tag{5.7}$$

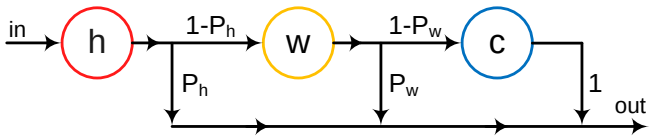


Figure 5.3: Three steps of look-up delays among pools.

5.4.2 VM Provisioning Sub-Model

Virtual Machine Provisioning Sub-Model (VMPSM) captures the instantiation, deployment and provisioning of VMs on a PM. VMPSM also incorporate the actual servicing of each task (VM) on a PM. Note that each task within super-task is provisioned with an individual VM. However RASM makes sure that all tasks within a super-task get provisioned at the same PM. In this model, we assume homogeneous PMs, VMs and tasks. As can be seen from Fig. 5.3, the look-up time for finding a proper PM among pools has Coxian distribution and the look-up time is a weighted sum of an exponentially distributed random variable with the probability of P_h, a 2-stage Erlang distributed random variable with the probability of $(1 - P_h)P_w$, and a 3-stage Erlang distributed random variable with the probability of $(1 - P_h)(1 - P_w)P_c$. Hence, the LST of look-up time can be calculated as:

$$\begin{aligned} B^*(s) &= P_h(\frac{\alpha_h}{s+\alpha_h}) + P_w(1-P_h)(\frac{\alpha_h}{s+\alpha_h})(\frac{\alpha_w}{s+\alpha_w}) \\ &+ (1-P_h)(1-P_w)(\frac{\alpha_h}{s+\alpha_h})(\frac{\alpha_w}{s+\alpha_w})(\frac{\alpha_c}{s+\alpha_c}) \end{aligned} \tag{5.8}$$

which allows us to calculate the mean, standard deviation and coefficient of variation (CoV) of the look-up time at the PM pools. As RASM is not an M/M/1 queuing system, the output process (which is also the arrival process to the PMs) is not Poisson [19] so, strictly speaking, the VMPSM cannot be modeled with a CTMC [57]. However, if the CoV of the arrival process to the PMs is not significantly lower than 1, we can approximate it with a Poisson process. This assumption, which will be justified in Section 5.5, allows us to model the process with sufficient accuracy. Using Eq. (5.8), we are able to calculate the mean, standard deviation and coefficient of variation (CoV) of the look-up time at pools. Such measures help us to verify the accuracy of the approximation. We will present and discuss this issue with further details in the numerical validation Section.

Fig. 5.4 shows the VMPSM (an approximated CTMC) for a PM in hot pool. A PM in warm or cold pool can be modeled with the same VMPSM, though, with different arrival and instantiation rate. Consequently, each pool (hot, warm and cold) can be modeled as a set of VMPSM with the same arrival and instantiation rate. Each state in Fig. 5.4 is labeled by (i, j, k) in which i indicates the number of tasks in PM's queue, j denotes the number of task that is under provisioning and k is the number of VM that are already deployed on the PM. Note that we set the queue size at each PM equal to the maximum number of VMs that can be deployed on a PM. Let ϕ_h be the rate at which a VM can be deployed on a PM at hot pool and μ be the service rate of each VM. So, the total service rate for each PM is the product of number of running VMs by μ. State $(0, 0, 0)$ indicates that the PM is empty and there is no task either in the queue or in the instantiation unit. Depending on the size of arriving super-task, model transits to one of the states in $\{(0, 1, 0), (1, 1, 0), \cdots, (MSS - 1, 1, 0)\}$, in which MSS is the maximum possible size of super-tasks. Since all tasks within a super-task are to be provisioned at the same PM, the maximum number of VMs on a PM must be at least equal to MSS. The arrival rate to each PM in the hot pool is given by:

$$\lambda_h = \frac{\lambda_{st}(1 - BP_q)}{N_h} \tag{5.9}$$

in which N_h is the number of PMs in the hot pool. Note that BP_q, used in (5.2), is obtained from RASM. In Fig. 5.4, $\{\lambda_i, i = 1..MSS\}$ is given by $\lambda_i = \lambda_h p_i$; here p_i is the probability of having a super-task of size i. In this work, the size of super-task can be generally distributed, but we use truncated geometric distribution for numerical experiment. The state transition in VMPSM can occur due to super-task arrival, task instantiation or service completion. From state $(0, 0, 0)$, system can move to state $(1, 1, 0)$ with rate λ_2 (i.e., super-

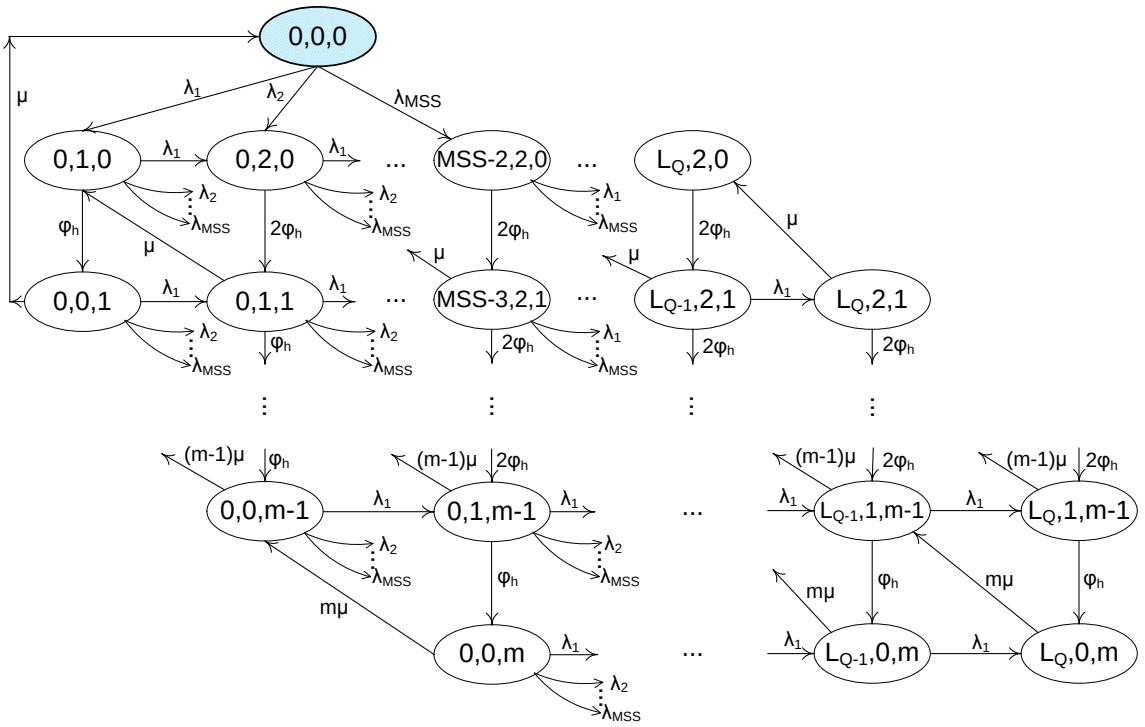


Figure 5.4: Virtual machine provisioning sub-model for a PM in the hot pool.

task with size 2). From $(1, 1, 0)$, system can transit to $(0, 1, 1)$ with rate ϕ_h (i.e., instantiation rate) or upon arriving a super-task with size k, moves to state $(k + 1, 1, 0)$. From $(0, 1, 1)$, system can move to $(0, 0, 2)$ with rate ϕ_h, transits to $(0, 1, 0)$ with rate μ (i.e., service completion rate), or again upon arriving a super-task with size k, system can move to state $(k, 1, 1)$ with rate λ_k. A PM can not accept a super-task, if there is no enough room to accommodate all tasks within super-task. Suppose that $\pi^h_{(i,j,k)}$ is the steady-state probability for the hot PM model (Fig. 5.4) to be in the state (i, j, k). Using steady-state probabilities, we can obtain the probability that at least one PM in hot pool can accept the super-task for provisioning. First we need to compute the probability that a hot PM cannot admit a super-task for provisioning (P^h_{na}). Let m be the maximum possible number of VMs in a

PM and F_h be the free capacity at each state:

$$F_h(i,j,k) = m - (i+j+k)$$

P_{na}^h is then given by:

$$P_{na}^h = \sum_{x \in \xi} \sum_{t=F_h+1}^{MSS} \pi_x^h p_t \tag{5.10}$$

where p_t is the probability of having a super-task of size t and ξ is a subset of VMPSM states with the following definition:

$$\xi = \{(i,j,k) |\ F_h(i,j,k) < MSS\}$$

Therefore, probability of success provisioning (P_h) in the hot pool can be obtained as

$$P_h = 1 - (P_{na}^h)^{N_h} \tag{5.11}$$

Note that P_h is used as an input parameter in the resource allocation sub-model (Fig. 5.2). The provisioning model for warm PM is the same with the one for hot PM, though, there are some differences in parameters:

(a) The arrival rate to each PM in warm pool is

$$\lambda_w = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)}{N_w} \tag{5.12}$$

where N_w is the number of PMs in warm pool.

(b) Every PM in warm pool requires extra time to be ready (hot) for first instantiation. This time is assumed to be exponentially distributed with mean value of γ_w.

Instantiation rate in warm pool is the same as in hot pool (ϕ_h). Like VMPSM for a hot PM (Fig. 5.4), the model for a warm PM is solved and the steady-states probabilities ($\pi_{(i,j,k)}^w$),

are obtained. The success probability for provisioning a super-task in warm pool is:

$$P_w = 1 - (P_{na}^w)^{N_w} \tag{5.13}$$

A PM in the cold pool can be modeled as in the warm and hot pool but with different arrival rate and start up time. The effective arrival rate at each PM in the cold pool is given by:

$$\lambda_c = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)(1 - P_w)}{N_c} \tag{5.14}$$

in which N_c is the number of PMs in the cold pool. A cold PM (off machine), requires longer time than a warm PM to be ready (hot) for the first instantiation. We assume that the start up time is also exponentially distributed with mean value of γ_c. The success probability for provisioning a super-task in the cold pool is given by:

$$P_c = 1 - (P_{na}^c)^{N_c} \tag{5.15}$$

From VMPSM, we can also obtain the mean waiting time at PM queue ($\overline{wt_{PM}}$) and mean provisioning time ($\overline{pt}$) by using the same approach as the one that led to Eq. (5.6). As a result, the total delay before starting of actual service time, is given by:

$$\overline{td} = \overline{wt} + \overline{lut} + \overline{wt_{PM}} + \overline{pt} \tag{5.16}$$

Note that all success probabilities (i.e., P_h, P_w and P_c) are dependent on the arrangement (i.e., number of hot, warm and cold PMs) of pools.

5.4.3 Interaction among Sub-Models

The interactions among sub-models is depicted in Fig. 5.5. VM provisioning sub-models (VMPSM) compute the steady state probabilities (P_h, P_w and P_c) that at least one

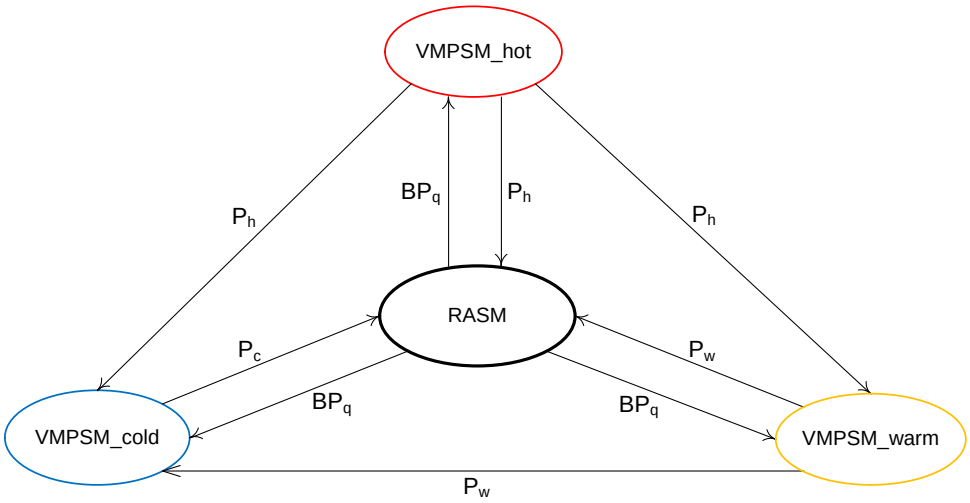


Figure 5.5: Interaction diagram among sub-models.

PM in a pool (hot, warm and cold, respectively) can accept a super-task for provisioning. These probabilities are used as input parameters to the resource allocation sub-model (RASM). Hot PM sub-model (VMPSM_hot) computes P_h which is the input parameter for both warm and cold sub-models and warm PM sub-model (VMPSM_warm) computes P_w which is the input parameter for the cold sub-model (VMPSM_cold). The resource allocation sub-model compute the blocking probability, BP_q, which is the input parameter to VM provisioning sub-models. As can be seen, there is an inter-dependency among sub-models. This cyclic dependency is resolved via fixed-point iterative method [65] using a modified version of successive substitution approach (Algorithm 1).

5.5 Numerical Validation - Pure Performance Model

The resulting sub-models have been implemented and solved using Maple 15 from Maplesoft, Inc. [66]. The successive substitution method (Algorithm 1) is continued until

the difference between the previous and current value of blocking probability in global queue (i.e., BP_q) is less than 10^{-6}. Usually the integrated model converges to the solution in less than 15 iterations. To validate the analytical solution, we have also built a discrete event simulator of the cloud center using the Artifex engine [79].

Under different configurations and parameter settings, we obtain two important performance metrics, namely, rejection probability of super-tasks (STs) and total response delay. Mean service time of each task within STs ranges form 20 to 140 minutes. However, it should be noted that, beyond a certain degree of virtualization (i.e., number of deployed VMs), the actual service times will increase due to the overhead required to run several VMs concurrently. To model this effect, we have used publicly available data about server performance [6]. The continuous line in Fig. 5.6 shows the VMmark performance for a family of advanced servers under varying degree of virtualization [96], including both computation and communication time; from this data, we have extracted the normalized response time, shown by circles, which has subsequently been used in our experiments. Mean service time represents both computation and communication time for a task within super-tasks. Identifying higher moments of service time requires more data or benchmarks from cloud providers [12].

Arrival rate ranges from 1 to 7 STs per minute and global queue size is set to 50 STs. We assume 15 to 20 PMs in each pool and each PM can run up to 10 VMs simultaneously. The look-up time for finding a proper PM is assumed to be independent of the number PMs in the pool and the type of pool (i.e., hot, warm and cold). Look-up rate is set to 10 searches per minute. Mean preparing delay for a warm and cold PM to become a hot PM (i.e., be ready for first VM instantiation) are assumed to be 1 to 3 and 5 to 10 minutes,

Algorithm 1 Successive Substitution Method

Input: Initial success probabilities in pools: P_{h0}, P_{w0}, P_{c0}

Output: Blocking probability in Global Queue: BP_q

counter ← 0; max ← 30; diff ← 1

BP_{q0} ← RASM (P_{h0}, P_{w0}, P_{c0})

while diff $\geq 10^{-6}$ **do**

counter ← counter + 1

P_{h1} ← VMPSM_hot (BP_{q0})

P_{w1} ← VMPSM_warm (BP_{q0},P_{h1})

P_{c1} ← VMPSM_cold (BP_{q0},P_{h1},P_{w1})

BP_{q1} ← RASM (P_{h1}, P_{w1}, P_{c1})

$P_{h0} \leftarrow P_{h1}$; $P_{w0} \leftarrow P_{w1}$; $P_{c0} \leftarrow P_{c1}$

diff ← $|(BP_{q1} - BP_{q0})|$

$BP_{q0} \leftarrow BP_{q1}$

if counter = max **then**

break

end if

end while

if counter = max **then**

return -1

else

return BP_{q0}

end if

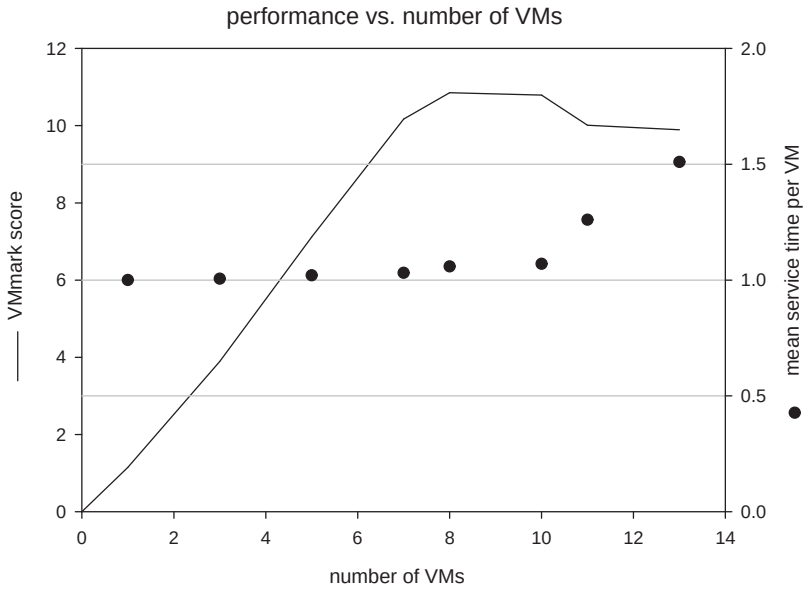
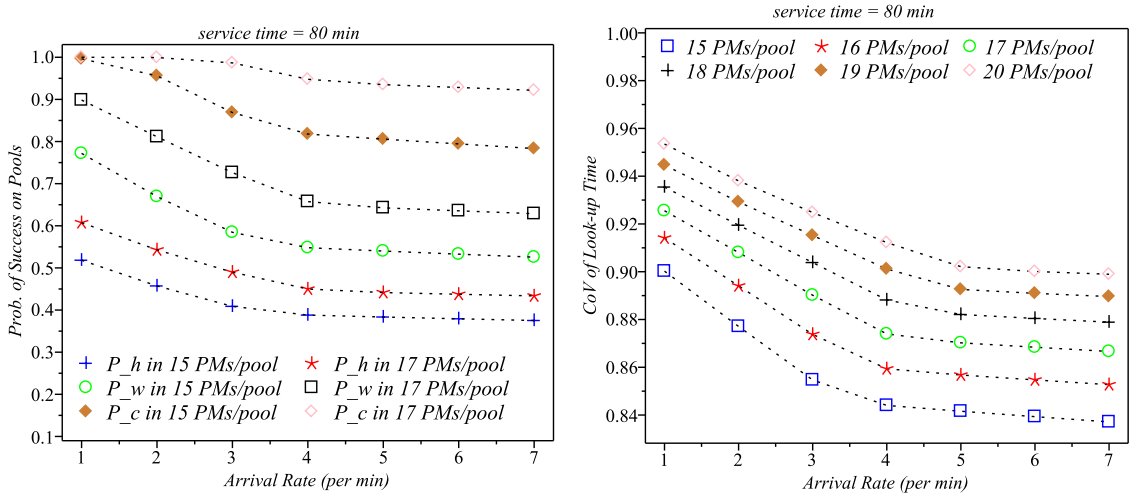


Figure 5.6: VMmark results and normalized mean service time.

respectively. Mean time to deploy a VM on a hot PM is assumed to be between 1 and 10 minutes. First VM instantiation on a warm and cold PM are to be 5 to 20 and 15 to 30 minutes, respectively. After first VM instantiation on a warm or cold PM, it has already become a hot PM so that the next VM can be deployed just like a hot PM. Note that in all plots, analytical model and simulation results are shown by lines and symbols respectively.

First, we characterize the look-up time in pools by presenting the effect of arrival rate on the success probabilities for two configurations (15 and 17 PMs per pool). As can be seen in Fig. 5.7(a), the success probabilities decrease smoothly by increasing the arrival rate. Also, it can be noticed that the curves for P_h, P_w and P_c are equidistant which reveals a linear dependency among them with respect to offered load and pool capacity. Fig. 5.7(b) shows that by increasing the arrival rate, the CoV of look-up time is slightly below one which corresponds to exponential distribution. More specifically, under high traffic

intensity, super-tasks have a lower chance to get provisioned in the hot pool so that RAM needs to check other pools more frequently. Under heavy load, distribution of look-up time will approach Coxian distribution with CoV between 0.84 and 0.90 while under low load it is close to exponential distribution. Overall, these results justify the Poisson approximation of the arrival process to the PMs, as does the degree of match between analytical and simulation results in Figs. 5.8, 5.9 and 5.10.

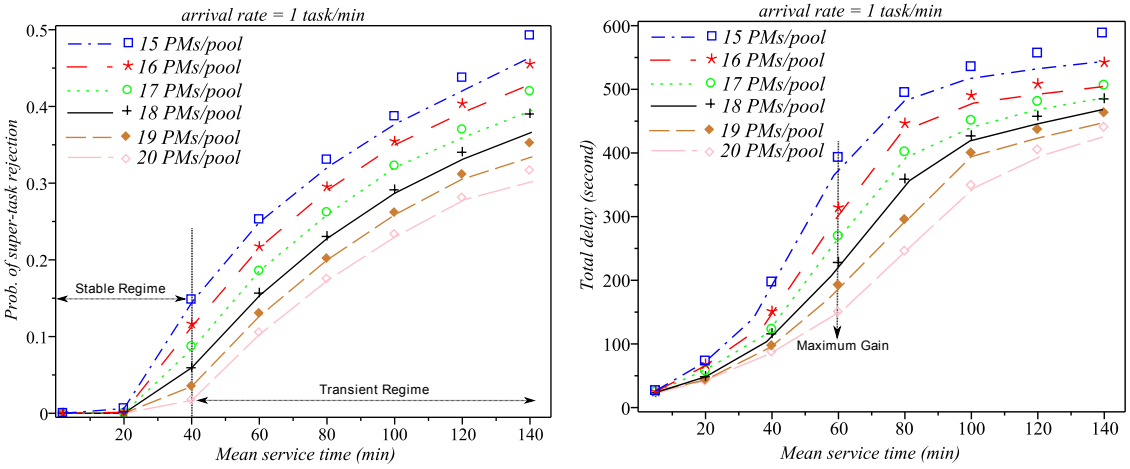


(a) Success probabilities on the pools.

(b) CoV of look-up time on the pools.

Figure 5.7: Look-up time characteristics.

We also analyze the effects of task service time on rejection probability and total delay imposed by the cloud center on STs before getting into service. In the first experiment, the results of which is shown in Fig. 5.8, STs have one task and PMs are permitted to run only one VM. Fig. 5.8(a) shows, at a fixed arrival rate (1 STs/min) and different numbers of PMs in each pool, increasing mean service time, raising the rejection probability linearly. Increasing the system capacity (i.e., the number of PMs in each pool) reduces the rejection

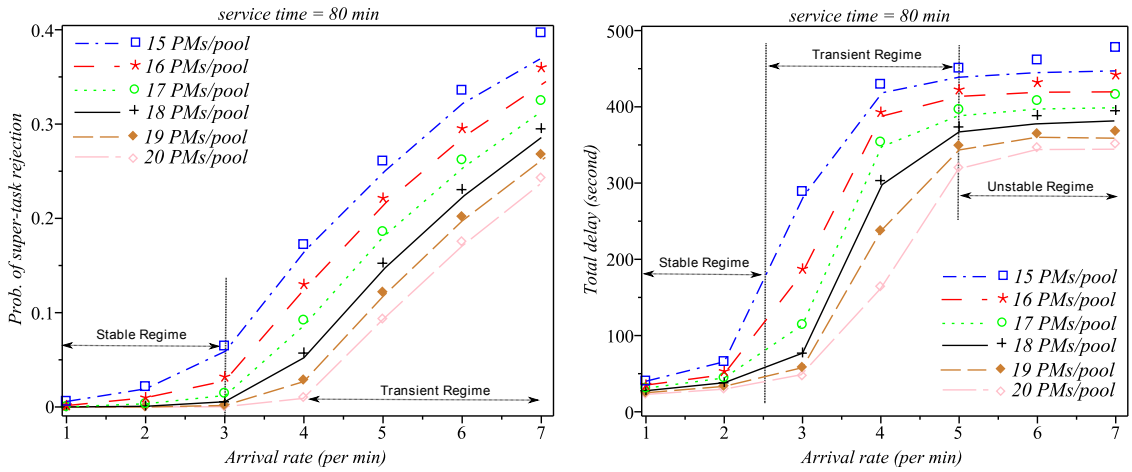


(a) Task Rejection probability.

(b) Total delay for a super-task.

Figure 5.8: Single VM deployed on each PM.

probability. Also, it can be seen that for tasks shorter than 40 min, extending the capacity of system has negligible effect on rejection probability, while for tasks of longer duration (i.e., from 40 to 140 min) rejection probability drops almost linearly. The maximum gain is obtained at 140 minutes. Fig 5.8(b) shows that longer service times will result in longer imposed delays on STs. It can be seen that for any capacity, there is a turning point beyond which total delay increases sharply. For instance when the capacity is 45 PMs (i.e., 15 PMs in each pool), the turning point is 20 minutes while for a capacity of 60 PMs (i.e., 20 PMs in each pool), the turning point is 40 minutes. Cloud providers should operate in the stable regime (i.e., before turning points). They can be alerted by such turning points to add more resources and avoid entering transient regime. In addition, it can be observed that for the given configuration, adding five PMs to each pool (i.e., comparing 15 PMs with 20 PMs at 60 min) can reduce the delay by up to a half.



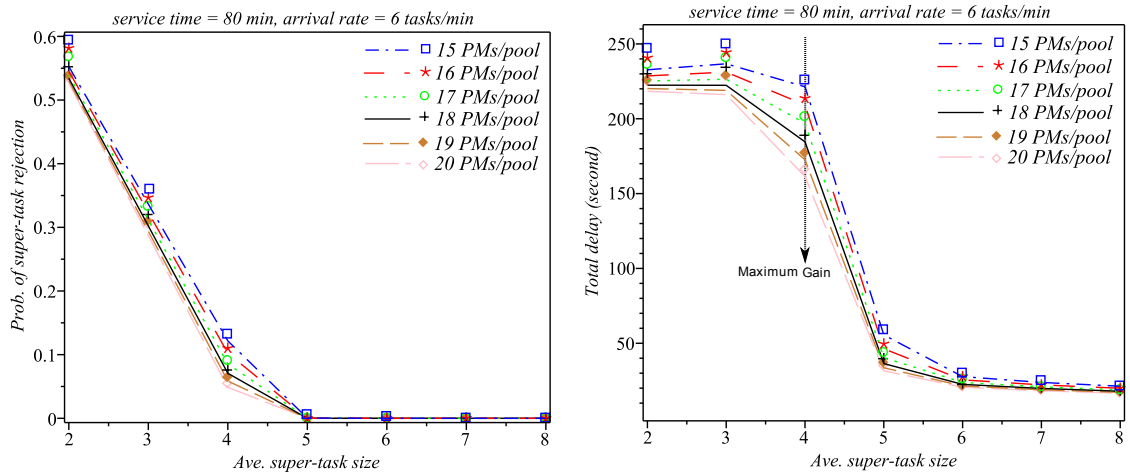
(a) Task Rejection probability.

(b) Total delay for a super-task.

Figure 5.9: Single VM on each PM.

We also calculate the two performance metrics under different arrival rates while fixing the average service time at 80 minutes (Fig. 5.9). One turning point can be noticed in rejection probability curves (Fig. 5.9(a)) at which rejection trend increases suddenly. Such points (e.g., Fig. 5.9(a), 4 STs/min for 20 PMs in each pool) may suggest the appropriate time to upgrade or inject more resources (i.e., PMs) so that prevent of an undesired rejection rate. In case of total delay, two turning points can be identified (Fig. 5.9(b)): after the first one , total delay grows sharply while at the second point increasing the arrival rate almost has no effect (i.e., unstable regime). The transient regime, the range between two turning points, is attributed to the size of global queue. The longer the global queue is, the larger the transient range is going to be which is expected.

In the last experiment, we examine the effect of super-task size on task rejection probability and total delay. Each PM can run up to ten VMs simultaneously and a super-task



(a) Task Rejection probability.

(b) Total delay for a super-task.

Figure 5.10: Ten VMs are allowed on each PM.

may request up to all of the available VMs in a PM. Note that the aggregate arrival rate is set to 6 tasks/hr, however the effective arrival rate of super-tasks is various for different super-task size. For instance, if the average super-task size was 3, then the effective arrival rate of super-tasks would be 2 STs/hr. As can be seen by increasing the size of super-tasks the rejection probability and total delay reduce sharply. The bigger super-task size will result in the lower number of arrivals as well as lower deviation of super-task size. In other words, in case of large super-tasks, they are more likely to be almost in the same size which improves the utilization of the cloud center. Notice that, in this work we adopted total acceptance policy for super-task servicing. Maximum gain is occurred at arrival rate of 4 STs/min for different pool arrangements. Also, it can be seen that the results from analytical model and the simulation are in a good agreement.

5.6 Failure and Repair Effects of Cloud Resources

In this Section, we integrate an availability model with the interacting analytical sub-models that we introduced in the previous sections of this Chapter. The availability model captures the failure/repair effects of cloud resources. Our results can be used by an admission control to prevent the cloud center from entering unstable regime of operation. The results also reveal practical insights into capacity planning for cloud computing centers.

5.7 Availability Model

The availability model deals with the failure of PMs, repairing process and employment of cold PMs in case of failure in running machines. This model realizes by a three-dimensional CTMC in which (i, j, k) indicate the initial number of PMs in the hot, warm and cold pool respectively. Fig. 5.11 depicts an example of availability model when the cloud center has 2, 1 and 2 PMs in the hot, warm and cold pool respectively. The hot and warm PMs can fail with rate θ_h and θ_w. To maintain a certain level of availability, upon failure of a hot PM, one warm PM substitutes immediately. Repair units repair broken PMs and return them to the hot, warm or cold pool. Each repair unit repairs a PM with rate β. If the hot pool is in need of a PM, the repaired PM goes to the hot pool; otherwise it goes to warm and if none of them (i.e., the hot and warm pools) require a PM, the repaired one shots down (i.e., goes to the cold pool). When the number of PMs in either hot or warm pool gets lower than a pre-defined value (e.g., in Fig. 5.11 lower than 2 and 1) cold PMs are fired up to serve the target pool. The mean start up time is assumed to be $1/SU_c$.

Fig. 5.12 represents a general availability model for arbitrary number of PMs in each

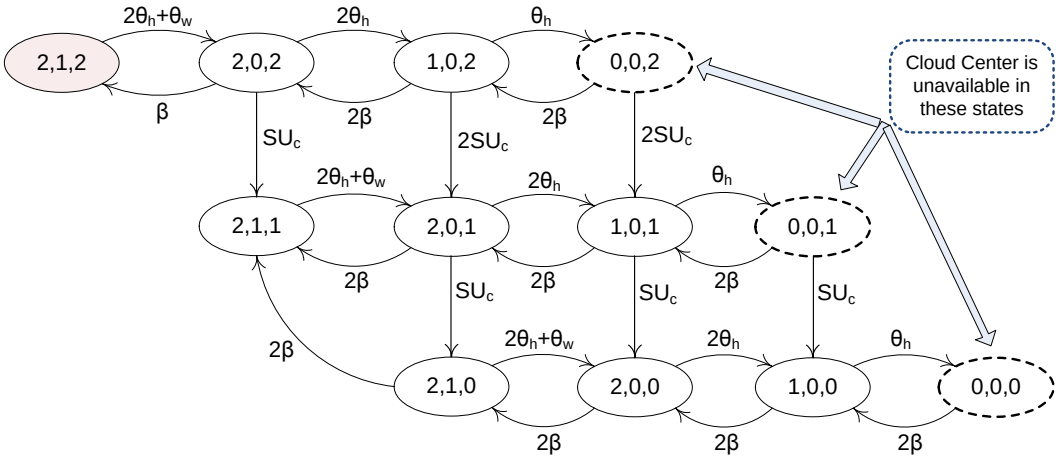


Figure 5.11: The availability model for 5 PMs in the pools.

pool.

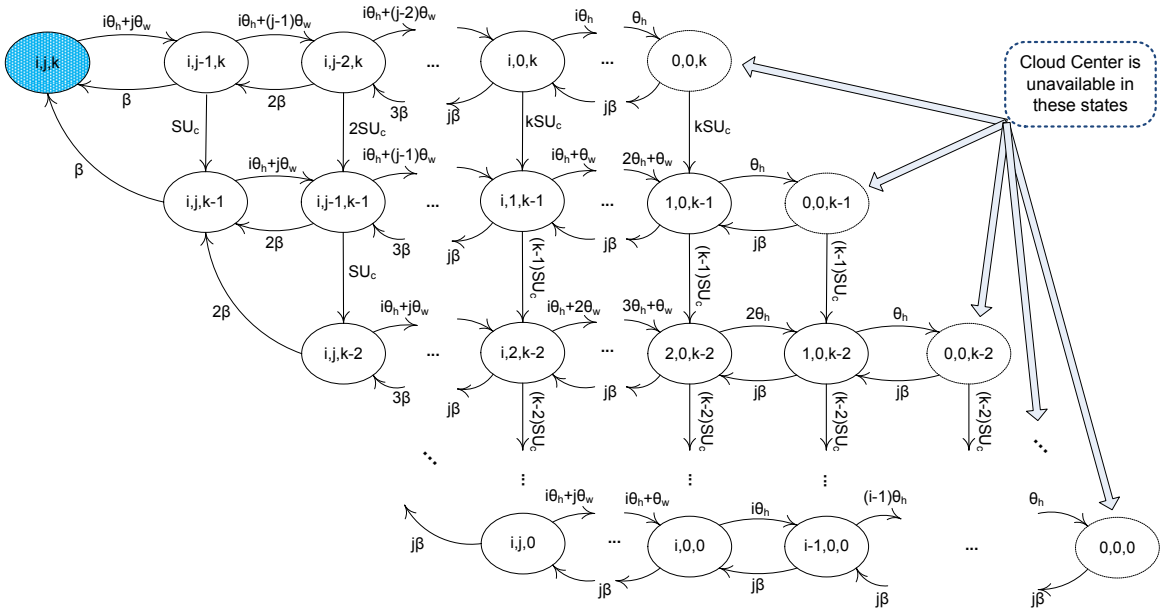


Figure 5.12: General availability model.

5.7.1 Interaction among Sub-Models

The interactions among sub-models is depicted in Fig. 5.13. VM provisioning sub-models (VMPSM) compute the steady state probabilities (P_h, P_w and P_c) that at least one PM in a pool (hot, warm and cold, respectively) can accept a super-task for provisioning. These probabilities are used as input parameters to the resource allocation sub-model (RASM). Hot PM sub-model (VMPSM_hot) computes P_h which is the input parameter for both warm and cold sub-models; warm PM sub-model (VMPSM_warm) computes P_w which is the input parameter for the cold sub-model (VMPSM_cold). The resource allocation sub-model (RASM) computes the blocking probability, BP_q, which is the input parameter to VM provisioning sub-models. Another type of interaction is between performance model (including RASM and VMPSMs) and the availability model. For each state in the availability model the desired statistics are calculated and then finally the effective value of each performance metrics is obtained. As can be seen, there is an inter-dependency among performance sub-models. This cyclic dependency is resolved via fixed-point iterative method [28] using a modified version of successive substitution approach.

5.8 Numerical Validation - Availability Model

The successive substitution method is continued until the difference between the previous and current value of blocking probability in global queue (i.e., BP_q) is less than 10^{-6}. The integrated model converges to the solution in 8 iterations or less.

We show the effects of changing arrival rate, task service time, number of PMs in each pool, number of VMs on each PM and super-task size on the interested performance

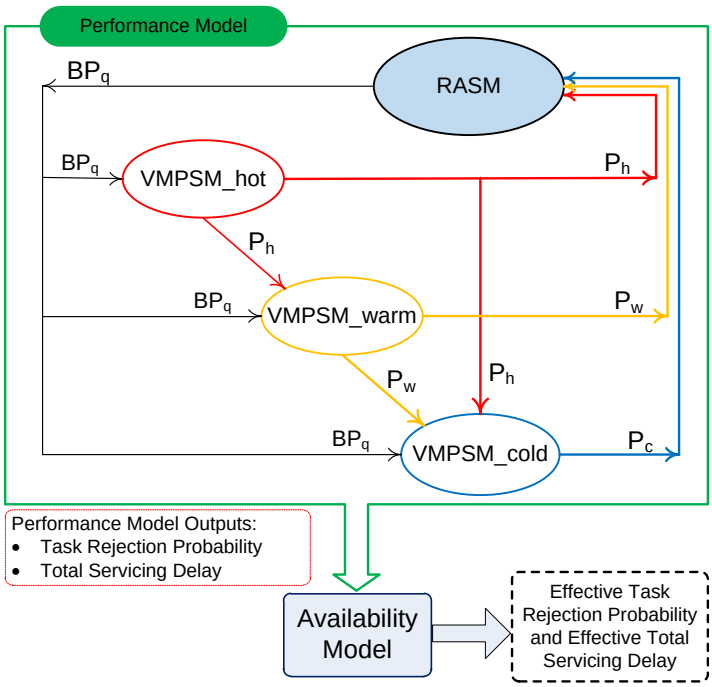
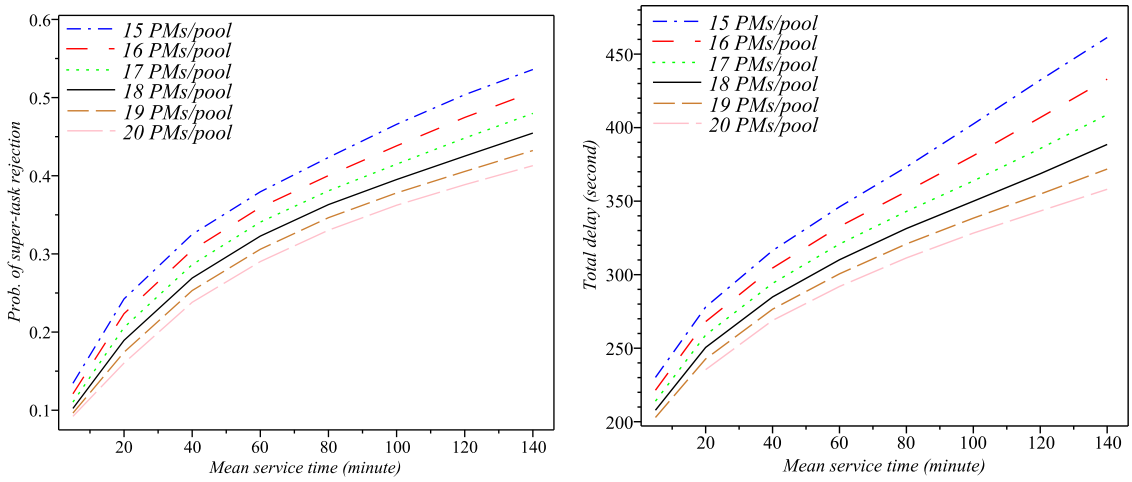


Figure 5.13: Interaction between sub-models.

indicators. Arrival rate ranges from 250 to 600 STs per hour. Mean service time of each task within STs ranges form 20 to 140 minutes. we assume 15 to 20 PMs in each pool and each PM can run 2 VMMs and up to 6 VMs simultaneously. The look-up time for finding a proper PM is assumed to be independent of the number of PMs in the pool and the type of pool (i.e., hot, warm and cold). Look-up rate is set to 825 searches per hour. Mean preparation delay for a warm and cold PM to become a hot PM (i.e., be ready for first VM instantiation) are assumed to be 1 to 3 and 5 to 10 minutes, respectively. Mean time to deploy a VM on a hot PM is assumed to be between 4 and 10 minutes. First VM instantiation on a warm and cold PM are to be between 5 to 20 and 15 to 30 minutes, respectively. After first VM instantiation on a warm or cold PM, it has already become a hot PM so that the next VM can be deployed just like a hot PM. The Mean Time To Failure

(MTTF) for hot and warm PMs ($1/\theta_h$ and $1/\theta_w$) are assumed to be in the range of 20 to 1000 and 100 to 3000 hours, respectively. The number of repair units is set to 5 and the range of 1 to 5 hours assumed for mean repair time ($1/\beta$) of a PM. The size of global queue is set to 50 super-tasks (ST).



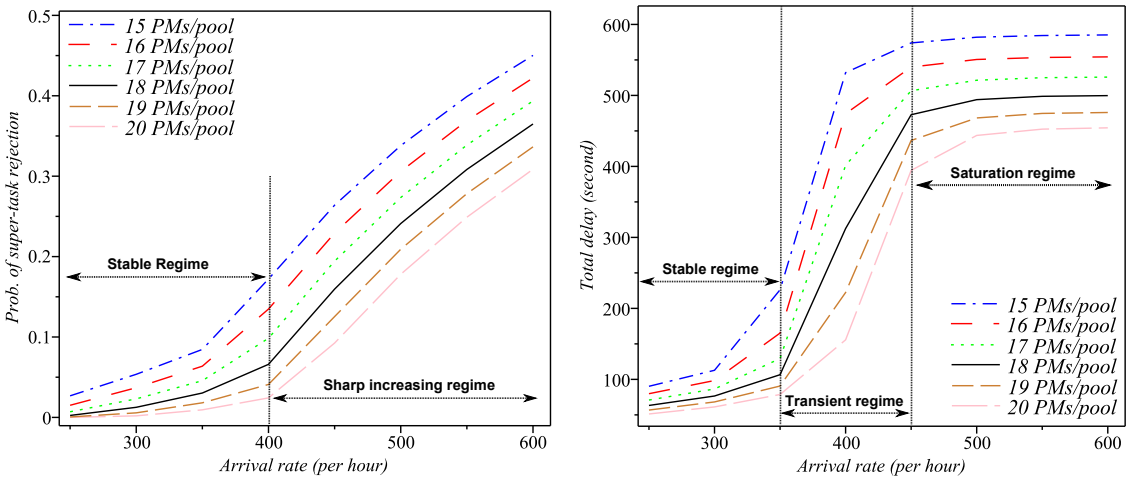
(a) Rejection probability.

(b) Total delay on a super-task.

Figure 5.14: arrival-rate=900 tasks/hour.

In the first experiment, the results of which is shown in Figs. 5.14(a) and 5.14(b), STs have one task and PMs are permitted to run 2 VMMs each of which hosts one VM. Fig. 5.14(a) shows (at a constant arrival rate of 900 STs/hr and different numbers of PMs in each pool) that by increasing the mean service time the rejection probability increases almost linearly. Also, we noticed that curves (for pool capacity 15-20 PMs/pool) in Fig. 5.14(a) are almost equidistant. Fig. 5.14(b) shows that a longer service time will result in a longer total delay on STs. In addition, it can be observed that by increasing the capacity of system (i.e., having more PMs in the pools) the maximum gains occurs at the longest service time

(i.e. 140 minutes).



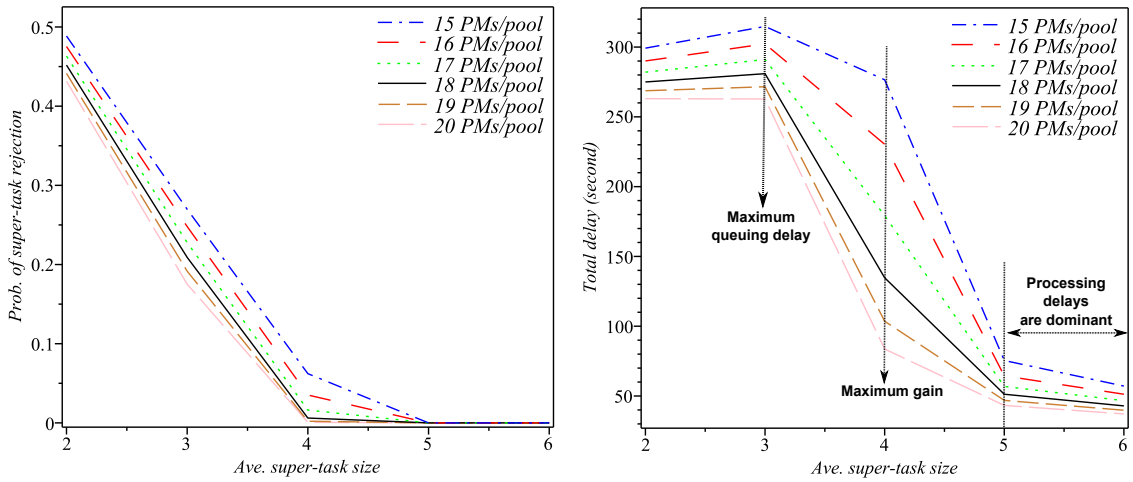
(a) Rejection probability.

(b) Total delay on a super-task.

Figure 5.15: service-time=120 minute.

Under different arrival rates, we also determine the two performance metrics while the average service time is set to 120 min (Figs. 5.15(a) and 5.15(b)). One turning point can be noticed in rejection probability curves (Fig. 5.15(a)) at which rejection probability increases suddenly. Such points (e.g., Fig. 5.15(a), 400 STs/hour for 20 PMs in each pool) may suggest the appropriate time to upgrade or inject more resources (i.e., PMs). However, in case of total delay, two turning points (three regimes of operation) can be identified (Fig. 5.15(b)): the first regime (i.e., stable regime) is the region in which the cloud providers would like to operate. An admission control may use provided information here and helps the cloud centers to operate in such regime by not accepting extra requests. Transient regime is in between two turning points which the total delay increases sharply. Cloud centers should avoid entering such region since the violation of SLA is about to happen.

However, since the transient regime is not too narrow, it is possible for cloud center to return to stable zone after a short time (i.e., the admission control has enough time to adjust the new admissions). The transient regime is attributed to the size of global queue. The longer the global queue is, the wider the transient regime is going to be. In saturation regime, the super-tasks are experiencing the highest possible amount of delay and this zone is not in the interest of both cloud service providers and cloud customers.



(a) Rejection probability.

(b) Total delay on a super-task.

Figure 5.16: service-time=120 minute and arrival-rate=2500 tasks/hour.

In the last experiment, (Figs. 5.16(a) and 5.16(b)) we examine the effect of super-task size on task rejection probability and total delay. Each PM run 2 VMMs and every VMM hosts 3 VMs; a super-task may request up to all of the available VMs in a PM (i.e., all 6 VMs). Note that the aggregate arrival rate is set to 2500 tasks/hr, however the effective arrival rate of super-tasks is various for different super-task size. For instance, if the average super-task size was five, then the effective arrival rate of super-tasks would be 500 STs/hr.

As can be seen by increasing the size of super-tasks the rejection probability and total delay reduce sharply. Since the size of super-tasks is truncated up to the maximum number of VMs allowed on each PM (here, up to 6 VMs), the bigger super-task size will result in the lower number of arrivals as well as lower deviation of super-task size. As can be seen, by increasing the capacity the maximum delay is occurred when the mean size of STs is 3 and the maximum gain (i.e., reduction of delay) is obtained when the mean size of STs is 4. For STs of size 5 or bigger, the dominant imposed delays are the processing delays (i.e, resource assigning and VM provisioning delays) as opposed to the queuing delays.

5.9 Summary of the Chapter

In this Chapter, we presented a performance model suitable for analyzing the service quality of large sized IaaS clouds, using interacting stochastic models. We examined the effects of various parameters including ST arrival rate, task service time, the virtualization degree, and ST size on task rejection probability and total response delay. The stable, transient and unstable regimes of operation for given configurations have been identified so that capacity planning is going to be a less challenging task for cloud providers.

Validation of analytical results through extensive simulation has shown that our analytical model is sufficiently detailed to capture all realistic aspects of resource allocation process, instance creation and instantiation delays of a modern cloud center, while maintaining a good tradeoff between accuracy and tractability. Consequently, cloud providers can obtain a reliable estimation of response time and blocking probability that will result in avoidance of SLA violation.

We have also evaluated an availability model on top of the interacting performance

model in order to capture the failure, repair and migration process of PMs among server pools. Using the availability model, we calculated the effective value of each performance measure.

Chapter 6

Pool Management and Heterogeneity in Cloud Centers

in this Chapter, we first develop and evaluate tractable functional performance sub-models and their interaction model. We construct separate sub-models for internal and external servicing of a complex cloud center. By introducing a new component to our analytical model, a Pool Management Module (PMM), our performance model now supports inter-pool communication [50]. PMs can be moved among pools in order to maintain acceptable availability, response time and power consumption at the same time. More specifically, PMM describes the dynamics of server pools and provides the steady-state pools arrangement. In the last part of this Chapter, we also propose and examine a layered stochastic performance model applicable to cloud centers with wide range of heterogeneous physical and virtual resources as well as task characteristics without sacrificing the scalability, granularity and simplicity [49].

This Chapter is organized as follows: in Section 6.1, we present the details of the pool

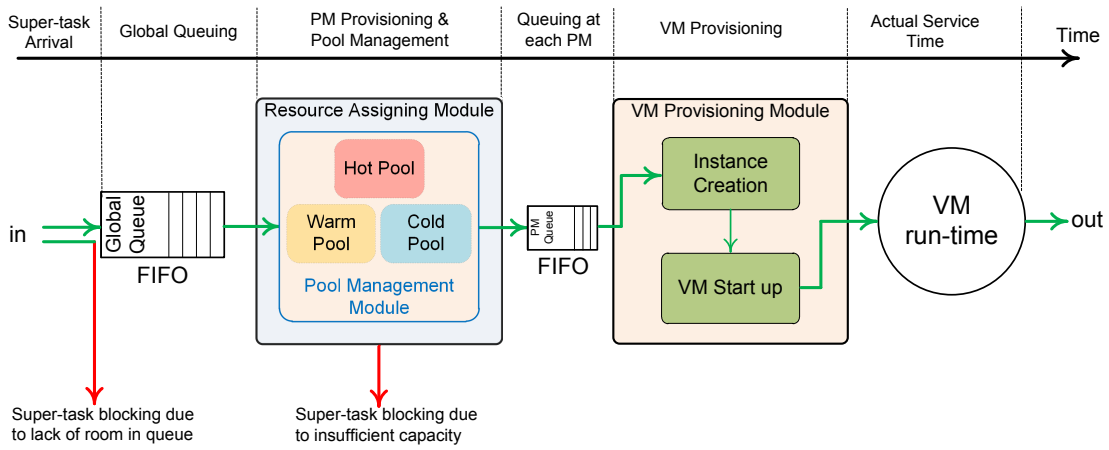


Figure 6.1: The steps of servicing and corresponding delays.

management scheme. Section 6.2 elaborates analytical sub-models; interactions and the degree of scalability for each sub-model are discussed. The numerical validation of pool schema is presented in Section 6.3. We describe our heterogeneous performance model in Section 6.4. Section 6.5 include the numerical results for the heterogeneous performance model. Finally, Section 6.6 summarizes and highlights the Chapter contributions.

6.1 Analytical Model

In IaaS cloud, when a request is processed, a pre-built or customized disk image is used to create one or more VM instances. In this work, we assume that pre-built images fulfill all user requests. We assume that Physical Machines (PMs) are categorized into three server pools: hot (i.e., with running VMs), warm (i.e., turned on but without running VM) and cold (i.e., turned off) [28, 27]. All PMs and VMs are homogeneous and each PM has two Virtual Machine Monitors (VMM) that configure and deploy VMs on the PM. Instantiation

of a VM from an image and provisioning it on hot PMs has the minimum delay compared to warm and cold PMs. Warm PMs require more time to be ready for provisioning and cold PMs require additional time to be started up before a VM instantiation. In this work we allow users to request more than one VM by submitting a super-task. A super-task may contain more than one task, each of which requires one VM. Each unit tasks has i.i.d. exponentially distributed service time; the size of super-tasks can be generally distributed.

If a user submits a super-task, it is very likely that individual tasks within the super-task require a high degree of interaction. As a result, we assume that all tasks within a super-task should be provisioned on the same PM. This assumption has two consequences. First, the super-task size must be limited to the number of VMs on a single PM. Second, all tasks within a super-task must begin service at the same time, otherwise the super-task will be rejected. The latter policy is often referred to as *total acceptance policy* [86]. The other approach, known as partial acceptance, may split a super-task so that individual tasks run on different PMs. While this policy may reduce the super-task rejection probability, it may also increase inter-task communication overhead and idle waiting, and, consequently, extend the overall service time. In this work, we assume total acceptance policy, while the performance of partial acceptance policy remains a promising topic for future work.

The steps incurred in servicing a super-task are shown in Fig. 6.1. User requests (super-tasks) are submitted to a global finite queue and then processed on a first-in, first-out basis (FIFO). A super-task that finds the queue full, will be rejected immediately. Once the super-task is admitted to the queue, it must wait until the Resource Assigning Module (RAM) processes it (first delay), as follows. First, RAM checks the hot pool for a PM with sufficient capacity. If such a PM can't be found, RAM moves on to the warm pool. If a PM

can't be found there, RAM moves on to the cold pool. Eventually, a super-task is either assigned to a PM (second delay), or rejected due to insufficient system capacity. Once the super-task is assigned to one of the PMs, it will have to wait in that PM's input queue (third delay) until the VM Provisioning Module (VMPM) instantiates and deploys the necessary VMs (fourth delay), when the actual service starts. The overall response time is, then, the sum of the delays mentioned above and the actual service time. When a running task finishes, the capacity used by the corresponding VM is released and becomes available for servicing the next (super-)task.

The management of pools and movement of PMs among them is the primary responsibility of the Pool Management Module (PMM). While the first criterion that governs PMM operation is to minimize the cloud center response time and task rejection probability, the obvious solution that would keep all PMs in the hot pool (i.e., always on) would lead to excessive energy consumption. Hence the PMM needs to switch idle PMs from hot to warm pool, or from warm to cold pool, after a certain interval of inactivity, the duration of which may be altered to suit the traffic load and other operational parameters, as will be explained below.

To model the behavior of this system, we design three stochastic sub-models, each of which captures the details of the respective management module (RAM, VMPM, and PMM) and combine them into an overall model. then, we solve this model to compute the cloud performance metrics: task rejection probability and mean response delay as functions of variations in workload (super-task arrival rate, super-task size, task mean service time, number of VMs per PM), system capacity (i.e., number of PMs in each pool), and design parameters such as the rate by which idle hot PM powers down to either warm or cold

pool. We describe our analysis in detail in the following sections, using the symbols and acronyms listed in Table 6.1.

Symbol	Description
RAM	Resource Assigning Module
VMPM_x	Virtual Machine Provisioning Module for a PM in the pool x; x could be hot, warm or cold
PMM	Pool Management Module
RASM	Resource Allocation Sub-Model
VMPSM_x	Virtual Machine Provisioning Sub-Model for a PM in the pool x; x could be hot, warm or cold
PMSM	Pool Management Sub-Model
MSS	Maximum Super-task Size
m	Maximum No. of VMs on a PM
p_i	Probability of having a super-task with size i
λ_{st}	Mean arrival rate of super-tasks
L_q	Size of global input queue
P_h, P_w, P_c	Probability of successful search in hot, warm and cold pool
$1/\alpha_h, 1/\alpha_w, 1/\alpha_c$	Mean look up time in hot, warm and cold pool
BP_q	Blocking probability due to lack of room in the global queue
BP_r	Blocking probability due to lack of capacity in the cloud center
P_{reject}	Total probability of blocking ($BP_q + BP_r$)
P_i	The probability by which a hot PM becomes idle
1/R	Mean searching time for finding an idle hot PM

$1/SU_w$	Mean start up time for a warm PM to become a hot PM
$1/SU_c$	Mean start up time for a cold PM to become a hot PM
$1/FR_w$	Mean time in which the lack of resource (PM) in the hot pool is detected and a PM from warm pool becomes hot
$1/FR_c$	Mean time in which the lack of resource (PM) in the hot pool is detected and a PM from cold pool becomes hot
σ	Desired No. of PMs that PMM tries to maintain in the warm pool for promoting the resiliency of the cloud center
$\overline{wt}$	Mean waiting time in global queue
$\overline{lut}$	Mean look up delay among pools
ϕ_h, ϕ_w, ϕ_c	Instantiation rate of a VM on a PM in the hot, warm or cold pool
$\lambda_h, \lambda_w, \lambda_c$	Arrival rate to a PM in the hot, warm and cold pool
N_h, N_w, N_c	Number of PMs in the hot, warm and cold pool
$\overline{PM_{wt}}$	Mean waiting time in a PM queue
$\overline{pt}$	Mean VM provisioning time
$\overline{td}$	Mean total delay for a task before getting into actual service

Table 6.1: Symbols and corresponding descriptions.

6.2 Integrated Stochastic Models

Due to high complexity of cloud centers, the proposed monolithic models were hard to analyze and extend. By introducing interacting analytical sub-models, the complexity of system is divided so that they can be analyzed accurately in order of processing steps. Moreover, the sub-models are scalable enough to capture the flexibility of the cloud computing paradigm.

In this paper, we implement the sub-models using interactive Continuous Time Markov Chain (CTMC). The sub-models are interactive such that the output of one sub-model is input to the other ones and vice versa. Table 6.2 shows the modules and their corresponding sub-models, which will be discussed in detail in the following sections.

Table 6.2: Modules and their corresponding sub-models.

Module	**Stochastic sub-model**
RAM	RASM
VMPM_hot	VMPSM_hot
VMPM_warm	VMPSM_warm
VMPM_cold	VMPSM_cold
PMM	PMSM

6.2.1 Resource Allocation Sub-Model

The resource allocation process is described in the Resource Allocation Sub-Model (RASM) shown in Fig. 6.2. RASM is a two dimensional CTMC that records the number of super-tasks in the global queue as well as the current pool on which provisioning is taking place. Since the number of potential users is high and a single user typically submits

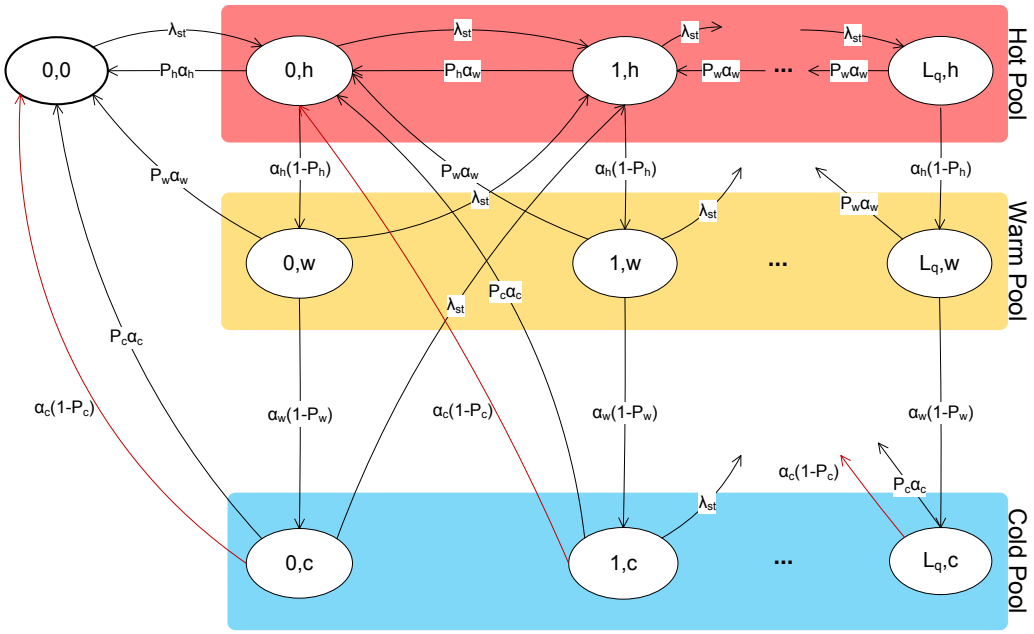


Figure 6.2: Resource Allocation Sub-Model (RASM).

super-tasks at a time with low probability, the super-task arrival can be adequately modeled as a Poisson process [30] with rate λ_{st}. Each state of Markov chain is labeled as (i, j), where i indicates the number of super-tasks in queue and j denotes the pool on which the leading super-task is under provisioning. State $(0, 0)$ indicates that system is empty which means there is no request under provisioning or in the queue. Index j can be h, w or c that indicate current super-task is undergoing provisioning on hot, warm or cold pool respectively. The global queue size is L_q and one more super-task can be at the deployment unit for provisioning thus, the capacity of system is $L_q + 1$.

Let P_h, P_w and P_c be the success probabilities of finding a PM that can accept the current super-task in the hot, warm and cold pool respectively. We assume that $1/\alpha_h$, $1/\alpha_w$ and $1/\alpha_c$ are the mean look up delays for finding an appropriate PM in hot, warm and cold

pool respectively. Upon arrival of first super-task, system moves to state $(0, h)$ which means the super-task will be provisioned immediately in the hot pool. Afterwards, depending on the upcoming event, three possible transitions can occur:

(a) Another super-task has arrived and system transits to state $(1, h)$ with rate λ_{st}.

(b) A PM in hot pool accepts the super-task so that system moves back to state $(0, 0)$ with rate $P_h\alpha_h$.

(c) None of PMs in hot pool has enough capacity to accept the super-task, so the system examines the warm pool (i.e., transit to state $(0, w)$) with rate $(1 - P_h)\alpha_h$.

On state $(0, w)$, RASM attempts to provision the super-task on warm pool. If one of the PMs in the warm pool can accommodate the super-task, the system will get back to $(0, 0)$, otherwise RASM examines the cold pool (i.e., transition from state $(0, w)$ to $(0, c)$). If none of the PMs in cold pool can provision the super-task, the system moves back from $(0, c)$ to $(0, 0)$ with rate $(1 - P_c)\alpha_c$ which means the user request (super-task) will get rejected due to insufficient resources in the cloud center. During the provisioning in cold pool, another super-task may arrive and takes the system to state $(1, h)$ which means there is one super-task in deployment unit and one awaiting in global queue. Finally, the super-task under provisioning decision leaves the deployment unit, once it receives a decision from RASM and the next super-task at the head of global queue will go under provisioning. In this sub-model, arrival rate of super-task (λ_{st}) and look up delays ($1/\alpha_h, 1/\alpha_w$ and $1/\alpha_c$) are exogenous parameters and success probabilities (P_h, P_w and P_c) are calculated from the VM provisioning sub-model. The VM provisioning sub-model will be discussed in the next Section.

Using steady-state probabilities $\pi_{(i,j)}$, blocking probability can be calculated. Super-tasks may experience two kinds of blocking events:

(a) Blocking due to a full global queue occurs with the probability of

$$BP_q = \pi_{(L_q,h)} + \pi_{(L_q,w)} + \pi_{(L_q,c)} \tag{6.1}$$

(b) Blocking due to insufficient resources (PMs) at pools occurs with the probability of [34]

$$BP_r = \sum_{i=0}^{L_q} \frac{\alpha_c(1-P_c)}{\alpha_c + \lambda_{st}} \pi_{(i,c)} \tag{6.2}$$

The probability of reject is, then, $P_{reject} = BP_q + BP_r$. In order to calculate the mean waiting time in queue, we first establish the probability generating function (PGF) for the number of super-tasks in the queue [85], as

$$Q(z) = \pi_{(0,0)} + \sum_{i=0}^{L_q} (\pi_{(i,h)} + \pi_{(i,w)} + \pi_{(i,c)}) z^i \tag{6.3}$$

The mean number of super-tasks in queue is [86]

$$\overline{q} = Q'(1) \tag{6.4}$$

Applying Little's law [57], the mean waiting time in the global queue is given by (first delay):

$$\overline{wt} = \frac{\overline{q}}{\lambda_{st}(1 - BP_q)} \tag{6.5}$$

Look up time among pools can be considered as a Coxian distribution with 3 steps (Fig. 6.3).

Therefore according to [93], the look-up time (second delay) can be calculated as follows.

$$\overline{lut} = \frac{1/\alpha_h + (1-P_h)((1/\alpha_w) + (1-P_w)(1/\alpha_c))}{1 - BP_q} \tag{6.6}$$

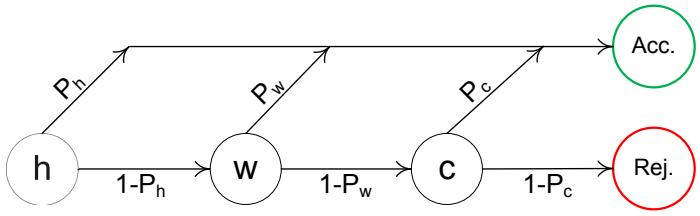


Figure 6.3: Three steps of look-up delays among pools.

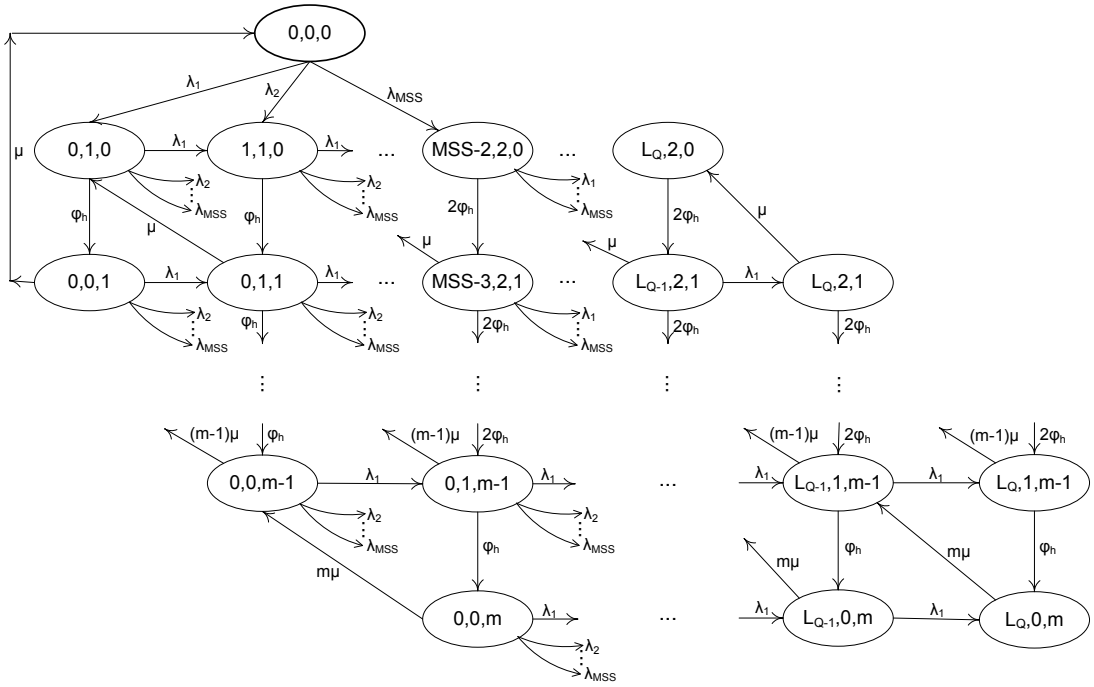


Figure 6.4: Virtual Machine Provisioning Sub-Model for a PM in the hot pool (VMPSM_hot).

6.2.2 VM Provisioning Sub-Model

Virtual Machine Provisioning Sub-Model (VMPSM) captures the instantiation, deployment and provisioning of VMs on a PM. VMPSM also incorporates the actual servicing of each task (VM) on a PM. Note that each task within super-task is provisioned with an individual VM, but RASM makes sure that all tasks within a super-task are provisioned at the

same PM. Fig. 6.4 shows the VMPSM (a CTMC) for a PM in the hot pool. As we assume homogeneous PMs, VMs, and tasks, all VMPSMs for the PMs in the hot pool are identical in terms of arrival and instantiation rates. The same holds for the PMS in the warm and cold pool – they can be modeled with the same VMPSM, though with a different arrival and instantiation rates. Consequently, each pool (hot, warm and cold) can be modeled as a set of VMPSMs.

Each state in Fig. 6.4 is labeled by (i, j, k) in which i indicates the number of tasks in PM's queue, j denotes the number of tasks that is under provisioning by VMMs and k is the number of VM that are already deployed on the PM. Since we assume two VMMs installed on each PM, at most tasks can be provisioned simultaneously on a PM. Note that we set the queue size at each PM to m, the maximum number of VMs that can be deployed on a PM. Let ϕ_h be the rate at which a VM can be deployed on a PM at hot pool and μ be the service rate of each VM. So, the total service rate for each PM is the product of number of running VMs by μ. State $(0, 0, 0)$ indicates that the PM is empty and there is no task either in the queue or in the instantiation unit. Depending on the size of arriving super-task, model transits to one of the states in $\{(0, 1, 0), (0, 2, 0), \cdots, (MSS - 2, 2, 0)\}$. Since all tasks within a super-task are to be provisioned at the same PM, maximum super-task size (MSS) is smaller or equal to m. The arrival rate to each PM in the hot pool is given by:

$$\lambda_h = \frac{\lambda_{st}(1 - BP_q)}{N_h} \tag{6.7}$$

in which N_h is the number of PMs in the hot pool. Note that BP_q, used in (6.1), is obtained from RASM. In Fig. 6.4, $\{\lambda_i, i = 1..MSS\}$ is given by $\lambda_i = \lambda_h p_i$; here p_i is the probability of having a super-task of size i. We examine truncated geometric and uniform distributions for numerical experiment.

The state transition in VMPSM can occur due to super-task arrival, task instantiation or service completion. From state $(0, 0, 0)$, system can move to state $(0, 2, 0)$ with rate λ_2 (i.e., super-task with size 2). From $(0, 2, 0)$, system can transit to $(0, 1, 1)$ with rate ϕ_h (i.e., instantiation rate) or upon arriving a super-task with size k, moves to state $(k, 2, 0)$. From $(0, 1, 1)$, system can move to $(0, 0, 2)$ with rate ϕ_h, transits to $(0, 1, 0)$ with rate μ (i.e., service completion rate), or again upon arriving a super-task with size k, system can move to state $(k - 1, 2, 1)$ with rate λ_k. A PM can not accept a super-task, if there is not enough room to accommodate all tasks within a super-task. Suppose that $\pi^h_{(i,j,k)}$ is the steady-state probability for the hot PM model (Fig. 6.4) to be in the state (i, j, k). Using steady-state probabilities, we can obtain the probability that at least one PM in hot pool can accept the super-task for provisioning. At first we need to compute the probability that a hot PM cannot admit a super-task for provisioning (P^h_{na}). Let F_h be the free capacity of the hot PM at each state:

$$F_h(i, j, k) = m - (i + j + k)$$

P^h_{na} is then given by:

$$P^h_{na} = \sum_{x \in \xi} \sum_{t=F_h+1}^{MSS} \pi^h_x p_t \tag{6.8}$$

where p_t is the probability of having a super-task of size t and ξ is a subset of VMPSM's states in which there is a chance of blocking for super-tasks:

$$\xi = \{(i, j, k) |\ F_h(i, j, k) < MSS\}$$

Therefore, probability of successful provisioning (P_h) in the hot pool can be obtained as

$$P_h = 1 - (P^h_{na})^{N_h} \tag{6.9}$$

Note that P_h is used as an input parameter in the resource allocation sub-model (Fig. 6.2). The provisioning model for warm PM is the same with the one for hot PM, though, there are some differences in parameters:

(a) The arrival rate to each PM in the warm pool is

$$\lambda_w = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)}{N_w} \tag{6.10}$$

where N_w is the number of PMs in warm pool.

(b) Every PM in warm pool requires extra time to be ready (hot) for first instantiation. This time is assumed to be exponentially distributed with mean value of $1/\gamma_w$.

Instantiation rate in warm pool is the same as in hot pool (ϕ_h). Like VMPSM for a hot PM (Fig. 6.4), the model for a warm PM is solved and the steady-states probabilities ($\pi^w_{(i,j,k)}$), are obtained. The success probability for provisioning a super-task in warm pool is:

$$P_w = 1 - (P^w_{na})^{N_w} \tag{6.11}$$

A PM in the cold pool can be modeled as in the warm and hot pool but with different arrival rate and start up time. The effective arrival rate at each PM in the cold pool is given by:

$$\lambda_c = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)(1 - P_w)}{N_c} \tag{6.12}$$

in which N_c is the number of PMs in the cold pool. A cold PM (off machine), requires longer time than a warm PM to be ready (hot) for the first instantiation. We assume that the start up time is also exponentially distributed with mean value of $1/\gamma_c$. The success probability for provisioning a super-task in the cold pool is given by:

$$P_c = 1 - (P^c_{na})^{N_c} \tag{6.13}$$

From VMPSM, we can also obtain the mean waiting time at PM's queue (third delay: $\overline{PM_{wt}}$) and mean provisioning time (fourth delay: $\overline{pt}$) by using the same approach as the one that led to Eq. (6.5). As a result, the total delay before starting of actual service time, is given by:

$$\overline{td} = \overline{wt} + \overline{lut} + \overline{PM_{wt}} + \overline{pt} \tag{6.14}$$

Note that all success probabilities (i.e., P_h, P_w and P_c) are input parameters to the resource allocation sub-model (Fig. 6.2).

6.2.3 Pool Management Sub-Model

Pool Management Sub-Model (PMSM) captures the details of pools management in the cloud center. We model PMSM as a three dimensional CTMC, Fig. 6.5. The system starts with state (i, j, k) which indicates that i, j and k PMs are in the hot, warm and cold pool respectively. If next search in hot pool was successful (probability of P_h), system will remain in the starting state, state (i, j, k), otherwise PMSM will bring one of the warm PMs into the hot pool (requires some time). Such transition occurs with the rate of FR_w which is defined as:

$$FR_w = \frac{1}{\lambda_{st} BP_q (1 - P_h) + (1/SU_w)} \tag{6.15}$$

in which $(1/SU_w)$ is the mean time that a warm PM becomes a hot PM.

In state (i, j, k), if one of the hot PM becomes idle (i.e., no running VM), the PMSM moves the idle hot PM to cold pool (i.e., turn off) by going to state $(i - 1, j, k + 1)$. If all PMs become idle, PMSM will move all the hot PMs to the cold pool and maintain a minimum number of PMs (here we assume 2 PMs) in the warm pool. As can be seen from Fig. 6.5, state $(i + j - 1, 1, k)$, PMSM turns on a cold PM machine whenever the number of

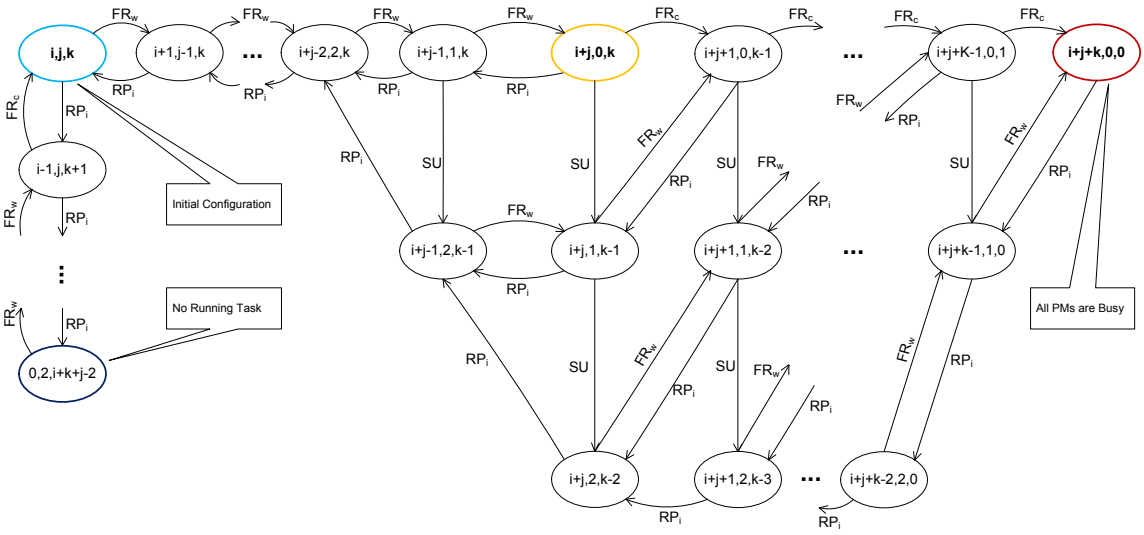


Figure 6.5: Pool Management Sub-Model (PMSM).

warm PMs gets lower than 2. If the system is in state $(i+j, 0, k)$, in which there is no warm PM at the moment, upon arrival of a new super-task, PMSM will try to accommodate the new super-task on the current hot PMs. In case of lack of capacity for new arrival, PMSM borrows a PM from cold pool with the rate FR_c, moving to state $(i+j+1, 0, k-1)$.

$$FR_c = \frac{1}{\lambda_{st} BP_q (1 - P_h) + (1/SU_c)} \tag{6.16}$$

where $(1/SU_c)$ is the mean time that a cold PM becomes a hot PM.

The state $(i+j+k, 0, 0)$ indicates that all PMs in the cloud center are hot and servicing users' tasks. At any state, if the number of PMs in the warm pool is less than a predefined minimum (e.g., 2 PMs in Fig. 6.5), an idle hot PM is moved to the warm pool at rate RP_i, otherwise the idle hot PM is moved to the cold pool (i.e., shut down).

6.2.4 Interaction among Sub-Models

The interactions among sub-models are depicted in Fig. 6.6. VM provisioning sub-models (VMPSM) compute the steady-state success probabilities (P_h, P_w and P_c) that at least a PM in one of the pools (hot, warm and cold, respectively) can accept the super-task. These success probabilities are used as input parameters to the resource allocation sub-model (RASM). The hot PM sub-model (VMPSM_hot) computes P_h which is the input parameter for both warm and cold sub-models. The warm PM sub-model (VMPSM_warm) computes P_w which is the input parameter for the cold sub-model (VMPSM_cold). In addition, the hot PM model provides P_h and the probability by which a hot PM becomes idle (P_i) for pool management sub-model (PMSM). In return, PMSM computes N_h, N_w and N_c as input for hot, warm and cold PM sub-models respectively. The resource allocation sub-model (RASM) computes the blocking probability, BP_q, which is the input parameter to VM provisioning sub-models as well as PMSM. Table 6.3 shows the sub-models and their corresponding outputs.

Table 6.3: Sub-models and their outputs.

Sub-model	**Output(s)**
RASM	BP_q, BP_r
VMPSM_hot	P_h, P_i
VMPSM_warm	P_w
VMPSM_cold	P_c
PMSM	N_h, N_w, N_c

As can be seen, there is an inter-dependency among sub-models. This cyclic dependency is resolved via fixed-point iterative method [65] using a modified version of succes-

sive substitution approach. For numerical experiments the successive substitution method

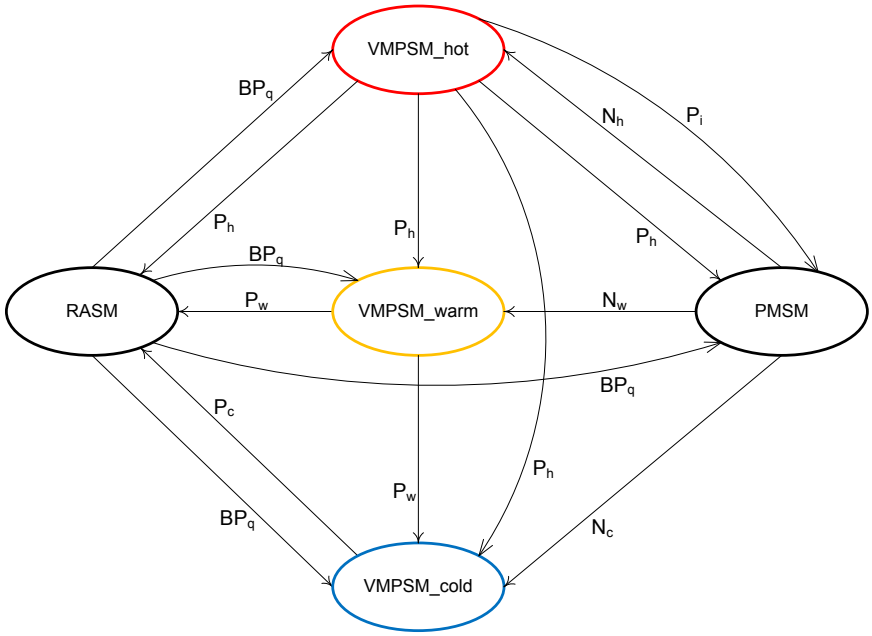


Figure 6.6: Interaction diagram among sub-models.

(Algorithm 2) is continued until the difference between the previous and current value of blocking probability in the global queue (i.e., BP_q) is less than 10^{-6}. Usually the integrated model converges to the solution in less than 8 iterations.

6.2.5 Scalability and Flexibility of Integrated Model

A cloud center may scale up or down in terms of global queue size, number of PMs in each pool and the degree of virtualization which are referred as design parameters. Table 6.4 shows the relationship between the number of states in each sub-model and their associated design parameters.

As can be seen from Table 6.4, the number of states in each sub-model has a linear

Algorithm 2 Successive Substitution Method

Input: Initial success probabilities in pools: P_{h0}, P_{w0}, P_{c0}

Input: Initial idle probability of a hot PM: P_{i0}

Output: Blocking probability in Global Queue: BP_q

counter ← 0; max ← 10; diff ← 1

BP_{q0} ← RASM (P_{h0}, P_{w0}, P_{c0})

$[N_h, N_w, N_c]$ ← PMM (P_{h0}, P_{i0})

while diff $\geq 10^{-6}$ **do**

counter ← counter + 1

$[P_h, P_i]$ ← VMPSM_hot (BP_{q0}, N_h)

P_w ← VMPSM_warm (BP_{q0},P_h, N_w)

P_c ← VMPSM_cold (BP_{q0},P_h,P_w, N_c)

$[N_h, N_w, N_c]$ ← PMM (P_h, P_i)

BP_{q1} ← RASM (P_h, P_w, P_c)

diff ← $|(BP_{q1} - BP_{q0})|$

$BP_{q0} \leftarrow BP_{q1}$

if counter = max **then**

break

end if

end while

if counter = max **then**

return -1

else

return BP_{q0}

end if

Table 6.4: Relationship between the size of sub-models and design parameters.

Sub-Model	**Design Parameters**	**Number of States: f(i)**
RASM	L_q	f(L_q) = 3L_q + 1
VMPSM	m	f(m) = 3, if m=1 f(m) = 6, if m=2 f(m) = $2f(m-1) - f(m-2) + 1$, if m>2
PMSM	N_w, N_c, σ	f(N_w, N_c, σ) = $N_w + N_c + 1 + \ \sigma \sum_{s=1}^{\sigma}(N_w + N_c - \sigma - s)$

dependency on design parameters which guarantees the scalability of the integrated model. Note that VMPSMs are identical for all PMs in the same pool. Therefore, we need only to solve three VMPSMs (i.e., VMPSM_hot, VMPSM_warm and VMPSM_cold) regardless of how many PMs are in the cloud center.

6.3 Numerical Validation

Under different configurations and parameter settings, we obtain two important performance metrics, namely, rejection probability of super-tasks (STs) and total response delay. We show the effects of changing arrival rate, task service time, number of PMs in each pool, number of VMs in each PM and super-task size on the said performance indicators. We also obtain the steady-state arrangement of pools (i.e., number of PMs in each pool) under above-mentioned parameter setting as well as various pool checking rates.

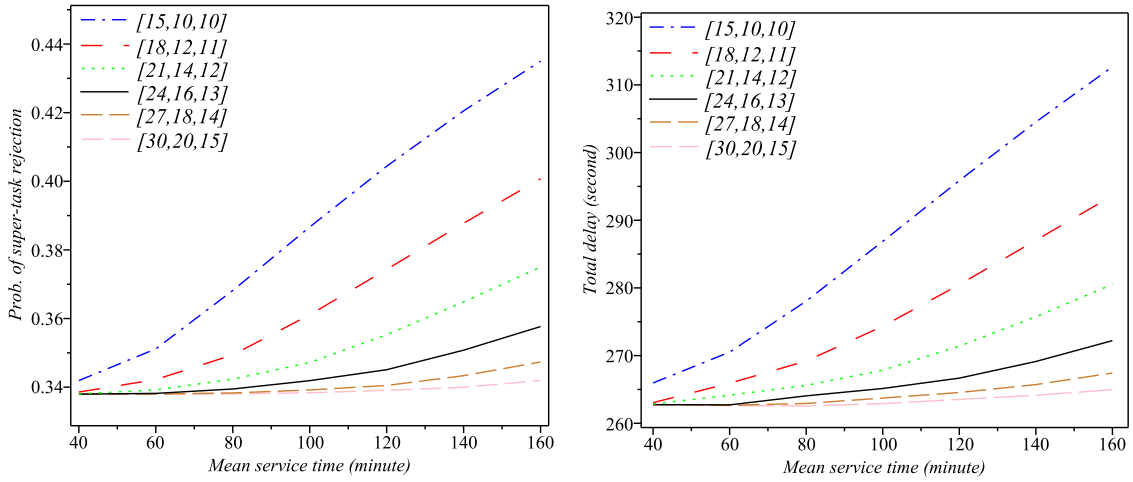
Table 6.5 presents the range of individual parameter values. Also, in all subsequent discussions and diagrams, the notation $[h, w, c]$ will refer to a cloud center with h, w, and c

PMs (i.e., servers) in the hot, warm, and cold pool, respectively.

Table 6.5: Range of parameters for numerical experiment.

Parameter	**Range**	**Unit**
Arrival rate	400 . . 1000	super-tasks/hour
Mean service time	40 . . 160	minute
No. of PMs in the hot pool	15 . . 30	N/A
No. of PMs in the warm pool	10 . . 20	N/A
No. of PMs in the cold pool	10 . . 15	N/A
No. of VMMs on each PM	2	N/A
No. of VMs on each PM	2 . . 10	N/A
Mean Look-up rate in the hot pool	625	searches/hour
Mean Look-up rate in the warm and cold pool	825	searches/hour
Required time for a warm PM to become hot	1 . . 5	minute
Required time for a cold PM to become hot	5 . . 10	minute
VM deployment time on a hot PM	1 . . 5	minute
First VM deployment delay on a warm PM	3 . . 10	minute
First VM deployment delay on a cold PM	8 . . 15	minute
Size of global queue	50 . . 100	super-tasks
Queue size at PMs	2 . . 10	tasks

Our first set of experiments investigates the effects of task service time on rejection probability and total delay imposed by the cloud center on super-tasks before getting into service. In the first experiment, the results of which are shown in Fig. 6.7, super-tasks have one task each, while each PMs are permitted to run only two VMs; the super-task arrival rate is constant at 1000 STs/hr. As can be seen from Fig. 6.7(a), increasing the mean service time of individual tasks leads to an almost linear increase in rejection probability, while the increase of the system capacity (i.e., the number of PMs in each pool) reduces



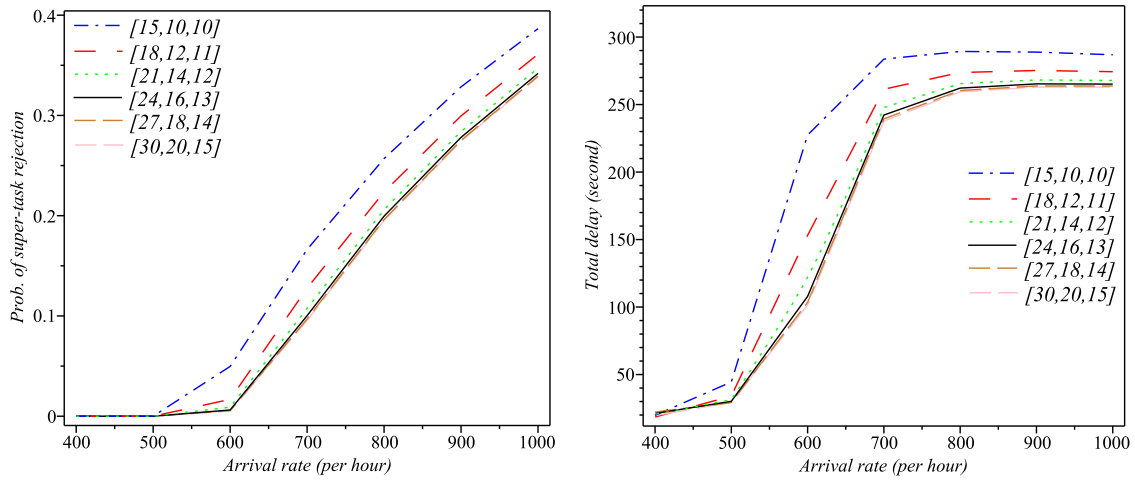
(a) Super-task rejection probability.

(b) Total delay on a super-task.

Figure 6.7: 2 VMMs on each PM, 1 VM on each VMM, arrival rate = 1000 STs/hour, super-task-size = 1.

the rejection probability. The latter effect is more pronounced for tasks with longer service time, in particular for those with service time over 80 minutes. At the same time, extending system capacity has a negligible impact of rejection probability for tasks shorter than 60 min. These results allow us to identify a suitable arrangement for each service time, by applying a trade-off between the performance gain and the expenditure of required resources. Regarding total delay, Fig. 6.7(b) shows that, for all pool arrangements, longer service times will result in longer delays, since the actual service time has a direct effect on the amount of waiting time of queued super-tasks. This effect is more noticeable for 'weaker' configurations such as [15, 10, 10] (i.e., 15 PMs in the hot pool, and warm and cold pools with 10 PMs each), as opposed to the 'stronger' ones such as [30, 20, 15] where the delay increases is barely perceptible in the observed range of mean service times.

System performance when the mean service time is fixed at 100 minutes, but the super-

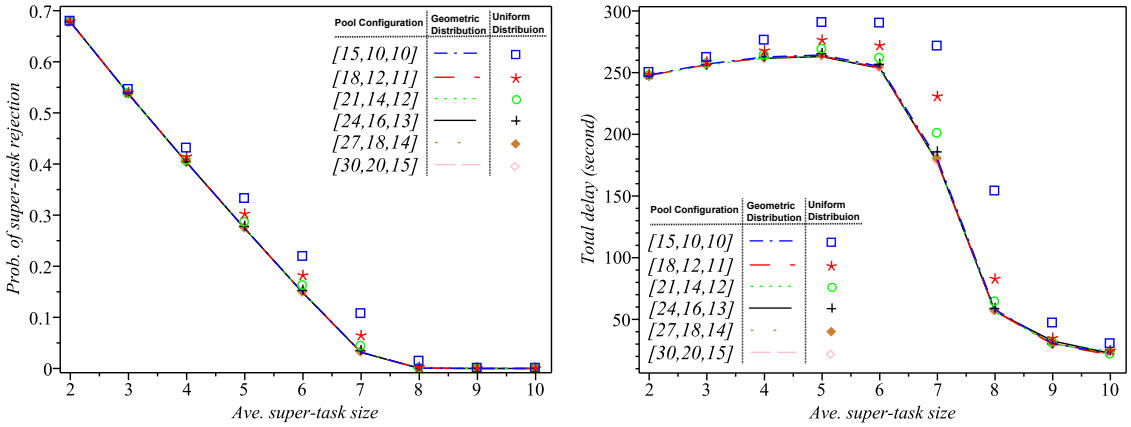


(a) Super-task rejection probability.

(b) Total delay on a super-task.

Figure 6.8: 2 VMMs on each PM, 1 VM on each VMM, service time = 100 min, super-task-size = 1.

task arrival rate (which, in this case, is equal to the task arrival rate) is variable, is shown in the diagrams in Fig. 6.8. Again, the rejection probability, shown in Fig. 6.8(a), exhibits a sharp rise above a certain value of (super-)task arrival rate, which is mainly due to insufficient size of the global (input) queue. However, the total delay, shown in Fig. 6.8(b), exhibits a sharp increase at an even lower value of the (super-)task arrival rate; when the rejection rate becomes non-negligible, the delay flattens because the actual number of tasks serviced is decreasing. Thus a simple increase in the size of the global queue is ill advised, as it would reduce the rejection probability at the expense of much higher delays. Consequently, if a cloud provider observes unacceptable delay and/or task rejection performance, a prudent course of action would be to increase the system capacity (i.e., the number of PMs) first, and extend the global queue only if the observed total response time is well below the prescribed limit.



(a) Super-task rejection probability.

(b) Total delay of a super-task.

Figure 6.9: 2 VMMs, 5 VMs on each VMM, service time = 100 min, arrival rate = 4500 tasks/hour.

Our second set of experiments focuses on the impact of super-task size on system performance. We have conducted the calculations using both geometric and uniform distribution for the number of tasks in a super-task. In the former case, we truncate the distribution at the value of maximum super-task size (MSS) which corresponds to the maximum number of VMs running on a single PM, so the probability of having a super-task of size i is

$$p_i = \begin{cases} (1-p)^{i-1}p, & \text{if } i < \text{MSS} \\ (1-p)^{MSS-1}, & \text{if } i = \text{MSS} \end{cases} \tag{6.17}$$

in which p is the parameter of geometric distribution. In the latter case (i.e., for uniform distribution) the probability of having a super-task of size i is

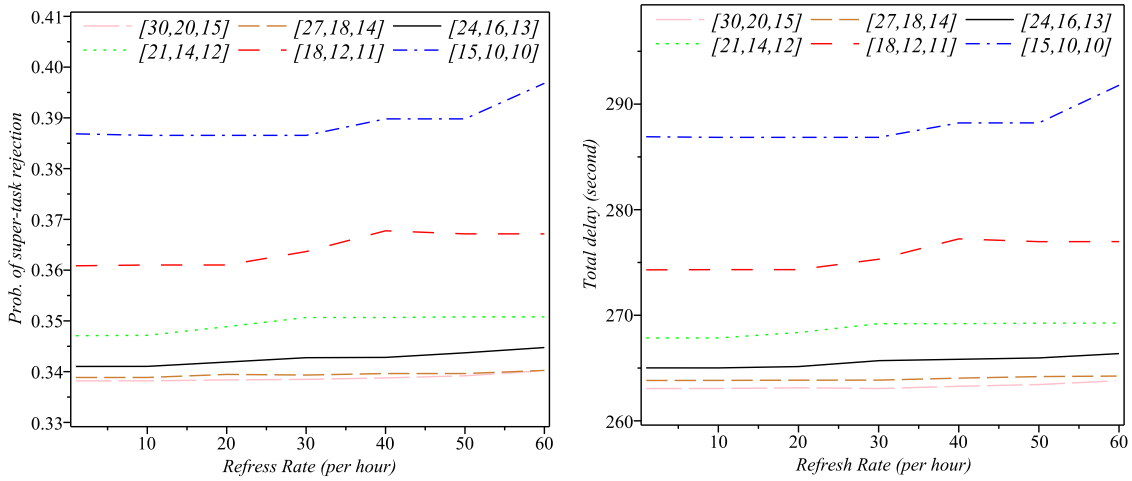
$$p_i = \frac{1}{MSS} \tag{6.18}$$

Regarding the value of MSS, it is worth noting that our model does not impose any restriction whatsoever on the degree of virtualization, but we still need a practical limit. We

are aware that a mainframe class computer can indeed run tens, or perhaps even a hundred VMs that correspond to a standard PC machine. However, a recent survey has shown that 61% of the respondents run less than 10 VMs per PM, while a mere 5% run more than 25 VMs on a PM [81]. Moreover, VMs running on multi-core CPUs are vulnerable to cache interference, as a VM running on one core may degrade the performance of a VM running on another core [84, 105]. Hence, we have assumed that each PM runs two VMMs, each of which manages up to five VMs, for a total of up to 10 VMs per PM.

From the diagrams in Fig. 6.9(a), we can see that an increase in super-task size leads to a reduction of rejection probability for both geometric and uniform distributions. As for the total task delay, shown in Fig. 6.9(b), the curves show a relatively flat peak, and then fall off rapidly. This is due to the fact that 4500 tasks/hour is the aggregate task arrival rate: as the mean super-task size increases, the actual super-task arrival rate will decrease. In case of geometric distribution (shown with lines), larger mean size means lower variability of actual sizes, which makes the resulting rejection probability and total delay lower. In fact, in the boundary case where the average super-task size is 10, all super-tasks have the same size, and the super-task arrival rate is one-tenth of the aggregate rate. As the result, each incoming super-task will obtain an entire PM which, together with the corresponding reduction in super-task arrival rate, guarantees a shorter waiting time in the global queue and virtual elimination of blocking at that queue. The processing delay becomes, then, the dominant part of the total delay. We note that lower variability of super-task size has been shown to increase the server utilization in multi-server systems as well [9, 82].

Similar behavior can be observed in the case of uniform distribution (shown with symbols), although the rejection probability and total delay are somewhat higher than in the



(a) Super-tasks rejection probability.

(b) Total delay on a super-task.

Figure 6.10: 2 VMs on each PM, 1 VM on each VMM, super-task-size = 1, arrival rate = 1000 STs/hour, service time = 100 min.

case of geometric distribution. The rationale for this behavior is that super-tasks of all sizes are equi-probable under uniform distribution, whereas larger size super-tasks (which are more likely to get rejected under the total acceptance policy) have lower probability under geometric distribution. Also, larger super-tasks release more capacity when finished, which helps reduce the overall delay.

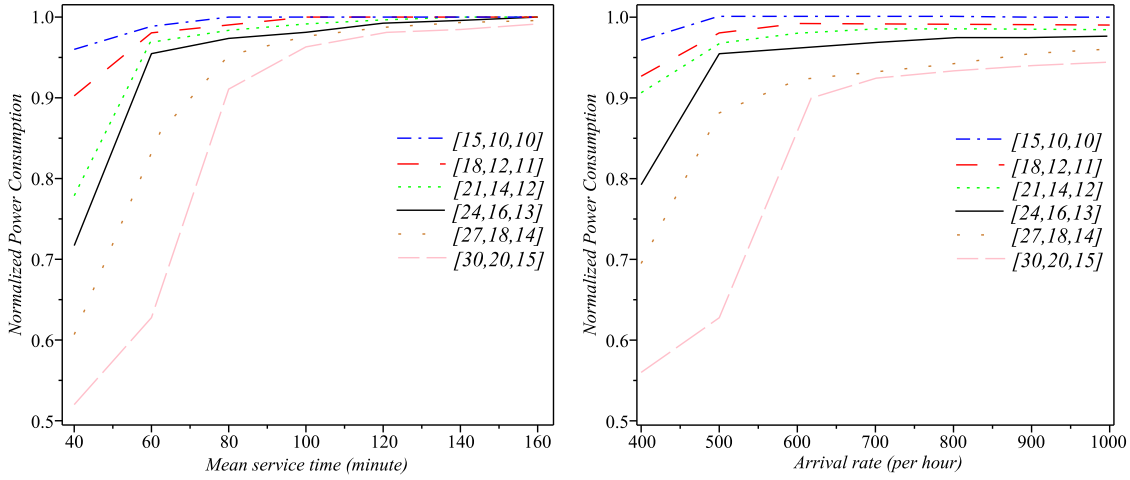
Our next experiment investigates the effects of pool checking rate on the performance metrics. The pool checking rate is the rate by which PMM searches for idle hot PMs and switches them to the warm or cold pool, respectively. Note that an idle hot PM would be first switched to the warm pool, and later to the cold pool; however, if there are more than σ PMs in the warm pool, it will be immediately moved to the cold pool (i.e., shut down). In previous experiments the mean pool checking rate was fixed at 15 checks per hour. For obvious reasons, higher pool checking rate will result in lower power consumption,

however a rate that is too high may bring about the ping-pong effect. Namely, a recently shut-down PM may be needed soon so PMM has to turn it on again, and frequent back and forth movement of PMs may lead to longer delays and higher rejection probability, not to mention reduced hardware reliability.

In our experiments we have calculated the performance metrics for a number of different pool configurations. As can be seen from Fig. 6.10, 'stronger' configurations easily outperform the 'weaker' ones. When power consumption, shown in Fig. 6.11, is also taken into account, the results reveal that any given pool arrangement has a unique pool check rate that results in optimum balance between power consumption, on one side, and availability and throughput, on the other side. From the results shown in Figs. 6.10 and 6.11, it can be seen that for [15,10,10] arrangement the optimum pool check rate is 30/hour, while for [18,12,11] the best rate is 20/hr. For other configurations, a pool check rate of 10/hour can be considered as a good choice.

Using PMM, we also study the steady-state arrangement of pools in a cloud center. In other words, we try to identify the most probable arrangement of PMs in the pools in a long run. We examine the model for vast variety of parameter space and present the results for 3 distinct settings (Table 6.6) in the Table 6.7.

In Table 6.7, the first column indicates the initial configuration of pools by which we start the experiment. Then under 3 different settings and for 6 pool configurations the steady-state arrangements are identified (the arrangements that are in bold face). Under such arrangement the rejection probability is zero and the cloud center imposes the minimum delay on its users' requests. For instance, for setting (A) the best arrangement is **[41,5,7]**. Such steady-state configuration suggests a cloud provider to arrange its server



(a) arrival rate = 500 (per hour).

(b) service time = 60 min.

Figure 6.11: Normalized power consumption: 2 VMMs on each PM, 1 VM on each VMM, super-task-size = 1.

Table 6.6: Three configuration settings for the experiment.

	Setting		
Design Parameters	**(A)**	**(B)**	**(C)**
Mean arrival rate (hour)	500	600	450
Mean service time (min)	80	60	100
Mean pool check rate (hour)	60	30	50
Desired No. of warm PMs	10	15	20
No. of VM on each PM	2	2	10
No. of VMM on each PM	2	2	2
Mean super-task size	1	2	4
Global queue size	50	75	100

pool accordingly in order for achieving the highest user satisfaction. However, if some amount of rejection or delay is tolerable, our experiment could suggest other configurations that can balance cloud providers' interests such as rejection probability, response delay, power consumption, and etc.

Table 6.7: Steady-state configuration of pools.

Initial	**Steady-State Configurations**		
Configuration	**Setting (A)**	**Setting (B)**	**Setting (C)**
$[15, 10, 10]$	$[33, 2, 0]$	$[34, 1, 0]$	$[34, 1, 0]$
$[18, 12, 11]$	$[38, 2, 1]$	$[40, 0, 1]$	$[39, 1, 1]$
$[21, 14, 12]$	$[40, 6, 1]$	$[45, 0, 2]$	**[41,3,3]**
$[24, 16, 13]$	**[41,5,7]**	$[48, 5, 0]$	$[39, 2, 12]$
$[27, 18, 14]$	$[40, 2, 17]$	$[51, 8, 0]$	$[39, 10, 10]$
$[30, 20, 15]$	$[38, 1, 26]$	**[52,5,8]**	$[39, 9, 17]$

We also quantify the effects of PMM on power consumption for different arrival rates as well as service rates. The normalized power consumption can be calculated as

$$Pow([N_h, N_w, N_c]) = \frac{N_h}{N_t} + \frac{N_w}{N_t} * \psi \tag{6.19}$$

in which $N_t = N_h + N_w + N_c$ and ψ is the ratio of power consumption of a warm PM to a hot PM (set to 60% according to [67, 24]). Therefore, when all PMs are busy the normalized power consumption of the cloud center is 1. As can be seen in the Fig. 6.11, in case of weaker configurations (i.e., [15,10,10] and [18,12,11]), PMM cannot achieve significant power saving since all of PMs are in hot mode most of the time. However, in case of larger configurations, PMM can achieve substantial power savings, even up to 50%, by turning down the idle PMs while still maintaining an acceptable level of QoS.

6.4 Heterogeneous Resources and Requests

Khazaei et al. [44, 48] employed homogeneous integrated performance models to overcome the complexity of cloud computing centers. In this Section however, we introduce

heterogeneity to our performance model. More specifically, we permit PMs to be different among pools and also VMs are to be different in terms of number of virtual CPUs (vCPU) [49]. Each pool includes specific type of PMs. PMs in pools are different in terms of virtual CPU (vCPU), number of cores per CPU, RAM, disk and the degree of virtualization (i.e the max number of running VMs on the PM).

Instantiation of a VM from an image and provisioning it on hot PMs has the minimum delay compared to warm and cold PMs. Warm PMs require some time to be ready for provisioning and cold PMs require additional time to be started up before a VM instantiation. Each VM needs one vCPU and each vCPU may run on multiple physical CUP cores [96]. The size of super-task (i.e., number of VMs) and size of vCPU (i.e., number of cores assigned to a vCPU) can be generally distributed. For numerical validation the super-task size is assumed to be uniformly or geometrically distributed but we truncate the distribution of the value by MSS (Maximum Super-task Size). MSS is the maximum number of VMs running on a single PM. A super-task is shown in Fig. 6.12. We also use uniform distribution for the vCPU size in the numerical Section.

# of vCPUs per VM	# of Cores per vCPU	RAM (Gig)	Disk (Gig)	# of VMs

Figure 6.12: Specification of a typical super-task submitted by users.

Due to high interaction among tasks within a super-task, each super-task will be provisioned on the same PM. A super-task may request up to all available VMs or CPU cores on a single PM. In this work, all VMs are the same in terms of RAM and disk while each VM may request a different number of CPU cores.

Server Assigning Module (SAM) starts with the hot pool to provision the super-task on a PM machine. If the process is not successful then SAM examines the warm pool. if there is no PM in warm pool that can accommodate the super-task, SAM refers to the cold pool. Finally, SAM either assigns a PM in one of the pools to a super-task or rejects the super-task. As a result, user requests (super-tasks) may get rejected either due to lack of space in global queue or insufficient resource at the cloud center. When a running task finishes, the capacity used by that VM is released and becomes available for servicing the next task.

We identify four main delays imposed by cloud computing centers on a user requests: at first, it should be determined whether cloud center has sufficient resources to fulfill the super-task. Using SAM, we capture the details of delays in the global queue as well as decision process. Then, VM Provisioning Module (VMPM) undertakes the provisioning of VMs. Therefore each task within super-task may experience queuing delay at PM's queue and VM provisioning process delay. After all, the actual service can start. Therefore, the response time can be considered as the sum of four above mentioned delays and the actual service time.

In order to model each module precisely, we design two stochastic sub-models for SAM and VMPM. We then connect these sub-models into an overall performance model to compute the task rejection probability and mean response delay. We describe our analysis in the following sections. SAM is realized by server allocation sub-model (SASM) that is identical with RASM in Chapter 5, Section 5.4.1, Fig. 5.2. Therefore, we only describe the VMPM in the next Section.

6.4.1 VM Provisioning Sub-Model

Virtual Machine Provisioning Sub-Model (VMPSM) captures the instantiation, deployment and running of VMs on a PM. Fig. 6.13 shows the VMPSM (a 4-dimensional CTMC) for a PM in the hot pool. For the sake of simplicity in representation, Fig. 6.13 just shows the VMPSM when each super-tasks has only one task. A PM in warm or cold pool can be modeled as a hot PM though, with some minor differences. Consequently, each pool can be modeled as a set of VMPSM with the same arrival and instantiation rate. Each state in Fig. 6.13 is labeled by (i, j, k, l) in which i indicates the number of tasks in PM's queue, j denotes the number of task that is under provisioning, k indicates the number of busy cores and l is the number of VM that are already deployed on the PM. Note that we set the queue size at each PM equal to the maximum number of VMs that can be deployed on a PM. Therefore, a super-task can be accepted by a PM if first, there is enough room in the PM's queue for all tasks within the super-task and second, if the PM has sufficient number of free cores for the given super-task. Let ϕ_h be the rate at which a VM can be deployed on a PM at hot pool and μ be the service rate of each VM. So, the total service rate for each PM is the product of number of running VMs by μ. State $(0, 0, 0, 0)$ indicates that the PM is empty and there is no task either in the queue or in the instantiation unit. The arrival rate to each PM in the hot pool is given by:

$$\lambda_h = \frac{\lambda_{st}(1 - BP_q)}{N_h} \tag{6.20}$$

in which N_h is the number of PMs in the hot pool.

The state transition in VMPSM can occur due to super-task arrival, task instantiation or service completion. From state $(0, 0, 0, 0)$, system can move to state $(0, 1, 0, 0)$ with rate λ_h; at this state, because the instantiation unit is free the super-task will go under

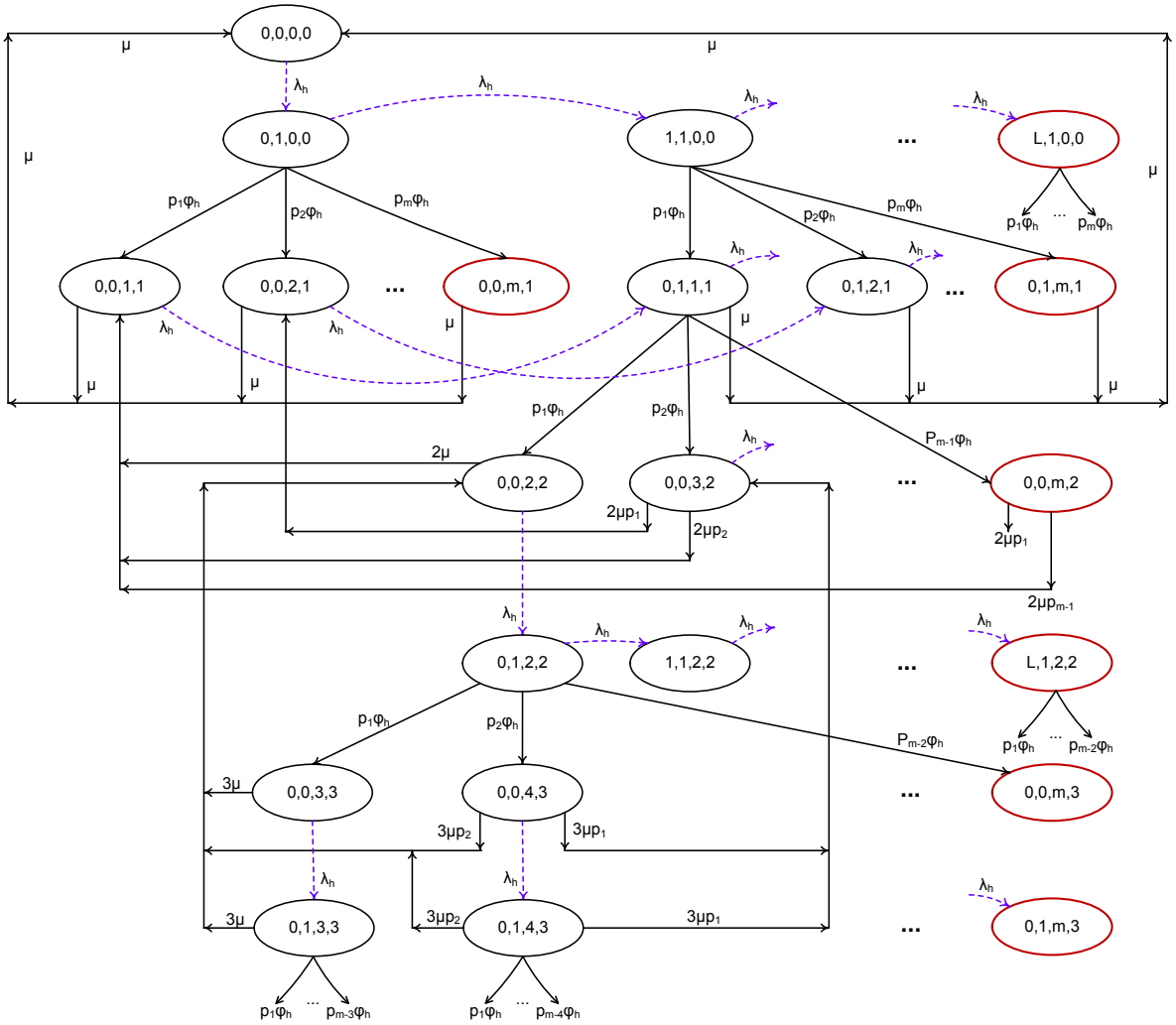


Figure 6.13: Virtual machine provisioning sub-model for a PM in the hot pool.

provisioning immediately. From $(0,1,0,0)$, system may transit to $(1,1,0,0)$ upon arrival of another super-task. From state $(0,1,0,0)$ system also may move to one of the states $\{(0,0,1,1),(0,0,2,1),\cdots,(0,0,m,1)\}$ with the rate of $p_i\phi_h$. Probability p_i is the probability by which a super-task may request i cores for a single VM. From states $\{(0,0,1,1),(0,0,2,1),\cdots,(0,0,m,1)\}$ system can get back to $(0,0,0,0)$ with rate μ (i.e.,

service completion rate), or again upon arriving a super-task, system can move to one of the states $\{(0,1,1,1),(0,1,2,1),\cdots,(0,1,m,1)\}$ with rate λ_h. Now consider the state $(0,0,3,2)$ indicates two VMs are running in which one of them employs one core and the other one engages two cores. System can move to $(0,0,2,1)$ or $(0,0,1,1)$ with rate $2\mu p_1$ (i.e., completed VM releases one core) or $2\mu p_2$ (i.e., completed VM releases two cores) respectively. Also, system may move to $(0,1,3,2)$ upon arrival of another super-task.

Suppose that $\pi^h_{(i,j,k,l)}$ is the steady-state probability for the hot PM model (Fig. 6.13) to be in the state (i,j,k,l). Using steady-state probabilities, we can obtain the probability that at least on PM in hot pool can accept the super-task for provisioning. At first we need to compute the probability that a hot PM cannot admit a super-task for provisioning (P^h_{na}). Let m be the total number of cores on a PM and $MCS \leq m$ be the maximum number of cores that can be requested by a super-task. Since each VM requires at least one vCPU (i.e., one core), the maximum number of VMs on a single PM (MSS) is not going beyond m. In Fig. 6.13, let $F_h(i,j,k,l) = m-(i+j+l)$ and $G_h(i,j,k,l) = m-k$ be the free capacity and cores at each state respectively. P^h_{na} is then given by:

$$P^h_{na} = \sum_{x\in\xi}\sum_{t=F_h+1}^{MSS}\pi^h_x P^*_t + \sum_{y\in\zeta}\sum_{s=G_h+1}^{MCS}\pi^h_y p_s \tag{6.21}$$

where P^*_t is the probability of arrival a super-task with size t and ξ is a set of states in which the system might block an arrival super-task due to lack of space:

$$\xi = \{(i,j,k,l)|\ F_h(i,j,k,l) < MSS\}$$

and p_s is the probability by which a super-task may request for s cores and ζ is a set of states in which the system might block an arrival super-task due to lack of free cores:

$$\zeta = \{(i,j,k,l)|\ G_h(i,j,k,l) < MCS\}$$

Therefore, probability of successful provisioning (P_h) in the hot pool can be obtained as $P_h = 1 - (P_{na}^h)^{N_h}$. where N_h is the number of PMs in the hot pool. Note that P_h is used as an input parameter in SASM. The provisioning model for warm PM is almost the same with the one for hot PM, though, there are some differences in parameters:

(a) The arrival rate to each PM in warm pool is

$$\lambda_w = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)}{N_w} \tag{6.22}$$

(b) Every PM in warm pool requires extra time to be ready for first instantiation. This time is assumed to be exponentially distributed with mean value of $1/\gamma_w$.

(c) The first instantiation is occurred by rate ϕ_w and the rest by ϕ_h.

(d) The specifications of PMs in the warm pool are different from hot pool. (i.e., MSS, MCS and m)

Like VMPSM for a hot PM (Fig. 6.13), the model for a warm PM is solved and the steady-states probabilities ($\pi^w_{(i,j,k,l)}$), are obtained. The success probability for provisioning a super-task in warm pool is $P_w = 1 - (P_{na}^w)^{N_w}$. A PM in the cold pool can be modeled as in the warm pool but with different arrival rate, instantiation rate and start up time. The effective arrival rate at each PM in the cold pool is given by:

$$\lambda_c = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)(1 - P_w)}{N_c} \tag{6.23}$$

A cold PM (off machine), requires longer time than a warm PM to be ready (hot) for the first instantiation. We assume that the start up time is also exponentially distributed with mean value of $1/\gamma_c$. The success probability for provisioning a super-task in the cold pool

is $P_c = 1 - (P_{na}^c)^{N_c}$. From VMPSM, we can also obtain the mean waiting time at PMs' queue ($\overline{PM_{wt}}$) and mean provisioning time ($\overline{PM_{pt}}$) by using the same approach as the one that led to Eq. (6.5). As a result, the total delay before starting of actual service time, is given by:

$$\overline{td} = \overline{wt} + \overline{lut} + \overline{PM_{wt}} + \overline{PM_{pt}} \tag{6.24}$$

Note that all success probabilities (i.e., P_h, P_w and P_c) are input parameters to the server allocation sub-model.

6.4.2 Interaction among Sub-Models

The interactions among sub-models is depicted in Fig. 6.14. VM provisioning sub-models (VMPSM) compute the steady state probabilities (P_h, P_w and P_c) that at least one PM in a pool (hot, warm and cold, respectively) can accept a super-task for provisioning. These probabilities are used as input parameters to the server allocation sub-model (SASM). Hot PM sub-model (VMPSM$_{hot}$) computes P_h which is the input parameter for both warm and cold sub-models; warm PM sub-model (VMPSM$_{warm}$) computes P_w which is the input parameter for the cold sub-model. The server allocation sub-model (SASM) computes the blocking probability, BP_q, which is the input parameter to VM provisioning sub-models. As can be seen, there is an inter-dependency among performance sub-models. This cyclic dependency is resolved via fixed-point iterative method using a modified version of successive substitution approach.

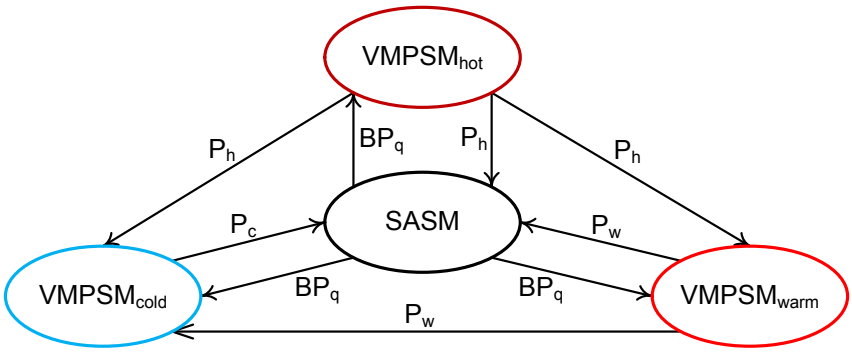


Figure 6.14: Interaction among sub-models.

6.5 Numerical Validation: Heterogeneous Centers

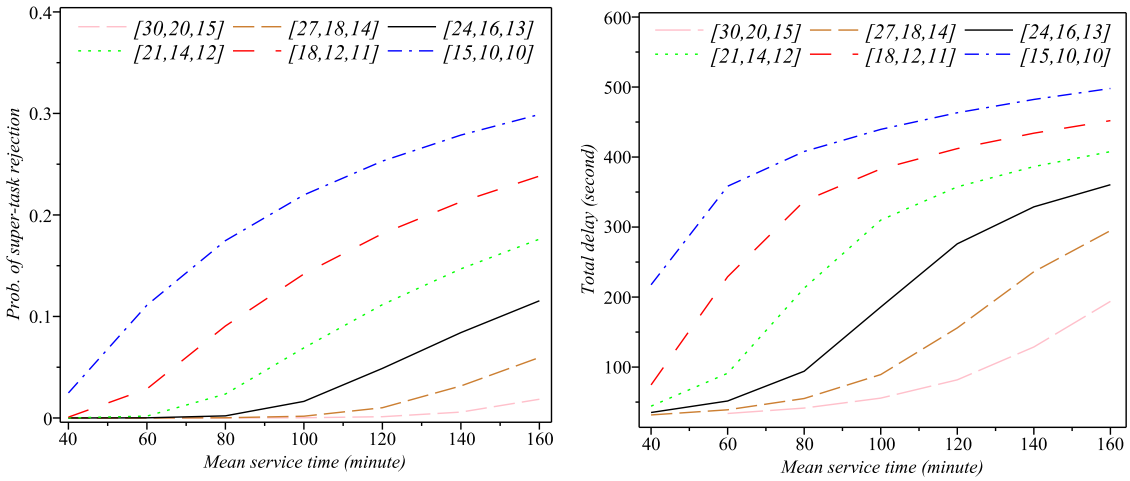
The successive substitution method is continued until the difference between the previous and current value of blocking probability in global queue (i.e., BP_q) is less than 10^{-6}. The integrated model converges to the solution in 8 iterations or less. Table 6.8 shows the specifications of PMs in each pool that we use for the numerical validation.

Table 6.8: Specifications of PMs in each pool.

Parameters	**Hot Pool**	**Warm Pool**	**Cold Pool**
# of CPU cores	10	8	4
RAM (GB)	40	25	10
Disk (GB)	500	250	200
Max # of VMs	10	8	2
# of cores per vCPU	1,2,3 or 4	1,2 or 4	2 or 4

We show the effects of changing arrival rate, task service time, number of PMs in each pool, number of VMs on each PM and super-task size on the interested performance indicators. Arrival rate ranges from 400 to 1000 STs per hour. Mean service time of each task within STs ranges form 40 to 160 minutes. we assume 35 to 65 PMs in pools. The

look-up time for finding a proper PM is assumed to be independent of the number of PMs in the pool and the type of pool (i.e., hot, warm and cold). Look-up rate is set to 625 and 825 searches per hour for warm and cold pools. Mean preparation delay for a warm and cold PM to become a hot PM (i.e., be ready for first VM instantiation) are assumed to be 1 to 3 and 5 to 10 minutes, respectively. Mean time to deploy a VM on a hot PM is assumed to be between 4 and 10 minutes. First VM instantiation on a warm and cold PM are to be between 4 to 7 and 8 to 12 minutes, respectively. After first VM instantiation on a warm or cold PM, it has already become a hot PM so that the next VM can be deployed just like a hot PM. The size of global queue is set to 100 super-tasks (ST) and the number of cores per vCPU is assumed to be uniformly distributed.



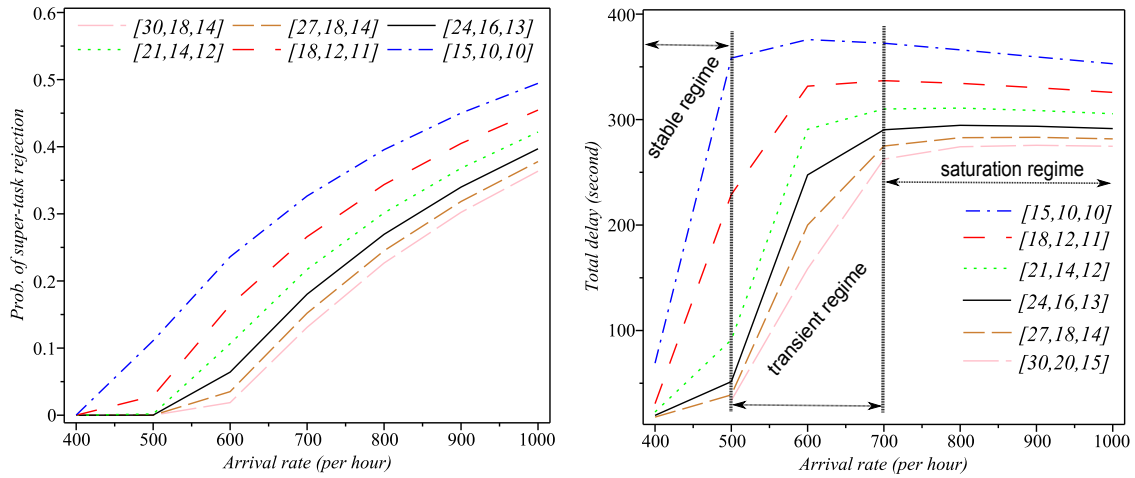
(a) Rejection probability.

(b) Total delay on a super-task.

Figure 6.15: arrival-rate=500 tasks/hour.

In the first experiment, the results of which is shown in Figs. 6.15(a) and 6.15(b), STs have one task each, and PMs are permitted to run only one VM. Fig. 6.15(a) shows (at

a constant arrival rate of 500 STs/hr and different numbers of PMs in each pool) that by increasing the mean service time the rejection probability increases almost linearly. The configuration [21,14,12] seems to be a reasonable choice to maintain the rejection probability below 15%. In addition, it can be observed that by increasing the capacity of system (i.e., having more PMs in the pools) the maximum gain occurs at the longest service time (i.e. 160 minutes). Fig. 6.15(b) shows that a longer service time will result in longer total delay on STs. Unlike the rejection probability, Fig. 6.15(a), there is no unique service time that yield the maximum gain for all pool configurations.



(a) Rejection probability.

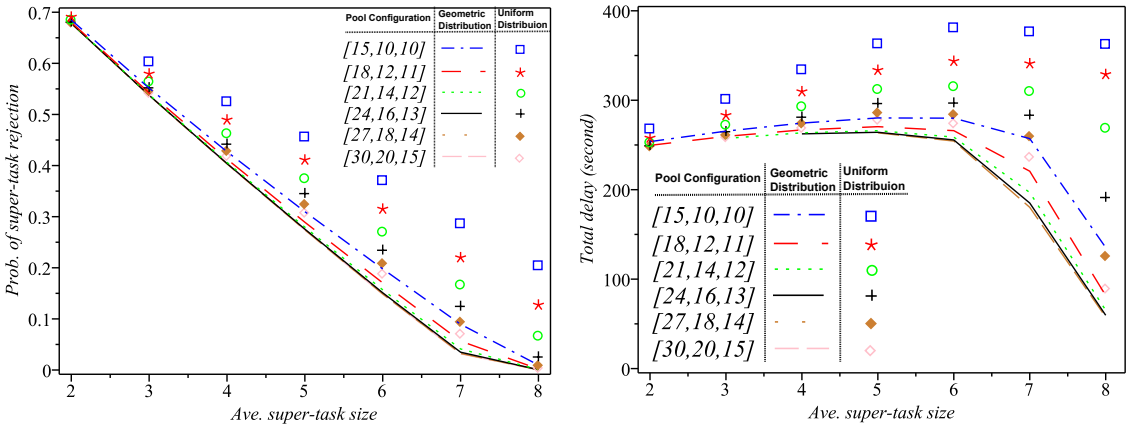
(b) Total delay on a super-task.

Figure 6.16: service-time=60 minute.

Under different arrival rates, we also determine the two performance metrics while the average service time is set to 60 min, Figs. 6.16(a) and 6.16(b). One turning point can be noticed in rejection probability curves at which rejection probability increases suddenly. Such points, e.g., Fig. 6.16(a), 500 STs/hour for configuration [27,18,14], may suggest

the appropriate time to upgrade or inject more resources (i.e., PMs). However, in case of total delay, two turning points (three regimes of operation) can be identified, Fig. 6.16(b): the first regime (i.e., stable regime) is the region in which the cloud providers would like to operate. (Admission control may use provided information here and helps the cloud centers to operate in such regime by not accepting extra requests) Transient regime is in between two turning points which the total delay increases sharply. Cloud centers should avoid entering such region since the violation of SLA is about to happen. As the transient regime is not too narrow, it is possible for cloud center to return to stable zone after a short time (i.e., the admission control has enough time to adjust the new admissions). The transient regime is attributed to the size of global queue. The longer the global queue is, the wider the transient regime is going to be. In saturation regime, the total delay for some configurations such as [15,10,10], [18,12,11] and [21,14,12] is even getting decreased at the expense of high rejection rate of super-tasks. This area of operation is totally unpleasant for both cloud providers and users since the cloud center is mostly unavailable for majority of the users.

In the last experiment, Figs. 6.17(a) and 6.17(b), we examine the effect of super-task size on task rejection probability and total delay using of geometric (shown with lines) and uniform (shown with symbols) distributions for super-task size. Each VM will be assigned one vCPU and each vCPU may use multiple cores (details in Table 6.8). Number of cores per vCPU is assumed to be uniformly distributed. A super-task may request up to all of the available VMs and cores on a PM. Note that the aggregate arrival rate is set to 3500 tasks/hr, however the effective arrival rate of super-tasks is various for different super-task size. For instance, if the average super-task size was five, then the effective arrival rate of super-tasks would be 700 STs/hr. As can be seen by increasing the size of super-tasks the



(a) Rejection probability.

(b) Total delay on a super-task.

Figure 6.17: service-time=60 minute, aggregate arrival-rate=4500 tasks/hour.

rejection probability reduces sharply for both geometric and uniform distribution. Since the size of super-tasks is truncated to the maximum number of VMs allowed on each PM (here, up to 8 VMs), the bigger super-task size will result in the lower number of arrivals as well as lower deviation of super-task size. As can be seen, by increasing the capacity the maximum delay occurrs when the mean size of STs is 6, and the maximum reduction of delay is obtained when the mean size of STs is 8. For STs of size 6 or bigger, the dominant imposed delays are the processing delays (i.e, resource assigning and VM provisioning delays) as opposed to the queuing delays. Due to higher deviation, the uniform distribution imposes more rejections and delay on super-tasks.

6.6 Summary of the Chapter

In this Chapter, we have developed an interacting analytical model that captures important aspects including resource assigning process, virtual machine deployment, pool management and power consumption of nowadays cloud centers. The performance model can assist cloud providers to predict the expected servicing delay, task rejection probability, steady-state arrangement of server pools and power consumption. We carried out extensive numerical experiments to study the effects of various parameters such as arrival rate of super-tasks, task service time, virtualization degree, super-task size and pool check rate on the task rejection probability, response time and normalized power consumption. The behavior of cloud center for given configurations has been characterized in order to facilitate the capacity planning, SLA analysis, cloud economic analysis and trade offs by cloud service providers. Using the proposed pool management model, the most appropriate arrangement of server pools and the amount of required electricity power can be identified in advance for anticipated arrival process and super-task characteristics.

We have also presented a performance model suitable for cloud computing centers with heterogeneous requests and resources using interacting stochastic models. More specifically, a user task may request different types of VMs; VMs can be varied in terms of CPU cores per virtual CPU. Also, unlike previous performance models that have been presented in Chapters 2, 3, 4 and 5, our model in this Chapter can support heterogeneous PMs.

Chapter 7

Summary and Future Work

This chapter concludes the dissertation with a brief summary of contributions and some promising directions for future research.

7.1 Contributions

The main contribution of this thesis is developing performance models for cloud computing centers. We started off with a basic analytical model and then gradually extended the scope of the model in order to capture most important aspects of nowadays cloud centers. The course steps of the performance model development process are as follows:

We developed the analytical model based on queuing theory. First, we adopted $M/G/m$ queuing system as the abstract model for performance evaluation due to the characteristics of cloud computing centers: having Poisson arrival of task requests, generally distributed service time and large number of servers. Since there wasn't any solution to be accurate and efficient enough in case of large m at first place and large value of coefficient variation

(more than one), We develop our own approach to characterize the queuing system and thus the cloud center. Next We extended our model to incorporate the cloud centers which have hyper-exponentially family (those distributions that have coefficient of variation more than one) distribution of service time. Later on we added an assumptions of finite capacity for cloud center, which made our model closer to a real cloud centers. So we incorporate necessary assumptions that are required for having a real performance model of cloud centers:

(i) Poisson arrival process

(ii) Single task arrival

(iii) Generally distributed task service time

(iv) Hyper/Hypo exponentially distributed task service time)

(v) Scale (Large number of servers)

For the first time, we employed $M^{[x]}/G/m/m + r$ queuing system, which is our abstract model, in order to deal with the performance evaluation of a cloud center under batch arrival (super-tasks). We assumed generally distributed batch size, Markovian arrival process, generally distributed service time, large number of servers and a finite capacity of input buffer for task requests. More specifically, the performance model was based on a two-stage approximation technique where the original non-Markovian process is first modeled with an embedded semi-Markov process, which is then modeled by an approximate embedded Markov process but only at the time instants of super-task arrivals; the number of departures between two successive super-task arrivals is counted approximately. This

technique provides quite accurate computation of important performance indicators such as the mean number of tasks in the system, queue length, mean response and waiting time, blocking probability and the probability of immediate service. While the model is approximate, its accuracy is more than sufficient under a wide range of input parameter values; in particular, it allows accurate modeling of cloud centers with a large number of PMs.

We have extended our analytical model for performance evaluation of highly virtualized cloud computing centers. Performance evaluation is particularly challenging in scenarios where virtualization is used to provide a well defined set of computing resources to the users and even more so when the degree of virtualization, i.e., the number of virtual machines (VMs) running on a single physical machine (PM) is high. We permitted each PM to run up to 200 VMs, just like state-of-the-art PMs, and considered the effects of virtualization on PMs performance based on the real data.

Our experience with previous performance models revealed that a monolithic model may suffer from intractability and poor scalability due to large number of parameters. The model ought to cover vast parameter space while it is still tractable. As a result, we developed and evaluated tractable functional sub-models and their interaction model while iteratively solve them. We constructed separate sub-models for different servicing steps in a complex cloud center and the overall solution obtain by iteration over individual sub-model solutions. We assumed that the cloud center consists of a number of PM that are allocated to users in the order of task arrivals. More specifically, user requests may share a PM using virtualization technique. Since many of the large cloud centers employ virtualization to provide the required resources such as PMs, we consider PMs with a high degree of virtualization.

We have extended our proposed interacting analytical model to capture important aspects including pool management, power consumption, resource assigning process and virtual machine deployment of nowadays cloud centers. The performance model can assist cloud providers to predict the expected servicing delay, task rejection probability, steady-state arrangement of server pools and power consumption. We carried out extensive numerical experiments to study the effects of various parameters such as arrival rate of super-tasks, task service time, virtualization degree, super-task size and pool check rate on the task rejection probability, response time and normalized power consumption. The behavior of cloud center for given configurations has been characterized in order to facilitate the capacity planning, SLA analysis, cloud economic analysis and trade offs by cloud service providers. Using the proposed pool management model, the most appropriate arrangement of server pools and the amount of required electricity power can be identified in advance for anticipated arrival process and super-task characteristics.

Finally, we have also presented a performance model suitable for cloud computing centers with heterogeneous requests and resources using interacting stochastic models. More specifically, a user task may request different types of VMs; VMs can be varied in terms of CPU cores per virtual CPU. Also, unlike previous performance models that have been presented in Chapters 2, 3, 4 and 5, our final performance model (last part of 6) can support heterogeneous PMs.

7.2 Future Work

As we just discussed in the Section 7.1, the obtained results gave us enough motivation to consider following as future research topics to be investigated as a continuation of this

Ph.D. thesis.

We plan to extend the heterogeneity of our performance model to other parts of IaaS cloud computing centers. We would like to support full heterogeneity in VMs, PMs and user tasks; more specifically, VMs are required to be varied in terms of memory, disk, CPU core and vCPU; PMs may differ in computation capacity, networking, memory, disk, Graphics Processing Unit (GPU) and hypervisor (VMM) capabilities; user tasks may request various number of resources for different amount of time (i.e., different probability distributions for task service time).

Assuming heterogeneous environment brings another challenge which is known as decision making of task assignment policy. Although this issue has been extensively studied in the area of cluster and grid computing, proposed techniques are not quite applicable/useful to cloud computing paradigm due to different nature of cloud architecture and user requests. One direction of future research can be proposing an efficient task assignment strategy whereby the utilization of cloud resources as well as the performance indicators (e.g., waiting time, response time, blocking probability and probability of immediate service) are improved simultaneously.

Bibliography

[1] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, and J. S. Vetter. Early evaluation of IBM BlueGene. *2008 SC International Conference for High Performance Computing Networking Storage and Analysis*, pages 1–12, May 2008.

[2] Amazon Elastic Compute Cloud. *User Guide*. Amazon Web Service LLC or its affiliate, August 2012.

[3] An amazon.com company. Amazon Elastic Compute Cloud, Amazon EC2. Website, Last accessed October 2012. `http://aws.amazon.com/ec2`.

[4] T. M. Apostol. *Mathematical analysis*. Addison Wesley, 2nd edition, 1974.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[6] A. Baig. UniCloud Virtualization Benchmark Report. white paper, March 2010. `http://www.oracle.com/us/technologies/linux/intel-univa-virtualization-400164.pdf`.

[7] J. Baker, C. Bond, J. Corbett, J. J. Furman, A. Khorlin, J. Larsonand, J. M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. In *Conference on Innovative Data Systems Research (CIDR)*, pages 223–234, January 2011.

[8] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Trans. on Comput.*, 39:1232–1250, October 1990.

[9] S. C. Borst. Optimal probabilistic allocation of customer types to servers. *SIGMETRICS Performance Evaluation Rev.*, 23:116–125, May 1995.

[10] O. J. Boxma, J. W. Cohen, and N. Huffel. Approximations of the mean waiting time in an $M/G/s$ queueing system. *Operations Research*, 27:1115–1127, 1979.

[11] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *IEEE International Conference on High Performance Computing and Communications*, pages 5–13, September 2008.

[12] J. Cao, W. Zhang, and W. Tan. Dynamic control of data streaming and processing in a virtualized environment. *IEEE Transactions on Automation Science and Engineering*, 9(2):365–376, April 2012.

[13] A.L.E. Corral-Ruiz, F.A. Cruz-Perez, and G. Hernandez-Valdez. Teletraffic model for the performance evaluation of cellular networks with hyper-erlang distributed

cell dwell time. In *71st IEEE Vehicular Technology Conference (VTC 2010-Spring)*, pages 1–6, May 2010.

[14] G. P. Cosmetatos. Some practical considerations on multi-server queues with multiple poisson arrivals. *Omega*, 6(5):443–448, 1978.

[15] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. *International Conference for High Performance Computing Networking Storage and Analysis*, C(November):1–12, February 2008.

[16] T. V. T. Duy, Y. Sato, and Y. Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, April 2010.

[17] D. Efrosinin and V. Rykov. On performance characteristics for queueing systems with heterogeneous servers. *Automation and Remote Control*, 69:61–75, 2008.

[18] A. Federgruen and L. Green. An M/G/c queue in which the number of servers required is random. *Journal of Applied Probability*, 21(3):583–601, 1984.

[19] D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Online Book, June 2012. Version 0.36.

[20] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini. QoS-aware clouds. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 321–328, July 2010.

[21] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, GCE '08*, pages 1–10, November 2008.

[22] J. Fu, W. Hao, M. Tu, B. Ma, J. Baldwin, and F.B. Bastani. Virtual services in cloud computing. In *IEEE 2010 6th World Congress on Services*, pages 467–472, June 2010.

[23] B. Furht. Cloud Computing Fundamentals. In *Handbook of Cloud Computing*, pages 3–19. Springer US, 2010.

[24] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, September 2010.

[25] D. F. García, J. García, J. Entrialgo, M. García, P. Valledor, R. García, and A. M. Campos. A QoS control mechanism to provide service differentiation and overload protection to internet scalable servers. *IEEE Transaction on Service Computing*, 2:3–16, January 2009.

[26] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Quantifying resiliency of IaaS cloud. *IEEE Symposium on Reliable Distributed Systems*, 0:343–347, January 2010.

[27] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi. Modeling and performance analysis of large scale iaas clouds. *Future Generation Computer Systems*, July 2012.

[28] R. Ghosh, K. S. Trivedi, V. K. Naik, and D. S. Kim. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models ap-

proach. In *Proceedings of the 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, pages 125–132, December 2010.

[29] GOGRID. GoGrid Cloud. Website, Last accessed July 2012. `http://www.gogrid.com`.

[30] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 3rd edition, Jul 2010.

[31] E. Hand. Head in the clouds. *Nature*, 449(7165):963–963, October 2007.

[32] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd. Allocating non-real-time and soft real-time jobs in multiclusters. *IEEE Trans. Parallel Distrib. Syst.*, 17:99–112, February 2006.

[33] Hewlett-Packard Development Company, Inc. An overview of the VMmark benchmark on HP Proliant servers and server blades. white paper, May 2012. `ftp://ftp.compaq.com/pub/products/servers/benchmarks/VMmark_Overview.pdf`.

[34] D. P. Heyman and M. J Sobel. *Stochastic Models in Operations Research*, volume 1. Dover, 2004.

[35] P. Hokstad. Approximations for the $M/G/m$ queues. *Operations Research*, 26:510–523, 1978.

[36] IBM. IBM Cloud Computing. Website, Last accessed July 2012. `http://www.ibm.com/ibm/cloud/`.

[37] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, June 2011.

[38] D. N. Joanes and C. A. Gill. Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1):183–189, 1998.

[39] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal load balancing in distributed computer systems*. Springer, 1997.

[40] H. Khazaei, J. Mišić, and V. B. Mišić. Performance analysis of cloud computing centers. In *7th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine)*, November 2010.

[41] H. Khazaei, J. Mišić, and V. B. Mišić. Modeling of cloud computing centers using $M/G/m$ queues. In *The First International Workshop on Data Center Performance*, March 2011.

[42] H. Khazaei, J. Mišić, and V. B. Mišić. On the performance and dimensioning of cloud computing centers. In Lizhe Wang, Rajiv Ranja, Jinjun Chen, and Boualem Benatallah, editors, *Cloud computing: methodology, system, and applications*, chapter 8, pages 151–165. FL: CRC Press, 2011.

[43] H. Khazaei, J. Mišić, and V. B. Mišić. Performance analysis of cloud centers un-

der burst arrivals and total rejection policy. In *IEEE Global Telecommunications Conference (GLOBECOM 2011)*, pages 1–6, December 2011.

[44] H. Khazaei, J. Mišić, and V. B. Mišić;. A fine-grained performance model of cloud computing centers. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1, 2012.

[45] H. Khazaei, J. Mišić, and V. B. Mišić. Performance analysis of cloud computing centers using $M/G/m/m + r$ queueing systems. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):1, 2012.

[46] H. Khazaei, J. Mišić, and V. B. Mišić. Performance evaluation of cloud data centers with batch task arrivals. In Hussein T. Mouftah and Burak Kantarci, editors, *Communication Infrastructures for Cloud Computing: Design and Applications*, chapter X, pages X–Y. IGI Global, December 2012.

[47] H. Khazaei, J. Mišić, and V. B. Mišić. Performance of cloud centers with high degree of virtualization under batch task arrivals. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1, 2012.

[48] H. Khazaei, J. Mišić, V. B. Mišić, and N. Beigi-Mohammadi. Availability analysis of cloud computing centers. In *Globecom 2012 - Communications Software, Services and Multimedia Symposium (GC12 CSSM)*, December 2012.

[49] H. Khazaei, J. Mišić, V. B. Mišić, and N. Beigi-Mohammadi. Modeling the performance of heterogeneous IaaS cloud centers. In *Submitted to the 3rd international workshop on Data Center Performance*, July 2013.

[50] H. Khazaei, J. Mišić, Vojislav B. Mišić, and S. Rashwand. Analysis of a pool management scheme for cloud computing centers. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints), 2012.

[51] S. Kieffer, W. Spencer, A. Schmidt, and S. Lyszyk. Planning a Data Center. white paper, February 2003. `http://www.nsai.net/White_Paper-Planning_A_Data_Center.pdf`.

[52] D. S. Kim, F. Machida, and K. S. Trivedi. Availability modeling and analysis of a virtualized system. In *Proceedings of the 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, PRDC '09, pages 365–371, November 2009.

[53] T. Kimura. Diffusion approximation for an $M/G/m$ queue. *Operations Research*, 31:304–321, 1983.

[54] T. Kimura. Optimal buffer design of an $M/G/s$ queue with finite capacity. *Communications in Statistics and Stochastic Models*, 12(6):165–180, 1996.

[55] T. Kimura. A transform-free approximation for the finite capacity $M/G/s$ queue. *Operations Research*, 44(6):984–988, 1996.

[56] T. Kimura and T. Ohsone. A diffusion approximation for an $M/G/m$ queue with group arrivals. *Management Science*, 30(3):381–388, 1984.

[57] L. Kleinrock. *Queueing Systems, Volume 1, Theory*. Wiley-Interscience, 1975.

[58] L. Kleinrock. A vision for the internet. *ST Journal of Research*, 2:4–5, November 2005.

[59] K. Li. Optimizing average job response time via decentralized probabilistic job dispatching in heterogeneous multiple computer systems. *The Computer Journal*, 41(4):223–230, May 1998.

[60] K. Li. Minimizing the probability of load imbalance in heterogeneous distributed computer systems. *Mathematical and Computer Modelling*, 36(9-10):1075 – 1084, December 2002.

[61] K. Li. Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments. *J. Syst. Archit.*, 54:111–123, January 2008.

[62] K. Li. Optimal load distribution for multiple heterogeneous blade servers in a cloud computing environment. *IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 0:948–957, 2011.

[63] F. Longo, R. Ghosh, V. K. Naik, and K. S. Trivedi. A scalable availability model for infrastructure-as-a-service cloud. *Dependable Systems and Networks, International Conference on*, pages 335–346, June 2011.

[64] B. N. W. Ma and J. W. Mark. Approximation of the mean queue length of an $M/G/c$ queueing system. *Operations Research*, 43:158–165, 1998.

[65] V. Mainkar and K. S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *Software Engineering, IEEE Transactions on*, 22(9):640–653, September 1996.

[66] Maplesoft, Inc. Maple 15. Website, Last accessed March 2011. `http://www.maplesoft.com`.

[67] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44(3):205–216, March 2009.

[68] M. Miyazawa. Approximation of the queue-length distribution of an $M/GI/s$ queue by the basic equations. *Journal of Applied Probability*, 23:443–458, 1986.

[69] H. J. Moon, Y. Chi, and H. Hacigümüs. SLA-aware profit optimization in cloud services via resource scheduling. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 152–153, July 2010.

[70] NephoScale. The NephoScale Cloud Servers. Website, Last accessed July 2012. `http://www.nephoscale.com/nephoscale-cloud-servers`.

[71] S. A. Nozaki and S. M. Ross. Approximations in finite-capacity multi-server queues with poisson arrivals. *Journal of Applied Probability*, 15:826–834, 1978.

[72] E. Page. Tables of waiting times for $M/M/n$, $M/D/n$ and $D/M/n$ and their use to give approximate waiting times in more general queues. *J. Operational Research Society*, 33:453–473, 1982.

[73] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids: a viable solution? *Proceedings of the 2008 international workshop on Dataaware distributed computing DADC 08*, pages 55–64, January 2008.

[74] A. Patrizio. IDC sees cloud market maturing quickly. *Datamation*, March 2011.

[75] Rackspace. The Rackspace Cloud. Website, Last accessed July 2012. `http://www.rackspace.com/cloud/`.

[76] C. G. Rommel. The probability of load balancing success in a homogeneous network. *IEEE Trans. Softw. Eng.*, 17:922–933, September 1991.

[77] A.L. Rosenberg and R.C. Chiang. Toward understanding heterogeneity in computing. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–10, April 2010.

[78] K. W. Ross and D. D. Yao. Optimal load balancing and scheduling in a distributed computer system. *J. ACM*, 38:676–689, July 1991.

[79] RSoft Design. *Artifex v.4.4.2*. RSoft Design Group, Inc., 2003.

[80] S. Saini, D. Talcott, D. Jespersen, J. Djomehri, H. Jin, and R. Biswas. Scientific application-based performance comparison of sgi altix 4700, ibm power5+, and sgi ice 8200 supercomputers. *2008 SC International Conference for High Performance Computing Networking Storage and Analysis*, pages 1–12, December 2008.

[81] SearchDataCenter.com. The data center purchasing intentions survey report. Special Report, August 2008. `http://searchdatacenter.techtarget.com`.

[82] J. Sethuraman and M. S. Squillante. Optimal stochastic scheduling in multiclass parallel queues. *SIGMETRICS Performance Evaluation Rev.*, 27:93–102, May 1999.

[83] B. A. Shirazi, K. M. Kavi, and A. R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. Wiley-IEEE Computer Society Press, 1995.

[84] G. Somani and S. Chaudhary. Application performance isolation in virtualization.

In *IEEE International Conference on Cloud Computing, CLOUD '09.*, pages 41–48, September 2009.

[85] H. Takagi. *Queueing Analysis*, volume 1: Vacation and Priority Systems. North-Holland, 1991.

[86] H. Takagi. *Queueing Analysis*, volume 2: Finite Systems. North-Holland, 1993.

[87] Y. Takahashi. An approximation formula for the mean waiting time of an $M/G/c$ queue. *J. Operational Research Society*, 20:150–163, 1977.

[88] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, ICPP '00, page 373, May 2000.

[89] A. N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32:445–465, April 1985.

[90] T. Thein and J. S. Park. Availability analysis of application servers using software rejuvenation and virtualization. *J. Comput. Sci. Technol.*, 24:339–346, March 2009.

[91] H. C. Tijms. Heuristics for finite-buffer queues. *Probability in the Engineering and Informational Sciences*, 6:277–285, 1992.

[92] H. C. Tijms, M. H. V. Hoorn, and A. Federgru. Approximations for the steady-state probabilities in the $M/G/c$ queue. *Advances in Applied Probability*, 13:186–206, 1981.

[93] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, second edition, July 2001.

[94] L. M. 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), June 2009.

[95] VMware, Inc. VMmark: A Scalable Benchmark for Virtualized Systems. Technical Report, September 2006. `http://www.vmware.com/pdf/vmmark_intro.pdf`.

[96] VMware, Inc. VMware VMmark 2.0 benchmark results. Website, Last accessed September 2012. `http://www.vmware.com/a/vmmark/`.

[97] J. Voas and J. Zhang. Cloud computing: New wine or just a new bottle? *IT Professional*, 11(2):15–17, March-April 2009.

[98] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *LOGIN*, 33(5):18–23, October 2008.

[99] L. Wang, G. von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu. Cloud computing: a perspective study. *New Generation Computing*, 28:137–146, 2010.

[100] L. Wang, J. Zhan, W. Shi, Y. Liang, and L. Yuan. In cloud, do mtc or htc service providers benefit from the economies of scale? *Proceedings of the 2nd Workshop on ManyTask Computing on Grids and Supercomputers MTAGS 09*, 2:1–10, December 2010.

[101] K. Xiong and H. Perros. Service performance and analysis in cloud computing. In *IEEE 2009 World Conference on Services*, pages 693–700, February 2009.

[102] B. Yang, F. Tan, Y. Dai, and S. Guo. Performance evaluation of cloud service considering fault recovery. In *First Int'l Conference on Cloud Computing (CloudCom) 2009*, pages 571–576, December 2009.

[103] D. D. Yao. Refining the diffusion approximation for the $M/G/m$ queue. *Operations Research*, 33:1266–1277, 1985.

[104] D. D. Yao. Some results for the queues $M^x/M/c$ and $GI^x/G/c$. *Operations Research Letters*, 4(2):79–83, 1985.

[105] K. Ye, X. Jiang, D. Ye, and D. Huang. Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server. In *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 281–288, September 2010.

[106] S. Yeo and H. Lee. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer*, 44:55–62, 2011.

[107] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann. C-meter: A framework for performance analysis of computing clouds. In *CCGRID09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 472–477, April 2009.

[108] L. Youseff, R. Wolski, B. Gorda, and R. Krintz. Paravirtualization for hpc systems. In *Proc. Workshop on Xen in High-Performance Cluster and Grid Computing*, pages 474–486. Springer, March 2006.

[109] L. M. Zhang, K. Li, and Y. Zhang. Green task scheduling algorithms with speeds

optimization on heterogeneous cloud servers. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM'10, pages 76–80, December 2010.