RESOURCE ALLOCATION AND TASK ADMISSION CONTROL IN CLOUD SYSTEMS

by

Haleh Khojasteh

M.Sc. in Computer Science, Ryerson University, Toronto, Canada, 2011

B.Sc. in Computer Engineering, Shahid Beheshti University, Tehran, Iran, 1997

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Program of

Computer Science

Toronto, Ontario, Canada, 2016

Author’s Declaration

Abstract

RESOURCE ALLOCATION AND TASK ADMISSION CONTROL IN CLOUD SYSTEMS

©Haleh Khojasteh 2016

Doctor of Philosophy
Computer Science
Ryerson University

The focus of this thesis is solving the problem of resource allocation in cloud datacenter using an Infrastructure-as-a-Service (IaaS) cloud model. We have investigated the behavior of IaaS cloud datacenters through detailed analytical and simulation models that model linear, transitional and saturated operation regimes. We have obtained accurate performance metrics such as task blocking probability, total delay, utilization and energy consumption. Our results show that the offered load does not offer complete characterization of datacenter operation; therefore, in our evaluations, we have considered the impact of task arrival rate and task service time separately.

To keep the cloud system in the linear operation regime, we have proposed several dynamic algorithms to control the admission of incoming tasks. In our first solution, task admission is based on task blocking probability and predefined thresholds for task arrival rate. The algorithms in our second solution are based on full rate task acceptance threshold and filtering coefficient. Our results confirm that the proposed task admission mechanisms are capable of maintaining the stability of cloud system under a wide range of input parameter values.

Finally, we have developed resource allocation solutions for mobile clouds in which offloading requests from a mobile device can lead to forking of new tasks in on-demand manner. To address this problem, we have proposed two flexible resource allocation mechanisms with different prioritization: one in which forked tasks are given full priority over newly arrived ones, and another in which a threshold is established to control the priority. Our results demonstrate that threshold-based priority scheme presents better system performance than the full priority scheme. Our proposed solution for clouds with mobile users can be also applied in other clouds which their users' applications fork new tasks.

Acknowledgments

There are many people to whom I owe credit for this thesis. Without the support and encouragement of these wonderful people it would not be possible to finish this work.

First and foremost, I express my deepest gratitude to my supervisor, Dr. Jelena Mišić for accepting me as her PhD student and guiding me with patience and enthusiasm. I was greatly inspired by her excitement, vision, and creativity. She has been abundantly helpful and has assisted me in numerous ways. Without her continual support and thoughtful mentoring, many of the ideas in this thesis would never have come to fruition.

Next, I am profoundly indebted to Dr. Vojislav B. Mišić for his generous contribution, great ideas and invaluable advice.

I would also like to appreciate my thesis committee, Dr. Isaac Woungang, Dr. Cherie Ding and Dr. Lian Zhao from Ryerson University and Dr. Ben Liang from University of Toronto for taking the time and making the effort to review this thesis and provide me with their insightful comments.

I am very grateful to my parents for their endless love and support.

Finally, I dedicate this thesis to my husband, Ali Tabatabaei and my son, Hirad who supported me unconditionally and have been the source of motivation and inspiration for me and made my time more enjoyable during these years.

To my beloved husband and son for their support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cloud Computing at a Glance

Cloud computing has become a tremendous paradigm for hosting and delivering services over the Internet. The goal of this computing model is to make a better use of various computer resources, keep together in order to achieve higher throughput and to resolve problems requiring high performance computations. It allows users to develop and leverage available computer resources in pay-as-you go manner.

In this thesis, we adopt the definition of cloud computing provided by The National Institute of Standards and Technology (NIST) [46] as it covers the essential aspects of cloud computing:

NIST definition of cloud computing: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

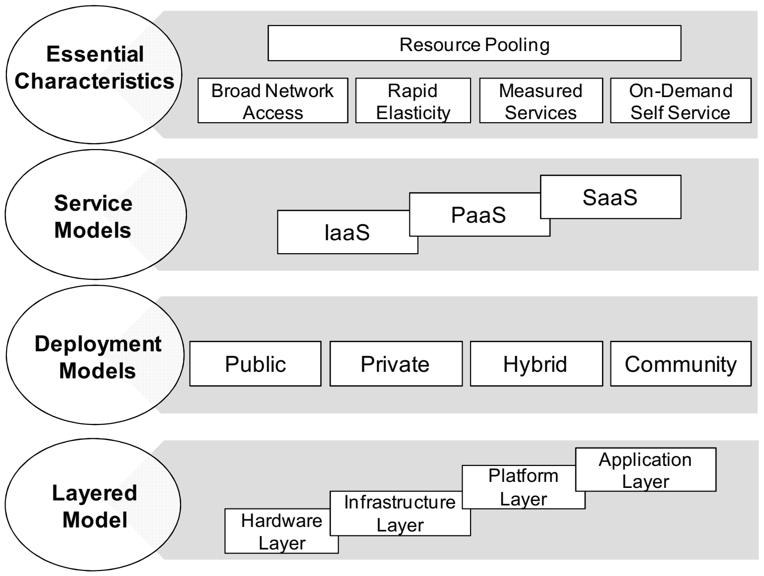


Figure 1.1: The NIST Cloud Model [46].

This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Fig. 1.1 illustrates the cloud's essential characteristics, service models, deployment models and the layers defined by NIST [46].

1.1.1 Essential Characteristics of Cloud

Five essential characteristics of cloud computing is clearly articulated in these definitions [46]:

a) **On-demand self-service:** a user can provision cloud resources, such as server time and network storage, automatically without requiring human interaction with each service provider.

b) **Broad network access:** cloud resources are available over the internet and accessed through diverse terminals such as thin or thick client platforms (e.g., computers, mobile phones, tablets etc.).

c) **Resource pooling:** the computing resources are pooled together to serve multiple consumers in a multi-tenant manner. These physical and virtual resources are assigned based on the users' requirements. User generally has no control or knowledge over the exact location of the provided computer resources, but s/he may be able to specify location at a country, state or datacenter levels. Computer resources provided to users include storage, processing, memory, and network bandwidth.

d) **Rapid elasticity:** computer resources provisioned to customers can be elastically provisioned and released and can be scaled rapidly outward and inward appropriately with demand. User generally is able to scale out resources that appear to be "unlimited" and can be provisioned in any quantity at any time.

e) **Measured service:** computer resources can be automatically controlled and optimized using a metering capability appropriate to the type of service provisioned (e.g., storage, processing, bandwidth, and active user accounts). Resource utilization can be monitored, controlled, and reported for both the provider and user to provide transparency.

1.1.2 The Service Models of Cloud

Following cloud service models have been used to classify the services in cloud community [46]:

I) **Software as a Service (SaaS):** cloud user release their applications on a hosting environment, which can be accessed through internet from diverse terminals (e.g. web browser, PDA, etc.). Cloud users do not have control over the multi-tenant cloud infrastructure. Cloud users' applications are managed in a single virtual environment on the SaaS to leverage optimized amount of resources in terms of availability, speed, security, maintenance and disaster recovery. Some examples of SaaS are SalesForce.com, Google Mail, Google Docs, Workday, Concur, Citrix GoToMeeting and Cisco WebEx.

II) **Platform as a Service (PaaS):** PaaS is a development platform supporting the full "Software Lifecycle" which allows cloud users to develop cloud services and applications (e.g. SaaS) directly on the PaaS cloud. Hence the difference between SaaS and PaaS is that SaaS only hosts completed cloud applications whereas PaaS offers a development platform that hosts both completed and in-progress cloud applications. This requires PaaS, in addition to supporting application hosting environment, to possess development infrastructure including programming environment, tools, configuration management, and so forth. Some examples of PaaS are Google AppEngine, Force.com and Apache Stratos.

III) **Infrastructure as a Service (IaaS):** cloud users directly use IT infrastructures (processing, storage, networks, and other fundamental computing resources) provided in the IaaS cloud. Virtualization is extensively used in IaaS cloud in order to integrate/decompose physical resources in an ad-hoc manner to meet growing or shrinking resource demand from cloud users. The basic strategy of virtualization is to set up independent Virtual Machines (VMs) that are isolated from both the underlying hardware and other VMs. Notice that this strategy is different from the multi-tenancy model, which aims to transform the application software architecture so that multiple instances (from multiple cloud users) can run on a single application (i.e. the same logic machine). Some examples of IaaS are Amazon EC2, Microsoft Azure, Google Compute Engine (GCE), Rackspace and Joyent.

1.1.3 The Deployment Models of Cloud

Four cloud deployment models have been defined in cloud community [46], [52], [1]:

a) **Public cloud:** a cloud that can be used by general public. The cloud may be less secure due to its openness. Public cloud are the dominant form of current cloud computing deployment models and they require significant investment and are usually owned by large corporations such as Microsoft (Azure), Google (AppEngine) or Amazon (EC2, S3).

b) **Private cloud:** a cloud that is exclusively used by one organization, company or one of its customers. The cloud may be operated by itself or a third party. Private cloud offers increased security at greater cost. The St. Andrews Cloud Computing Co-laboratory and Concur Technologies (Lemos, 2009) are illustration associations that have a private cloud.

c) **Community cloud:** a cloud that is shared by two or more several organizations or companies and is usually set up for their specific requirements. This cloud is typically used for a shared concern (e.g. such as schools within a university).

d) **Hybrid cloud:** the cloud infrastructure is a combination of two or more clouds (private, community, or public). In this model, the clouds remain as unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds). Organizations use the hybrid cloud model in order to optimize their resources to increase their core competencies by offloading peripheral business functions onto the cloud while controlling core activities on-premise through private cloud.

1.1.4 The Layered Models of Cloud

Cloud computing architecture is divided into four layers [46]:

I) **The hardware layer** refers to managing the physical resources of the cloud computing infrastructure, including physical servers, routers, switches, power and cooling systems. This layer is implemented in datacenters and contains thousands of servers which are arranged in racks and interconnected through switches, routers or other techniques.

II) **The infrastructure layer** (or the virtualization layer) manages a pool of storage and computing resources by partitioning the physical resources using virtualization techniques (such as Xen, KVM and VMware).

III) **The platform layer** includes operating systems and application frameworks. The aim of this layer is to minimize the burden of deploying applications directly into virtual machine containers.

IV) **The application layer** consists of the cloud applications which can be exploited with the automatic-scaling feature to achieve better performance, availability and lower operating cost.

1.1.5 The Advantages of Cloud Computing

Cloud computing offers numerous advantages both to end users and businesses of all sizes. Some of the key advantages offered by cloud computing are listed as [47]:

- Lower-Cost Computers for Users
- Improved Performance
- Lower IT Infrastructure Costs
- Reducing Business Risks and Maintenance Issues
- Lower Software Costs
- Instant Software Updates
- Increased Computing Power
- Unlimited Storage Capacity
- Increased Data Safety
- Improved Compatibility Between Operating Systems
- Improved Document Format Compatibility
- Easier Group Collaboration
- Universal Access to Documents
- Latest Version Availability
- Removing the Tether to Specific Devices

1.2 Major Challenges in Cloud Computing

Cloud computing indeed provides several compelling features that make it attractive to business owners. However, there are some open challenges which are not completely addressed yet. The major challenges in this field are listed in [56], [1], [53]:

1.2.1 Resource Allocation Issues in Cloud

- **Predictable and unpredictable workloads:** The load of requests send to the cloud may change drastically and increasingly as the demand of applications grows. This type of workload can be classified as unpredictable workload and it is difficult to manage this type of workload.

- **Homogenous and heterogeneous workloads:** Homogenous workloads primarily are hosted on VMs which have similar type of configuration, i.e., required number of CPUs, RAM, storage and mean time required to execute. Whereas, heterogeneous workloads are accommodated on VMs with different configurations. The resource allocation for both types of workloads can be challenging for the cloud system.

- **Batch workloads and transactional workloads:** The difference between batch and transactional workloads is that batch workloads do not need user intervention such as the input from user. But transactional workloads need user intervention. Example of transactional workloads is online transactional systems. Batch workloads are non-preemptive whereas transactional workloads can be preemptive. Batch workloads can be unchanging, but transactional workloads fluctuate. To deal with both types of workloads, an efficient resource allocation algorithm is to be addressed.

- **Elasticity:** Elasticity in cloud means how much it can dynamically handle the fluctuation in the requirements of the resources. The demand for resources may grow in time; the cloud must automatically detect the demand and scale the resources. An efficient resource management in cloud should have a clear solution for this issue.

- **Minimization of costs and maximization of resource utilization:** The two important constraints to be met in cloud resource allocation are the minimization of overall operation cost and maximization of resource utilization. A robust cloud system should provide services to its users with the intention to entrust them with continuing their services. In order to reach this goal, the service provider should be able to give services to the users with low cost. This can be done by having adequate techniques to monitor the resource usage and minimize the cost for the users and maximize the system utilization.

- **VM Migration:** Virtual Machine migration is one of the methods which can be utilized by load balancer or used in energy preservation policies. To accommodate the resources, VMs can be migrated between hosts. VM Migration in cloud computing can be used to increase efficiency, balance load across the datacenters and it enables robust and highly responsive provisioning in datacenters. However, the implementation of VM Migration can be challenging due to the additional overhead of deploying and managing VMs.

- **Handling high availability for long running tasks:** The tasks on cloud can be running for a long time such as many hours or days. Therefore, it is required to have resources which are available for tasks without any interruption or failure. The techniques are to be addressed to automatically detect any failure or unavailability and shift the tasks to the available resources. Moreover, these techniques should handle the situation in a short time and in a way that users do not sense the unavailability.

- **Parallel task scheduling:** Computing tasks in parallel would increase the make span of the task. The tasks can be independent or dependent. Independent tasks can be executed in several virtual machines in parallel. Whereas, dependent tasks can experience communication issues, so they must be scheduled carefully.
- **Networked Cloud:** Cloud datacenters have resource allocation problem within their intra domains. In case of distributed clouds, the resources allocation algorithms suitable for local environments are not compatible with the distributed clouds. Some of the research issues associated with distributed clouds is communication delay, optimal assignment of networked resources, virtualization of networked resources, etc. The problem of mapping virtualized network resources is referred as Virtualized Network Embedding (VNE) problem.
- **Contention among requests:** Resource contention arises when two applications want to access the same resource at the same time. In a decentralized cloud environment, user requests may be stuck in contention while searching for suitable resources.
- **Scarcity of resources:** this issue arises when there are limited resources.
- **Resource fragmentation:** this situation arises when the resources are isolated. There will be enough resources, but they are not available for allocation.

- **Over-provisioning of resources:** This case happens when the application gets more resources than the demanded ones.
- **Under-provisioning of resources:** this issue occurs when the application is assigned to less resources than it requires.

1.2.2 Cloud Security, Privacy and Reliability

Two of the key issues in cloud computing are security and privacy. A company's confidential data must be unavailable and inaccessible by other companies. Companies offering cloud computing services must satisfy security concerns. Customers only pay to these companies when they are convinced in security measures.

Privacy is another issue. As user's data can be accessed from any location, proper authentication techniques should be developed. Using authorization, each user can access only the data and applications relevant to his or her job. Replication time and costs is important for data resiliency.

Reliability is another concern which should be considered in cloud system. Server's stability should be provided to avoid outrage in cloud servers.

1.2.3 Server Consolidation

Server consolidation is an approach to maximize resource utilization while minimizing energy consumption in a cloud computing environment. Some under-utilized servers can be set to an energy-saving state by combining VMs located on several servers onto a single server using VM migration.

1.2.4 Energy Management

Increasing energy efficiency in cloud computing and designing energy-efficient datacenters is one of the important issues in this area. Some solutions to this problem include the development of Energy-efficient hardware architecture which deploys techniques such as slowing down CPUs' speed, turning off partial hardware components, energy-aware job scheduling and server consolidation used to reduce power consumption by turning off unused machines and studying energy-efficient network protocols and infrastructures. These techniques are intended to achieve a trade-off between energy savings and application performance.

1.2.5 Bandwidth

One of the bottlenecks of cloud computing is bandwidth. Without utilizing efficient strategies, the network speed can decrease or the network can halt when a large number of users or some high transaction processes want to contact the datacenter.

1.2.6 Traffic Management and Analysis

Analysis of network traffic is important for datacenters to increase the system's efficiency. Furthermore, network operators need to know how traffic flows through the network in order to make management and planning decisions. Also, it is important to improve network performance in datacenter networks.

1.3 General Issues of Mobile Cloud Computing

Although there are many advantages offered by mobile cloud computing, still there are many problems that hinder mobile cloud's success and adoption; these issues should be seriously tackled to realize the potential of promising mobile clouds.

The major challenges in mobile cloud computing come from the characteristics of the involved elements, namely, mobile devices, wireless network and cloud [28], [55], [21]; this makes the implementation of mobile cloud computing more complicated than stationary clouds. The major issues and challenges in mobile cloud computing are as following:

- **Limitations of the Mobile devices:** compared to personal computers, mobile devices have limited storage capacity, poor display, less computational power and energy resource. Although smart phones have improved significantly, they still have battery power constraint.

- **Network Bandwidth and Latency:** mobile cloud computing may face the challenge from the transmission channel due to the intrinsic nature and constraints of wireless networks and devices. Furthermore, the cloud services may be located far away from mobile users, which in turn increase the network latency.

- **Heterogeneity:** heterogeneity can appear among mobile devices, wireless networks and cloud servers; this can cause major issues in the integration, interoperability, compatibility, portability and efficiency of different elements. It is even possible to experience heterogeneity in a single type of mobile devices, wireless network and cloud.

- **Service Availability:** mobile clients may not be able to connect to the cloud due to traffic congestion, network failures and being out of signal.

- **Security and Privacy:** mobile devices usually lack the computing power to execute sophisticated security algorithms. Moreover, it is difficult to enforce a standardized credential protection mechanism due to the variety of mobile devices. Clouds and wireless networks also have security and privacy problems.

- **Reliability:** maintenance, fault tolerance and reliability can be an issue in all levels and elements of mobile cloud computing.

- **Context-awareness issues:** context-aware and socially-aware computing are inseparable traits of contemporary handheld computers. Designing resource-efficient applications which are environment-aware is an essential need.

- **Elasticity:** cloud providers confront situations in which there is more demand than available resources. The negative impact of cloud-resource unavailability and service interruption for mobile clients is more severe than stationary clients connected to the wall power and stationary networks. Frequent suspension of energy-constraint mobile clients due to resource scarcity, not only shrinks usefulness of cloud outsourcing for Mobile users, but also divests privilege of intensive computation anytime, anywhere from mobile users. Hence, several challenging tasks (e.g. resource provisioning without service interruption, quick disaster recovery, and high service availability) need to be realized since service unavailability and interruption prolong execution time, increase monitoring overhead, and consume smartphones' local resources, especially battery; therefore, emerging solutions must be employed to allocate cloud resources on demand.

- **Live VM migration issues:** executing resource-intensive mobile application via VM migration-based application offloading involves encapsulation of application in VM instance and migrating it to the cloud, which is a challenging task due to additional overhead of deploying and managing VM on mobile devices.

- **Cloud Policies for Mobile Users:** cloud service providers apply certain policies to restrain service quality to a desired level. Also, service provisioning, controlling, balancing and billing are often matched with the requirements of desktop clients rather than mobile users. Considering the great differences in wired and wireless communications, disregarding mobility and resource limitations of mobile devices in design and maintenance of cloud structures can significantly impact on feasibility of mobile cloud solutions. Metrics such as bandwidth quota and number of API calls per day limit users and have impact on users' experience. Therefore, it is essential to amend restriction rules and policies to meet mobile users' requirements.

- **Service Execution and Delivery:** cloud-mobile users, require an efficient monitoring means to measure and evaluate the quality of service they receive. Considering the mobile clouds' dynamism, several network challenges such as inconsistent bandwidth and packet delivery ratio, delay, jitter, and network blips hamper service delivery and they raise ambiguity in SLA. This ambiguity can increase dispute between cloud vendors and mobile cloud end-users. Therefore, current static SLA can be fleshed out with more powerful, dynamic representation and monitoring techniques established in heterogeneous wireless environment for cloud mobile users.

- **Mobile Cloud Billing:** user mobility in the rapidly changing environment diverges the cloud billing system in mobile clouds from billing scheme in cloud computing. Designing an appropriate billing system for mobile clouds with dynamic heterogeneous environment requires considering additional parameters compared to stationary clouds. Interception latency, jitter, session reestablishment delay, bandwidth capacity, and quality of security are examples of major parameters in designing a mobile cloud billing system.

1.4 Motivations

Managing a cloud system is complex due to its unpredictable environment. It is extremely challenging to obtain accurate information on the state of the system. Moreover, it contains large resources which are shared and require complex policies to manage them. The main factors affecting the resource management in cloud are performance, functionality and cost.

Resource management in cloud computing is associated with fluctuating workloads which pose a major challenge to elasticity of cloud computing. This fluctuation can occur in planned or unplanned spikes. For the first case, the situation can be predicted in advance and resource allocation can be done in advance. For the second case, resources have to be allocated on-demand and reallocated when needed. This concept is called "Auto-scaling" in cloud [45]. The cloud service requirements indicate that the adopted resource management policies in cloud computing should be different from the policies applied to the traditional network systems.

For a cloud provider, predicting the dynamic nature of cloud users' application demands is inapplicable. For the cloud users, the task should be completed on time with minimal cost. An efficient resource allocation scheme requires overcoming lack of limited resources, resource heterogeneity, locality restrictions, environmental necessities and dynamic nature of resource demand. Finding an optimum resource allocation strategy in huge systems like datacenters is challenging due to dynamic or uncertain resource demand and supply [1].

Resource allocation in cloud can be classified into two main steps:

The first step is mapping the VMs onto Physical Machines (PMs). Resources in cloud include the software and hardware required to execute user workloads. Examples of such resources are memory, CPU, bandwidth, storage and network. Resource allocation is the process of allocating optimal resources to the tasks requested by the cloud users. In a cloud environment, resource allocation generally means allocating VMs while satisfying the configurations specified by the user. The configurations include the operating system, MIPS, network bandwidth, storage, etc.

The second step is mapping the workloads onto the VMs: There is another situation where the cloud contains a set of existing Virtual Machines and a built environment with predefined memory, CPU and bandwidth. The users submit their workloads which may be time varying and deadline based. These workloads need to be allocated to the optimal resources such that the workloads are processed efficiently. This step of allocation is referred as mapping the workloads onto the VMs.

1.4.1 The Key Definitions in Cloud Resource Management

The important definitions in cloud resource management [56], [73] are highlighted as follows:

- **Resource allocation:** refers to the process of provisioning available computer resources for the cloud users' requests. This includes the provisioning of VMs in PMs and also, user task requests on VMs.
- **Virtualization:** Virtualization is a technology that abstracts away the details of physical hardware and provides virtualized resources for high-level applications. A virtualized server is commonly called a VM. Virtualization forms the foundation of cloud computing, as it provides the capability of pooling computing resources from clusters of servers and dynamically assigning or reassigning virtual resources to applications on-demand.
- **Quality of Service (QoS):** refers to metrics such as task blocking rate, task rejection rate, delay, response time, operational cost, throughput, maximization of profit and etc.
- **Workload balancing:** load balancing of tasks among the resources to improve the system utilization.
- **Energy Management:** refers to optimized use of energy in the datacenter.
- **Admission control:** one of the general policies to be considered in cloud resource management. It makes decision whether to admit a task/request to be processed in the cloud.

1.4.2 The Components of Cloud Resource Management System

The cloud resource management system contains the following basic components:

- **SLA Management:** The Service Level Agreement (SLA) Management module makes an agreement between the user and the service provider for the services requested by the user. The agreement contains the resource requirements of a user like the CPU, memory and other architectural configuration of a VM. It also includes QoS requirements such as task blocking rate, task rejection rate, delay, response time, task completion time, operational cost, etc. The user can also negotiate with service provider regarding the price of services. The SLA Management module communicates with the admission controller before making an agreement. After successful agreement, the user requests are sent to the scheduler.

- **Admission Controller:** The request made by the client through SLA is validated based on the availability of requested resources and other constraints specified in the SLA. If the requirements cannot be satisfied, the request is rejected by the service provider. This avoids provisioning of invalid tasks in the cloud resources.

- **Pricing:** Pricing is done for dynamic tasks based on their usage of resources. The time and cost of the consumed resources are calculated in a real time manner.

- **Scheduler:** The scheduler provisions the requested tasks in the available VMs or provisions a requested VM in a PM. There will be a queue to determine the priority of the task. Only one task can be processed on a VM at the same time. The scheduler should decide about which VM or PM the task has to be provisioned on. For this purpose, the scheduler should have information regarding the status of the datacenter such as available number of CPUs, available amount of memory, bandwidth, etc. The scheduler may get this information from a load balancer. After completion of tasks requested by the user, the pricing module is invoked to prepare the bill according to the resources consumed by the user.This information is sent to the user through the SLA management module.

- **Load Balancer:** The load balancer finds the overused and underutilized VMs or PMs. It balances the workload among the VMs to control the resource utilization. Also, energy can be saved by utilizing the idle resources. The load balancer interacts with scheduler and admission controller to balance the workload on VMs.

1.4.3 Resource Management Requirements

Resource management in cloud system is different from the traditional network systems and the characteristics of the cloud's services, users and architecture yields some requirements in resource management which should be carefully addressed [56], [54], [53].

- **Resource Provisioning:** Cloud providers should utilize robust resource management techniques in cloud environment which efficiently deal with the problem of VM provisioning and VM placement. It is crucial to have clear solution for how to match tasks to available VMs.

- **Job Scheduling:** Cloud providers should address the problem of job scheduling which is the order of giving service to the user requests or the order of execution of the applications.

  In all chapters of this thesis, we have used First Come First Served (FCFS) scheduling algorithm. However, in Chapter 5, we have also applied a priority-based scheduling algorithm. Our priority scheme includes two types of tasks with different levels of priority. In each level of priority, tasks are served on FCFS basis.

- **Scalability:** Cloud providers should use flexible mechanisms to dynamically scale up and down resources on demand in terms of VMs. A cloud user requires to access the resources on-demand. The cloud system may scale up to additional available resources when the system is experiencing high user demand and it may later scale down when the user demand decreases. A scalable cloud system also checks the idle VMs and turns them off if they are idle for more than a specific time.

  Increasing the workload on available resources is called scaling in and increasing the workload by adding resources on demand is called scaling out. A flexible cloud system adopts these scaling methods to increase the system's utilization.

- **Load balancing:** Cloud providers should utilize dynamic load balancing algorithms which can strongly control the abrupt changes in workload and provide better results in heterogeneous and dynamic environments.

  The applied load balancing algorithms should also prevent inappropriate utilization of the resources in the cloud system; under-utilization results in waste of resources and energy and over-utilization leads to slower response times of the applications. To tackle this problem, migration of VMs is one solution which is called as autonomic management of resources in cloud.

  However, as the distribution attributes become more complex and dynamic, some of the dynamic load balancing algorithms can become inefficient and cause excessive overhead which can result in degrading of system performance [49].

- **Pricing:** The primary attribute of cloud computing inspired from utility computing is its feature of pay-per-use model, i.e., services rendered through cloud are charged based on service consumption. The services may be hardware or software appliances. This leads to the need of a pricing model whose objective is profit maximization for the service providers and quality of service for the users. Moreover, there is a trade-off between these two objectives along with service request fluctuation in cloud.

- **Availability:** The system becomes unavailable due to reasons like hardware or software failures, load fluctuations, long waiting time of jobs, etc. The traditional method to provide high availability is providing extra idle resources to be used in case of failures which can result in wasting resources. Dynamic methods are required in cloud systems to automatically detect any failure or unavailability and shift the tasks to the available resources in a short time.

- **Quality of Service (QoS) constraints:** This includes various parameters like task blocking and rejection rate, delay, cost, response time, throughput, etc. A cloud system providing services to its customers should fulfill their quality of service requirements.

- **Overheads:** Workloads to be processed on virtualized cloud platforms may be CPU intensive or network I/O intensive. The in-depth understanding of these overheads and network bandwidth as a constraint is necessary for effective resource management in the cloud.

1.4.4 The Requirements of Mobile Cloud Computing

Mobile clouds have some requirements coming from the characteristics of their services and the involved networks; these requirements play an important role in the development of models and solutions in this field.

As we have discussed in Section 1.3, the integration of mobile devices, wireless networks and clouds raises some major issues. The coexistence of these heterogeneous networks and the characteristics of the mobile services, emerges some requirements which should be met.

Mobile cloud computing requires wireless connectivity with the following features:

- Mobile cloud computing requires an “always-on” connectivity for a low data rate cloud control signaling channel.
- Mobile cloud computing requires an “on-demand” available wireless connectivity with a scalable link bandwidth.
- Mobile cloud computing requires a network selection that takes energy-efficiency and costs into account.

The coexistence requirements in mobile cloud computing are not limited to mobile device and wireless network connectivity necessities. In order to represent the operational requirements in mobile clouds, we describe the general offloading scheme in mobile device. Fig. 1.2 illustrates the overview of the scheme.

The components of an offloading system can be divided in two planes: components on the client (i.e., the mobile device) and components in the environment (i.e., either a cloud, a cloudlet, a peer device, or a hybrid environment).

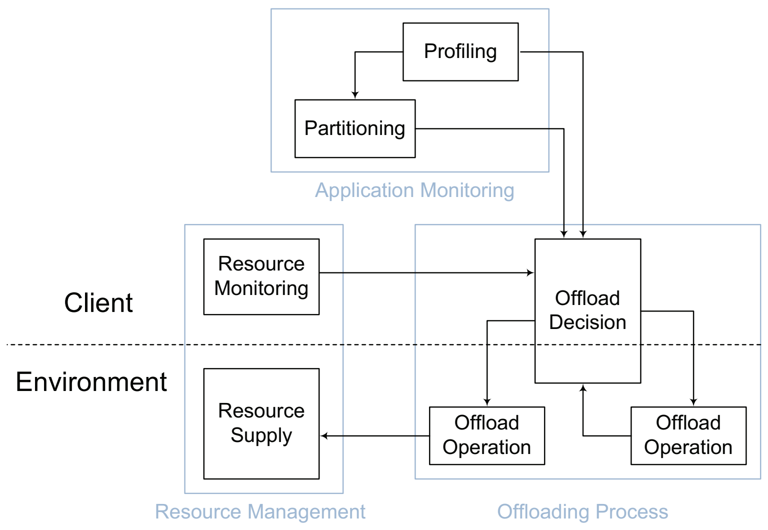


Figure 1.2: The general offloading scheme [50].

The profiling component in a mobile device is able to assess the way in which the application is functioning through various mechanisms, such as static or dynamic analysis. The information obtained from the profiling component may be used by the partitioning component, which aims to split the application in components of predefined granularities and identify which of them can be offloaded. There is also a need to assess the resources (both local resources in the mobile device and remote resources on the cloud) that are available for running the application. Thus, the resource management component spans both the mobile device and the cloud. In the mobile device, the resource monitoring component assesses parameters such as battery level, CPU load, wireless connection quality, etc. In the cloud, the resource supply component manages the external resources that may be used in offloading, through mechanisms such as discovery and provisioning. Resource discovery is useful in opportunistic approaches, such as cyber foraging, in which the mobile device tries to find available offloading targets in its environment [50]. Resource allocation is a proactive approach, highly utilized in cloud environments, in which resources are dynamically assigned to adjust to computing needs. The offload process itself needs to be an iterative process, due to mobility and the changing nature of the execution conditions. For example, the mobile user may switch from Wi-Fi to 3G/4G or may reach a critical battery level, which affects the offloading process. The offload decision component receives information from application monitoring and resource management of a mobile device, to assess the current offloading needs and conditions, and from previous iterations of the offload operation, to assess their benefits and defects.

Offload decision making scheme in a mobile device can choose what, when and where to offload. However, as we will describe in Chapter 5, the scheme of “what, where and when” offloading in the mobile devices can cause some serious issues for the cloud which is hosting the offloaded job requests; the coexistence issues between the offloaded jobs of mobile devices and the cloud servers is the main challenge we address in Chapter 5.

1.5 Objectives

In Section 1.1.1, we have introduced the five essential characteristics of clouds highlighted by NIST [46]. However, in Sections 1.2 and 1.3 which the key challenges of clouds were presented, we have seen that these characteristics are not realized in the current clouds appropriately. Also, the cloud requirements which are outlined in Section 1.4, emerge the need of developing dynamic and comprehensive models in this area.

In this thesis, we deal with resource allocation, elasticity and on-demand service provisioning issues in clouds and each chapter deals with some or all of these issues.

The major contribution of this thesis is providing efficient resource allocation solutions for cloud datacenters. Note that in this work, “resource” is referred to the VMs which are instantiated on PMs and the proposed resource allocation solutions provision the requested tasks on available VMs in PMs.

We have analyzed the behavior of IaaS cloud datacenters through our analytical and/or simulation models. In all cases, we have studied the system's performance in different operating regions including linear, transition to saturation, and saturation regimes. Performance metrics are in linear operation region if they are linearly dependent on the input parameters including mean task arrival rate, mean task service rate and number of active servers. The system is in saturated operation region when the performance metrics do not change significantly as the input parameters increase; this is due to the high traffic load imposed on the cloud system and the fullness of the system.

In the chapters which are mostly focused on stationary clouds (i.e., Chapters 3 and 4), servers are partitioned into pools of hot (i.e., always on), warm (i.e, switched on without any instantiated VM), and cold machines (i.e., switched on if there is a demand). The cold PMs can be switched on according to user request increment or the hot PMs can be switched off due to energy preservation policy. As an exception, in Section 3.2 the servers are only partitioned into hot and cold pools as we aim to evaluate the sensitivity of the energy expenditure on the partitioning threshold. The performance parameters which have been evaluated in this thesis include task blocking probability, total delay, utilization and energy consumption. Task blocking probability is the probability of rejecting the task due to the lack of room in the waiting queues or insufficient resource capacity.

In order to prevent the cloud system getting into the saturation region, we have developed dynamic algorithms to control the admission of task arrivals. These algorithms are discussed in Chapter 4 and they are based on controlling parameters such as full rate task acceptance threshold and filtering coefficient or establishing thresholds for task arrival rate and task blocking probability. Full rate task acceptance threshold is the threshold of accepting all of the arriving tasks into the cloud system. Filtering coefficient is the coefficient of task admission in the system and determines the acceptance rate of the arriving tasks.

The characteristics of mobile devices and wireless network makes the development of mobile cloud computing more complicated than the stationary clouds. Offloading requests from a mobile device usually require quick response, may be infrequent, and are subject to variable network connectivity. Also, the volume of workload which is going to be offloaded is not predefined or known for a mobile device when it starts the offload process. There is a possibility that the mobile device offloads a large burst of tasks toward the clouds or it may scarcely offload any application to the cloud; from this definition we can conclude that the mobile device has a stochastic behavior during the offload process. In this thesis, we address the elasticity in mobile cloud computing and the heterogeneity issues between cloud and mobile devices. We have developed a solution which allocates resources for on-demand job requests in the mobile clouds. This solution can also be applied to the stationary clouds which their users' applications fork new tasks. The solution uses prioritization method to provide resources for offloaded tasks from the mobile devices. The details of the proposed solution for mobile requests and the priority schemes are outlined in Chapters 5.

1.6 Main Contributions

This section introduces our main contributions in the cloud systems' resource allocation and admission control.

Table 1.1: The illustration of main contributions in this thesis.

Thesis Structure	Contributions	Challenges
Chapter 3: Investigating The Behavior of IaaS Cloud Datacenters	Examining the the trade-off between performance and power consumption in the proposed model in different operating regions. Evaluating the effects of pool threshold and mean look up time in hot and cold pools on the system performance	Lack of analysis of the behavior of IaaS Clouds in the linear and transition regimes
Chapter 4: Task Admission Control for Cloud Server Pools	Developing several task admission control algorithms based on thresholds for task arrival rate and task blocking probability or based on two controlling parameters, full rate task acceptance threshold and filtering coefficient	Lack of control over moving to the saturation operating region
Chapter 5: Prioritization of Overflow Tasks to Improve Performance of Mobile Cloud	Deploying two flexible resource allocation mechanisms which use prioritization and developing a virtual machine provisioning model	Lack of efficient resource management schemes for mobile clouds and clouds which their users' applications fork new tasks

- In Chapter 3, we utilize an analytical model which consists of three interactive stochastic submodels and is solved via successive fixed point iteration. In Section 3.1, we evaluate the behavior of an IaaS cloud datacenter in which servers are partitioned into pools of hot, warm and cold machines. We examine the behavior of this pool management scheme, specifically, the trade-off between performance and power consumption in different operating regions including linear operation, transition to saturation, and saturation. In Section 3.2, we consider a model in which partitioning only uses hot and cold pools. Also, we investigate the effects of pool threshold and mean look up time in hot and cold pools on the performance of an IaaS cloud system in the different operating regions.

- In Chapter 4, we have added several admission control algorithms of task arrivals to the resource allocation model introduced in Section 3.1. We have defined two admission control algorithms of task arrivals in Section 4.1 to keep the cloud system in the non-saturation operating region. These admission control algorithms are executed in two steps: first step is adjusting the partitioning coefficient according to the mean task arrival rate; the second tier is tracking the task blocking probability in the system and adjusting the task acceptance rate according to the predefined task blocking probability thresholds.

  In Section 4.2, we have proposed two other task admission algorithms to keep the system in the stable operating region. We have used two controlling parameters, full rate task acceptance threshold and filtering coefficient, to deploy task admission policies. First algorithm is lightweight and appropriate for the cloud systems with smooth changes of task arrival rate. This algorithm is based on long-term estimation of average utilization and offered load. Second algorithm is suitable for highly dynamic systems and is based on instantaneous utilization. Offered load represents the traffic load in the system and is computed as the mean task arrival rate into the cloud system divided by the mean task service rate.

- In Chapter 5, we have defined two flexible resource allocation mechanisms for clouds where their users' requests fork new tasks, especially, mobile clouds. These mechanisms use prioritization method to provide appropriate service for offloaded tasks from the mobile devices. The first mechanism gives the full priority to the forked tasks over newly arrived tasks. The second mechanism establishes a threshold to control the priority of the forked tasks over newly arrived tasks. This chapter also includes a model for virtual machine provisioning.

Table 1.1 outlines the main contributions presented in this thesis.

1.7 Thesis Organization

The rest of this thesis is organized as following:

- Chapter 2 discusses the state-of-the-art research works in the fields of resource allocation and admission control in cloud computing.
- In Chapter 3, we evaluate the performance and power consumption of the proposed IaaS cloud model in different operating regions and the trade-off between performance and power consumption. Also, we have analyzed the effect of provisioning overhead time on the proposed IaaS cloud model.
- In Chapter 4 we have introduced a model for task admission control based on thresholds of task arrival rate and blocking probability and we have evaluated the performance this model. Also, we have presented two dynamic task admission control algorithms which have been developed based on task filtering policy.
- In Chapter 5, we have developed a solution which allocates resources to cloud job requests which can fork new tasks, specifically, on-demand job requests in the mobile clouds. In this model, we have prioritized serving the overflow tasks over the new incoming tasks. Also, we have modeled virtual machine provisioning in this solution.

- Chapter 6 concludes the thesis and and discusses some directions for future work.

Table A.2 lists the parameters defined in this work.

Chapter 2

Related Work

2.1 Resource Allocation in Stationary Cloud Computing

Resource provisioning in stationary cloud computing has been extensively studied with different focuses. In this section, we are going to explore the current research works in this area.

In [70], energy-efficient virtual resource allocation for the cloud has been formulated as a multi-objective optimization problem which is solved using an intelligent optimization algorithm. The work presented in [65] has proposed a congestion control method using an index for evaluating fair resource allocation in case of congestion. In [63], authors have proposed a dynamic resource allocation in a cloud environment which considers computing job requests that are characterized by their arrival and teardown times, as well as a predictive profile of their computing requirements during their activity period. Two algorithms to adjust resource allocation and task scheduling adaptively based on the actual task execution time have been proposed in [42].

In [39], allocating VMs to applications with real-time tasks is formulated as a constrained optimization problem. Since an exhaustive search for solutions has exponential complexity, polynomial-time heuristic model was proposed to solve the problem. Moreover, the cost obtained by this heuristic model was compared with the optimal solution and an Earliest Deadline First (EDF-greedy) approach.

In [68], an ad hoc parallel data processing framework has been presented to exploit the dynamic resource allocation for both task scheduling and task execution in IaaS clouds. Specific tasks of a processing job can be assigned to different types of VMs. The algorithm presented in [20] formed groups of VM instances according to their runtime deadlines and packed VMs in the same group on the same servers. Moreover, it shuts down some servers in time when the service request decreases in order to reduce energy consumption.

The work presented in [25] has proposed a resource allocation model using combinatorial auction mechanisms which uses energy parameters. Three algorithms have been introduced for different aspects of resource allocation.

[48] has applied the queueing theory to evaluate the average response time and explore the trade-off between performance and cost in the hybrid cloud (collaboration of private cloud and public cloud). By taking advantage of Lyapunov optimization techniques, an online decision algorithm is designed for request distribution which achieves the average response time arbitrarily close to the theoretically optimum and controls the outsourcing cost based on a given budget.

The work in [72] has presented a multi-resource cloud computing model with resource trade-offs. In this model, each user is allowed to specify multiple resource requirements (corresponding to the requirements of different task implementations) and the utility a user derives from the resources allocated to him is the maximum number of tasks he can complete with these resources. The work has proposed two different mechanisms, which reflect two different classical economic approaches for fairly allocating resources: the Nash Bargaining (NB) mechanism and the Lexicographically Max- Min Fair (LMMF) mechanism.

The work presented in [74] has proposed a randomized auction mechanism based on an application of smoothed analysis and randomized reduction, for dynamic VM provisioning (pricing tailor-made VMs on the spot) and pricing in geo-distributed cloud data centers. This auction achieves truthfulness in expectation, polynomial running time in expectation, and $(1 - \epsilon)$-optimal social welfare in expectation for resource allocation, where ϵ can be arbitrarily close to 0.

[75] has used online procurement auction mechanisms to address the resource pooling issue in cloud storage systems. The online nature of the auction is in line with asynchronous user request arrivals in practice. With characterizing truthfulness conditions under the online procurement auction paradigm, authors have converted the mechanism design problem into an online algorithm design problem, with a marginal pricing function for resources as variables set by cloud storage service providers for online procurement auction.

The problem of optimal placement of VMs in clouds was tackled in [3] for minimizing latency. Complexity was reduced by recurring to a hierarchical split of the placement problem into two reduced complexity sub-problems of choosing the datacenters, then choosing the specific racks and servers, and applying a partitioning of the application placement graph.

A resource allocation problem in [10] was formulated. In this model, later tasks could reuse resources released by earlier tasks and the problem was solved with an approximation algorithm that could yield close to optimum solutions in a polynomial time.

However, to the best of our knowledge, no research has been done on the effects of the variation of offered load on cloud computing centers' behavior in linear and transition to saturation region. This is particularly important for congestion and admission control. We have examined the effects of the variation of offered load on cloud systems' performance in the Chapters 3 to 5.

2.2 Task Admission Control in Cloud Systems

Resource allocation and admission control in clouds have been the topic of much research effort in recent years. Some of the solutions were based on queueing theory: for example, [66] made use of the queueing information available in the system to make online control decisions. It used Lyapunov Optimization to design an online admission control, routing, and resource allocation algorithm for a virtualized datacenter. The algorithm considers a joint utility of the average application throughput and energy costs of the datacenter in decision making. In [41], authors proposed an autonomous scheme for admission control in cloud services aiming at preventing overloading, guaranteeing target response time and dynamically adapting the admitted workload to compensate for changes in system capacity. They employed an adaptive feedback control scheme alongside with a queue model of the application. Cloud service, modeled using queuing theory, and controlled through adaptive proactive controllers that estimate whether services need some of the resources in the near future or not, was discussed in [2].

A Markov Decision Process (MDP) framework was proposed in [17] to model admission control in cloud while Approximate Dynamic Programming (ADP) paradigm was utilized to devise optimized admission policies. In [24] authors formulated an optimization problem for dynamic resource sharing of mobile users in Mobile Cloud Computing hotspot with a cloudlet as a Semi-Markov Decision Process (SMDP) which was, then, transformed into a Linear Programming (LP) model to obtain an optimal solution. A unified framework of admission control and resource allocation was provided in [15] which modeled the system's dynamic behavior with a group of state-space models, scaled between different desired operation points and used a set-theoretic control technique to solve admission control and resource allocation problems.

A technique for determining the effective bandwidth for aggregated flow was developed in [23] to make admission decisions using network calculus. Authors also examined the relationship between effective bandwidth and equivalent capacity for aggregated flow, while [36] modeled the admission control problem in a cloud using the General Algebraic Modeling System (GAMS) and solved it under provider-defined settings. Same authors in [37] presented a technique for admission control of a set of horizontally scalable services and their optimal placement into a federated Cloud environment. They have considered in their model that a request may also be partially accommodated in federated external providers, if needed or if it is more convenient.

A session-based adaptive admission control approach for virtualized application servers was presented in [4]. Also, a session deferment mechanism was implemented to reduce the number of rejected sessions. In [64] three key characteristics of cloud services and IaaS management practices was identified which are burstiness in service workloads, fluctuations in virtual machine resource usage over time and virtual machines being limited to pre-defined sizes only. Based on these characteristics, paper proposed scheduling and admission control algorithms that incorporate resource overbooking to improve utilization. A combination of modeling, monitoring, and prediction techniques was used to avoid exceeding the total infrastructure capacity.

Our task admission control solutions presented in Chapter 4 are unified with our resource allocation solution discussed in Section 3.1. The proposed task admission control algorithms are based on establishing thresholds for task arrival rate and task blocking probability in one solution, and based on full rate task acceptance threshold and filtering coefficient in another solution.

2.3 Resource Allocation in Mobile Cloud Computing

Several research studies have proposed solutions to address the issues of computational power and battery lifetime of mobile devices by offloading computing tasks to cloud. CloneCloud [12] has been an approach for extending the concept of VM-based clone cloud offloading from LAN surrogates to cloud servers. OS supporting VM migration was introduced in CloneCloud. MAUI [14] has provided method level code offloading based on the .NET framework. MAUI aimed to optimize energy consumption of a mobile device by estimation and evaluating the trade-off between the energy consumed by local processing versus the transmission of code and data for cloud offloading. Decision process in MAUI is based on information and complex characteristics of the mobile environment. A framework for moving smartphone application processing to the cloud centers was introduced in ThinkAir [38]. This framework is based on the concept of smartphone virtualization in the cloud and addresses lack of scalability by creating VM of a complete smartphone system on the cloud. ThinkAir provides on-demand resource allocation by dynamically managing VMs in the cloud via using an execution controller. The execution controller handles decision-making and communication with the cloud server. It considers execution time, energy, and cost to make decision in order to achieve optimum performance. With regard to the network profile parameters, device profile parameters, and program profile parameters of the smartphone, ThinkAir dynamically allocates the available cloud resources to the programs. CMcloud [9] is a mobile-to-cloud offloading platform which attempts to minimize both the server costs and the user service fee by offloading as many mobile applications to a single server as possible, while trying to satisfy the target performance of all applications. To achieve such goals, CMcloud exploited architecture performance modeling and server migration techniques. In POMAC framework (Properly Offloading Mobile Applications to Clouds) [22], other than offloading decision making technique, an offloading mechanism was designed through method interception at Dalvik virtual machine level to allow mobile applications to offload their computation intensive methods.

In most of the works related to resource allocation in mobile cloud computing, there are some trade-offs among power consumption, QoS parameters and costs. These objectives are usually dependent on cloud resources, applications profiles and network parameters. COSMOS (Computation Offloading as a Service for Mobile Devices) system [57] has provided computation offloading as a service to mobile devices. The COSMOS system receives mobile user computation offload demands and allocates them to a shared set of compute resources that it dynamically acquires (through leases) from a commercial cloud service provider.

The partitioning of elastic mobile datastream applications was formulated in [30] as an optimization problem by minimizing the cost function which is combination of communication energy and computation energy.

In [19], a model has been built that considers some characteristics of the workflow's software and the network's hardware devices. With this model, the objective functions have been constructed which guide the offloading decisions. A heuristic algorithm was presented in this model that produced offloading plans according to these objective functions and their variations.

A technique of coalesced offloading was proposed in [69], which exploited the potential for multiple applications to coordinate their offloading requests with the objective of saving additional energy on mobile devices. Authors believe that by sending these requests in bundles, the period of time that the network interface stays in the high-power state can be reduced. Two online algorithms were presented, collectively referred to as Ready, Set, Go (RSG), that make near-optimal decisions on how offloading requests from multiple applications are to be best coalesced.

In [51], some practices for managing the resources required for the mobile cloud model has been formulated, namely energy, bandwidth and cloud computing resources. These practices were realised with the authors' mobile cloud middleware project, featuring the Cloud Personal Assistant (CPA). In order to realise resource allocation in their work, they have estimated a cost model for each VM running on a server in the cloud and they have calculated the summation of the costs required to run the physical resources required on the server.

In another attempt to connect mobile devices to cloud servers in [29], given an application described by a task dependency graph, an optimization problem was formulated to minimize the latency while considering prescribed resource utilization constraints. Authors have proposed Hermes, a fully polynomial time approximation scheme (FPTAS) algorithm to solve this problem. Hermes provides a solution with latency no more than $(1 + \epsilon)$ times of the minimum while incurring complexity that is polynomial in $1/\epsilon$ and the problem size.

The proposed model in [67] is based on the wireless network cloud (WNC) concept and a multi-objective optimization approach using an event-based finite state model and dynamic constraint programming method has been used to determine the appropriate transmission power, process power, cloud offloading and optimum QoS profiles. However, estimation and approximation techniques used in this work approximate the parameters linearly and the main objective of this paper concerns performance metrics of mobile devices and users and the challenges of cloud computing centers have not been covered.

In [58], other than presenting a study on virtual machine deployment, authors have evaluated the impact of VM deployment and management for application processing by analyzing the parameters such as VM deployment and execution time of applications. The work presented in [59], has analyzed the impact of performance metrics on the execution of applications (cloudlets). Performance metrics associated with the execution of cloudlets in distributed mobile cloud computing environment are presented as cloudlet offloading, cloudlet allocation to VM, cloudlet scheduling in VM, cloudlet migration in datacenter and cloudlet reintegration.

The work in [31] has presented a task scheduling and resource allocation scheme which used the continually updated data from the loosely federated General Packet Radio Service (GRPS) to automatically select appropriate mobile nodes to participate informing clouds, and to adjust both task scheduling and resource allocation according to the changing conditions due to the dynamicity of resources and tasks in an existing cloud.

Unlike most of existing works that either rely on a linear programming formulation or on intuitively derived heuristics that offer no theoretical performance guarantees, our solution to the resource allocation problem in mobile clouds presented in Chapter 5 does not simplify the computational complexity of offloading problem to make it solvable. It should be mentioned that our model is not dealing with the offloading decision process in the mobile devices and we assume that offloading decision has been made and our scheme allocates cloud resources to the offloaded applications when they arrive in the cloud data centers.

2.4 The Markov Models with Multiple Priority Classes

The Markov models with multiple priority classes have been used in different fields. For example, a Markov chain flow decomposition for a two class priority queue in presented in [8].

Also, threshold-based priorities have been utilized in the development of Markov models. For instance, in [43], a multiserver queueing system with two priority classes have been employed which has a threshold defined according to number of servers in the system.

In another approach, a threshold based Markov chain system has been deployed in [62] to model elastic and inelastic traffic flows in TCP-friendly admission control; the threshold was defined according to some inelastic flow parameters.

In Chapter 5, we have utilized a Markovian multiserver queueing system with two priority levels to model the two types of mobile cloud tasks as two classes of services.

2.5 Chapter Summary

In this chapter, we have surveyed the current research works in the area of resource allocation in stationary and mobile cloud computing. Also, we have outlined the literature related to task admission control in cloud. Finally, we have introduced some Markov Models with Multiple Priority Classes which remotely inspired us in the development of our resource allocation model in mobile clouds presented in Chapter 5.

Chapter 3

Investigating The Behavior of IaaS Cloud Datacenters

To save energy, physical machines (servers) in cloud datacenters are partitioned in different pools, depending on whether they are kept on at all time and/or whether they have virtual machines instantiated. Partitioning (pooling) of servers not only affects the power consumption of the datacenter but also the performance and responsiveness to user requests. In Section 3.1, we examine the behavior of pool management scheme in different operating regions which correspond to linear operation, transition to saturation, and saturation, in particular the trade-off between performance and power consumption. Our results in Section 3.1.2 show that the definition of offered load presented in [13] does not offer complete characterization of data center operation; instead, the impact of task arrival rate and task service time must be considered separately. In [13], offered load is defined as the traffic load in the system and is computed as the mean task arrival rate into the cloud system divided by the mean task service rate.

In Section 3.2, we analyze an efficient pool management model for cloud systems that partitions the PMs into a hot and cold pool to improve energy efficiency. Servers are moved from one pool to the other as needed to fulfill the incoming task requests, which are provisioned on virtual machines running on the servers. The model features two levels of task admission control: one at the global input, the other at each server separately. We examine the behavior of the cloud system in the regions of linear operation, transition to saturation and saturation, from the viewpoint of task rejection rates and energy consumption. Furthermore, we evaluate the sensitivity of task blocking probability and energy consumption to the pool partitioning threshold and the value of mean look up time in hot and cold pools, respectively, in different test scenarios.

This chapter is organized as follows: Section 3.1 provides the proposed model of resource allocation in a pooled IaaS cloud and presents the analysis of trade-off between performance and power consumption. In Section 3.2, we evaluate the effects of pool threshold and mean look up time in hot and cold pools on the system performance. Section 3.3 outlines the summary of this chapter.

3.1 Characterizing Energy Consumption of IaaS Clouds in Non-saturated Operation

Energy efficiency is one of the top priorities in a cloud datacenter [71], which is why ways to reduce energy consumption without sacrificing performance are among the most important research topics. If the cloud datacenter is configured so that user tasks are provisioned on VMs executing on servers or PMs, energy efficiency may be improved by dividing the PMs in three groups or pools [33]. In this approach, PMs in the *cold* pool are normally kept switched off, while PMs in the *warm* and *hot* pool are normally switched on, the latter having VMs already instantiated and ready to provision user tasks. When a user request that asks for one or more VMs arrives, the pools are checked in sequence from hot through warm to cold; if necessary, PMs are switched on and/or VMs instantiated. Upon termination of user tasks, hot PMs with idle VMs are moved back to the warm pool, while excess warm PMs are switched off and, thus, moved back to the cold pool.

In this manner, power consumption is made dependent on the datacenter load, which leads to improved energy efficiency and 'greener' computing [5]. At the same time, the manner in which the PMs are partitioned into hot, warm, and cold pools will affect the performance of the datacenter with respect to request response time and probability that a request will be blocked due to insufficient resource availability.

Earlier performance studies of pool management schemes [33] have focused on IaaS clouds operating in saturation regime where all the PMs are essentially busy (almost) all the time. This region is not really usable for cloud operators since blocking of user requests reaches values which are unacceptably high. Instead, in this chapter we focus on linear and transition regimes which are more important in practice. We analyze the boundaries between the operating regimes with respect to overall system load, which is a function of the distribution of task arrival rate, task service time, and look-up overhead needed to decide whether the task can be accommodated or not. We evaluate the energy efficiency of this model in different scenarios.

Interestingly enough, our results indicate that characterizing the load with a single value, which is customary in the performance evaluation of communication networks [35],

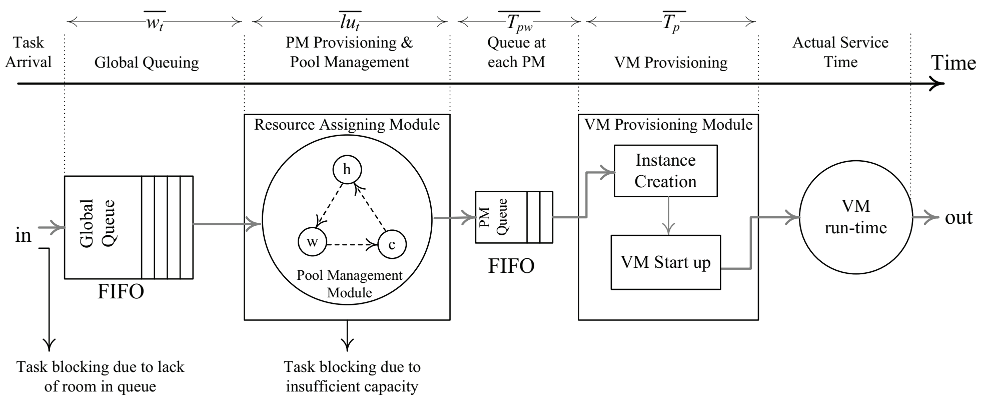


Figure 3.1: The steps of servicing and corresponding delays (adapted from [33]).

is not quite appropriate for cloud datacenters. Namely, searching for the appropriate PM (or PMs) on which tasks from the current user request will be provisioned, necessitates a certain overhead which depends on the characteristics of the request but also on the current load of the system. Therefore, it is of interest to evaluate the performance of the scheme under both varying user request arrival rate and user task service time.

3.1.1 Analytical Model

We assume that the IaaS cloud center has a common input queue, while the three PM pools have separate queues of their own. A single PM can host a number of VMs simultaneously; this number is limited in order to ensure satisfactory performance level of the VMs. Without loss of generality, we assume that all PMs are homogeneous as are the VMs. Furthermore, we assume that a single pre-built VM image can satisfy all requests [33]; however, the model can easily be extended to cover PMs and/or VMs with different characteristics. We assume that a PM can simultaneously run up to 10 VMs. VM Provisioning on hot PMs has the minimum delay compared to warm and cold PMs. Warm PMs need more time to be ready for provisioning and cold PMs require additional time to be started up before a VM instantiation. In this chapter, system has single task arrivals.

According to Fig. 3.1, when a task arrives, it will get accepted or rejected due to lack of room in the global queue. A task should wait until the Resource Assigning Module (RAM) processes it. RAM either assigns a PM in hot, warm or cold pool (indicated by h, w and c respectively) to the task or rejects it due to insufficient capacity. If one of the PMs accepts the task, it would be queued in that PM's queue. At last, the VM Provisioning Module (VMM) starts the instantiation and deployment of VMs for that task and the actual service starts.

RAM processes the tasks in the global queue on a First-In, First-Out (FIFO) basis; RAM first tries to provision the task on a PM machine in the hot pool. If the process is not successful then RAM tries the warm pool and if there is no PM available in warm pool, RAM tries the cold pool. If RAM cannot assign a PM to a task, it gets rejected. When a running task is done, the assigned capacity will be released. If there is no running task on a PM for certain time, the Pool Management Module (PMM) moves the PM to warm or cold pool according to the pool policy and traffic condition. PMM is responsible for management of pools during the operation of the cloud center. PMM moves the PMs among the pools depending on the load, according to two criteria: first, achieving minimal response time and blocking probability and second, in order to have less energy consumption, idle hot PMs should power down to either warm or cold mode after certain idle time.

To model the performance of an IaaS cloud center, we have constructed a probabilistic model that consists of three different submodels with one, three, and one instance each, respectively. Our model closely follows the one presented in [33], but our emphasis is on

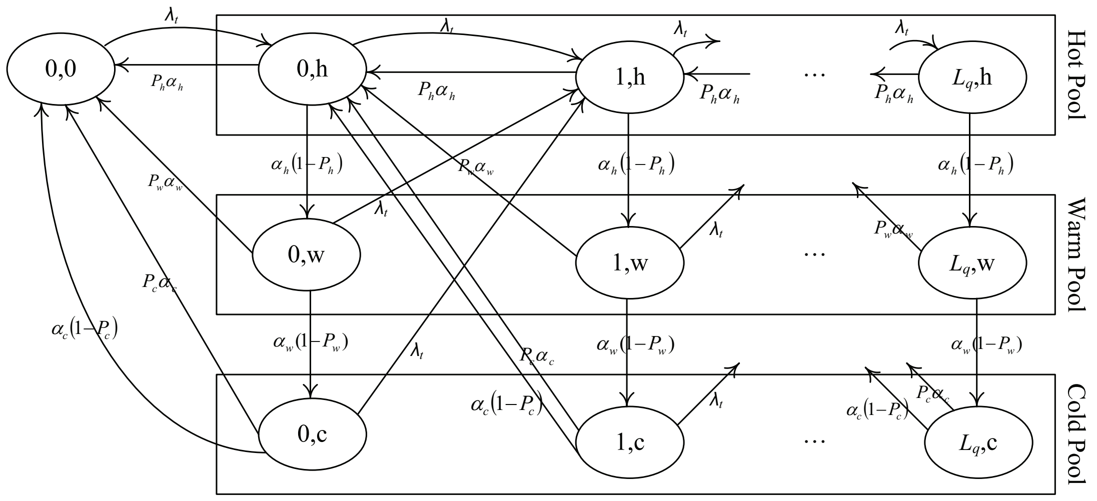


Figure 3.2: CTMC of the Resource Allocation module (adapted from [33]).

energy-related issues in the context of a non-saturated IaaS cloud center.

Resource Allocation

Task requests arrive according to a Poisson process with arrival rate λ_t. An incoming request will be processed by the RAM, shown with the two-dimensional Continuous Time Markov chain (CTMC) in Fig. 3.2. RAM checks the hot, warm, and cold pools (in that order) to find whether there is a sufficient number of PMs and idle VMs to accommodate the request; $1/\alpha_h, 1/\alpha_w$ and $1/\alpha_c$ are the mean look up delays in the respective pools. A hot PM can immediately begin service of a request, provided it has VMs to spare. If there is no hot PM with the required capacity, a warm PM may be used, but it must instantiate the required number of VMs before provisioning the request. If there is no warm PM either, a cold PM will be used, but must be switched on before instantiating the VMs. Obviously, the delays for the three types of PMs will differ, with hot PMs providing the shortest one.

Once an idle VM is found, RAM will allocate the request. A request may be rejected if there is no space in the input queue, or a suitable PM can't be found. The probability of the former event is $P_{bq} = \pi(L_q, h) + \pi(L_q, w) + \pi(L_q, c)$, where $\pi(i, j)$ denotes the probability of pool j having i requests while L_q is the capacity of the input queue; the probability of the latter is $P_{br} = \sum_{i=0}^{L_q} \frac{\alpha_c(1 - P_c)}{\alpha_c + \lambda_t} \pi(i, c)$. Total blocking probability is, then, $P_{blk} = P_{bq} + P_{br}$.

If we define the probability generating function (PGF) for the number of tasks in the queue [61] as

$$V(z) = \pi(0, 0) + \sum_{i=0}^{L_q} (\pi(i, h) + \pi(i, w) + \pi(i, c))z^i, \tag{3.1}$$

mean waiting time can be calculated using Little's law [35] as

$$\overline{w_t} = \frac{\overline{v}}{\lambda_t(1 - P_{bq})}, \tag{3.2}$$

and mean look up time among pools can be obtained [33] as

$$\overline{lu_t} = \frac{1/\alpha_h + (1 - P_h)((1/\alpha_w) + (1 - P_w)(1/\alpha_c))}{1 - P_{bq}}. \tag{3.3}$$

Virtual Machine Provisioning

The request then waits in the PM input queue until a VM is ready to provision it. Provisioning is described with a VMM modeled as another CTMC, shown schematically in Fig. 3.3. The complete analytical model employs three such modules with identical structure corresponding to hot, warm, and cold pools, respectively.

Let ϕ_h as the rate at which a VM can be provided on a PM in the hot pool, and let μ be

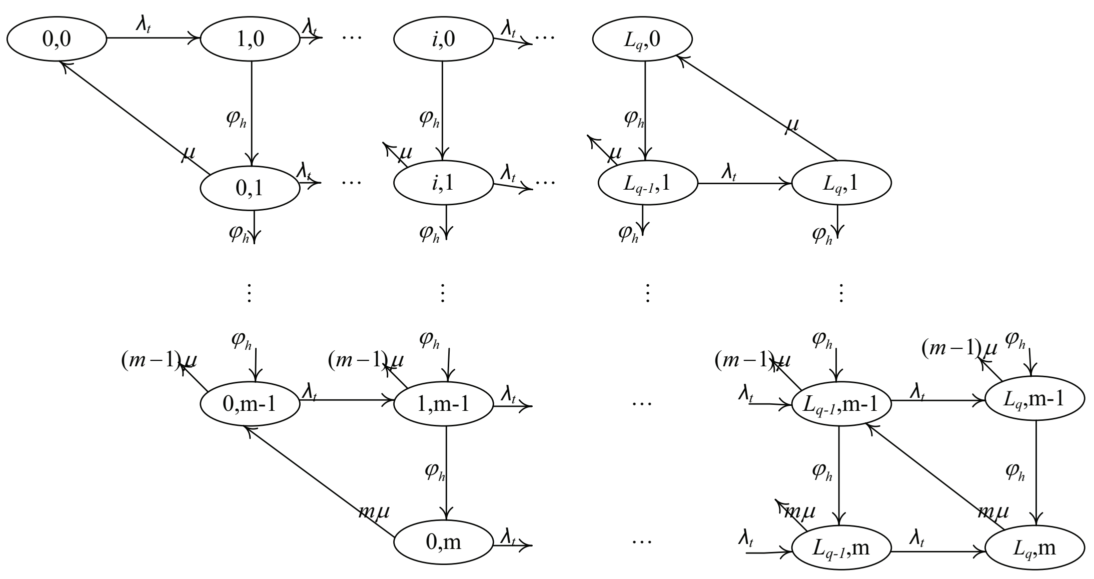


Figure 3.3: CTMC of the Virtual Machine Provisioning module of the hot pool (adapted from [33]).

the service rate of each task. The arrival rates can be calculated as

$$\begin{aligned}\lambda_h &= \frac{\lambda_t(1-P_{bq})}{N_h}\\ \lambda_w &= \frac{\lambda_t(1-P_{bq})(1-P_h)}{N_w}\\ \lambda_c &= \frac{\lambda_t(1-P_{bq})(1-P_h)(1-P_w)}{N_c}\end{aligned} \tag{3.4}$$

for hot, warm, and cold pool, respectively, where N_h, N_w, and N_c denote the number of PMs in the respective pools. $P_h = 1 - (P^h_{na})^{N_h}$ denotes the probability of a PM in the hot pool accepting the task; the complementary probability may be calculated as $P^h_{na} = \sum_{x\in\eta} \pi^h_x$. In the last expression, $\eta = \{(i,j,k)|i = L_q\}$ denotes a subset of VMM states in which a task may be blocked.

By the same token, success probability for provisioning a task in warm and cold pool

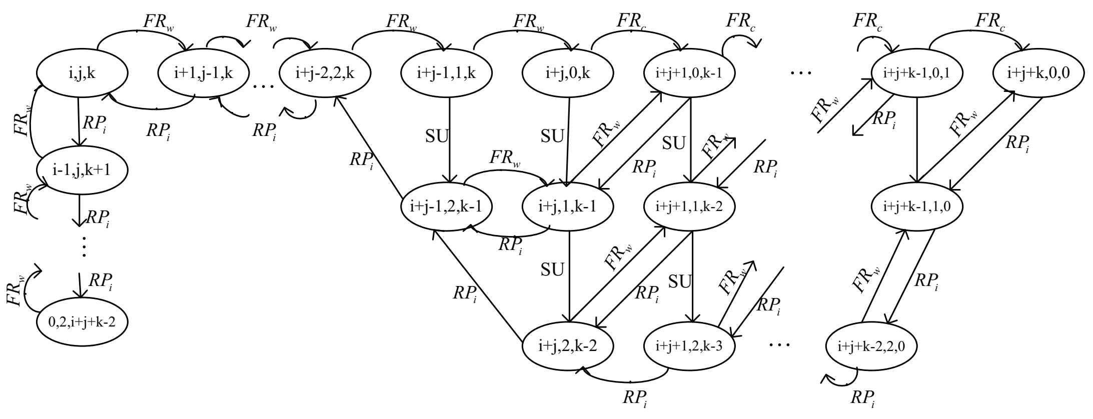


Figure 3.4: CTMC of the Pool Management module (adapted from [33]).

is $P_w = 1 - (P^w_{na})^{N_w}$ and $P_c = 1 - (P^c_{na})^{N_c}$, respectively.

Pool Management

Provisioning of requests may require that PMs are moved between pools under the control of the PMM, modeled with a CTMC shown in Fig. 3.4. The transition from a warm to hot PM occurs at a rate of $FR_w = (\lambda_t P_{bq}(1 - P_h) + (1/SU_w))^{-1}$, where $1/SU_w$ is the mean time required for a warm PM to switch to hot state. Similarly, a cold PM can be moved to the hot pool at a rate of $FR_c = (\lambda_t P_{bq}(1 - P_h) + (1/SU_c))^{-1}$, where $1/SU_c$ is the mean time needed for a cold PM to switch to hot state.

Conversely, if, upon a task ends execution and the number of idle hot PMs exceeds a predefined threshold, an idle hot PM can be moved to the warm pool to reduce energy consumption, or even to the cold pool if the number of PMs in the warm pool is above the predefined threshold, at a rate RP_i.

Integrated Model

The overall model, consists of three interactive stochastic submodels; this reduces complexity of the model itself but also the computational complexity of solving the model. It is, then, solved via successive fixed point iteration [44], shown as pseudocode in Algorithm 1. Algorithm ends when the difference between the values of probabilities in successive iterations drops below a predefined threshold ($\Delta = 10^{-6}$).

The interactions among sub-models are presented in Fig. 3.5. VM Provisioning Sub-Model (VMSM) computes the mean success probabilities (P_h, P_w and P_c) that at least one PM in a pool (hot, warm and cold, respectively) can be assigned to a task. Success probabilities are used as input parameters to the Resource Allocation Sub-Model (RASM). The hot PM Sub-Model (VMSM_hot) computes P_h which is the input parameter for both warm and cold sub-models. The warm PM submodel (VMSM_warm) computes P_w which is the input parameter for the cold sub-model (VMSM_cold). Moreover, 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). On the other hand, PMSM computes N_h, N_w and N_c as input for hot, warm and cold PM sub-models respectively. The RASM computes the blocking probability, P_{bq}, which is the input parameter to VM provisioning sub-models and PMSM.

Task waiting time is obtained as the sum of four components: waiting time in the global input queue, until the processing by RAM; RAM processing time; waiting time in the PM queue; lastly, the time for VM instantiation and deployment. Total response time is obtained by adding the waiting time to the duration of the actual service time.

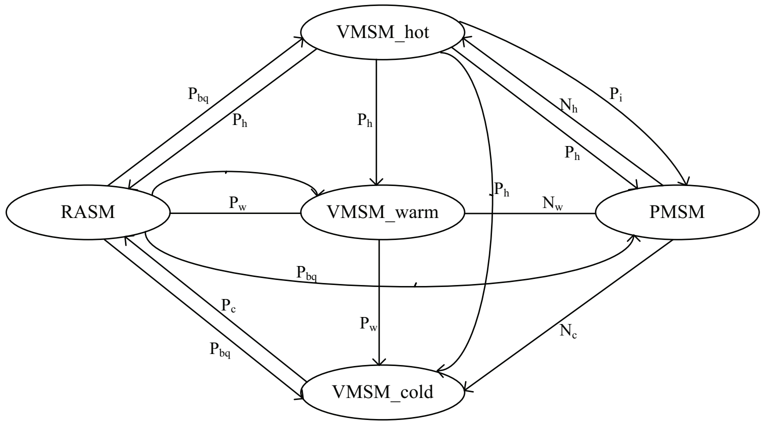


Figure 3.5: Interaction diagram among sub-models (adapted from [33]).

Algorithm 1 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 common input queue: $P_{bq}$;
count $\longleftarrow 0$; maximum $\longleftarrow 30$; $\Delta \longleftarrow 1$;
$P_{bq0} \longleftarrow$ RAM $(Ph0, Pw0; Pc0)$;
$[N_h, N_w, N_c] \longleftarrow$ PMM $(P_{h0}, P_{i0})$
**while** $\Delta \geq 10^{-6}$ **do**
    count $\longleftarrow$ count +1;
    $[P_h, P_i] \longleftarrow$ VMM_hot $(P_{bq0}, N_h)$;
    $P_w \longleftarrow$ VMM_warm $(P_{bq0}, P_h, N_w)$;
    $P_c \longleftarrow$ VMM_cold $(P_{bq0}, P_h, P_w, N_c)$;
    $[N_h, N_w, N_c] \longleftarrow$ PMM $(P_h, P_i)$
    $P_{bq1} \longleftarrow$ RAM $(P_h, P_w; P_c)$;
    $\Delta \longleftarrow |(P_{bq1} - P_{bq0})|$;
    $P_{bq0} \longleftarrow P_{bq1}$;
    **if** count = maximum **then**
        break;
    **end if**
**end while**
**if** count = maximum **then**
    **return** -1;
**else**
    **return** $P_{bq0}$;
**end if**
```

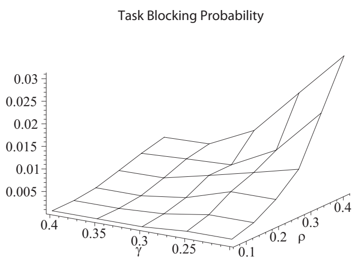
3.1.2 Performance

The analytical model has been solved using Maple 15 from Maplesoft Inc. [26].

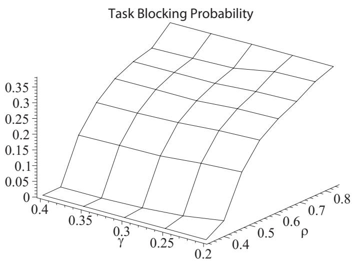
To evaluate the performance of the pooled cloud system and to investigate the performance-energy trade-off in more detail, we have solved the model for two scenarios. First, we kept the task service time constant and varied task arrival rate. Second, we kept the task arrival rate constant but varied task service time. In both cases, we have kept the total number of PMs constant at $N = 100$ while varying the initial proportion of PMs allocated to different pools: γN PMs in the hot and warm pools each, and $(1 - 2\gamma)N$ PMs in the cold pool. In addition, the number of PMs in the hot pool was kept at or above γN, while the other two pools were allowed to change. Transition to the warm pool was initiated when the number of idle hot PMs exceeded 2, and the same threshold was used for the transition from warm to cold pool.

Mean provisioning time for a task consists of the following components: mean waiting time in the input queue, mean time for pool look-up (which was shown to have a Coxian distribution [33]), mean waiting time in the queue of the allocated PM, and mean waiting time for VM provisioning. To obtain the total service time, we need to add the actual time of request execution.

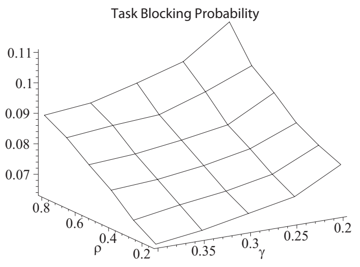
In the performance evaluation process, we have assumed that the service time changes in the range of 20 to 120 minutes based on the cloud users' experience. The task arrival rate in a cloud datacenter can change during the different times of day or night. In order to capture the behavior of the system during different times, in the first scenario we have varied task arrival rate in the range of 100 to 1200 tasks per hour.



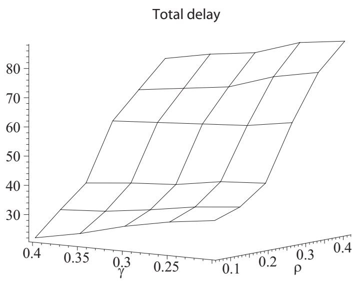
(a) Blocking at service time 40 min, task arrival rate variable from 100 to 600 per hour.



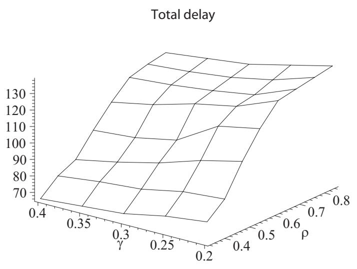
(b) Blocking at service time 40 min, task arrival rate variable from 500 to 1200 per hour).



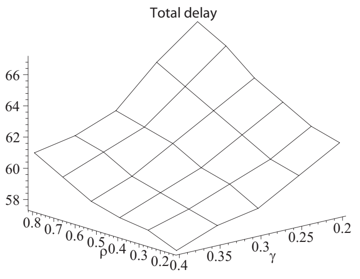
(c) Blocking at task arrival rate 400 per hour, mean service time variable from 20 to 120 minutes.



(d) Total delay (seconds) at service time 40 min, task arrival rate variable from 100 to 600 per hour.



(e) Total delay (seconds) at service time 40 min, task arrival rate variable from 500 to 1200 per hour.

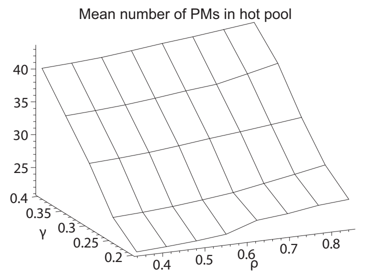


(f) Total delay (seconds) at task arrival rate 400 per hour, mean service time variable from 20 to 120 minutes.

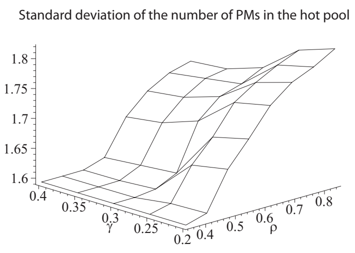
Figure 3.6: Task blocking probability and total delay.

Task Blocking and Total Task Delay

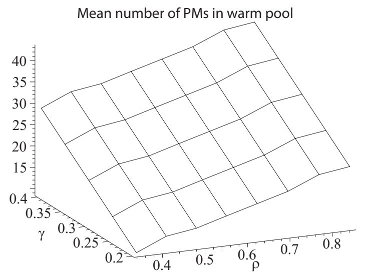
In Fig. 3.6, we analyze the effect of service time and task arrival rate on blocking probability and total delay for a task. To facilitate comparison under different combination of fixed and variable independent variables, we have plotted the diagrams as functions of γ and offered load calculated as $\rho = \frac{\lambda_t}{10N\mu_{tot}}$, where the maximum number of VMs running on a single PM is assumed to be 10. For clarity, we have divided the range of offered loads for the scenario with constant service time into two sub-ranges; the corresponding results are shown in the diagrams in the leftmost and middle columns of Fig. 3.6.



(a) Mean number of PMs in the hot pool.



(b) Standard deviation of the number of PMs in the hot pool.



(c) Mean number of PMs in the warm pool.

Figure 3.7: Pertaining to steady-state partitioning of PMs into pools. Mean service time fixed at 40 minutes, task arrival rate variable from 500 to 1200 per hour).

As expected, both request blocking and total delay increase with load. Below the load of $\rho \approx 0.3$ to 0.4, Figs. 3.6a and 3.6d, the cloud datacenter operates in linear regime with low blocking and reasonably small delay.

Beyond this load, however, blocking rapidly increases as does the delay. Figs. 3.6b and 3.6e show the increase of delay which appears to be gradual but only because a large number of task requests (over 10%) is rejected.

On the other hand, increasing the mean service time whilst keeping the task arrival rate constant, shown in Figs. 3.6c and 3.6f, results in an increase of blocking and delay which are much smoother. Note, however, that the datacenter operates in linear regime, well beyond the saturation limit, even though the value of ρ does increase above the threshold identified in the other four diagrams.

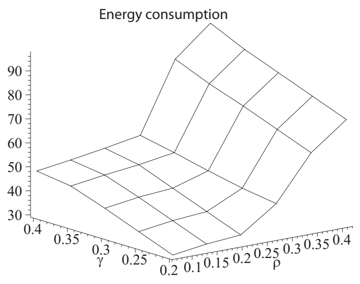
We note that for the same offered load, the cloud center appears to be more sensitive to the task arrival rate than to task service time, which is due to the overhead imposed by the provisioning process which increases with the number of tasks but is independent of the task service time.

Pool Management

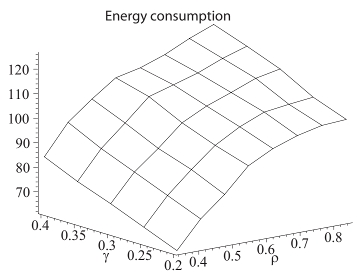
The observations above can be corroborated by calculating the steady-state number of PMs in different pools. The number of PMs in both hot and warm pools, shown in Figs. 3.7a and 3.7c, exhibit a steady increase which is approximately linear function of the offered load, in one dimension, and similarly a linear function of the parameter γ. (We note that the distinction between variable arrival rate and variable service time does not apply here, since the overhead is incurred *before* the actual provisioning of tasks.) As can be seen, higher load leads to a shift in the partitioning of PMs, from the initial ratio defined by γ, towards an ever increasing number of PMs in the hot and warm pools, and the corresponding depletion of the cold pool.

However, the increase is far from being stationary, as witnessed by the diagram of standard deviation of the number of PMs in the hot pool in Fig. 3.7b. As the offered load increases, the standard deviation exhibits a rapid increase, which means that the fluctuation of the number of PMs increases, as many PMs are being moved to and from the hot (and warm) pool in order to cater to the variable load.

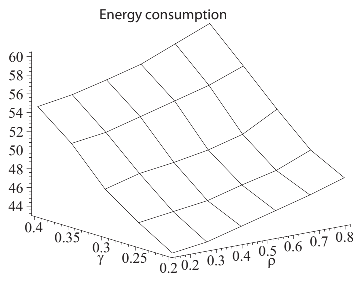
At the same time, at low values of offered load, the mean number of PMs in the warm pool, Fig. 3.7c, is noticeably lower than that in the hot pool, Fig. 3.7a. Only when the load increases towards a rather high value of $\rho = 0.8$ do the two numbers begin to converge, meaning that most PMs are either hot or warm, and only a handful ever remains in the cold pool.



(a) Energy consumption at service time 40 min, task arrival rate variable from 100 to 600 per hour.



(b) Energy consumption at service time 40 min, task arrival rate variable from 500 to 1200 per hour.



(c) Energy consumption at task arrival rate 400 per hour, mean service time variable from 20 to 120 minutes.

Figure 3.8: Energy consumption as function of offered load and partitioning of PMs into pools.

3.1.3 Energy Consumption

The fluctuation of the number of PMs in the three pools is reflected on energy consumption of the cloud center. Let the power consumption of a hot PM be δ_p, while that of a warm PM be $W_c\delta_p$ (the power consumption of a PM in the cold pool is, obviously, zero). If we denote the mean time spent in each state of the CTMC for the PMM, Fig. 3.4, with $\overline{T_{st}}$, the total energy consumption is

$$E_c = \sum_{s\in\zeta}(N_{h_s} + W_c N_{w_s})\delta_p\overline{T_{st}} \tag{3.5}$$

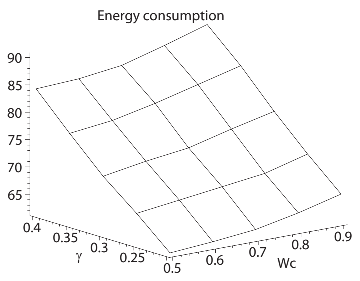
where ζ is the set of PMM states and N_{h_s} and N_{w_s} denote the number of PMs in hot and warm pool, respectively, in state s of the PMM.

Our first experiment follows closely the scenarios outlined in the previous Section, with the ratio of power consumption of a warm PM vs. that of a hot one fixed at $W_c = 0.5$. (For simplicity, we express all energy consumption values relative to the energy consumption of a hot PM with all 10 VMs instantiated.) The results are shown in Fig. 3.8; as before, we have split the range of observed values for the offered load in order to highlight the difference between linear and saturation regimes. As can be seen, the shape of surfaces obtained under variable task request arrival rate, Figs. 3.8a and 3.8b, clearly indicate the boundaries of the linear regime in which the energy consumption is low and not very dependent on the offered load. Of course, if the initial partitioning of PMs gives preference to PMs in the hot and warm pools, higher energy consumption will result.

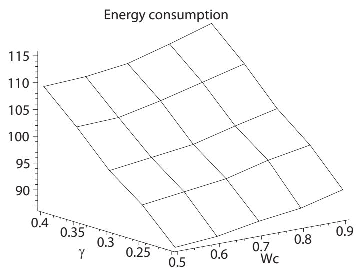
However, as soon as the task arrival rate exceeds the value of $\rho \approx 0.3$, energy consumption begins to rise at a considerable rate, due to higher proportion of PMs being in the hot pool, but also due to longer time spent in switching to and from hot state. (We assume that energy expenditure of a PM during switching is equal to that of a fully loaded hot PM.) Energy consumption appears to flatten at high loads above $\rho \approx 0.6$, but only because most of the PMs are in hot and warm pools most of the time, switching only occasionally to the cold state.

When the task arrival rate is constant, energy consumption exhibits a nearly linear dependency on the mean task service time and the partitioning coefficient γ, as can be seen from Fig. 3.8c. This behavior is similar to that observed for blocking probability and total task delay in Fig. 3.6.

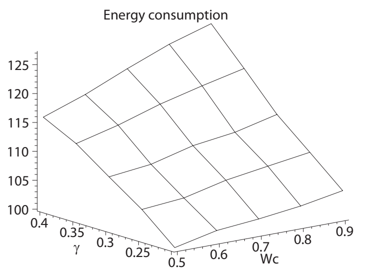
Our final experiment involved energy expenditure under constant task request arrival rate and mean service time (i.e., under constant offered load), but with variable power consumption ratio W_c of a PM in the warm state vs. that of a PM in the hot state. The resulting diagrams are shown in Fig. 3.9, where energy expenditure (relative to the energy consumption of a single fully loaded PM) is nearly linearly dependent on both γ and W_c.



(a) Under mean arrival rate of 400 tasks per hour.



(b) Under mean arrival rate of 700 tasks per hour.



(c) Under mean arrival rate of 1000 tasks per hour.

Figure 3.9: Energy consumption as function of the ratio of warm-vs.-hot PM power consumption and partitioning of PMs into pools. Mean service time fixed at 40 minutes.

However, it is much more sensitive to the former than to the latter, as the consequence of the fact that the partitioning parameter imposes a lower bound on the number of PMs in the hot pool. Thus the energy expenditure will not drop as much when reducing the load, as in Fig. 3.9a, since the minimum number of PMs in the hot pool is still limited by the value of the partitioning parameter γ. This hints that partitioning into three pools may be inefficient, and that better results with respect to energy consumption could be obtained by simply having two pools, a hot and cold one, despite performance degradation possibly incurred in this setup. This remains a promising direction for further research.

3.2 Analyzing The Impact of Provisioning Overhead Time in Cloud Computing Centers

In an IaaS cloud, an incoming task service request undergoes several processing steps before being actually provisioned on a VM executing on a designated PM [33]. These processing steps generally pertain to the search for a suitable PM that will provision the task; moving the task to the selected PM; and, finally, waiting for the instantiation of the VM on which the task will execute. Admission control through task rejection or blocking is thus performed at two steps along the way: at the global input queue and at the input queue of the selected PM.

To reduce energy expenditure incurred by PMs running idle, PMs are often partitioned into pools. For example, the model presented in [32] deals with three pools, hot pool with PMs that are always on and VMs already instantiated, warm pool with PMs that are on but without any instantiated VMs, and cold pool that consists of PMs switched off. In this setup, PMs are moved from cold to warm to hot state in order to fulfill the incoming task requests, and moved back to warm or cold pool when there is idle capacity in order to reduce energy consumption. The performance of the cloud center is then evaluated using the tools of probabilistic analysis and queueing theory. However, the emphasis was on the performance in saturation region, which is not the preferred operational regime for cloud providers as the achievable performance levels, esp. task blocking rate, is not very good; moreover, the analysis in that paper does not consider energy consumption in detail.

An extension of that analysis is presented in Section 3.1 where the focus was on linear and transitional regime, rather than on saturation regime. An interesting finding was that the aggregated metric of offered load, similar to the metric used to characterize networking systems, is insufficient to characterize the behavior of the cloud system. This is caused by the overhead incurred in processing the incoming task requests and admission control activities, which are dependent on the task arrival rate but not on the task service time.

In this section, we consider a model in which partitioning uses two pools: the hot pool in which PMs are running with the maximum number of VMs instantiated and ready to provision user requests, and the cold pool in which PMs are turned off. We assume that the number of PMs in the hot pool does not fall below a predefined threshold, which serves to maintain the performance at an acceptable level. Our focus is again on the linear and transitional operational regimes which are much more interesting to cloud service providers, rather than the saturation regime in which task blocking reaches values that are unacceptably high in practice. We evaluate the performance of the cloud system expressed as task blocking rate and energy expenditure. Furthermore, we evaluate the sensitivity of the energy expenditure on the partitioning threshold, as well as on the mean look up time in the hot and cold pools.

3.2.1 The Model of The Cloud System

An IaaS cloud center consists of a number of PMs (servers) running a number of VMs each that fulfill task requests submitted by users. All task requests are provisioned using a customized disk image [33]. For simplicity, we assume that the PMs are homogeneous as are the VMs, and that pre-built images satisfy all service requests. Also, we assume that each task request can be provisioned on a single VM.

We implement functional sub-models and their interactions to obtain an overall solution. First, we implement separate sub-models for different provisioning steps of a cloud service. In the next step, we obtain the overall solution by iteration over individual sub-model solutions and deploying interaction among these sub-models (Fig. 3.10). Our typical cloud center has a number of PMs which will allocate resources to perform tasks in the order of their arrival.

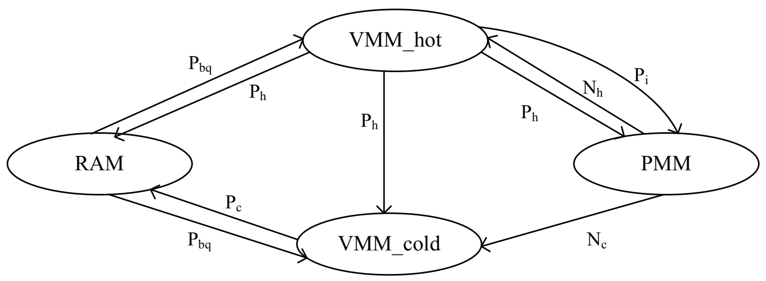


Figure 3.10: Interaction diagram among sub-models.

Resource Allocation

An incoming task request is first received in the global input queue, where it waits processing by a resource allocation module. This module will search the hot and cold pools for the required resources, i.e., PMs with idle capacity. If such a PM is found, the module will allocate the task to it, possibly instructing the PM to be switched on in the process; otherwise, the task may be rejected in what is effectively the first level of admission control. The operation of the resource allocation module can be described with a CTMC shown in Fig. 3.11. Task request arrivals are modeled as a Poisson process with rate λ_t. Task requests are lined up in the global finite queue with a maximum capacity of L_q to be processed. Each state of the CTMC in Fig. 3.11 is labeled as (i, j), where i denotes the number of tasks in the queue and j presents the pool on which the current task is under provisioning. Index h and c indicate hot and cold pool respectively. P_h and P_c present success probabilities of finding a PM that can accept the current task in the hot and cold pool respectively. $1/\alpha_h$ and $1/\alpha_c$ are mean look up times for finding an adequate PM in hot and cold pool respectively. The resource allocation module also calculates the task blocking probability, P_{bq}, which is the input parameter for the modules described below.

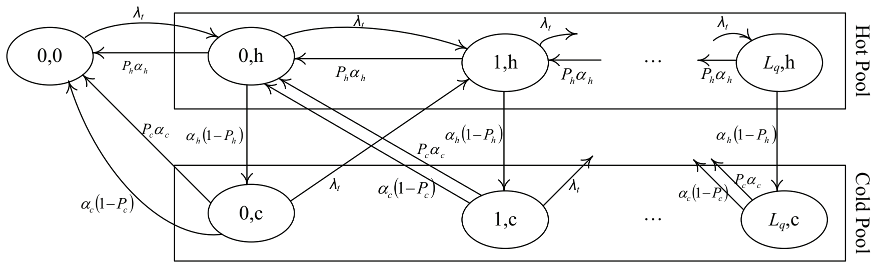


Figure 3.11: Resource allocation.

Virtual Machine Provisioning

If the task is allocated to one of the PMs, it is transferred to the input queue of the PM, which effectively implements the second level of task admission control. Once the task reaches the head of the said queue, the required VM is instantiated and the task is actually provisioned, i.e., the actual service starts. The operation of virtual machine provisioning module managing the hot PM pool can be described with a CTMC shown in Fig. 3.12. Each state of Markov chain is labeled by (i, j) in which i denotes the number of tasks in PM's queue and j is the number of VM that are already deployed on the PM. λ_h is the arrival rate to each PM in the hot pool and is in proportion to λ_t) and N_h. φ_h is the rate at which a VM can be deployed on a PM in hot pool and μ is the service rate of each VM.The total service rate for each PM is the product of number of running VMs by μ. The maximum number of VMs on a PM is equal to m and in this model, it is set to 10. The virtual machine provisioning module calculates the success probabilities (P_h and P_c) that at least one PM in a given pool (hot and cold, respectively) can be assigned to a task. Success probabilities are, then, used as input parameters for the resource allocation module and the other virtual machine provisioning module, as well as for the pool management module described below. In addition, the hot PM model provides the probability by which

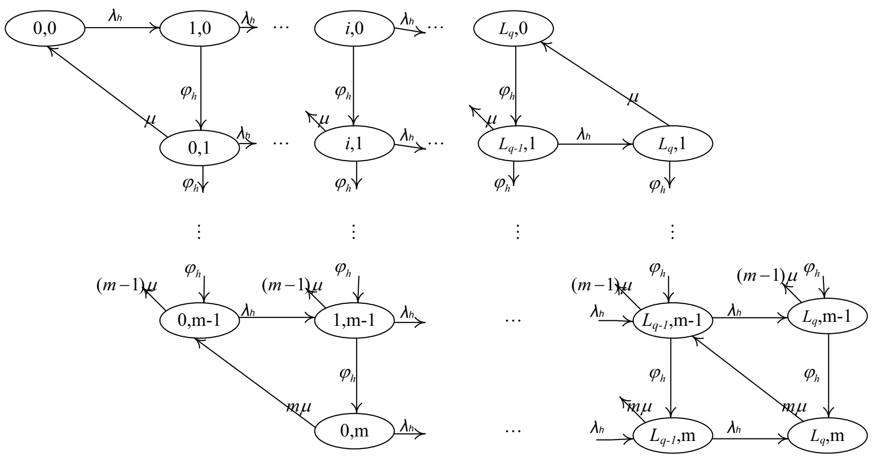


Figure 3.12: Virtual machine provisioning in the hot pool.

a hot PM becomes idle, P_i. An analogous CTMC, but with different transition rates, can be drawn for the corresponding module managing the cold pool.

Also, pool management module computes N_h and N_c as input for hot and cold PM sub-models respectively.

Pool Management

As noted above, a cold PM is switched on and populated with the required number of VMs when there is no capacity in the hot pool to satisfy user requests. Conversely, if a hot PM is idle for a predefined time, it is switched off. However, the number of PMs is never below a predefined threshold in order to maintain acceptable performance. The management of PM pools is done by a separate module, the operation of which can be described by a two dimensional CTMC shown in Fig. 3.13. The system starts from the state (i, j) which indicates that i and j PMs are in the hot and cold pool, respectively. If next search in the hot pool is successful (with probability of P_h), system will remain in the starting state, otherwise one of the cold PMs will be moved into the hot pool. This

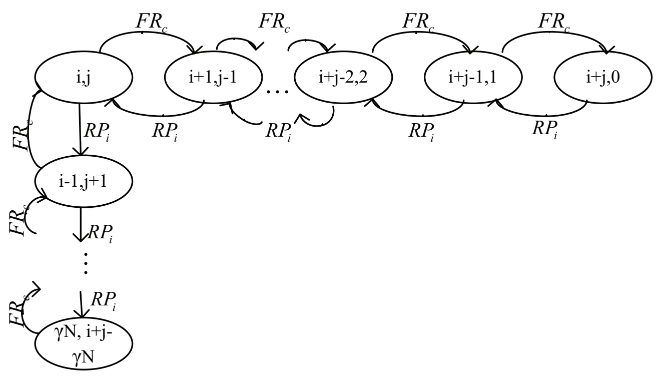


Figure 3.13: Pool management.

transition occurs with the rate of FR_c. With higher demand of PMs, cold PMs can become hot PMs gradually until all the PMs are hot. Also, due to the low demand of resources, idle hot PMs will be turned off gradually, but the number of hot PMs cannot get less than γN. The idle hot PM is moved to the cold pool by rate of RP_i.

Solving the Integrated Model

Our overall model thus consists of three inter-connected stochastic models which compute the cloud performance parameters such as task blocking probability and energy consumption. However, there is a cyclic inter-dependency among the modules, which can be resolved using fixed point iterative method [44] through a modified version of successive substitution approach presented in Algorithm 2.

3.2.2 Performance Evaluation

The proposed model has been solved using Maple 15 [26].

In our experiments, the total number of available PMs is maintained at a constant value of $N = 100$. The threshold coefficient γ determines the initial partitioning of the PMs

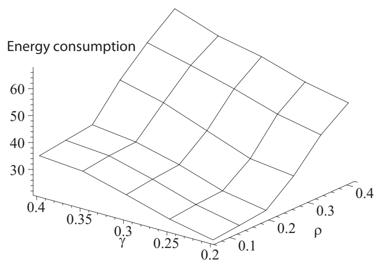
Algorithm 2 Iterative Algorithm for Substitution

```
Input: Starting success probabilities in pools: $P_{h0}$, $P_{c0}$;
Input: Starting idle probability of a hot PM: $P_{i0}$;
Output: First level of probability of blocking: $P_{bq}$;
count $\longleftarrow$ 0; maximum $\longleftarrow$ 30; $\Delta \longleftarrow 1$;
$P_{bq0} \longleftarrow$ RAM $(Ph0, Pc0)$;
$[N_h, N_c] \longleftarrow$ PMM $(P_{h0}, P_{i0})$;
while $\Delta \geq 10^{-6}$ do
    count $\longleftarrow$ count +1;
    $[P_h, P_i] \longleftarrow$ VMM_hot $(P_{bq0}, N_h)$;
    $P_c \longleftarrow$ VMM_cold $(P_{bq0}, P_h, N_c)$;
    $[N_h, N_c] \longleftarrow$ PMM $(P_h, P_i)$;
    $P_{bq1} \longleftarrow$ RAM $(P_h, P_c)$;
    $\Delta \longleftarrow |(P_{bq1} - P_{bq0})|$;
    $P_{bq0} \longleftarrow P_{bq1}$;
    if count = maximum then
        break;
    end if
end while
if count = maximum then
    return -1;
else
    return $P_{bq0}$;
end if
```

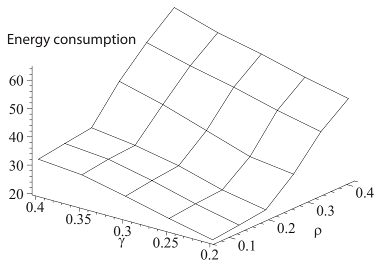
between hot and cold pools, with γN PMs in the hot pool and $(1-\gamma)N$ PMs in the cold pool. As noted above, the number of PMs in the hot pool can never drop below γN.

Due to space limitations, we do not show the actual equations describing the model, as they closely follow the analysis presented in Section 3.1. Instead, we show just the results obtained from the solution of the model.

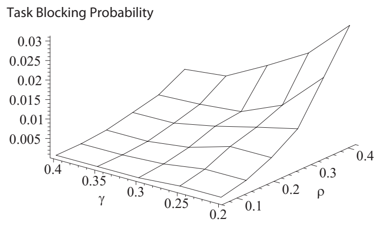
The response time or total delay is the sum of four provisioning times: waiting in the global queue, processing in the resource allocation module, waiting time in the PM queue, and VM instantiation and deployment, to which the actual service time should be added [32–34]. We note that:



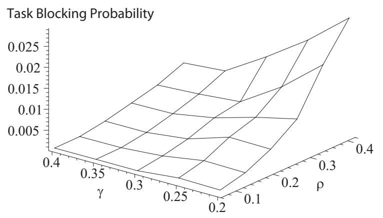
(a) Energy consumption, mean look up time in the cold pool 12 seconds.



(b) Energy consumption, mean look up time in the cold pool 7.2 seconds.



(c) Task blocking probability, mean look up time in the cold pool 12 seconds.



(d) Task blocking probability, mean look up time in cold pool 7.2 seconds.

Figure 3.14: Performance of cloud datacenter at mean service time 40 minutes, mean look up time in the hot pool 18 seconds, task arrival rate variable from 100 to 600 per hour.

- Mean waiting time in global queue ($\overline{wt}$) is exponentially distributed with mean value of task arrival rate ($1/\lambda_t$).
- Mean look up time between the pools ($\overline{lut}$) has a Coxian distribution with two steps of look-up time between hot and cold pools [33].
- Mean waiting time in the input queue of the select PM ($\overline{PM_{wt}}$) is exponentially distributed for each pool with mean values of arrival rate ($1/\lambda_h$ and $1/\lambda_c$ that are in proportion to $1/\lambda_t$).
- Mean waiting time for VM provisioning ($\overline{pt}$) is exponentially distributed with mean value of task arrival rate ($1/\lambda_t$).

Cloud System Behavior at Low Loads

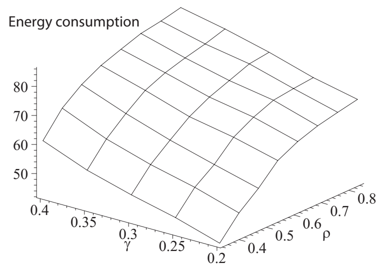
In our first experiment, we have investigated the impact of system load and pool threshold on energy consumption and task blocking, under the conditions of low system load (values of ρ up to 0.4). Mean task service time is assumed to be constant at 40 minutes, so that the adjustment of system load is achieved through the adjustment of task arrival rate. Task rejection probability is obtained from the model. Energy consumption is calculated by considering the mean time $\overline{T_{st}}$ spent in each state of the Markov chain in Fig. 3.13 and power consumption δ_p of a hot machine, as

$$E_c = \sum_{s \in \xi} N_{h_s} \, \delta_p \, \overline{T_{st}} \tag{3.6}$$

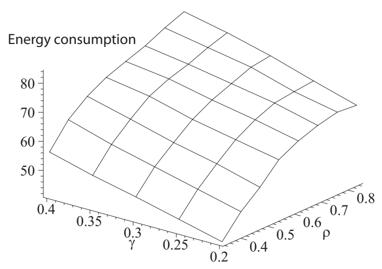
where ξ is the set of states of the Markov chain from Fig. 3.13, while N_{h_s} is the number of PMs in hot pool in state s. In this work, δ_p is assumed to be 1, as all PMs are assumed to be homogeneous; the resulting values of energy expenditure may be interpreted as 'the mean number of PMs that are on.' The resulting diagrams are shown in Fig. 3.14.

As can be seen, energy consumption, shown in Figs. 3.14a and 3.14b, is approximately linearly dependent on the pool threshold, as could be expected. However, it is initially nearly independent of the system load, up about $\rho = 0.2$, but then increases rapidly. This is caused by the fact that in the first part of the range of values for ρ, the current number of PMs in the hot pool suffices to service the incoming task requests. However, the increase of the load in second part of the range of values for ρ means that more PMs are switched on (i.e., moved into the hot pool), with the associated increase in energy consumption.

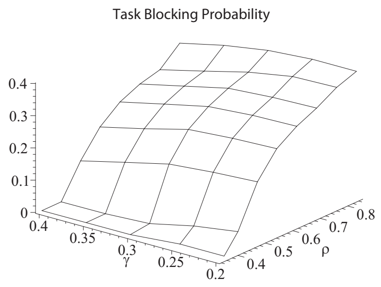
At the same time, the blocking probability in Figs. 3.14c and 3.14d is comparatively



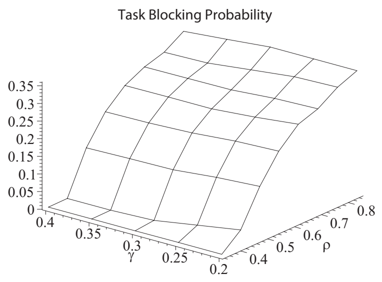
(a) Energy consumption, mean look up time in the cold pool 6 seconds.



(b) Energy consumption, mean look up time in the cold pool 4.5 seconds.



(c) Task blocking probability, mean look up time in the cold pool 6 seconds.



(d) Task blocking probability, mean look up time in cold pool 4.5 seconds.

Figure 3.15: Performance of cloud datacenter at mean service time 40 minutes, mean look up time in the hot pool 7.2 seconds, task arrival rate variable from 500 to 1200 per hour.

low, well below 1% in the most part of the observed range, but it does rise when the system load increases. Also, blocking probability remains low as long as the pool threshold is kept near the maximum value of $\gamma = 0.4$, but it does increase at lower values of γ. The rationale for such behavior is simple: when the pool threshold is low, most PMs are kept in the cold pool (i.e., they are switched off), and they are brought in to the hot pool only when there is a need. However, this action incurs a certain overhead which may cause some tasks to be rejected. Keeping more PMs in the hot pool helps reduce the number of rejected tasks and keeps the blocking probability low; since the mechanism of moving the PMs between pools is continuously adaptive to the instantaneous load, the increase in blocking probability is gradual and does not exhibit a distinct threshold such as the one observed in the diagrams of energy consumption.

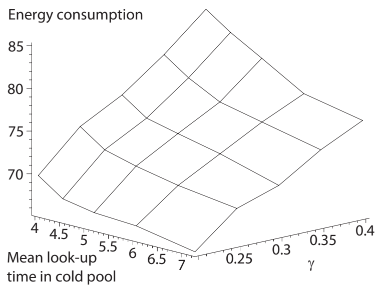
We have also used two values for the mean look up time in the cold pool which correspond to the mean look up rates of 300 (diagrams on the left) and 500 searches per hour (diagrams on the right). The mean look up time in the hot pool was kept constant at 200 searches per hour. We note that faster search reduces both the energy consumption and the task rejection probability, although the differences are not significant in either case.

Cloud System Behavior at High Loads

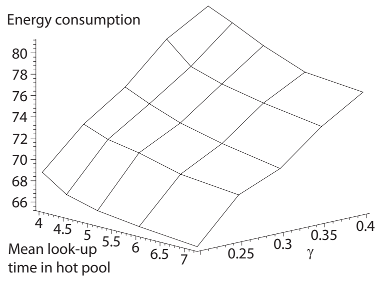
We have repeated the first experiment but with the range of system loads that is wider and extends well into higher loads, up to $\rho = 0.8$; the mean look up times have been reduced in order to make the system more responsive. As before, the changes in system load were accomplished by changing the mean task arrival rate while the mean task service time was kept constant. The results are shown in Fig. 3.15.

As can be seen, the higher values of system load lead to considerable increase in energy consumption as well as in task blocking. As the task arrival rates are higher, a distinct threshold may be observed in the diagrams for task blocking probability, Figs. 3.15c and 3.15c. The corresponding thresholds in the diagrams for energy consumption can't be seen – in fact, they would be visible if the range of system loads were extended downward towards values of $\rho < 0.2$.

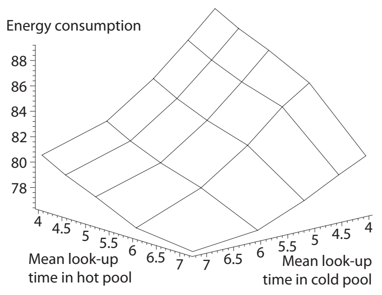
But in either case, higher system load leads to higher energy consumption and considerably higher task rejection rate. The values seem to flatten at high system loads, but only because the system has effectively entered saturation, with a large number of incoming



(a) Energy consumption as the function of mean look up rate in the cold pool; mean look up rate in the hot pool constant at 7.2 seconds.



(b) Energy consumption as the function of mean look up rate in the hot pool; mean look up rate in the cold pool constant at 7.2 seconds.



(c) Energy consumption as the function of mean look up rates in the hot and cold pools; pool threshold kept at a constant value of $\gamma = 0.4$.

Figure 3.16: The impact of look up times on energy consumption. Mean task arrival rate 1000 per hour, mean task service time 40 minutes.

tasks rejected (values are well over 10%). Operation in this regime is most likely unacceptable in practice and should be avoided.

As before, shorter mean look up times (i.e., faster searches) give a slight improvement in performance. This has motivated us to conduct the experiments described in the next Subsection.

The Impact of Look up Times on Energy Consumption

Finally, we have investigated the impact of look up times on energy consumption; the resulting diagrams are shown in Fig. 3.16. As can be seen, lower values of the mean look up time in the cold pool results in higher energy consumption which is due to faster searches and quicker bringing up of new PMs into the hot pool. The increase is somewhat higher than in the case of corresponding mean look up time in the hot pool, which is caused by the additional overhead needed to turn on a cold PM. The difference in impact between the mean look up times in the cold and hot pools is confirmed in Fig. 3.16c.

3.3 Chapter Summary

In Section 3.1 we have examined the behavior of an IaaS cloud data center in which servers are partitioned into pools of hot, warm, and cold machines. We have focused on the operation in the linear regime, and we have shown that the transition to saturation is clearly visible from the diagrams of task request blocking probability. We have shown that the manner in which servers are partitioned in the pools affects the performance, sometimes even more than the variations of offered load. We have also shown that the task arrival rate is more critical parameter affecting the performance of the cloud data center than mean task service time, due to the overhead generated in resource allocation and provisioning. We have also shown that the energy expenditure is highly dependent on the manner in which servers are partitioned into pools, and that reduced power consumption of servers kept in the warm state does not result in commensurate savings in energy.

Also, we have analyzed the effects of pool threshold and mean look up time in hot and cold pools on the performance of an IaaS cloud system where PMs are partitioned into a hot and a cold pool in Section 3.2. Our results confirm that the system should operate well below saturation in order to provide acceptably low task rejection probability and energy consumption. Also, we have shown that the performance is more sensitive to the mean look up time for the PMs in the cold pool.

A continuation to the work discussed in this chapter can be investigation of the performance of the networked cloud system as well as of the system with migration of live VMs between servers (PMs).

Chapter 4

Task Admission Control for Cloud Server Pools

This chapter provides a model for task admission control and resource allocation in the cloud infrastructure. In Section 4.1, we have investigated the operating regions of the proposed cloud model and evaluated the performance of resource allocation in the proposed model. In order to prevent the cloud system getting into the saturation region, we have presented two algorithms to control the admission of incoming tasks. These algorithms are based upon establishing thresholds for task arrival rate and task blocking probability. Our experiments indicate that these simple admission control algorithms can vastly improve system performance.

In Section 4.2, we develop two task admission control algorithms which are based on task filtering policy. First, we present a lightweight task admission control algorithm which is appropriate for the cloud systems with smooth changes of task arrival rate. This algorithm is based on long-term estimation of average utilization and offered load. Second, we introduce a task admission algorithm which is efficient for highly dynamic systems and is based on instantaneous utilization. The performance of the proposed schemes is evaluated against varying intensities of offered load. Performance evaluation of the proposed schemes in Section 4.2.2 confirms that both of them are able to ensure that the system is kept in the stable operating region.

In Section 4.2, we have also conducted experimental simulations to examine the short-term variability of the system. We have also investigated the effect of the size of tasks' waiting queue and the filtering coefficient on the system performance through the simulation model. Our simulation results show that utilizing larger waiting queue improves the task blocking rate. Also, our observations confirm that using more aggressive scheme of filtering coefficient in the lightweight algorithm decreases the task blocking rate and delay.

This chapter is organized as follows: Section 4.1 presents our task admission control solution which are based on establishing thresholds for task arrival rate and task blocking probability. Section 4.2 describes the proposed admission control mechanisms which are based on task filtering policy. Section 4.3 concludes the chapter.

4.1 Task Admission Control in Cloud Based on Thresholds of Task Arrival Rate and Blocking Probability

In Chapter 3, we have introduced virtualization as an effective technique for maximizing the utilization of physical servers (hosts) in an IaaS cloud datacenter, with a number of VMs running on a given host. Furthermore we have discussed that energy expenditure of such servers can be minimized by pooling of hosts. In Section 3.1, servers are partitioned

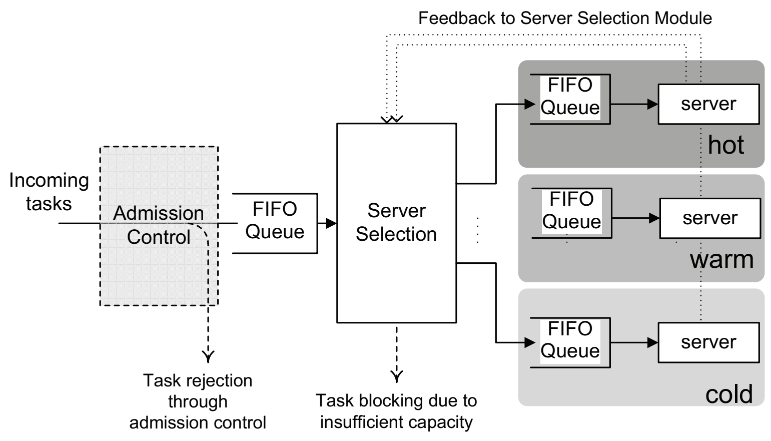

Feedback to Server Selection Module
Incoming tasks
Admission Control
FIFO Queue
Server Selection
FIFO Queue
server
hot
FIFO Queue
server
warm
FIFO Queue
server
cold
Task rejection through admission control
Task blocking due to insufficient capacity

Figure 4.1: System model.

into a hot pool (always on and with VMs instantiated and ready to run), a warm pool (on but without VMs instantiated), and a cold pool with hosts turned off. Servers are, then, moved from one pool to another as needed to fulfill user requests or to conserve energy.

In this setup, maintaining desired performance levels is a major concern. In particular, accepting tasks when they arrive, only to reject them later, might lead to performance deterioration for tasks already taken into service and might also damage the reputation of the cloud service provider. Admission control is an effective mechanism to enforce those performance levels and fulfill their respective SLAs with users as required. In this section, we propose a simple yet effective admission control mechanism that can be easily added to an existing cloud center and investigate its performance as the function of system load and baseline partitioning of servers into pools.

4.1.1 System Model and Its Performance

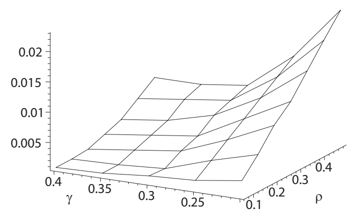
System Model

The system model of a cloud server pool with a number of servers or hosts is shown in Fig. 4.1. We assume that the system has a common input queue. Task requests arrive according to a Poisson process with arrival rate λ and they are served in the FIFO order. Without loss of generality, we assume that all servers are homogeneous as are the VMs; furthermore, we assume that a single VM image can satisfy all requests for a task.

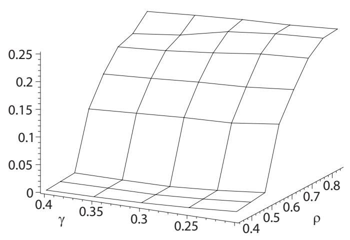
To accommodate a request, the server selection module checks the server pools to find whether there is a server with sufficient number of idle VMs. The hot pool is checked first; if a server with sufficient number of idle VMs is found, the task can be serviced immediately. Otherwise, the warm pool is checked; if a warm server is to be used, it must first instantiate the required number of VMs which incurs some delay. Finally, the cold pool is checked, but bringing a server from the cold pool to the hot one requires additional delay for server start-up and VM instantiation. Either way, the request is routed to the server FIFO queue where it awaits for the VM that will provision it.

If there is no server with sufficient capability, the request will be rejected. Requests can also be rejected if there is no space in the input queue.

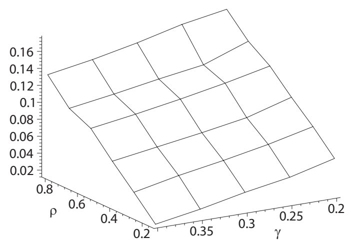
Initial partitioning of available N servers is performed as follows: γN PMs are allocated to the hot and warm pools each, and the remaining $(1-2\gamma)N$ PMs are allocated to the cold pool. Servers are 'upgraded' (i.e., moved from cold to warm, or from warm to hot pool) when requests need to be provisioned; they are 'downgraded' as soon as requests finish service so that all VMs on a hot server become idle. To keep the system performance at an acceptable level, the number of hosts in the hot pool is never reduced below γN; the



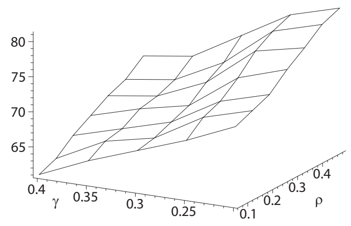
(a) Blocking at service time 40 min, task arrival rate variable from 100 to 600 per hour.



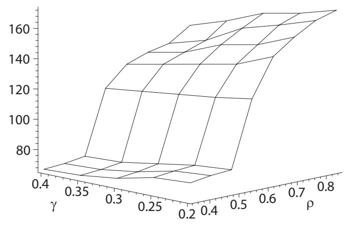
(b) Blocking at service time 40 min, task arrival rate variable from 500 to 1200 per hour).



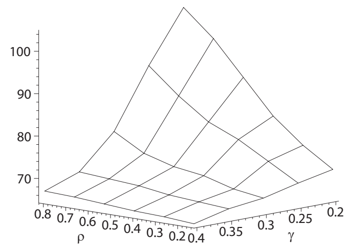
(c) Blocking at task arrival rate 400 per hour, mean service time variable from 20 to 120 minutes.



(d) Total delay (seconds) at service time 40 min, task arrival rate variable from 100 to 600 per hour.



(e) Total delay (seconds) at service time 40 min, task arrival rate variable from 500 to 1200 per hour).



(f) Total delay (seconds) at task arrival rate 400 per hour, mean service time variable from 20 to 120 minutes.

Figure 4.2: Task blocking probability and total delay.

other two pools are allowed to change as needed.

Performance of The Original Pooled System

To investigate the performance of the pooled cloud system described above we have built a discrete event simulator using MATLAB R2013a with the Simulink component [27]. Simulink is a block diagram environment for multi-domain simulation and model-based design. It supports system-level design, simulation, automatic code generation and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries and solvers for modeling and simulating dynamic systems.

Moreover, it enables the designers to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis. Regarding the nature of service/resource provisioning in cloud computing datacenters and also our analytical model which has been developed based upon the queuing systems, we required an efficient tool which could support discrete-event simulation (DES) approach. According to the model's requirements and Simulink's features, we found it a suitable simulator.

The results are shown in Fig. 4.2. The diagrams in the top row show blocking probability as the function of the partitioning coefficient (i.e., default proportion of hot servers) γ and system load $\rho = \frac{\lambda}{m \cdot N\mu_{tot}}$, assuming the distribution of service times is exponential; diagrams in the bottom row show total delay under the same conditions. First two diagrams in each row are obtained by varying the task arrival rate under constant service time; the rightmost diagram is obtained by varying the task service time under constant task arrival rate. For clarity, we have separated the range of offered loads into two sub-ranges: one from $\rho = 0.1$ to 0.5, shown in the diagrams in the leftmost column, and another from $\rho = 0.4$ to 0.8, shown in the diagrams in the middle column; both correspond to mean service time of 40 minutes. The diagrams in the rightmost column were obtained under task arrival rate of 400 tasks per hour.

As can be seen, task blocking rate decreases with γ, which could be expected. When the task arrival rate increases, task blocking also increases. In the lower half of the range of load values, Fig. 4.2a, this increase begins to show at low values of γ and higher values of ρ. However, when we look at the upper range of load values, Fig. 4.2b, we see that the system actually enters saturation when task blocking rapidly increases beyond $\rho = 0.5$. The increase in blocking is more gradual when task arrival rate is constant while mean service time increases, but the overall increase is quite steep, from around 0.02 (i.e., 2% of rejected tasks) to above 0.15 (i.e., 15%); again, this could be expected.

Saturation is also evident on the diagrams that show total task delay, especially in Fig. 4.2e, even though the delay is only about twice of the maximum value in the lower part of the range, Fig. 4.2d. Saturation is more noticeable in the diagram of delay obtained under constant task arrival rate and variable service rate, Fig. 4.2f.

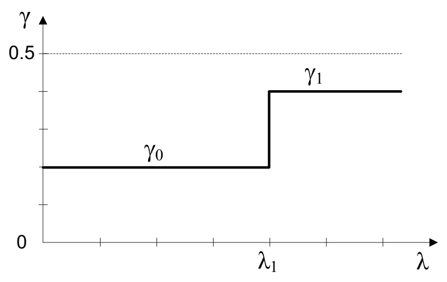
Obviously, if we want to keep the cloud center out of saturation, in other words if we need to keep the delay and rejection rate low, some kind of admission control is desirable. By rejecting task arrivals that stand little to no chance of being serviced immediately, cloud service provider will improve the quality of service for the existing customers.

4.1.2 Simple Admission Control Mechanisms

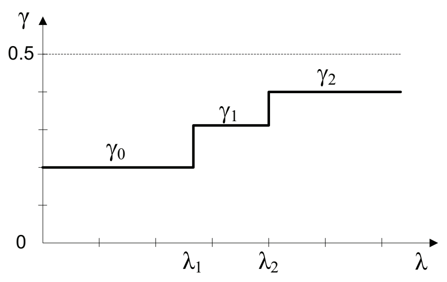
The aim of applying admission control schemes to the pool management system is to keep the cloud system in the linear operation region with low blocking and reasonably small delay. We have seen that the partitioning coefficient γ affects blocking, and we can use this feature to improve performance. Therefore, admission control mechanism executes in two steps.

1. In the first tier, it adjusts the partitioning coefficient according to the mean task arrival rate. When the task arrival rate increases, the partitioning coefficient is increased to soften the impact of increased load, and vice versa. As task arrivals are random events, fluctuations of the task arrival rate are smoothed out by using an exponentially weighted moving average.

2. As the second tier of adjustment, if the probability of the task being rejected by the



(a) Changes of γ according to Algorithm 3.



(b) Changes of γ according to Algorithm 4.

Figure 4.3: Dynamics of changing of γ according to admission control algorithms.

server selection module exceeds a predefined threshold P_{blk0}, the task is rejected outright. Again, the rejection probability is smoothed out via an exponentially weighted moving average.

In this manner, admission control filters out tasks so that most of the rejected ones are rejected outright, whereas only a small predefined percentage of them are rejected only after being processed by the server selection module, which takes additional time. This helps to maintain the rejection rate of the admitted tasks low. Furthermore, the elimination process of the admission control module can be used in a different way to improve performance, i.e., to redirect superfluous task requests to another cloud server center instead; however, the elaboration of this approach is beyond the scope of the present work.

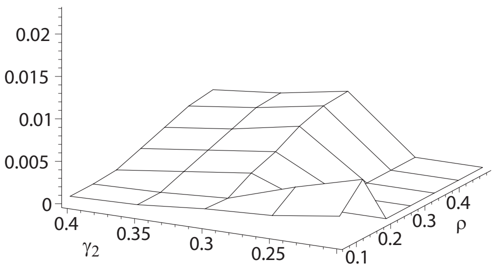
Performance of Admission Control

To investigated the performance of the admission control, we have defined two algorithms. The first one uses two distinct values of the partitioning coefficient, the default value γ_0 and the high-load value γ_1, as shown in Fig. 4.3a; it is shown as Algorithm 3 below. The second one uses a 'softer' changeover with three values for the partitioning coefficient: the default value γ_0, the mid-range value γ_1, and the high range value γ_2, as shown in Fig. 4.3b; it is shown as Algorithm 4 below.

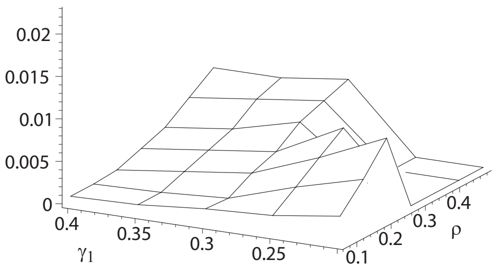
Algorithm 3 Admission control mechanism 1.

```
Set $\lambda \longleftarrow \lambda_0$
while true do
    With each task arrival, recalculate $\overline{\lambda}$
    if $\overline{\lambda} > \lambda_1$ then
        Set $\gamma \longleftarrow \gamma_1$;
    else
        Set $\gamma \longleftarrow \gamma_0$;
    end if
    With each task rejection, recalculate $\overline{P_{blk}}$
    if $P_{blkl} < \overline{P_{blk}} < P_{blk0}$ then
        Drop most recently arrived task with the rate of $\alpha = f(\overline{\lambda})$;
    else if $\overline{P_{blk}} \geq P_{blk0}$ then
        Drop most recently arrived task;
    end if
end while
```

The resulting blocking probability under the two algorithms is shown in Fig. 4.4, with initial partitioning coefficient and system load as independent variables. (Load is varied by varying the task arrival rate.) For Algorithm 3, the thresholds were $\lambda_1 = 300$ tasks per hour, $\gamma_0 = 0.2$, and $\gamma_1 = 0.4$. In the blocking probability range of P_{blkl} to P_{blk0}, dropping rate (α) is a function of $\overline{\lambda}$ which means that by increasing of mean task arrival rate, dropping rate is going to increase as well. In defining α, we have been remotely motivated by packet dropping rate in Random Early Detection (RED) mechanism used in TCP congestion control [18]. For Algorithm 4, the thresholds were $\lambda_1 = 300$ tasks per hour, $\lambda_2 = 400$ tasks per hour, $\gamma_0 = 0.2$, $\gamma_1 = 0.3$ and $\gamma_2 = 0.4$. Mean service time was kept at 40 minutes. As can be seen, both algorithms, together with pool management of the proposed approach, manage to maintain the overall blocking probability within reasonably low range. Interestingly enough, the 'softer' adjustment provided by Algorithm 4 is less



(a) According to Algorithm 3.



(b) According to Algorithm 4.

Figure 4.4: Task blocking probability with admission control.

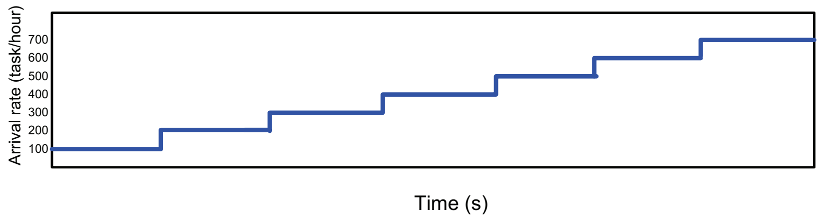
successful in keeping the overall blocking probability low than the 'harder' one provided by Algorithm 3.

Algorithm 4 Admission control mechanism 2.

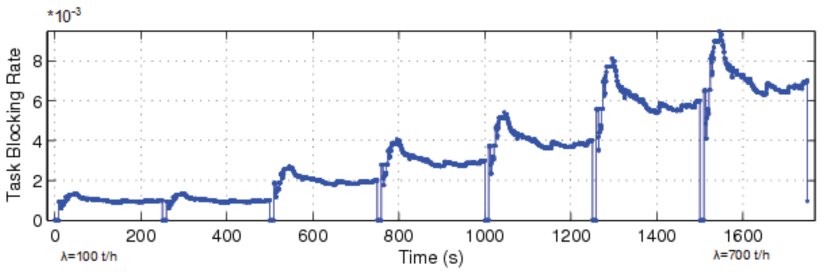
```
Set $\lambda \longleftarrow \lambda_0$
while true do
  With each task arrival, recalculate $\overline{\lambda}$
  if $\overline{\lambda} > \lambda_2$ then
    Set $\gamma \longleftarrow \gamma_2$;
  else
    if $\overline{\lambda} > \lambda_1$ then
      Set $\gamma \longleftarrow \gamma_1$;
    else
      Set $\gamma \longleftarrow \gamma_0$;
    end if
  end if
  With each task rejection, recalculate $\overline{P_{blk}}$
  if $\overline{P_{blk}} \geq P_{blk0}$ then
    Drop most recently arrived task;
  end if
end while
```

As further validation of the efficiency of Algorithm 3, we have plotted the task blocking rate of the system without admission control, with the task arrival rate periodically increased from 100 to 700 tasks per hour after every 250 seconds. Mean task service time was fixed at 40 minutes. As can be seen from the timing diagram in Fig. 4.5b, task blocking rate is steadily increasing with each increment of the task arrival rate. However, when admission control according to Algorithm 3 is applied, task blocking rate is kept reasonably constant, as shown in Fig. 4.5c. The initial increase is due to the smoothing algorithm, which takes some time after the change in task arrival rate to adjust the mean task arrival rate as well as the mean blocking rate. Mean delay that tasks experience is also kept reasonably constant, as shown in Fig. 4.5d, with the same caveat about transitory regime as above. In fact, the delay even decreases slightly, which is due to the fact that some tasks are not admitted in the first place, and therefore do not affect the performance.

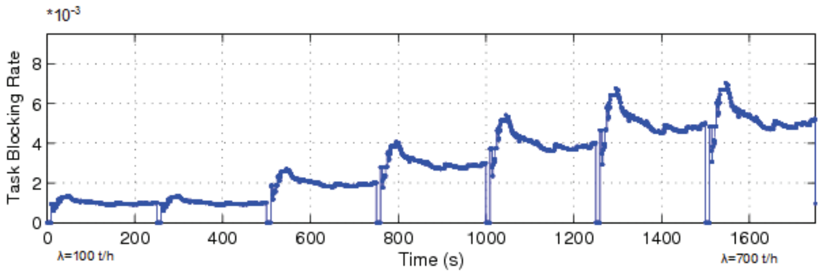
In another test case which its results have been presented in Fig. 4.6, we have increased the task arrival rate from 100 to 700 tasks per hour and then we have decreased the rate to 100 tasks per hour periodically after every 250 seconds. Mean service time was set to 40 minutes. It can be seen that in Fig. 4.6c, using Algorithm 3 is improving the task blocking rate. Moreover, Figs. 4.6d and 4.6e present the changes of task rejection rate and γ respectively; these are the controlling parameters in Algorithm 3 and when task arrival rate increases, task rejection rate and γ will increase as well to reduce the task blocking rate. On the other hand, with decreasing of task arrival rate repeatedly, γ will be reduced to its minimum value and task rejection rate will get to zero.



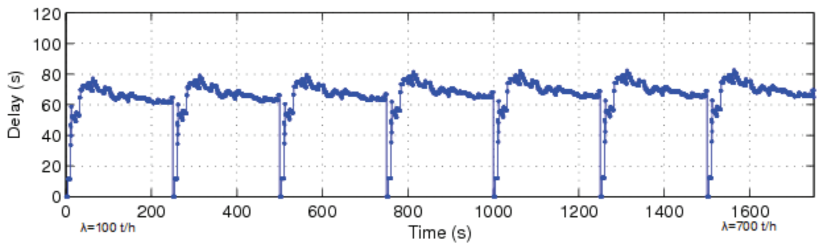
(a) Task arrival rate changes during the test time.



(b) Task blocking rate without admission control.

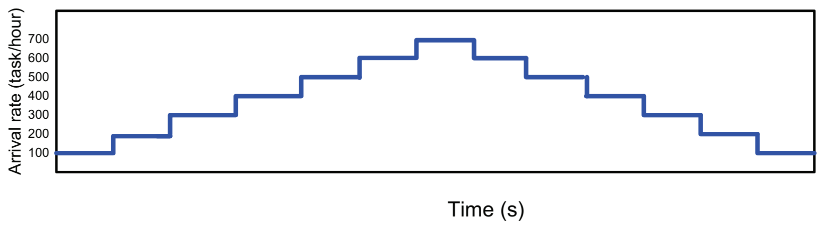


(c) Task blocking rate with admission control (Algorithm 3).

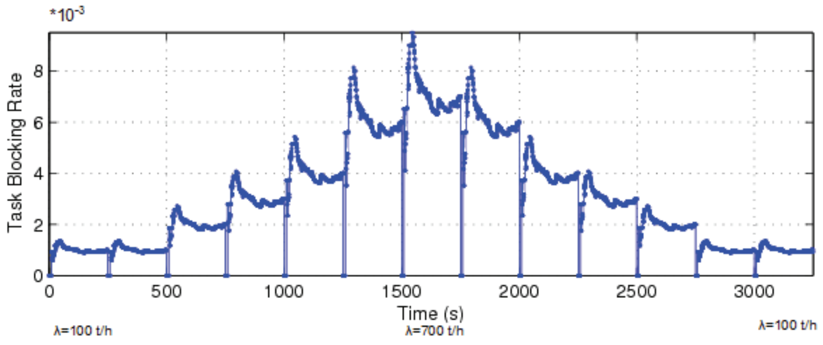


(d) Total delay with admission control (Algorithm 3).

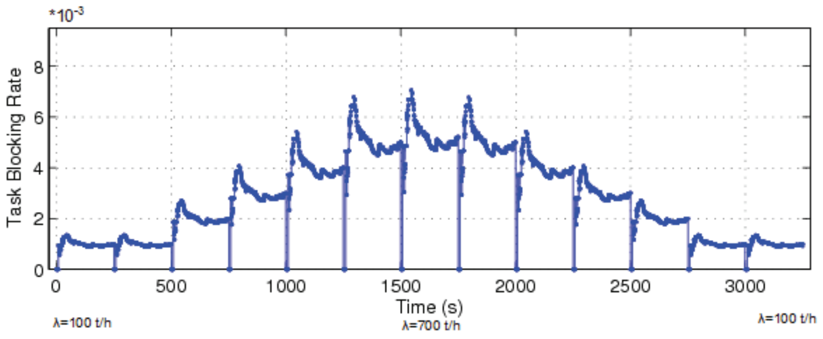
Figure 4.5: Test case of task arrival rate variable from 100 to 700 per hour, service time 40 min.



(a) Task arrival rate changes during the test time.

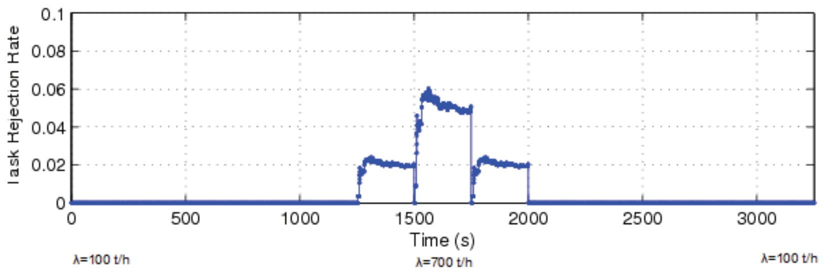


(b) Task blocking rate without admission control.

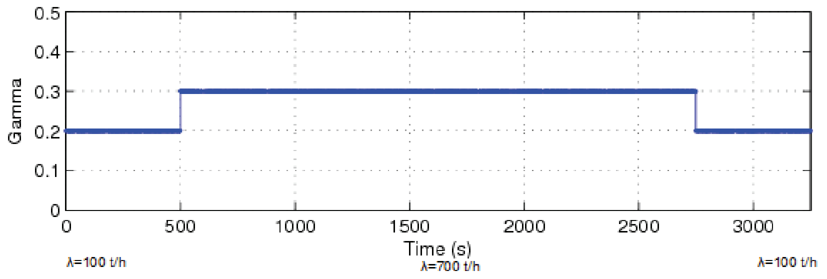


(c) Task blocking rate with admission control (Algorithm 3).

Figure 4.6: Test case of task arrival rate variable from 100 to 700 per hour and down to 100 per hour, service time 40 min.



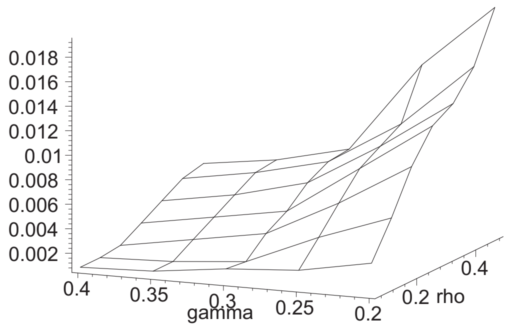
(d) Task rejection rate with admission control (Algorithm 3).



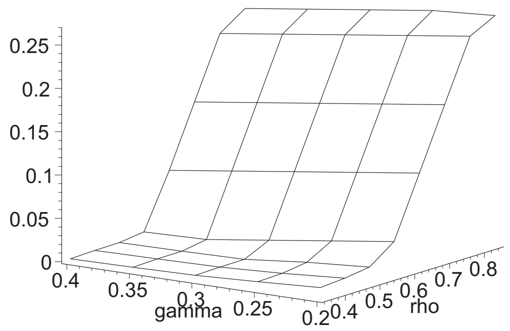
(e) γ change with admission control (Algorithm 3).

Figure 4.6: Test case of task arrival rate variable from 100 to 700 per hour and down to 100 per hour, service time 40 min.

In another test case presented in Fig. 4.7, we have increased the task arrival rate from 100 to 1200 tasks per hour and in a similar case in Fig. 4.8, we have increased the task arrival rate from 100 to 1000 tasks per hour periodically after every 250 seconds and mean service time has been fixed to 40 minutes in both figures. In this test case, we have applied Algorithm 3 to control the admission of arriving tasks and we have ignored the task blocking threshold of P_{blk0} to get into the higher rates of incoming tasks and examine the task blocking rate. The task rejection rate is increased according to the increasing of incoming task rate. As can be seen, in Fig. 4.7a, task blocking rate is less than the case of not using any admission control mechanism (Fig. 4.2a). Furthermore, when task arrival rate increases in the ranges of 500 to 1200 tasks per hour (Fig. 4.7b), task blocking rate will get into the saturation regime in higher rates of task arrival compared to Fig. 4.2b which



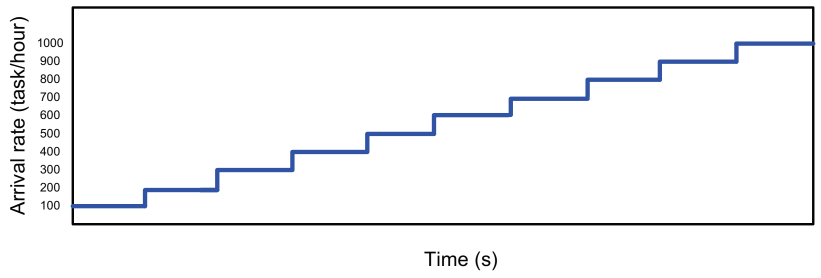
(a) Blocking at service time 40 min, task arrival rate variable from 100 to 600 per hour.



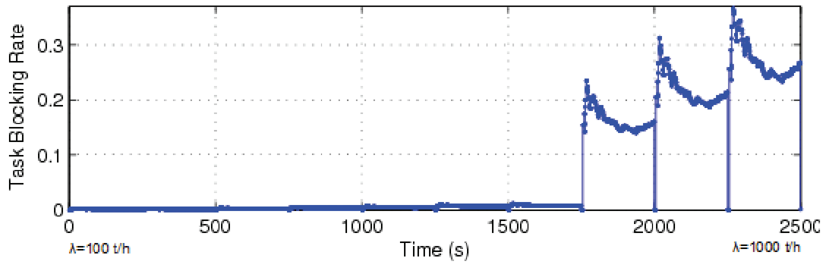
(b) Blocking at service time 40 min, task arrival rate variable from 500 to 1200 per hour).

Figure 4.7: Task blocking probability with admission control (Algorithm 3 without considering task blocking threshold of P_{blk0}).

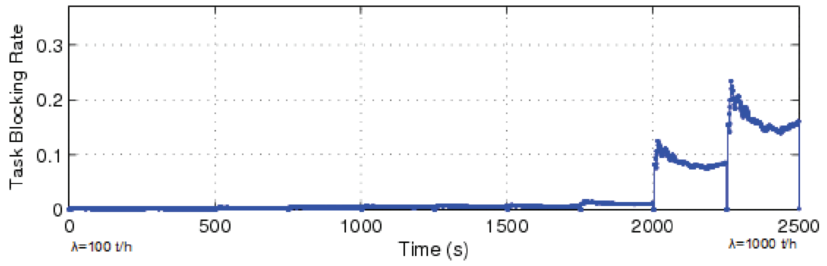
no admission control has been applied. In the timing diagrams in Figs. 4.8b and 4.8e, task blocking rate and total delay are increasing respectively with each increment of the task arrival rate. When Algorithm 3 is applied to model, task blocking rate and delay have been reduced as shown in Figs. 4.8c and 4.8f. According to Fig. 4.8d as the incoming task rate increases, the applied admission control mechanism will increase the task rejection rate to keep the total delay and task blocking below the threshold.



(a) Task arrival rate changes during the test time.

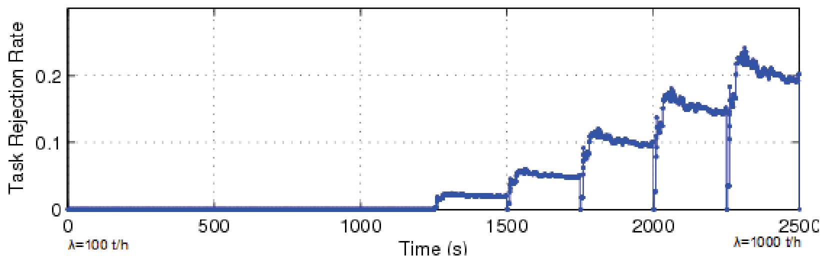


(b) Task blocking rate without admission control.

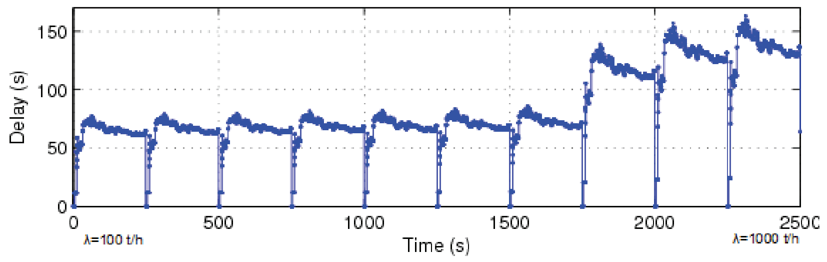


(c) Task blocking rate with admission control (Algorithm 3 without considering task blocking threshold of P_{blk0}).

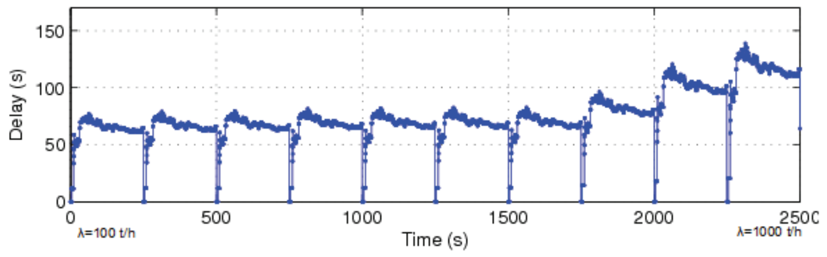
Figure 4.8: Test case of task arrival rate variable from 100 to 1000 per hour, service time 40 min.



(d) Task rejection rate with admission control (Algorithm 3 without considering task blocking threshold of P_{blk0}).



(e) Total delay without admission control.



(f) Total delay with admission control (Algorithm 3).

Figure 4.8: Test case of task arrival rate variable from 100 to 1000 per hour, service time 40 min.

4.2 Task Filtering as a Task Admission Control Policy in Cloud Server Pools

In a cloud server pool, resources are comprised of VMs which are deployed on PMs. In this work, every task is served with a single VM. A cloud server pool cannot serve unlimited number of tasks simultaneously. The decision whether to admit a service request or reject it is not trivial. However, with utilizing appropriate admission control and resource allocation mechanisms, it is possible to improve the usage of resources and provide a satisfactory level of utilization.

In this section we propose a dynamic controlling algorithm to devise task admission policies. First, we have implemented a lightweight task admission control algorithm which is based on the measurements of utilization and offered load. Second, we have developed the analytical model of the scheme as a queueing model and demonstrated its performance. Third, we have presented an alternative task admission algorithm for the cloud systems which experience abrupt changes in their task arrival rates. Finally, we have analyzed the short-term variability of the system through simulation model. Also, we have examined the effect of the size of tasks' waiting queue on the system performance in our simulations. We have also utilized a different calculation scheme for filtering coefficient in the lightweight task admission control algorithm which is more aggressive toward rejecting the incoming tasks. We have evaluated the effect of the different filtering coefficients on the system performance.

4.2.1 Admission Control Policy

Cloud providers need to have clear policies for task admission control and they should deploy robust task admission mechanisms to have a trade-off between SLA requirements and cloud system's resources. Admission control policies are defined according to different requirements in a cloud system and users' expectations.

In the current work, we have assumed that at the beginning of resource allocation process all of the users' incoming requests can get into the server pool and they will be provisioned in the resource allocation process. With utilizing an appropriate resource allocation mechanism, resources which in this case are VMs, will be assigned to the tasks. The system's performance is monitored continuously. During this process, admission control mechanism can reject some of the incoming tasks according to the system's requirements, e.g., when a utilization threshold is defined as a goal or when a definite proportion of VMs are assigned to the tasks; the target threshold is defined regarding to the system's requirements and it is not limited to the mentioned goals.

Controlling Parameters and Related Policies

In this work, we aim to keep the system in the stable operating region of the utilization threshold, U_{thr}. In order to achieve this goal, we have utilized two controlling parameters in the cloud system.

Fig. 4.9 illustrates the proposed task admission and resource allocation scheme in a server pool. Also, it depicts the overview of task admission controlling parameters. Using filtering coefficient results in selective task acceptance. The system's current resource availability is given as a feedback to the admission control mechanism. The resource allocation mechanism has been developed as three interactive stochastic sub-models including Resource Assigning Module, Pool Management Module and VM Provisioning Module; the overall solution is obtained by iteration over individual sub-model solutions. The details of resource allocation mechanism and related provisioning processes are presented in details in Section 3.1.

Filtering policy can be defined according to the preferences of the operator; for example, cloud management system can set a lower position as the admission threshold and

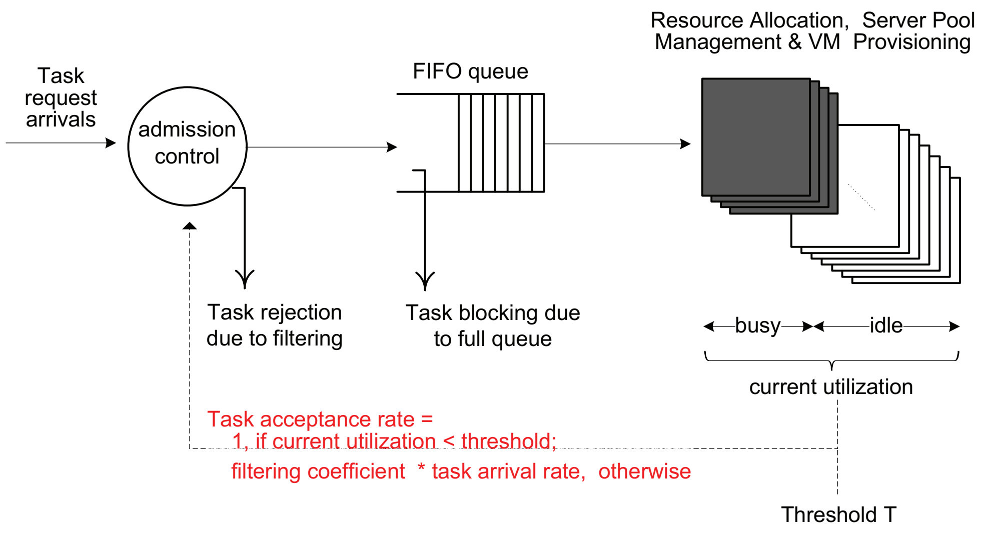

Task request arrivals
admission control
FIFO queue
Resource Allocation, Server Pool Management & VM Provisioning
Task rejection due to filtering
Task blocking due to full queue
busy
idle
current utilization
Task acceptance rate =
1, if current utilization < threshold;
filtering coefficient * task arrival rate, otherwise
Threshold T

Figure 4.9: The task admission control and resource allocation scheme in a cloud server pool.

chose a larger filtering coefficient (i.e., lower task dropping rate) or it can select a higher position threshold with a lower filtering coefficient (i.e., higher task dropping rate). In the former approach, system starts to drop the tasks earlier with a lower rate; whereas in the latter approach, system will start task dropping later, but with the higher rates. Cloud management system can select the appropriate policy according to the users' preference or system requirements.

Task Admission Scheme Based on Offered Load

Admission control mechanism used in this work is presented in Algorithm 5. In this algorithm, whenever a task arrives in the system or departs from it, some parameters will be re-calculated. First, average offered load (ρ_{av}) is calculated using the current task arrival rate. We have used Exponentially Weighted Moving Average (EWMA) technique [7] to smoothen the effect of sudden changes on the system. ρ_{av} is computed using EWMA smoothing factor (α), the previous average offered load (ρ_{avp}) and the system's current offered load (ρ_c) Estimated average utilization (u'_{av}) is calculated according to EWMA smoothing factor (β), the previous estimated average utilization (u'_{avp}) and the cloud system's current utilization (u_s). In case of task arrival, if u'_{av} is less than the utilization threshold, U_{thr}, the system accepts the incoming task; otherwise, the system calculates the filtering coefficient as $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$ and it will reject the incoming task with the probability of $1 - F_{cf}$. The offered load threshold, ρ_{thr} is set to 0.75. The utilization threshold, U_{thr} is set to 0.75.

Algorithm 5 Admission mechanism based on offered load.

```
Upon task departure:
  Re-calculate $\rho_{av}$, $u_s$ and $u'_{av}$;
Upon task arrival:
  Re-calculate $\rho_{av}$ as $\rho_{av} \longleftarrow \alpha \cdot \rho_c + (1-\alpha)\rho_{avp}$;
  Estimate new average utilization ($u'_{av}$) as
  $u'_{av} \longleftarrow \beta \cdot u_s + (1-\beta)u'_{avp}$;
  if $u'_{av} > U_{thr}$ then
    Calculate $F_{cf} \longleftarrow 1 - (\rho_{av} - \rho_{thr})$;
    Reject the incoming task with the probability of
    $1 - F_{cf}$;
  else
    Accept the incoming task;
  end if
```

Algorithm 5 is appropriate for systems which experience small steps of change in task arrival rate. It is easy to determine α and β in these systems as the systems' behavior is predictable. For the systems with unexpected arrival rate, the computation of parameters is more complicated and in Section 4.2.1, we have proposed Algorithm 6 to maintain task admission control in those systems.

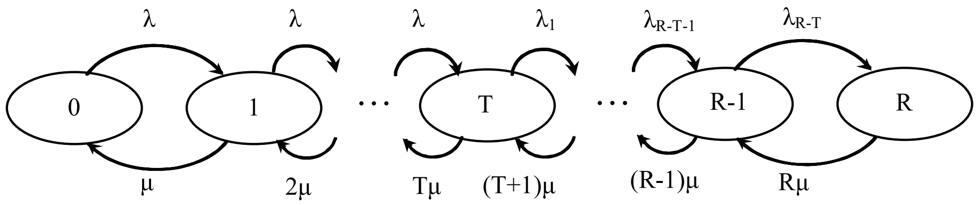


Figure 4.10: Markov chain model admission control.

Analytical Model of Admission Control

Admission control scheme can be modeled as a birth-death process which is a special case of CTMC. The task admission scheme can be analyzed using the Markov chain depicted in Fig. 4.10. In this setup, T and R represent the thresholds of full rate task acceptance and full rate task rejection, respectively. Namely, when the mean blocking probability of the system is below the first threshold, admission control accepts all incoming tasks. As the result, T is the last state where all of the arrived tasks will be admitted. Beyond this threshold, admission control begins to drop some of the incoming tasks. R is the full rejection threshold state beyond which all incoming tasks will be rejected and it is also known as the capacity of the system or the number of VMs in the model.

Transition rates of going from state i to state $i+1$ (i.e., that the task is accepted by the admission control) are calculated in Equation 4.2 and the accepted rate of incoming tasks, λ_f, can be set as

$$\lambda_f = \begin{cases} \lambda & \text{state } i = 0..T \\ F_{cf} \cdot \lambda & \text{state } i = T+1..R \\ 0 & \text{state } i > R \end{cases} \tag{4.1}$$

where λ is the task arrival rate and F_{cf} is filtering coefficient calculated as $F_{cf} = 1 - (\rho_i - U_{thr})$. Offered load in the state i presented as ρ_i and utilization threshold, U_{thr}, determine the value of filtering coefficient. The state probabilities of the Markov model can be calculated as

$$P_k = \begin{cases} \dfrac{1}{k!}\left(\dfrac{\lambda}{\mu}\right)^k \dfrac{1}{DD} & 0 \leq k \leq T \\ \dfrac{\lambda^T \prod\limits_{i=1}^{k-T} \lambda_i}{k!\ \mu^k} \dfrac{1}{DD} & T < k \leq R \end{cases} \tag{4.2}$$

where

$$DD = \sum_{0}^{T} \frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i + \left(\frac{\lambda}{\mu}\right)^T \sum_{T+1}^{R} \frac{\prod\limits_{i=1}^{i-T} \lambda_i}{i!\ \mu^{i-T}} \tag{4.3}$$

and μ is the mean service rate. The average utilization of the system, u_{av}, and the blocking probability of T threshold, P_{blk}, is given by

$$u_{av} = \frac{1}{R}\sum_{k=0}^{R} k P_k \tag{4.4}$$

$$P_{blk} = \sum_{k=T+1}^{R} P_k \tag{4.5}$$

We have obtained the value of threshold T using two different methods. In the first method, we have solved the system of equations (4.1) to (4.4) according to given utilization threshold, U_{thr}. In the second method, we have applied exhaustive search to find optimal value of T threshold which renders minimal difference between estimated utilization and utilization threshold, U_{thr}.

Task Admission Scheme Based on Current Utilization

Algorithm 6 Admission mechanism based on current utilization.

```
  $flg_o \longleftarrow 0$;
  Upon task arrival:
    Consider the current utilization including the
    pending task ($u'_c$);
    **if** $u'_c > U_{thr}$ **then**
      **if** $\left|u'_c - u'_{prev}\right| > M_{thr}$ **or** $flg_o = 0$ **then**
        **for** $T_i = U_{thr} \cdot R$ to $R$ **do**
          Adopt the instantaneous utilization ($u_c$) equation from the system of equa-
          tions (4.1) to (4.4);
          Apply $T_i$ threshold to $u_c$ to obtain $uc_i$;
          Calculate $Fc_i$ according to $uc_i$;
          Calculate $\delta_i \longleftarrow |uc_i - U_{thr}|$;
        **end for**
        Find the minimum $\delta_i$;
        Select the corresponding $T_i$ and $Fc_i$ as the
        optimal values of $T$ and $F_{cf}$ respectively;
        $flg_o \longleftarrow 1$;
      **else**
        Use the previous $T$ as the current $T$;
        Apply $T$ threshold to $u_c$ and calculate $F_{cf}$
        according to $u_c$;
      **end if**
      Reject the incoming task with the probability of
      $1 - F_{cf}$;
    **else**
      Accept the incoming task;
    **end if**
```

Task admission mechanism presented in Algorithm 5 is efficient for the cloud systems which go through smooth changes of incoming task rate. However, in the systems where change of arrival rate is highly dynamic, smoothing factors α and β, and Algorithm 5 does not predict the filtering coefficient with sufficient accuracy. Therefore, we propose an alternative algorithm which is better for handling the systems with abrupt changes of arrival rate. In this solution, system applies exhaustive search to find admission arrival rate which is derived by the closest value of instantaneous utilization to the utilization threshold, U_{thr}. In this iterative mechanism, system tracks U_{thr} as closely as possible. This solution is inspired by the Delta Modulation technique used in Pulse Code Modulation (PCM) [60]. Algorithm 6 demonstrates this approach. When a task arrives, offered load (ρ) is calculated using the current task arrival rate. Also, the current utilization including the pending task, u'_c, is calculated. if u'_c is less than the utilization threshold, U_{thr}, the system accepts the incoming task. Otherwise, if the difference between u'_c and the previous utilization before considering T threshold, u'_{prev}, is more than a threshold margin, M_{thr}, or if it is the first time occurrence of utilization more than U_{thr}, the system examines the threshold range of $U_{thr} \cdot R$ to R. In this case, the typical T_i (the threshold position of full rate task acceptance) will be replaced in current utilization (u_c) equation, adopted from the system of equations (4.1) to (4.4). It then calculates the filtering coefficient (Fc_i) corresponding to T_i. Then, uc_i will be computed using the values of (Fc_i) and T_i. Also, the difference between uc_i and U_{thr}, δ_i, is calculated. The minimum value of δ_i represents the closest value of utilization to U_{thr} and the corresponding T_i and Fc_i will be chosen as target T and F_{cf} values respectively. also, if the difference between u'_c and u'_{prev} is less than M_{thr} margin, system will use the previous T as the current T and after applying this T to u_c, F_{cf} can be calculated. The system will reject the incoming task with the probability of $1 - F_{cf}$. The utilization threshold, U_{thr} is set to 0.75.

4.2.2 Performance Evaluation

To investigate the behavior of the admission control model, we have evaluated different performance parameters under varying levels of offered load, ρ, calculated as $\rho = \frac{\lambda_f}{m \cdot N \cdot \mu_{tot}}$.

In our experiments, $N = 100$ is the number of PMs and $m = 10$ is the maximum number of VMs on each PM; therefore, the system's capacity is 1000 VMs. We have kept the task service time constant and varied the task arrival rate to achieve the offered load in the range of 0.4 to 0.95. In order to investigate the effect of mean service time varieties on the performance metrics separate from the effects of offered load and task arrival rate changes, we have examined four service times of 20, 40, 60 and 80 minutes for each parameter. The utilization threshold, U_{thr} is set to 75%; in this range of value, U_{thr} is relatively high and the system is in normal condition, not yet in the saturation operational region in which significant number of tasks will get blocked due to the excessive traffic load. We have used Maple 15 to develop this model [26].

Fig. 4.11 shows the accepted rate of incoming tasks. It can be seen that while offered load is under 0.75, with increasing of arrival rate, accepted rate increases as well. When offered load goes beyond 0.75, accepted rate of incoming tasks becomes flat to keep the average utilization in the defined threshold. Also, in order to have the same ranges of offered load in Fig. 4.11, with increasing of constant service time, the accepted rate of incoming tasks moves to the lower ranges (from maximum value of 0.64 tasks per second or 2300 tasks per hour to 0.16 tasks per second or 570 tasks per hour).

Performance Evaluation of Algorithm 5

In this section, we investigate the performance of task admission scheme that uses Algorithm 5.

Fig. 4.12 presents the threshold positions according to changing offered load for four constant service times. As can be seen in Fig. 4.12, same offered load can result in same

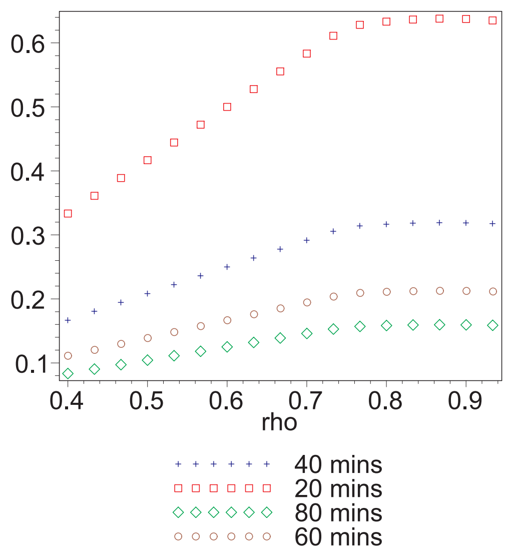


Figure 4.11: Accepted rate of incoming tasks at constant service time and variable task arrival rate.

threshold position. Also, with increasing of offered load or in other word, increasing of incoming task, threshold position moves to higher number of VMs in order to keep the average utilization around 75%. Boxes represent the obtained thresholds according to Equation 4.4 and crosses are the obtained thresholds of the exhaustive search method discussed in Section 4.2.1. As can be seen, the outcomes of two methods match quite well.

Fig. 4.13 illustrates the blocking probability of admission control process. In Fig. 4.13, when offered load is below 0.75, the blocking probability of admission control process is negligible, but when offered load goes beyond 0.75, the blocking probability increases when the offered load increases. This is because the system is getting full and it is blocking more tasks to maintain the desired level of utilization. Also, it can be seen that in different service times, the same offered load results in almost same blocking probability, despite the slight discrepancies which arise from rounding errors in the computations.

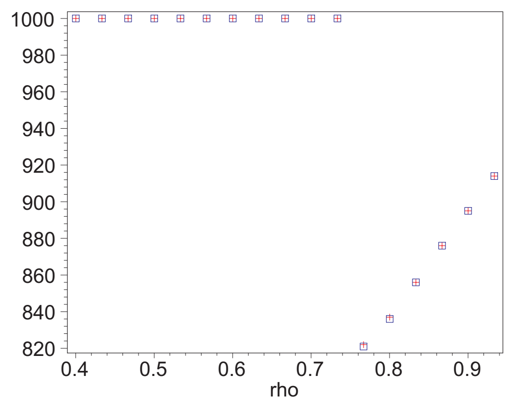


Figure 4.12: Threshold position at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 5.

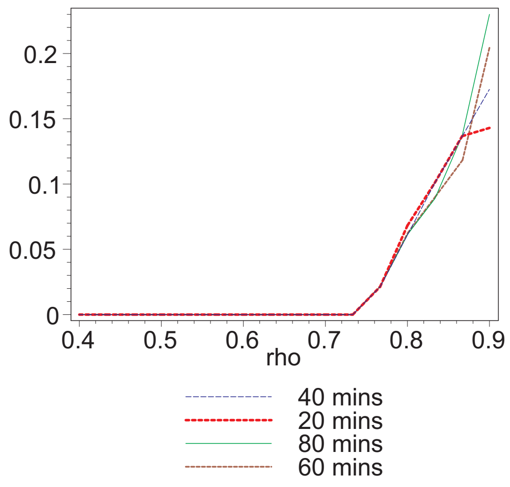


Figure 4.13: Blocking probability at constant service time and variable task arrival rate using Algorithm 5.

Fig. 4.14 shows the utilization of the admission control scheme. This figure presents the outcome of applying the obtained threshold to the system. As expected, in all cases with choosing appropriate threshold position, utilization is not degrading and with offered loads higher than 0.75, utilization gets into the stable condition of utilization threshold, U_{thr}. Therefore, our goal of keeping the utilization around a specific threshold is achieved.

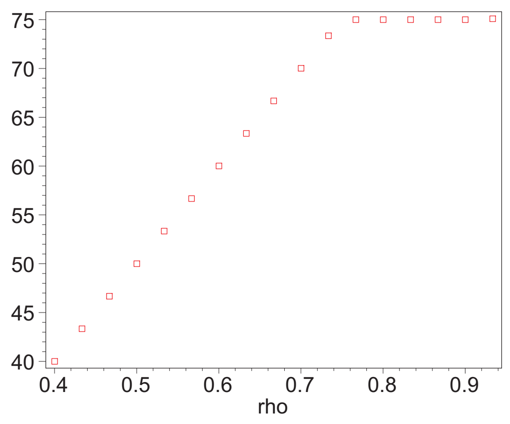


Figure 4.14: Utilization at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 5.

In Fig. 4.15, the value of filtering coefficient in four cases has been presented. As expected, in the same ranges of offered load, these coefficients are identical; this is because in Algorithm 5, filtering coefficient is calculated as a function of offered load and therefore, same offered load gives same filtering coefficient. Also, in all cases, when offered load is less than 0.75, the system does not filter the incoming tasks; but when offered load is getting higher than 0.75, system drops some of the arriving tasks. With increasing of offered load, dropping rate of arriving tasks increases linearly to compensate the target utilization threshold. In the worst case of $\rho = 95\%$, system only accepts 82% of the incoming tasks.

Fig. 4.16 illustrates the total delay in four cases. The total delay includes the total resource provisioning time in a server pool and it is obtained by forwarding the accepted incoming tasks to the pool management scheme presented in Fig. 4.9. The details of provisioning time calculation has been described in Section 3.1. As can be seen in Fig. 4.16, higher service time generally results in higher delay, but the admission control scheme prevents the system to experience dramatic changes in total delay and it keeps the total delay

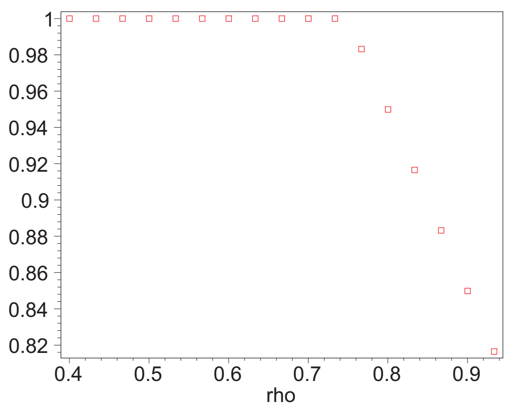


Figure 4.15: Filtering coefficient at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 5.

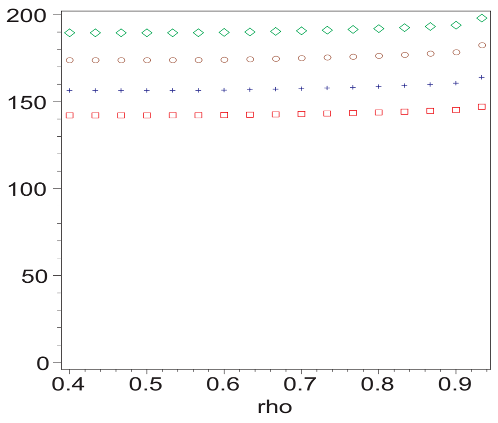


Figure 4.16: Total delay at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 5. Legends are same as the legends in Fig. 4.11.

in an almost steady range.

Performance Evaluation of Algorithm 6

The performance of task admission scheme that uses Algorithm 6 is presented in this section. It can be seen that the performance of Algorithm 6 is very close to Algorithm 5.

In Fig. 4.17, when offered load goes beyond 0.75, threshold position approximately

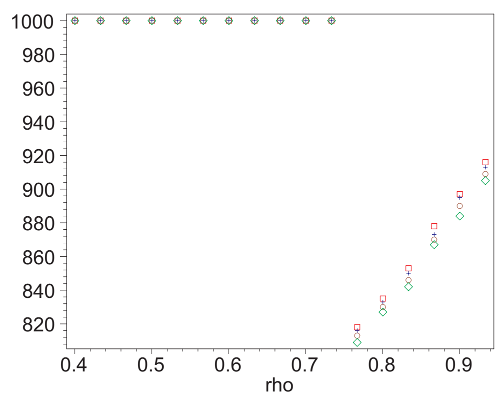


Figure 4.17: Threshold position at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 6. Legends are same as the legends in Fig. 4.11.

changes in the range of 805 to 915 VMs to keep the average utilization around 75%, similar to Fig. 4.12; although, in this case, threshold position is found through exhaustive search. As can be seen, similar to Fig. 4.12, same offered load gives presents almost same threshold position and the slight differences are the result of rounding errors in the calculations.

The blocking probability of the alternative solution is shown in Fig. 4.18. The blocking probability of this solution is slightly higher than the blocking probability presented in Fig. 4.13; however, the range and pattern of changes of blocking probabilities are close. In the case of offered load larger than 75%, the system is getting full and it is blocking more tasks to maintain the desired level of utilization. Also, it can be seen that in different service times, the same offered load results in almost same blocking probability, despite the slight discrepancies which come from rounding errors in the calculations.

The system's utilization in Fig 4.19 matches the utilization presented in Fig 4.14; although, in Fig 4.19, utilization is slightly changing over the different offered loads.

The pattern and the range of changes of filtering coefficient in Fig. 4.20 are similar to the pattern and ranges of changes illustrated in Fig. 4.15; although, in Fig. 4.20, the

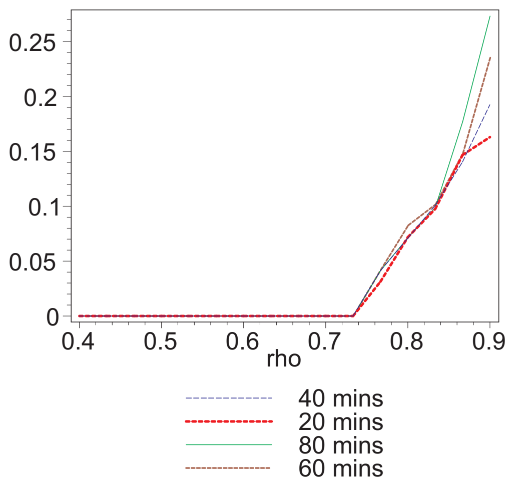


Figure 4.18: Blocking probability at constant service time and variable task arrival rate using Algorithm 6.

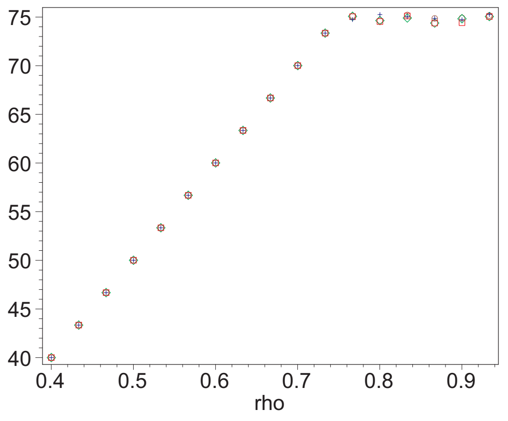


Figure 4.19: Utilization at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 6. Legends are same as the legends in Fig. 4.11.

filtering coefficient is decreasing slightly faster. Also, the slight discrepancies of filtering coefficients in Fig. 4.20 are the result of rounding errors in the computations and the same offered load presents almost same filtering coefficient.

Also, the system's delay shown in Fig. 4.21 matches the total delay in Fig. 4.16.

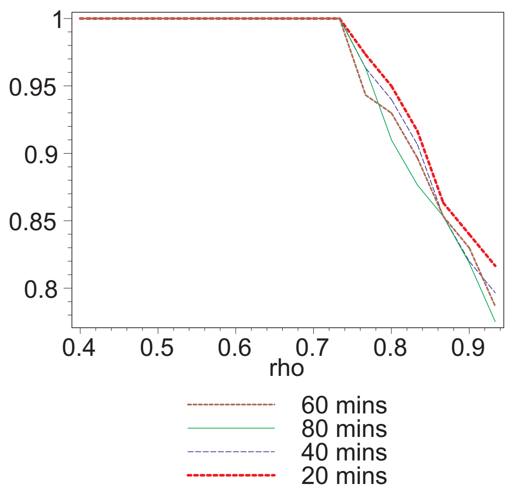


Figure 4.20: Filtering coefficient at constant service time and variable task arrival rate using Algorithm 6.

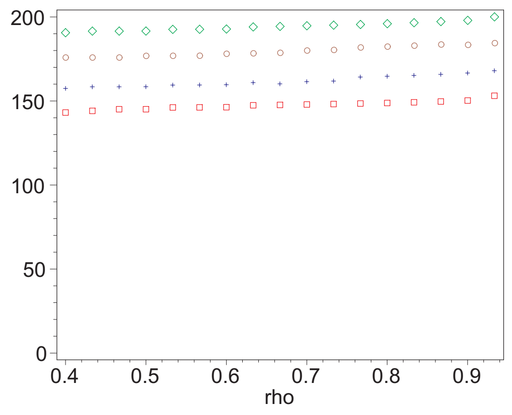


Figure 4.21: Total delay at constant service times of 20, 40, 60 and 80 mins and variable task arrival rate using Algorithm 6. Legends are same as the legends in Fig. 4.11.

Analyzing the Effect of The Size of Waiting Queue and Filtering Coefficient on System Performance

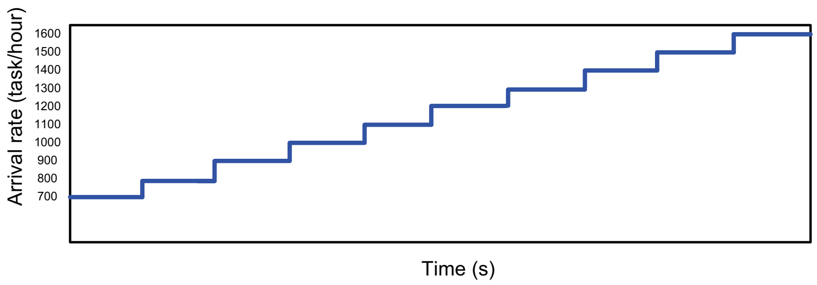
In this subsection, we analyze the effect of changing the size of FIFO waiting queue and filtering coefficient in Algorithm 5 on the system. In the results presented in Section 4.2.2

and 4.2.2, we have assumed that the size of waiting queue, Lq, is equal to 50. Also, filtering coefficient, F_{cf}, in Algorithm 5 is calculated as $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$. We have built a discrete event simulator using MATLAB R2013a with the Simulink component [27] to analyze the effect of changing parameters.

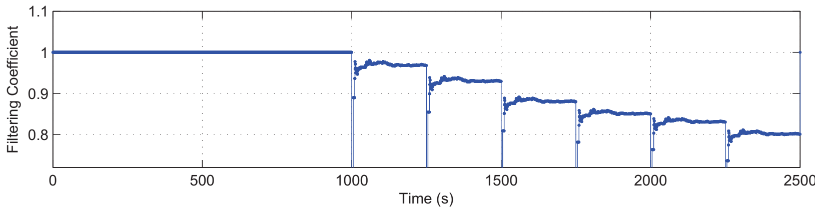
Simulation Experiments Using Short Waiting Queue Figs. 4.22 and 4.23 illustrate the simulation snapshots when Lq is equal to 50 and in the case of applying Algorithm 5, F_{cf} is calculated as $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.

As can be seen in Figs. 4.22a and 4.22b, when task arrival rate is less than 1100 tasks per hour or in other words, offered load is less than 0.75, system admits all arriving tasks; whereas, when offered load goes beyond 0.75, system starts rejecting some tasks and filtering coefficient decreases. According to task blocking rates presented in Figs. 4.22c and 4.22d, using Algorithm 5 results in almost 50% decreasing of the task blocking probability.

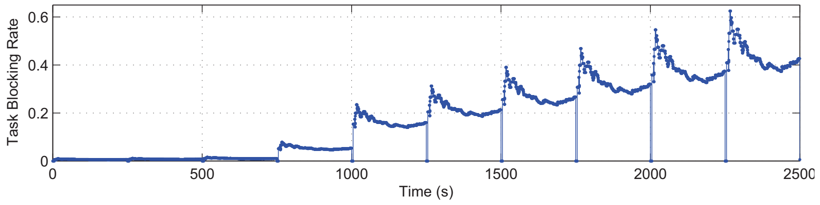
Also, comparison of Figs. 4.22e and 4.22f shows that using Algorithm 5 improves the total delay.



(a) Task arrival rate changes during the test time.

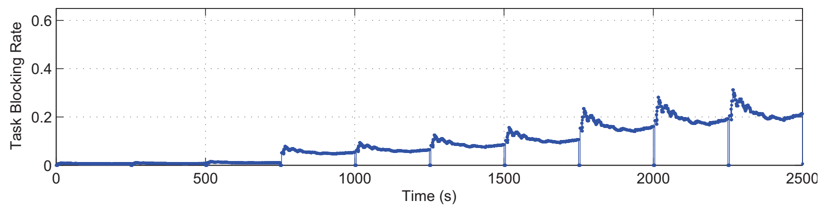


(b) Filtering coefficient with admission control (Algorithm 5).

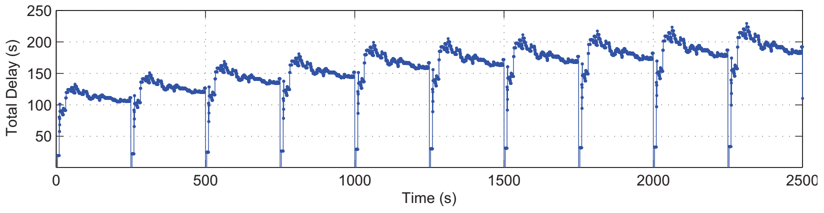


(c) Task blocking probability without admission control.

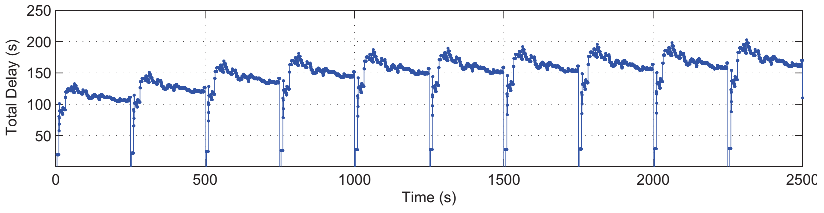
Figure 4.22: Test case of $Lq = 50$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.



(d) Task blocking probability with admission control (Algorithm 5).



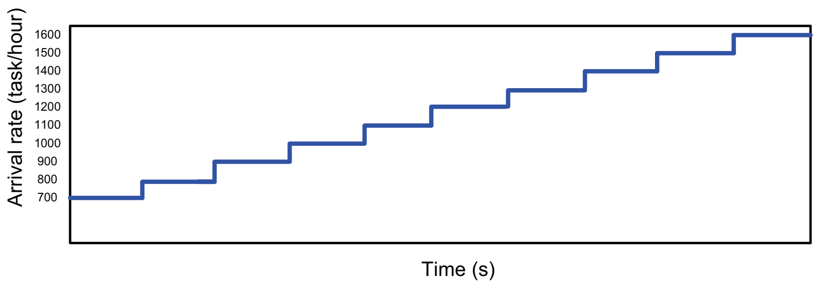
(e) Total delay without admission control.



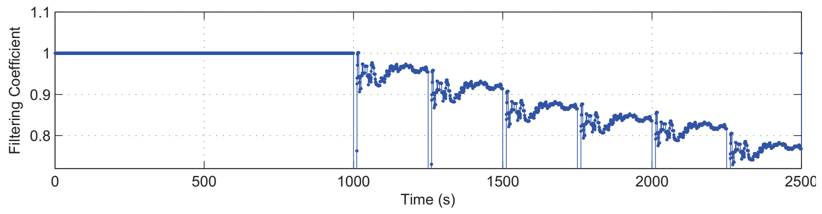
(f) Total delay with admission control (Algorithm 5).

Figure 4.22: Test case of $Lq = 50$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.

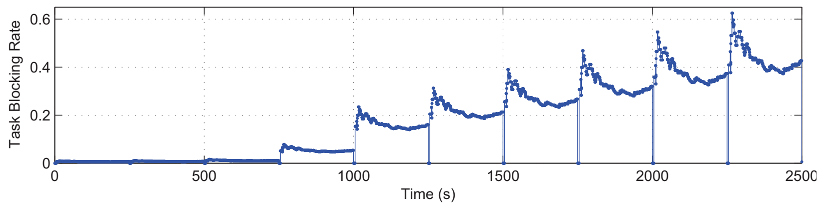
Fig. 4.23 illustrates the snapshots of performance parameters when Algorithm 6 is used and no task admission control is applied. In Fig. 4.23b, similar to Fig. 4.22b, when offered load goes beyond 0.75, filtering coefficient decreases; although, in the case of using Algorithm 6, the dropping rate is slightly higher. Also, according to the task blocking rate presented in Fig. 4.23d, using Algorithm 6 improves the task blocking rate; though, the blocking rate improvement is not as high as the case of Algorithm 5. This is due to the heavy load of computation in Algorithm 6.



(a) Task arrival rate changes during the test time.

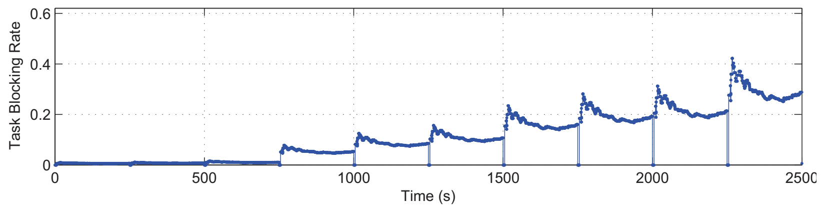


(b) Filtering coefficient with admission control (Algorithm 6).

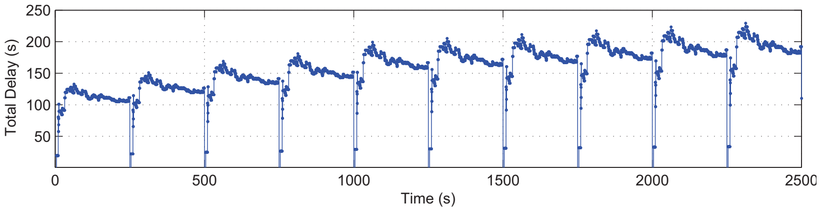


(c) Task blocking probability without admission control.

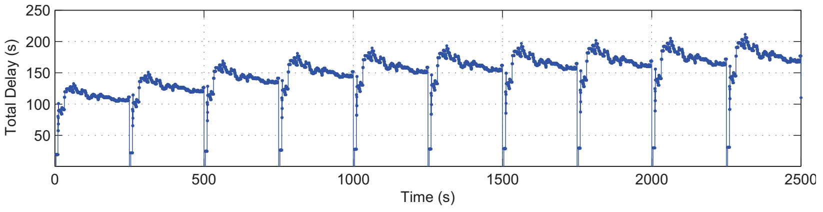
Figure 4.23: Test case of $Lq = 50$, task arrival rate variable from 700 to 1600 per hour, service time 40 min.



(d) Task blocking probability with admission control (Algorithm 6).



(e) Total delay without admission control.

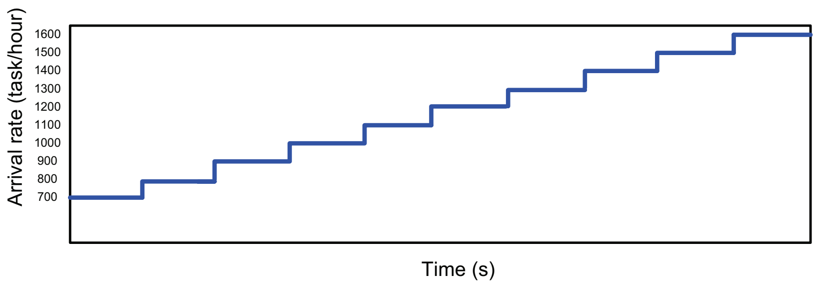


(f) Total delay with admission control (Algorithm 6).

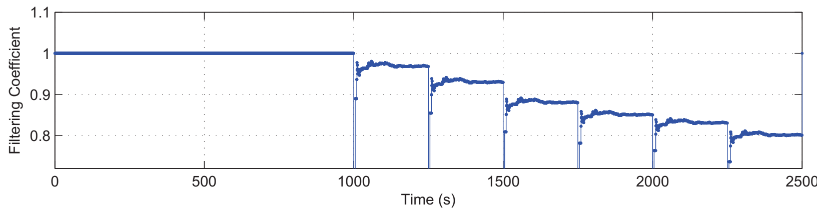
Figure 4.23: Test case of $Lq = 50$, task arrival rate variable from 700 to 1600 per hour, service time 40 min.

Simulation Experiments Using Large Waiting Queue Figs. 4.24 and 4.25 present the simulation snapshots when Lq is equal to 200 and in the case of applying Algorithm 5, F_{cf} is calculated as $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.

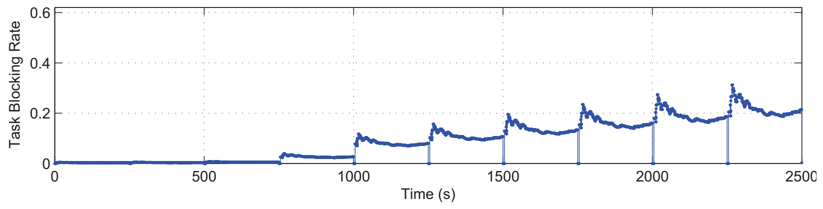
If we compare the Figs. 4.24c and 4.22c, we can see that even without using any admission control mechanism, increasing the size of waiting queue has a significant positive effect on task blocking rate; this is because more tasks have a chance to stay in the larger waiting queue and they do not get rejected. As shown in Fig. 4.24d, using Algorithm 5 also improves the task blocking rate. However, Figs. 4.24e and 4.24f indicate that using a larger waiting queue causes higher delay in the system; although, using Algorithm 5 still results



(a) Task arrival rate changes during the test time.



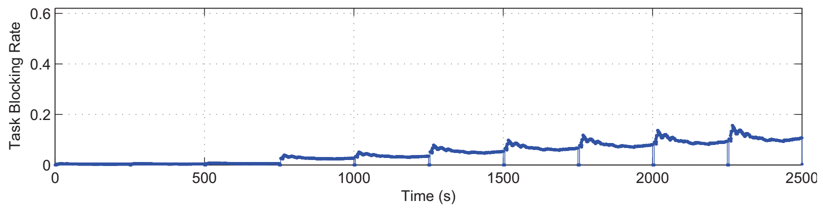
(b) Filtering coefficient with admission control (Algorithm 5).



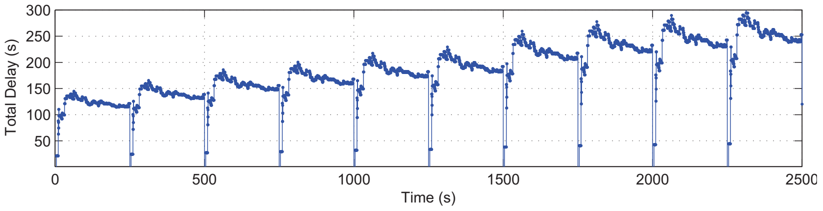
(c) Task blocking probability without admission control.

Figure 4.24: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.

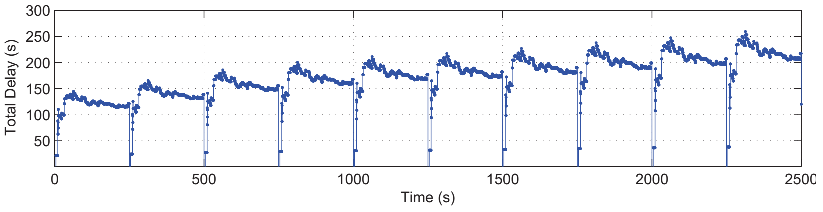
in lower delay.



(d) Task blocking probability with admission control (Algorithm 5).



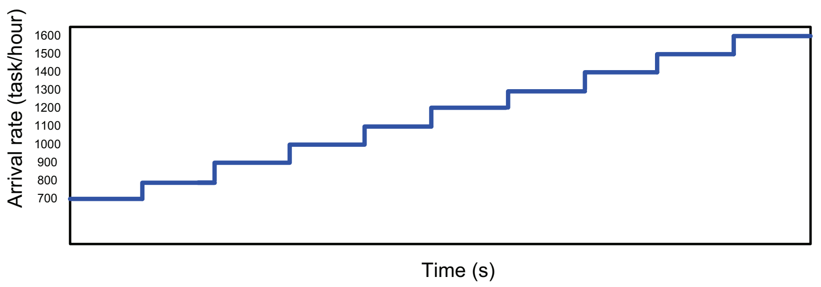
(e) Total delay without admission control.



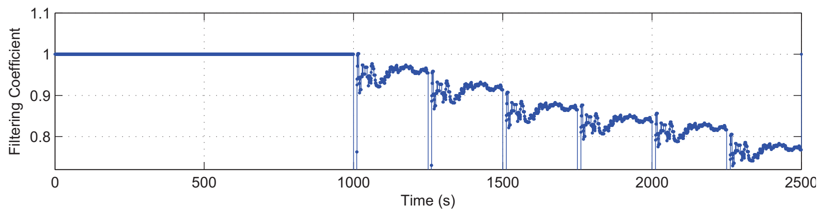
(f) Total delay with admission control (Algorithm 5).

Figure 4.24: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - (\rho_{av} - \rho_{thr})$.

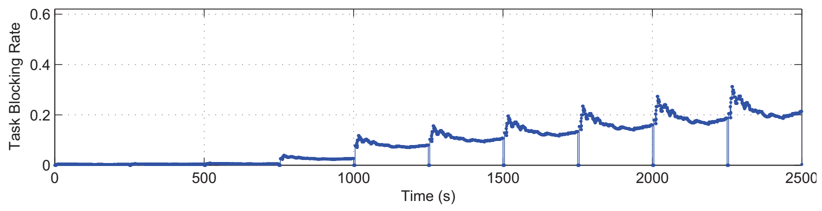
Similar to the case of Algorithm 5, using Algorithm 6 and increasing the size of waiting queue has a positive effect on task blocking rate (Fig. 4.25d) and Algorithm 6 improves the total delay (Fig. 4.25f compared to Fig. 4.25e); though, in case of using Algorithm 6, delay is slightly higher than case of Algorithm 5. Also, the larger waiting queue ($Lq = 200$) results in the higher delay than short waiting queue ($Lq = 50$). This is because of experiencing longer waiting times in larger queues.



(a) Task arrival rate changes during the test time.

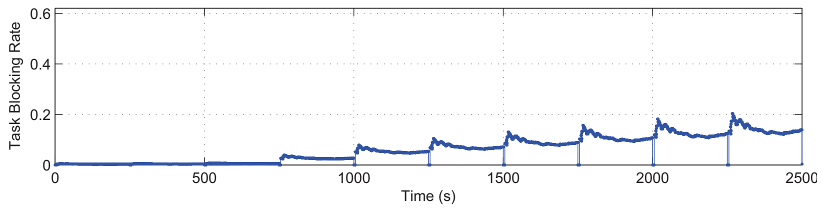


(b) Filtering coefficient with admission control (Algorithm 6).

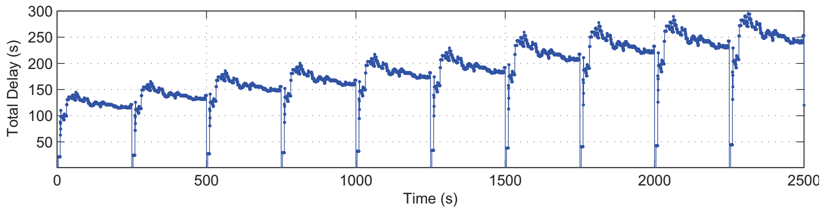


(c) Task blocking probability without admission control.

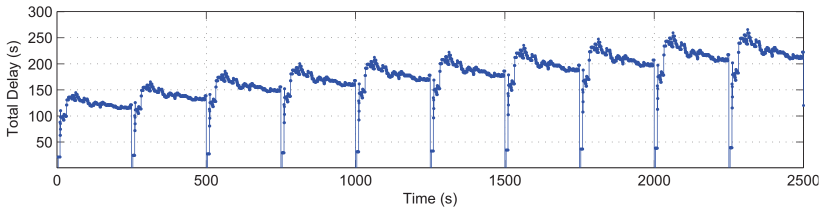
Figure 4.25: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min.



(d) Task blocking probability with admission control (Algorithm 6).



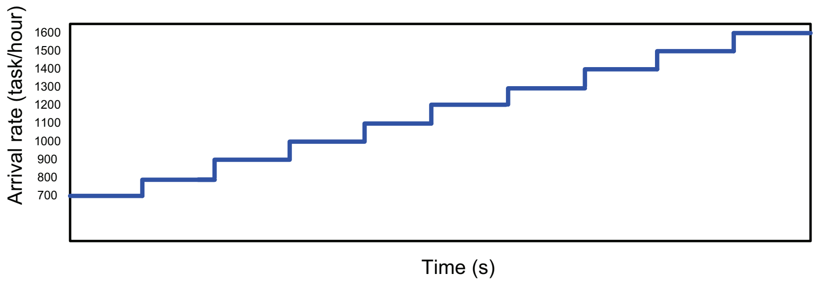
(e) Total delay without admission control.



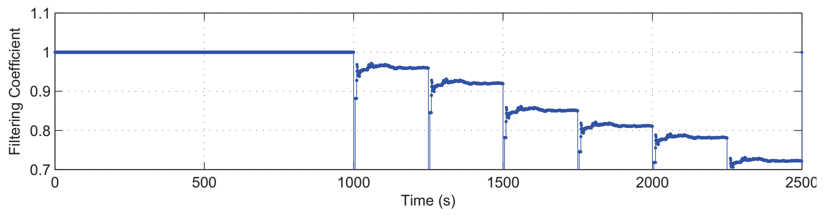
(f) Total delay with admission control (Algorithm 6).

Figure 4.25: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min.

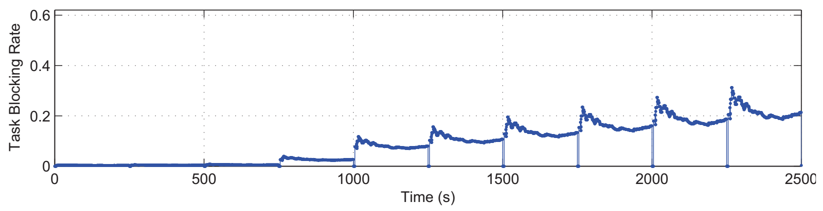
In order to achieve higher system performance in Algorithm 5, we have used a more aggressive calculation scheme for filtering coefficient. In this case, F_{cf} is calculated as $F_{cf} = 1 - ((\rho_{av} - \rho_{thr})/\rho_{thr})$. the results presented in Figs. 4.26d and 4.26f demonstrate that using this aggressive calculation scheme for filtering coefficient decreases the task blocking rate and total delay compared to its counterpart case presented in Figs. 4.24d and 4.24f. Hence, this new calculation scheme for F_{cf} results in further improvement of system performance.



(a) Task arrival rate changes during the test time.

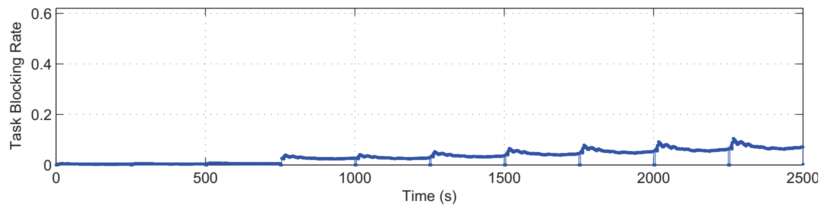


(b) Filtering coefficient with admission control (Algorithm 5).

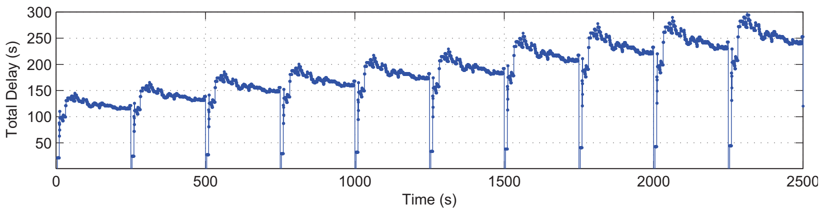


(c) Task blocking probability without admission control.

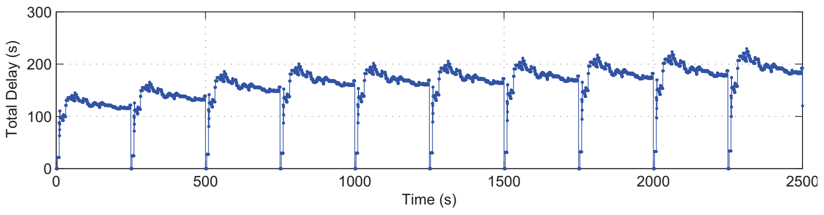
Figure 4.26: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - ((\rho_{av} - \rho_{thr})/\rho_{thr})$.



(d) Task blocking probability with admission control (Algorithm 5).



(e) Total delay without admission control.



(f) Total delay with admission control (Algorithm 5).

Figure 4.26: Test case of $Lq = 200$, task arrival rate variable from 700 to 1600 per hour, service time 40 min. $F_{cf} = 1 - ((\rho_{av} - \rho_{thr})/\rho_{thr})$.

4.3 Chapter Summary

In Section 4.1, we have examined the behavior of a cloud center in different operating through our simulation model. Also, two algorithms for task arrival admission control are presented to keep the cloud system in the non-saturation operating area. These admission control algorithms execute in two steps: first step is adjusting the partitioning coefficient according to the mean task arrival rate; the second tier is tracking the task blocking probability in the system and adjusting the task acceptance rate according to the predefined task blocking probability thresholds. Our simulation results confirm that the proposed admission control algorithms enhance system performance. However, adjustments provided by Algorithm 3 appeared to be more successful in keeping the overall blocking probability low than the adjustments provided by Algorithm 4. Although, the cost of this success is dropping some of the incoming tasks in the linear to transition operating regions in Algorithm 3.

The two proposed task admission algorithms presented in Section 4.2 are aimed to keep the system in the stable operating region defined by the operator. We have utilized two controlling parameters, full rate task acceptance threshold and filtering coefficient, to deploy task admission policies. The first task admission algorithm is lightweight, more conservative in making decisions and suitable for the cloud systems which are relatively stable in their task arrival rate; whereas, the second algorithm is appropriate for the systems with the wide ranges of change in arrival rate and it is more complicated.

The evaluation of our model shows that in different ranges of incoming task rate and mean service time, the average offered load will result in the similar average performance in the cloud system. Also, our numerical results confirm that both of the task admission mechanisms provide the cloud system's stability.

We have also analyzed the effect of the size of tasks' waiting queue using experimental simulation. We have observed that increasing the size of waiting queue reduces the task blocking probability; although, it causes higher delay due to experiencing longer waiting times in larger queues. Also, we have observed that using more aggressive scheme of filtering coefficient in the lightweight algorithm improves the task blocking rate and delay.

A continuation to the work presented in this chapter can be optimal placement of filtered tasks in a networked cloud system.

Chapter 5

Prioritization of Overflow Tasks to Improve Performance of Mobile Cloud

When the required computational resources exceed the capabilities of a mobile device, the application may be offloaded to a cloud and executed in a virtual machine running on a host. In many cases, this application forks new tasks which require virtual machines of their own that need to be provisioned on the same physical machine as was the original application. Achieving satisfactory performance level in such a scenario requires flexible resource allocation mechanisms in the cloud datacenter. In this chapter we present two such mechanisms which use prioritization: one in which forked tasks are given full priority over newly arrived ones, and another in which a threshold is established to control the priority. We analyze the performance of both mechanisms using a Markovian multiserver queueing system with two priority levels to model the resource allocation process, and a multi-dimensional Markov system based on a Birth-Death queueing system with finite population, to model virtual machine provisioning. We have examined the performance of the proposed system in Section 5.3 and found that the threshold-based priority scheme not only performs better, but can also be tuned to achieve the desired performance level.

This chapter is organized as follows: in Section 5.1, we introduce our resource allocation mechanism in mobile cloud computing. Section 5.2 describes the proposed resource allocation solution for mobile cloud computing. Section 5.3 discusses the performance of our system and the related outcomes. Section 5.4 summarizes this chapter.

5.1 Introduction

Mobile applications such as face recognition, natural language processing, interactive gaming, and augmented reality are demanding intensive computation and high energy consumption. However, mobile devices have limited computation resources and limited battery life. The tension between resource-hungry applications and resource-constrained mobile devices hence poses a significant challenge for future mobile platform development. Mobile cloud computing, where mobile devices can offload some computational jobs to the cloud is envisioned as a promising approach to address such a challenge [11].

The characteristics of mobile devices and wireless network makes the implementation of mobile cloud computing more complicated than stationary clouds. Offloading requests from a mobile device usually require quick response, may be infrequent, and are subject to variable network connectivity, whereas cloud resources incur relatively long setup times, are leased for long time quanta, and are not too sensitive to network connectivity [57].

Also, the volume of workload which is going to be offloaded is not predefined or known for a mobile device when it starts the offload process. There is a possibility that the mobile device offloads a large burst of tasks toward the clouds or it may scarcely offload any application to the cloud; from this definition we can conclude that the mobile device has a stochastic behavior during the offload process.

In this work, we address the elasticity in mobile cloud computing and the heterogeneity issues between cloud and mobile devices. We have developed a solution which allocates resources for on-demand job requests in the mobile clouds. Offloaded jobs are placed in PMs in mobile cloud. However, these jobs can fork new tasks which may not find sufficient resources in home PM. In that case, they need to be returned to resource allocation module in order to be assigned the resources they need; we call these tasks "overflow tasks". The proposed solution manages these two types of tasks as two classes of services with different levels of priority. We have utilized Markovian multiserver queueing system with two priority levels. In our solution, each type of task has its corresponding distinctive queue. Also, this solution prioritizes serving the overflow tasks over the new incoming tasks as we assume the overflow tasks generally have a shorter deadline than the new arrivals. Also, we have followed two approaches of priority differentiation. In the first approach, we assume that the overflow tasks have full priority and the new arrivals will not get service unless there is no overflow task left in the queue. In the second approach, we consider a threshold for the number of waiting overflow tasks. Below this threshold, each type of task has access to the resources based on its corresponding probability. If the number of waiting overflow tasks gets larger than the predefined threshold, the full priority is given to the overflow tasks and new arriving tasks should wait until the number of waiting overflow tasks becomes less than the threshold.

Dynamic side of offloaded jobs and existence of overflow tasks put a burden on resource management system. Major issue is how to assign priority to the tasks in order to satisfy bounds on job completion times. In this work, we consider soft bounds on completion times and relative priority between the tasks. Note that in this work, “job” is referred to the mobile device’s application offloaded toward cloud and it may include one or more “tasks”.

In this work, based on the characteristic of service demands of mobile devices, we assume that the size of a job can vary during its service time. A job initially has a single task, however, it may generate new tasks during its service time. The service times of the tasks are independent and identically distributed and each task requires a VM for its execution. Hence, the number of tasks in a job during its service time will be a random variable. A job is completed when all the tasks belonging to that job complete their service. In order to develop our VM provisioning scheme, we have proposed a model regarding the dynamic behavior of the mobile requests which is based on integration of multi-dimensional Markov system and Birth-Death queueing systems with finite population ($M/M/L//L$) inspired by the Birth-Death queueing systems developed in [35]. However, there is a possibility that mobile jobs offload a large burst of tasks toward cloud servers. The task provisioning would be impossible without limiting the size of offloaded job, i.e., the maximum number of tasks that it can contain. Also, since the cloud has finite number of PMs and every PM has a finite operational capacity (e.g., the number of CPUs and memory capacity) and assuming that single task gets a VM (VMs are homogenous with respect to the resources), there is a limit on maximum number of jobs that a PM can host. With these limitations, we have prevented the mobile devices to take the resources extensively and deteriorate the performance of the cloud and the quality of service presented to cloud users including the mobile users.

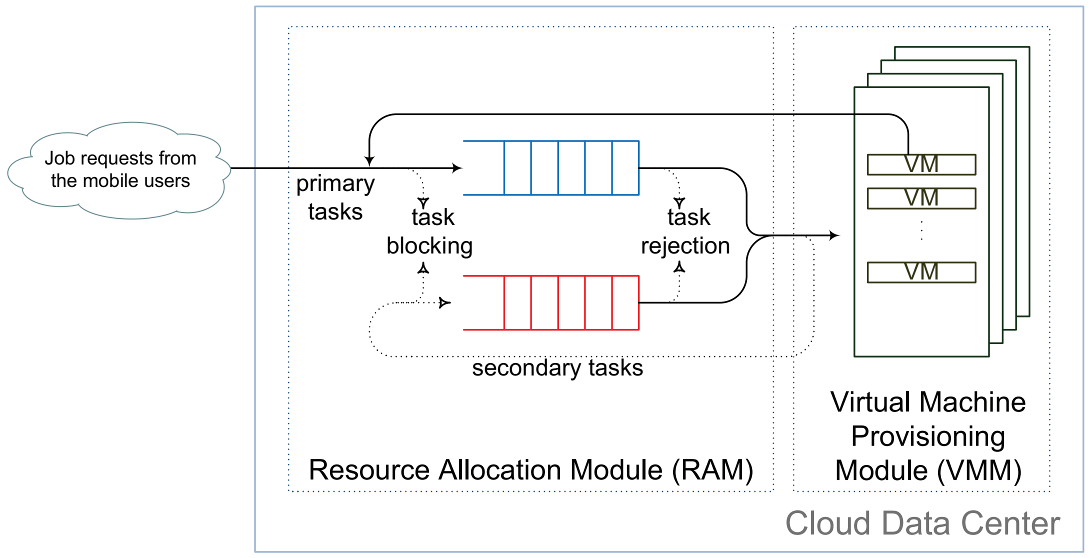

Job requests from the mobile users
primary tasks
task blocking
task rejection
secondary tasks
VM
VM
VM
Resource Allocation Module (RAM)
Virtual Machine Provisioning Module (VMM)
Cloud Data Center

Figure 5.1: The overview of system model.

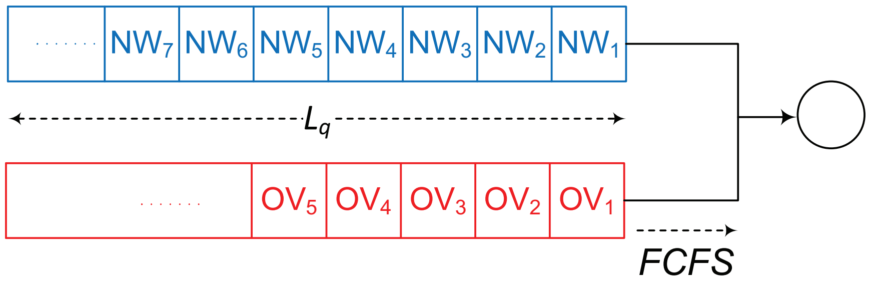
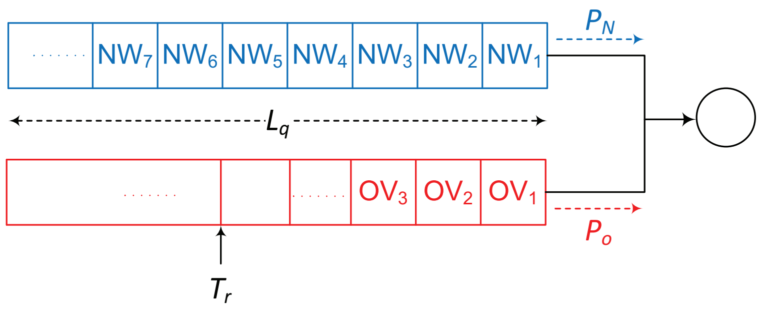


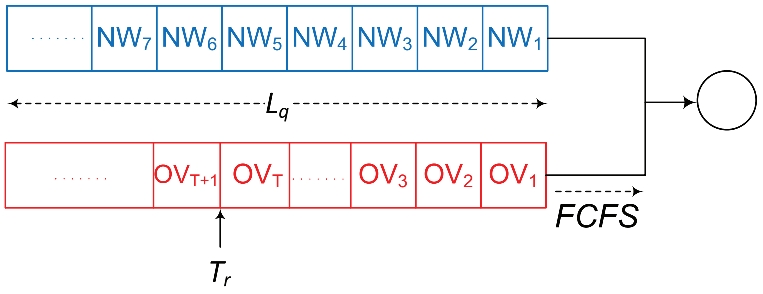
Figure 5.2: The service order in the queue when full priority is given to overflow tasks.

5.2 Resource Allocation Solution

Fig. 5.1 depicts the overview of our solution. When the offloaded task reaches the datacenter, RAM looks into the PM pool to find idle resources. If sufficient VMs are available at that moment, they will be assigned to the task. Otherwise, the task should wait for the VMs to become available. Once the task gets to the head of the queue, RAM will look into the server pool to assign appropriate VMs. Our system also has the ability to control the



(a) Before threshold: $N_{ov} < T_r$.



NW$_7$ NW$_6$ NW$_5$ NW$_4$ NW$_3$ NW$_2$ NW$_1$
L_q
OV$_{T+1}$ OV$_T$ OV$_3$ OV$_2$ OV$_1$
FCFS
T_r

(b) After threshold: $N_{ov} > T_r$.

Figure 5.3: The service order in the queue when threshold-based priority is given to overflow tasks.

overflow tasks in the PM, i.e., if the PMs cannot host the assigned tasks, they will be returned to RAM to get another chance to obtain required resources. The RAM gives higher priority to overflow task than the new incoming tasks. However, there is a possibility that a task is rejected because of insufficient resources inside a PM. Another possibility is that RAM gets full and as the result, arriving tasks into this module get blocked.

The order which the tasks get service is based on the priority scheme adopted in the queueing system shown in Fig. 5.1. We have assumed that each type of task (new arrival and overflow) have a dedicated queue and the size of each queue is L_q. If the queues are full, they cannot accept new tasks; therefore, these tasks will get blocked. We have considered two priority scheme, full priority of overflow tasks and the threshold-based priority of overflow tasks. If full priority is given to overflow tasks, the overflow tasks waiting in queue will get service in FCFS order before the new incoming tasks as shown in Fig. 5.2. Once there is no overflow task left in queue, the new incoming tasks will be served in FCFS order. If threshold-based priority is given to overflow tasks, as long as the number of overflow tasks in queue is less than a threshold ($N_{ov} < T_r$) as illustrated in Fig. 5.3a, overflow tasks and new incoming tasks get service according to the probabilities of P_o and P_N respectively; this approach is similar to Weighted Fair Queueing (WFQ) method presented in [16]. If the number of overflow tasks in queue is larger than the threshold ($N_{ov} > T_r$), the full priority is given to overflow tasks and they get service in FCFS order as presented in Fig. 5.3b. Once the number of overflow tasks becomes less than the threshold, new arriving tasks get the chance to access the resources and both type of tasks get weighted priorities.

5.2.1 The Queueing Model of Resource Allocation

We have assumed that task requests arrive according to a Poisson process with arrival rate λ. An incoming request will be processed in the RAM which is modeled as a multi-dimensional CTMC presented in Figs. 5.4 and 5.5. RAM checks server pool to find whether there is a sufficient number of idle VMs on a PM to accommodate every request. $1/\beta$ is the mean look up time to find appropriate PM in the server pool.

In this model, we have developed two types of tasks; the first type is the overflow tasks which can be generated in PMs because of resource shortage in PMs. The overflow tasks will return to RAM to get service. The second type of tasks is the new task arrivals into the system which includes the majority of the tasks. We have prioritized the overflow tasks over new arrivals because of the deadline of the tasks already accepted into the system. Priority is a useful scheduling method that allows different customer types to receive differentiated performance levels.

The first type of tasks which are the overflow tasks have a non-preemptive strict priority over the new incoming tasks.

We have followed two different approaches in defining the priority scheme. In the first approach, full priority is given to the overflow tasks and in the second approach, the overflow tasks have threshold-based priority over the new incoming tasks.

The Queueing Model of Resource Allocation with Full Priority of Overflow Tasks

In this approach, illustrated in Fig. 5.4, we have assumed that overflow tasks have the full priority, i.e., new arrival tasks will not be served if there is an overflow task left in queue. We have modeled RAM as Markovian multiserver queueing system with two priority levels.

The vertical direction illustrates the overflow tasks waiting in the queueing system. The horizontal direction demonstrates the new incoming tasks waiting in the queue to get service. The maximum number of the waiting tasks in queue is L_q. The states are labeled as (i, j, k): i denotes the number of overflow tasks waiting in queue, j indicates the number of new arriving tasks waiting in queue and k presents the admission mode: 'A' represents the acceptance of the task and 'R_o' and 'R_N' are corresponding to rejection of the overflow and new incoming task respectively.

The new arriving tasks can only get service in the first horizontal line direction which is corresponding to the case where no overflow task is left in queue ($i = 0$). The dashed state shown in Fig. 5.4 is corresponding to the case of new arriving task rejection and is represented as $(0, L_q, R_N)$. The new task arrives with rate of λ. If this task is accepted, the system moves to the state $(0, j-1, A))$ with the rate of $P_s\beta$. Otherwise, the system moves to $(0, j+1, A)$ which means the new task is added to the waiting new tasks. If the queue is full, a new task cannot be added to queue and system moves from state $(0, L_q, A)$ to $(0, L_q, R_N)$ with the rate of $\beta(1-P_s)$. Then, the system moves to $(0, L_q, A)$ with rate of η and the task is rejected. η is the clean up rate and we have assumed $\eta = 10\beta$. P_s is the success probability of finding appropriate VMs in VMM and later in this section, we will explain how it is calculated.

As can be seen, in the other horizontal lines, new incoming tasks arrive with rate of λ and they only wait for the completion of overflow tasks ($i = 0$) to have access to resources.

The overflow tasks can get served in all vertical line directions. This means that they have access to the resources despite the new arrivals. Therefore, they have full priority over new arriving tasks. The case of overflow task rejection is represent in the dotted states which are represented as (L_q, j, R_o). The overflow tasks arrive in RAM with rate of O_o. Similar to the new arriving tasks, if the overflow task is accepted, the system moves to the state $(i-1, j, A)$ with the rate of $P_s\beta$. Otherwise, the system moves to the state $(i+1, j, A)$ which means the overflow task is added to the waiting tasks. In this case, if the queue is full, system moves from state (L_q, j, A) to (L_q, j, R_o) with the rate of $\beta(1-P_s)$. Then, the system goes back to (L_q, j, A) with rate of η and the task is rejected.

In this approach of RAM, the task blocking probability, P_{bq}, which is due to having full

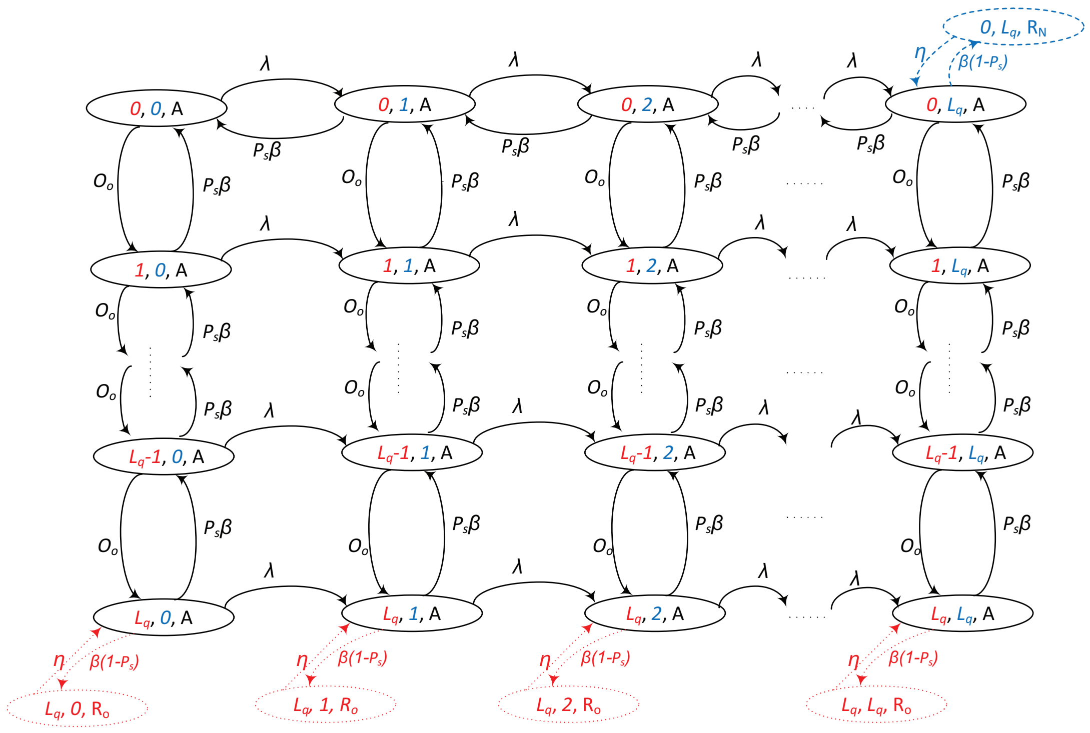


Figure 5.4: RAM: Overflow tasks have full priority over new arriving tasks.

RAM module, is computed as

$$P_{bq} = \sum_{i=0}^{L_q-1} \pi(i, L_q, A) + \sum_{j=0}^{L_q} \sum_{k \in S_2} \pi(L_q, j, k) + \pi(0, L_q, R_N) \tag{5.1}$$

where $S_2 = \{A, R_o\}$.

The task rejection probability, P_{br}, which is the rejection due to insufficient resources, is calculated as:

$$P_{br} = \frac{\beta(1-P_s)}{\eta}\pi(0, L_q, R_N) + \sum_{j=1}^{L_q} \frac{\beta(1-P_s)}{\eta}\pi(L_q, j, R_o) \tag{5.2}$$

The total rejection probability, P_{rj}, is the sum of the two above parameters:

$$P_{rj} = P_{bq} + P_{br} \tag{5.3}$$

The Queueing Model of Resource Allocation with Threshold-based Priority of Overflow Tasks

In order to decrease the waiting time of new arrivals and giving them more opportunity to have access to resources, we have followed a second approach. In this approach, overflow tasks have threshold-based priority over new arrivals, i.e., new arrival tasks and overflow tasks will be served based on the probabilities of P_N and P_o respectively until the number of waiting overflow tasks reaches a threshold, T_r. After this threshold, the overflow tasks have the full priority in the system and new arrivals should wait as long as number of waiting overflow tasks is larger than the threshold. P_N is calculated as $P_N = \frac{\lambda}{\lambda+O_o}$ and P_o is obtained as $P_o = \frac{O_o}{\lambda+O_o}$.

In Fig. 5.5, the vertical direction demonstrates the overflow tasks waiting in the queue. The horizontal direction illustrates the new incoming tasks waiting in the queue to get served. The maximum number of waiting tasks is L_q. The states are labeled similar to the first approach.

In this case, new arriving tasks can get service in the horizontal line direction while the number of waiting overflow tasks is less than a threshold ($i \leq T_r$) and overflow tasks do not have priority over the new incoming tasks. When the number of waiting overflow tasks is larger than T_r, new arrivals must wait until this number gets less than T_r. The dashed states illustrate the case of new incoming task rejection and the corresponding states are represented as (i, j, R_N) where $i \leq T_r$. The new tasks arrive with rate of λ. If the task is accepted, the system moves to the state $(i, j-1, A)$ with the rate of $P_N P_s \beta$. Otherwise, the system moves to $(i, j+1, A)$ and the task waits in queue. In this case, if the queue is full, system moves to the state of (i, L_q, R_N) with the rate of $\beta(1 - P_N P_s)$ and after rejecting the task, system will move to state (i, L_q, A) with clean up rate of η.

It can be seen that in the horizontal direction lines with a number of line less than or equal to T_r, both type of tasks get same type of service; whereas in the horizontal direction lines beyond the threshold, new incoming tasks arrive with rate of λ and they only wait for the number of waiting overflow tasks get less than the threshold value ($i \leq T_r$) to be served.

The overflow tasks can get service in all vertical line directions and they have access to the resources without considering whether the new arrivals are permitted to access the resources or not. Thus, overflow tasks have a higher priority than new arriving flow. The case of overflow task rejection is represented in the dotted states which are represented as (i, j, R_o). The overflow tasks reach in RAM with O_o rate. If the overflow task is accepted, the system moves to the state $(i-1, j, A)$ with the rate of $P_o P_s \beta$ or $P_s \beta$; this rate depends on i. if $i \leq T_r$, the rate is $P_o P_s \beta$ and if $i > T_r$, the rate is $P_s \beta$. On the other hand, if task is rejected, the system moves to $(i+1, j, A)$ and the overflow task is added to the waiting tasks. In this case, if the queue is full, system moves from state (L_q, j, A) to (L_q, j, R_o) with the rate of $\beta(1 - P_s)$. Then, the system goes back to $(L - q, j, A)$ with rate of η and the task is rejected.

In this case, task blocking probability, P_{bq} and task rejection probability P_{br} are calcu-

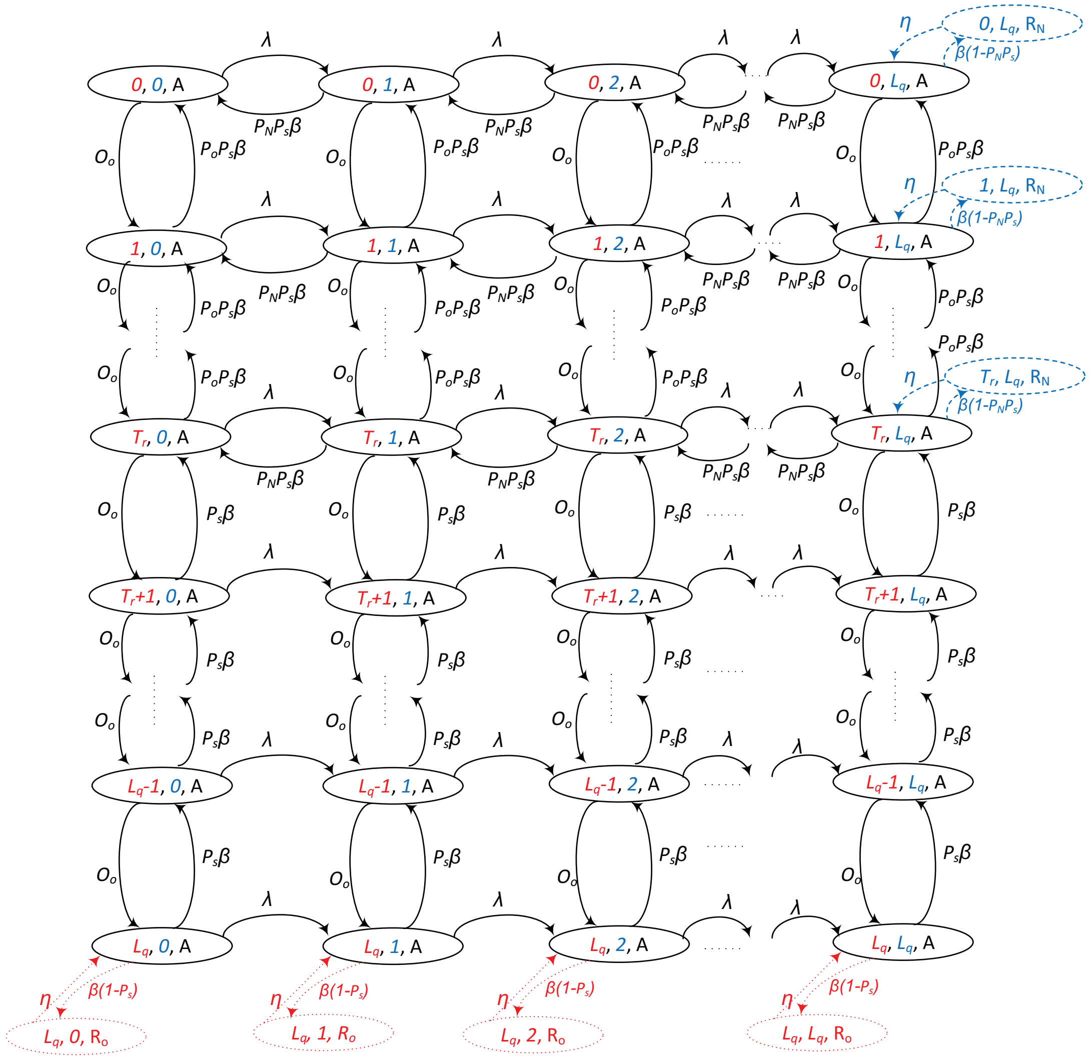


Figure 5.5: RAM: Overflow tasks have a threshold-based priority over new arriving tasks.

lated as follows:

$$P_{bq} = \sum_{i=0}^{T_r} \sum_{k \in S_1} \pi(i, L_q, k) + \sum_{j=0}^{L_q} \sum_{k \in S_2} \pi(L_q, j, k) \tag{5.4}$$

where $S_1 = \{A, R_N\}$ and $S_2 = \{A, R_o\}$.

$$P_{br} = \sum_{i=1}^{T_r} \frac{\beta(1 - P_N P_s)}{\eta} \pi(i, L_q, R_N) + \sum_{j=1}^{L_q} \frac{\beta(1 - P_s)}{\eta} \pi(L_q, j, R_o) \tag{5.5}$$

Total rejection probability, P_{rj}, is the sum of P_{bq} and P_{br}.

5.2.2 The Queueing Model of Virtual Machine Provisioning

In the cloud datacenter, each mobile device is associated with a cloud clone, which runs on VMs that can execute mobile applications on behalf of the mobile device. Our default setting for resource allocation is using only one VM which clones the data and applications of the mobile device called primary server. The primary server is always online, waiting for the mobile device to connect to it. If the mobile application needs more computation resources, system will use the second type of VMs. These VMs, called secondary servers, in general do not clone the data and applications of a specific mobile device and can be allocated to any user on demand. When the mobile device connects to the cloud, it communicates with the primary server which in turn manages the secondaries, informing them that a new client has connected. All interactions between the mobile device and the primary server are as cloud user-server, but now the primary server behaves as a (transparent) proxy for the secondaries. Every time when the mobile device asks for service requiring more than one VM, the primary server resumes the needed number of secondary clones.

We use the semantics of VM forking to clone the primary and secondary servers. The VM fork abstraction lets an application take advantage of cloud resources by forking multiple copies of its VM. VM fork preserves the isolation and ease of software development associated with VMs [40]. The basic idea behind VM forking is similar to the familiar process fork: a parent VM (in our case primary server) issues a fork call which creates a number of clones, or child VMs (secondary servers). Each of the forked VMs proceeds with an identical view of the system, but using a unique identifier (vmid) allows them to be distinguished from one another and from the parent. However, each forked VM has its own independent copy of the operating system and virtual disk, and state updates are not propagated between VMs. Forked VMs are entities whose memory image and virtual disk are discarded once they exit. Any application-specific state or values they generate must be explicitly communicated to the parent VM, for example by message passing or via a distributed file system. VM fork has to be used with care as it replicates all the processes and threads of the parent VM; if multiple processes within the same VM simultaneously invoke VM forking, conflicts can happen. Therefore, VM fork should be used in VMs that have been customized to run a single application or perform a specific task. Upon VM fork, each child is configured with a new IP address based on its vmid, and it is placed on the same virtual subnet as the VM from which it was created.

Regarding to the dynamic demands of mobile service requests, we assume that number of tasks in a job varies randomly during the time that job is in service. The arrival of the jobs to the system is according to the Poisson distribution with the rate of λ and arrival rate to each PM is a Poisson process with the rate of λ_i. A new arriving job initially demands service for a single task. A job generates random number of tasks according to a Poisson process with the rate of λ_{ci} during its service time. Each task requires a VM for its execution and task execution times are exponentially distributed. Service time of a job begins with its arrival to the system and it is completed when there are no more tasks belonging to that job left in the system.

The proposed VMM develops the instantiation, provisioning and deployment of VMs on PMs. The service time is corresponding to the task execution time of the allocated VM and time spent on generation of overflow in PMs. The model provisions all the tasks of a job request on a single PM.

Fig. 5.6 depicts the VMM model in a PM where the PM can accept up to three jobs. We have developed a multi-dimensional CTMC to model the accommodation of jobs. Each state is labeled as (i, j, k) where i indicates the jobs waiting in the PM's queue to get served, j denotes the number of primary servers in service process or in general, the number of jobs being served on the PM and k denotes the number of secondary servers which are serving a job. Each PM can host up to m VMs. Mean service rates of primary VMs and secondary VMs are respectively represented as μ and d and they are exponentially distributed. O_i denotes the incoming overflow rate which is directed from RAM to the PMs and O_o represents the overflow rate which is generated in the PM and will be forwarded to RAM for reallocation process. φ is the instantiation rate of a VM and it is also considered as the transition rate of adding a new serving job in the PM; as can be seen in Fig. 5.6, this transition rate is not constant through the Markov system and it varies according to the number of jobs which are in service in the PM. The details of the transition rate are presented later in this section. The arrival rate to each PM, λ_i, is computed as

$$\lambda_i = \frac{\lambda(1 - P_{bq})}{N} \tag{5.6}$$

where P_{bq} is the blocking probability obtained from RAM and N is the number of PMs in the system.

In Fig. 5.6, the main plane of the multi-dimensional Markov model illustrates the waiting queues and serving status of primary VMs. The horizontal rows represent how many primary tasks, more accurately how many mobile job requests, are waiting in line to get served; vertical rows are corresponding to the number of jobs in service.

In the secondary task queues which are branched off from the main plane of the Markov model, forked tasks are generated according to Poisson distribution with the rate of λ_{ci}. The mean service rate, d, is exponentially distributed. The maximum length of the secondary or forked task queue can be L which in this model we have considered: $L = 4$, i.e., every job is assigned to a primary VM and can get a maximum of 4 secondary VMs. The forking time is included in the service time of the secondary task queues.

The secondary task queues are modeled as Birth-Death queueing systems with finite population, L-server case ($M/M/L//L$). This queueing system is inspired by the Birth-Death queueing systems with finite population presented in [35]. The equilibrium probability, p_k, which is the stationary state probability of being in the kth state of the queue, can be obtained as

$$p_k = p_0 \left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k} \tag{5.7}$$

we compute p_k for the states of $0 \leq k \leq L$. p_0 is calculated as

$$p_0 = \left[\sum_{k=0}^{L} \left(\frac{\lambda_{ci}}{d} \right)^k \binom{L}{k} \right]^{-1} = \frac{1}{(1 + \lambda_{ci}/d)^L} \tag{5.8}$$

and p_k is computed as

$$p_k = \begin{cases} \frac{\left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k}}{(1+\lambda_{ci}/d)^L} & 0 \leq k \leq L \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

The average number of secondary tasks in the queue is

$$\overline{Q} = \sum_{k=0}^{L} k p_k = \frac{\sum\limits_{k=0}^{L} k \left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k}}{(1 + \lambda_{ci}/d)^L} = \frac{L\lambda_{ci}/d}{1 + \lambda_{ci}/d} \tag{5.10}$$

In order to decide the number of mobile job requests which can be accommodated on a single PM, we have defined some intermediate parameters. We have a limitation over the number of tasks in the job requests (i.e., $L + 1$) and a constant number of VM available on each PM (defined as m). Therefore, we can be sure that in case of $m > L$, at least one job can be accommodated on a PM. The number of jobs with size of $L + 1$ in a PM is at least

$$c = \left\lfloor \frac{m}{L + 1} \right\rfloor \tag{5.11}$$

Parameter c represents the minimum number of tasks which can be deployed on a PM. However, if we consider c jobs accommodable on a PM, some of the VMs on the PM will be underused. Since we want to make sufficient use of VMs, we go beyond c jobs and the number of tasks which we have deployed on a PM is considered as $c + \frac{c}{2}$. However, we

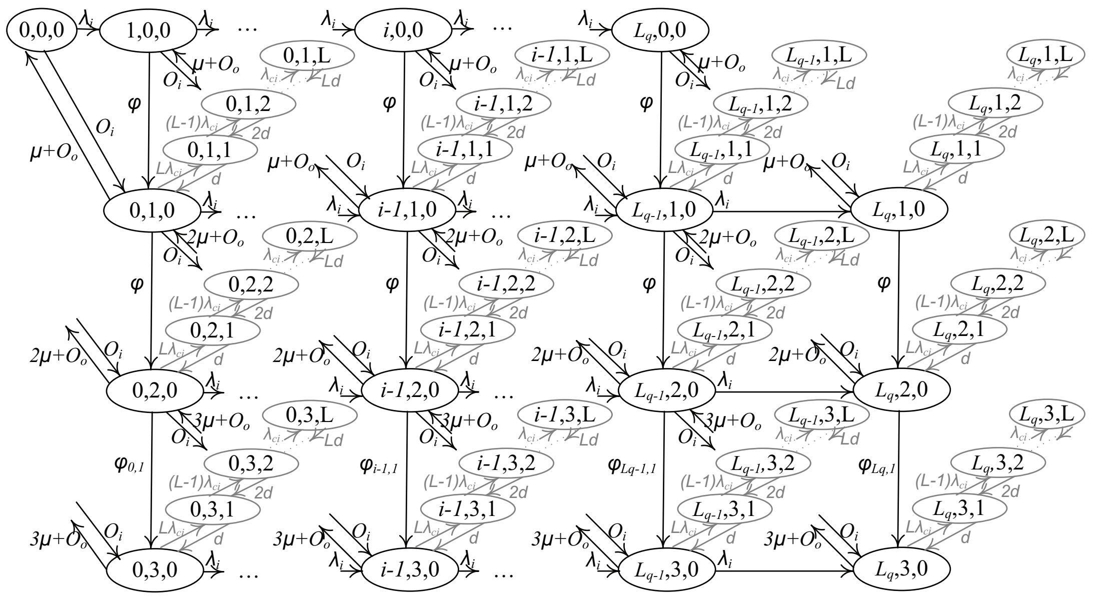


Figure 5.6: Virtual machine provisioning model of a PM with the ability to accept maximum three jobs.

have assumed that after deploying c jobs, the transition rate of moving to serve the next job is less than the previous jobs.

In our case: $m = 10$ and $L = 4$, i.e., a task of a mobile user can acquire maximum 5 VMs (1 primary and 4 secondary VMs). Also, $c = \left\lfloor \frac{10}{5} \right\rfloor = 2$ and the number of tasks on a PM is: $2 + 1 = 3$ which is the number we have used in the provisioning module illustrated in Fig. 5.6.

As can be seen, the transition rate in not constant in VMM: if number of current tasks on a PM is less than or equal to c, the transition rate is φ which corresponds to instantiation rate and we assume the instantiation time is equal to 1 sec; if number of current tasks hosted in a PM is more than c, the transition rate is $\varphi_{i,x}$ and is obtained as

$$\varphi_{i,x} = Pt_{i,x} \cdot \varphi \tag{5.12}$$

where x indicates the number of served jobs more than c. In the case illustrated in Fig .5.6, x is equal to 1. $Pt_{i,x}$ is the transition coefficient corresponding to level x where i jobs are waiting in line and is computed as

$$Pt_{i,x} = \frac{\sum\limits_{h=1}^{c+x} \sum\limits_{k=\overline{Q}}^{L} p(i, j_h, k)}{\sum\limits_{r=1}^{c+x} \sum\limits_{k=1}^{L} p(i, j_r, k)} \tag{5.13}$$

where $(c+x)$ is the number of primary VMs until the horizontal level x. $\overline{Q}$ is the average number of expected secondary tasks in a single secondary queue and its value is calculated as (5.10). In the nominator of (5.13), with $(c+x)$ jobs under service and i jobs waiting in line, the sum of steady-state probabilities where the length of each secondary queue is larger than $\overline{Q}$ is calculated and in denominator, the sum of steady-state probabilities for full length of each secondary queue is calculated.

The probability that job request cannot be deployed on the PM is calculated as

$$P_{na} = p(L_q, 0, 0) + \sum_{k=1}^{L} \sum_{j=1}^{c+\frac{c}{2}} p(L_q, j, k) + \sum_{i=0}^{L_{q-1}} \sum_{y \in \Phi} P_y \tag{5.14}$$

where $p(i, j, k)$ indicates the steady-state probability of the corresponding state and P_y is a member of Φ.

Φ is a set of product of probability of states (one state from each job's secondary queue in the vertical direction) which the sum of their corresponding secondary VMs is more than the capacity of secondary VMs on a PM. The probability that the total number of secondary VMs in a PM gets more than a specific number is a combinatorial probability and it can be computed as a sum of products in (5.14). Φ is represented as

$$\Phi = \left\{ \prod_{l=1}^{c+\frac{c}{2}} p(i, j_l, k_l) \middle| \sum_{l=1}^{c+\frac{c}{2}} k_l > m - (c + \frac{c}{2}) \right\} \tag{5.15}$$

where $m - (c + \frac{c}{2})$ denotes the number of allowed secondary VMs on a PM.

The successful provisioning probability can be obtained from

$$P_s = 1 - P_{na}^N \tag{5.16}$$

Finally, the overflow rate generated in a PM, O_o, is calculated as

$$O_o = \frac{\frac{(\lambda_i/\mu)^{c+\frac{c}{2}}}{(c+\frac{c}{2})!} \cdot \frac{(\lambda_{ci}/d)^{m-(c+\frac{c}{2})}}{[m-(c+\frac{c}{2})]!}}{\sum_{i=0}^{c+\frac{c}{2}} \sum_{j=0}^{m-(c+\frac{c}{2})} \frac{(\lambda_i/\mu)^i}{i!} \cdot \frac{(\lambda_{ci}/d)^j}{j!}} \tag{5.17}$$

In order to obtain O_o, we have computed the probability that a job request is blocked due to lack of resources. We have been inspired by Erlang B formula and the truncation of a system consisting two independent queues in a multi-dimensional Markov system [6] to determine this rate.

Algorithm 7 demonstrates how we have solved the VMM module and obtained its parameters. In Section 5.2.3, we will discuss the successive iteration model of RAM and

VMM. Algorithm 7 will only be deployed for one time during the first iteration of the integrated model. During the next iterations of the integrated model, same transition rates will be used in VMM module.

In Algorithm 7, first, we assume that all the transition rates in the module are equal to (φ) and we solve the model using this rate and obtain the steady-state probability of all the states and overflow rate; then, we calculate the $Pt_{i,x}$ for every level after the c jobs in the system; according to these new transition rates, we solve the module and we compute the new values of steady-state probability of all the states and using these values, we can calculate P_s. Overflow rate is independent of the steady-state probabilities; therefore, it is not needed to calculate it again.

Algorithm 7 First Time Solving of VMM Module

```
  Use basic transition/instantiation rate $(\varphi)$ to solve the sub-model;
  Compute outgoing overflow rate, $O_o$;
  Calculate $Pt_{i,x}$ coefficient for every corresponding state of $i$ and level of $x$;
  Solve the module with new transition rates ($\varphi$ or $\varphi_{i,x}$);
  Calculate $P_s$ in the new solved sub-model;
```

5.2.3 The Integrated Model

The overall model, consists of two interactive stochastic modules. This reduces complexity of the model itself but the computational complexity of solving the model is solved via successive fixed point iteration. The associated pseudocode is shown in Algorithm 8. Iteration ends when the difference between the values of probabilities in successive iterations drops below a predefined threshold ($\Delta = 10^{-6}$).

The interactions among modules are presented in Fig. 5.7. VMM computes the successful provisioning probability (P_s) and also the overflow rate (O_o). Success provisioning

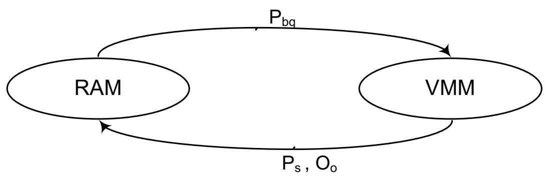


Figure 5.7: Interaction of provisioning modules.

probability and overflow rate are used as input parameters to RAM. On the other hand, RAM computes the task blocking probability, P_{bq}, which is the input parameter to VM provisioning module. Note that only during the first iteration, VM provisioning module will execute Algorithm 7 to find the transition rates.

Algorithm 8 The Integrated model Algorithm

```
**Input:** Initial successful provisioning probability and overflow rate: $P_{s0}, O_{o0}$;
**Output:** Blocking probability in the RAM: $P_{bq}$;
count $=$ 0; maximum $=$ 30; $\Delta = 1$;
$P_{bq0} \longleftarrow$ RAM $(P_{s0}, O_{o0})$;
**while** $\Delta \geq 10^{-6}$ **do**
    count $\longleftarrow$ count +1; $P_s \longleftarrow$ VMM $(P_{bq0})$;
    $O_o \longleftarrow$ VMM $(P_{bq0})$;
    $P_{bq1} \longleftarrow$ RAM $(P_s, O_o)$;
    $\Delta \longleftarrow |(P_{bq1} - P_{bq0})|$;
    $P_{bq0} \longleftarrow P_{bq1}$;
    **if** count == maximum **then**
        break;
    **end if**
**end while**
**if** count == maximum **then**
    **return** -1;
**else**
    **return** $P_{bq0}$;
**end if**
```

5.3 Performance Evaluation

The proposed model is solved using Maple 15 [26].

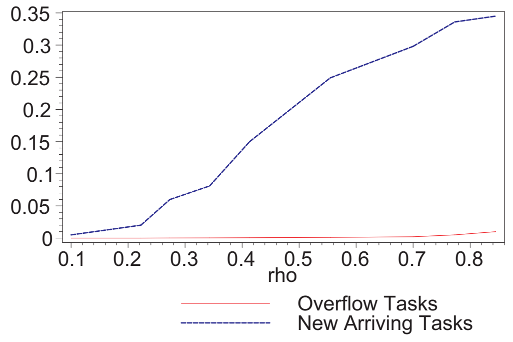
To evaluate the performance of the proposed mobile cloud system, we have solved the model for four scenarios. First, we kept the mean service time constant and varied task arrival rate; in this case, we have analyzed the effects of incoming mobile tasks on three important metrics, blocking probability, total delay and utilization. Second, we kept the task arrival rate constant but varied mean service time; in this case we investigate the impact of service time on blocking probability and total delay imposed by the cloud center. In both of these scenarios, the maximum length of the secondary task queue or in other words, the number of forked tasks accepted in a job, L, is equal to 4. Note that size of a job is equal to $L + 1$. In the third scenario, we kept mean service time and task arrival rate constant and varied L. We have analyzed the impact of changing the maximum number of accepted tasks in a job on blocking probability and total delay. In the fourth scenario, we evaluate the effect of changing threshold location in overflow queue on blocking probability in the case of threshold-based priority; in this scenario, arrival rate, mean service time and L are constant and threshold location is changed.

To facilitate comparison under different combination of fixed and variable independent variables, we have plotted the diagrams as functions of offered load, computed as $\rho = \frac{\lambda}{m \cdot N \cdot \mu}$, where m, the maximum number of VMs running on a single PM is assumed to be 10 and N, the total number of PMs in the system is 100. Also, we have assumed that $L_q = 50$ and except in fourth scenario represented in Fig. 5.15 which in threshold is variable, in other cases $T_r = 30$.

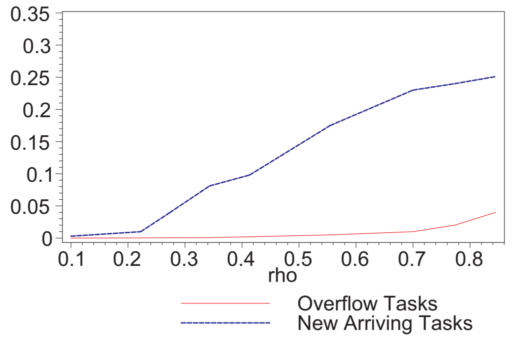
5.3.1 Task Blocking Probability

Task blocking probability obtained in two first scenarios is shown in Figs.5.8 and 5.9. As expected, probability of task blocking increases with the offered load. Overflow tasks are given priority and, consequently, easier access to resources under both full and threshold-based priority mechanisms, which is why the blocking probability is much lower for such tasks. However, when threshold-based priority is applied, blocking probability for newly arrived tasks is noticeably lower, while that for overflow tasks is slightly higher. This indicates that the performance for one or the other type of tasks may be adjusted within certain limits. In the worst case, less than 4of overflow tasks are blocked.

We have also investigated the blocking probability under constant offered load but with a variable limit to the number of secondary tasks L (i.e, third scenario); the results obtained under both service policies are shown in Fig. 5.10. Note that the case $L = 0$ corresponds to the absence of secondary tasks which, by extension, means that there are no overflow tasks; consequently, there is no corresponding data value for threshold-based priority curve. Again, new arriving tasks suffer a higher blocking rate which slowly increases with the task forking limit L; overflow tasks, on the other hand, are not affected much due to the dual-queue prioritization mechanism presented above. As before, threshold-based prioritization provides much better performance for new tasks than its full priority counterpart. We note that a rough upper bound for the probability that a job does not complete because a forked task is ultimately blocked may be obtained as the product of mean length of secondary task queue and probability of overflow, $P_{na}\overline{Q}$.

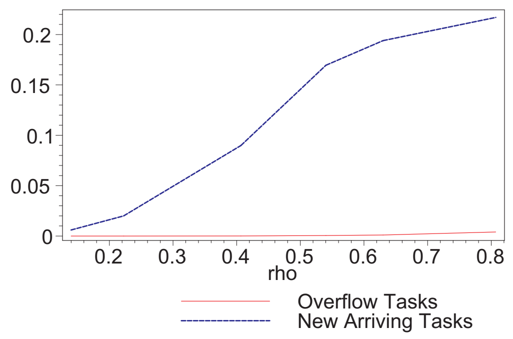


(a) Task Blocking Probability in case of full priority.

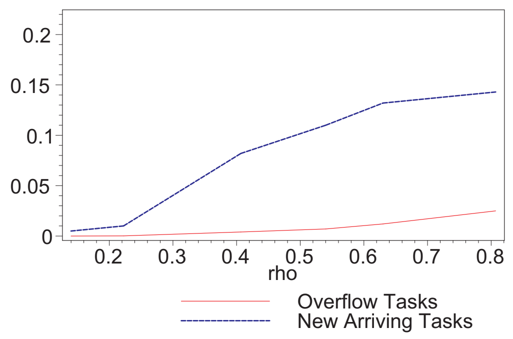


(b) Task Blocking Probability in case of threshold-based priority ($T_r = 30$).

Figure 5.8: Task Blocking Probability wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Service time = 1 min. Variable task arrival rate (100-1300 tasks/min). $L = 4$.



(a) Task Blocking Probability in case of full priority.

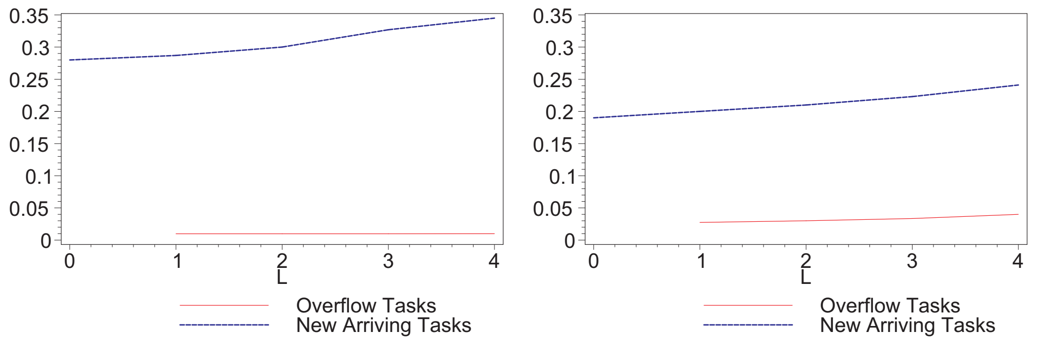


(b) Task Blocking Probability in case of threshold-based priority ($T_r = 30$).

Figure 5.9: Task Blocking Probability wrt constant arrival rate and variable service time in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 250 tasks/min. Variable service time (30-180 sec). $L = 4$.

5.3.2 Mean Task Delay

As for the mean task delays, threshold-based prioritization offers lower values (i.e., better performance), as can be seen in Figs. 5.11, 5.12 and 5.13. As can be expected, mean delays increase rather sharply with the offered load. As the system operates well below



(a) Task Blocking Probability in case of full priority.

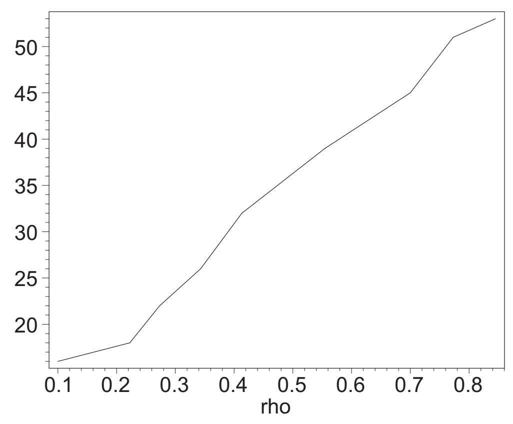
(b) Task Blocking Probability in case of threshold-based priority ($T_r = 30$).

Figure 5.10: Task Blocking Probability wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 1300 tasks/min. Service time = 1 min. Variable L (0–4).

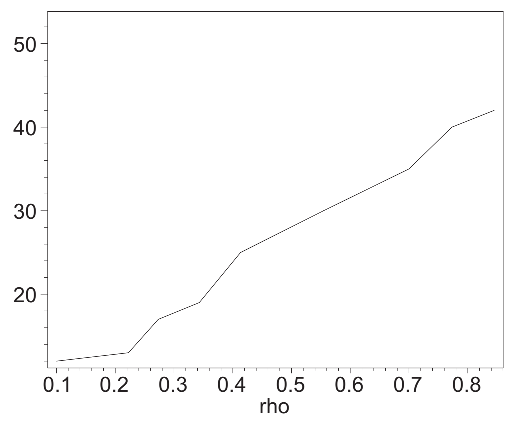
saturation, rise in delay values is approximately linear. In case of variable task forking limit, the rise is somewhat milder, but this may be due to the comparatively low value of offered load utilized to generate data for Fig. 5.13. We note that for the same offered load in the both priority cases, the cloud center generally appears to be more sensitive to the task arrival rate than to mean service time. This is due to the overhead imposed by the waiting times and provisioning processes which increases with the number of tasks but is independent of the task service time.

5.3.3 Utilization

Server utilization is shown in Fig. 5.14. Under both prioritization policies, utilization increases with the offered load. As the system does not enter saturation, the rate of rise is approximately linear in both cases, although some flattening may be observed at offered load $\rho = 0.7$ and above. We note that utilization is slightly lower when full priority is given to the overflow tasks, compared to the threshold-based policy, since the number of overflow

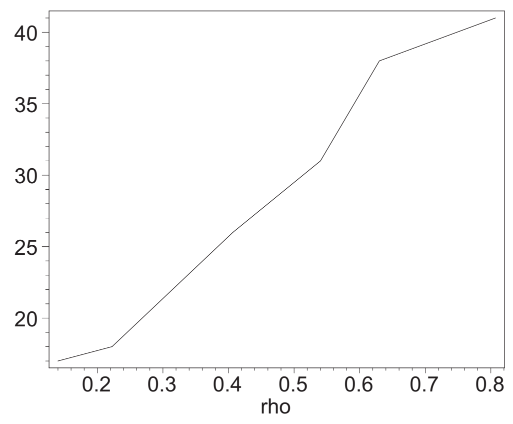


(a) Total Delay in case of full priority.

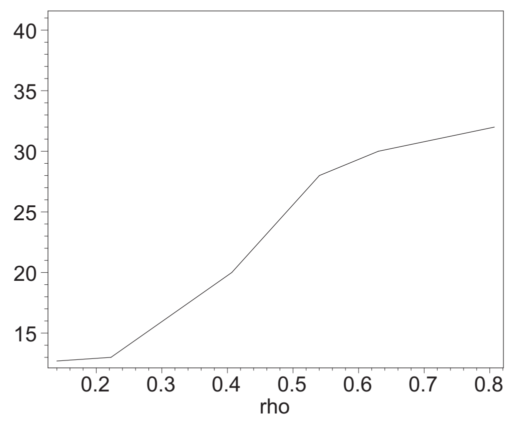


(b) Total Delay in case of threshold-based priority ($T_r = 30$).

Figure 5.11: Total Delay Probability wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Service time = 1 min. Variable task arrival rate (100-1300 tasks/min). $L = 4$.



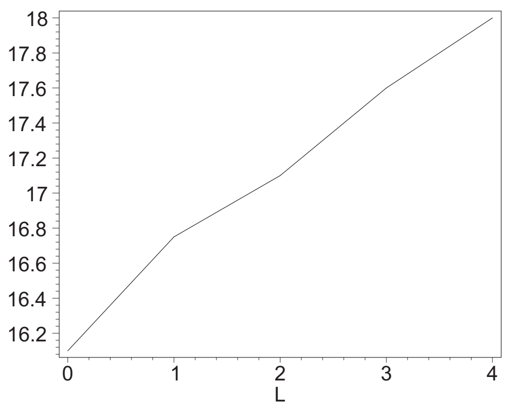
(a) Total Delay in case of full priority.



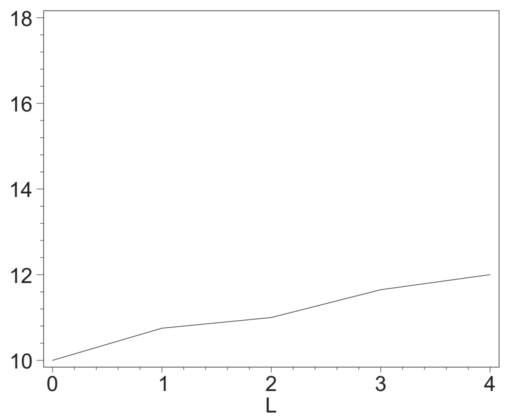
(b) Total Delay in case of threshold-based priority ($T_r = 30$).

Figure 5.12: Total Delay Probability wrt constant arrival rate and variable service time in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 250 tasks/min. Variable service time (30-180 sec). $L = 4$.

tasks is low.

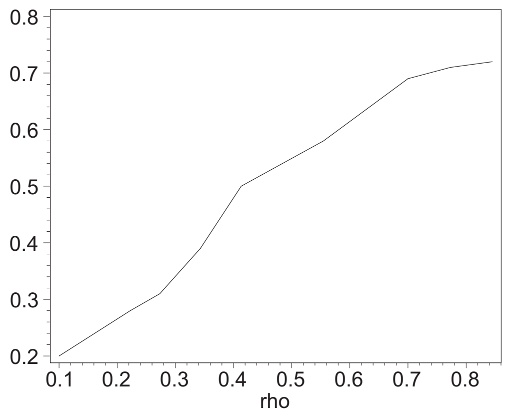


(a) Total Delay in case of full priority.

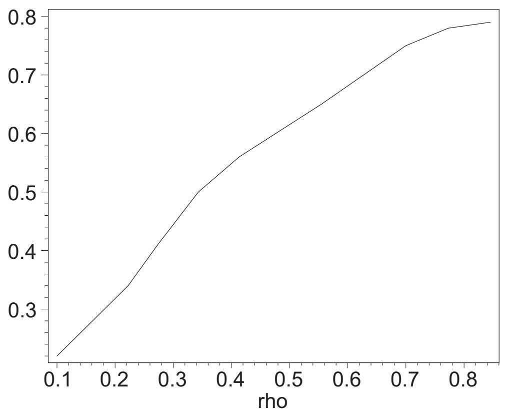


(b) Total Delay in case of threshold-based priority ($T_r = 30$).

Figure 5.13: Total Delay wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 250 tasks/min. Service time = 1 min. Variable L (0–4).



(a) Utilization in case of full priority.



(b) Utilization in case of threshold-based priority ($T_r = 30$).

Figure 5.14: Utilization wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Service time = 1 min. Variable task arrival rate (100-1300 tasks/min). $L = 4$.

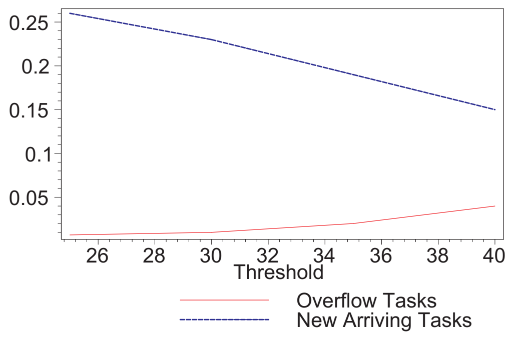


Figure 5.15: Task Blocking Probability wrt constant service time and variable arrival rate in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 1000 tasks/min. Service time = 1 min. $L = 4$. Variable T_r (25–40)

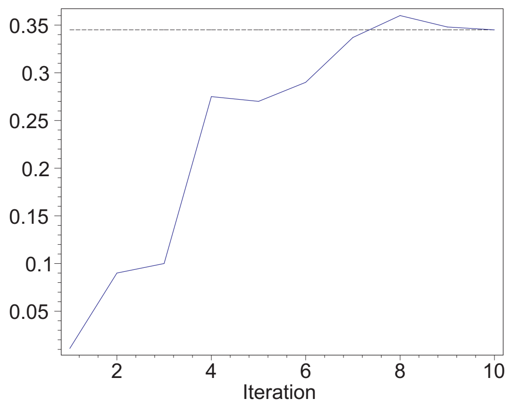
5.3.4 The Impact of Queue Threshold T_r

Fig. 5.15 shows the effect of queue threshold in the threshold-based prioritization policy on task blocking probability. As the threshold moves closer to the queue size of $L_q = 50$, value of the blocking probability for new and overflow tasks are getting closer to each other as the probability that the threshold will be exceeded diminishes. Conversely, lower values of the threshold push the system to behave in a manner closer to that under full priority in fact, full priority policy is equivalent to a threshold-based one with threshold value of $T_r = 0$.

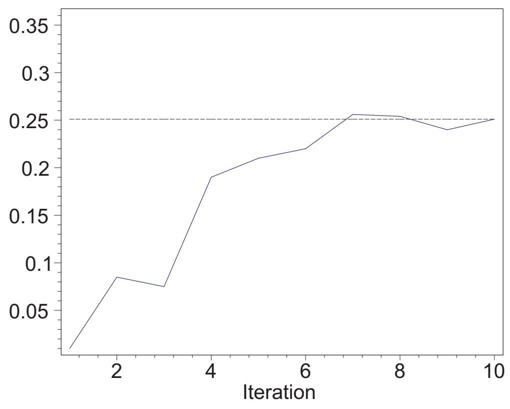
5.3.5 Task Blocking Probability Through Iterations

Fig.5.16 illustrates task Blocking Probability of new incoming and overflow tasks through iterations in case of giving full and threshold-based priorities to overflow tasks. The value of last iteration is considered as the final value of each parameter.

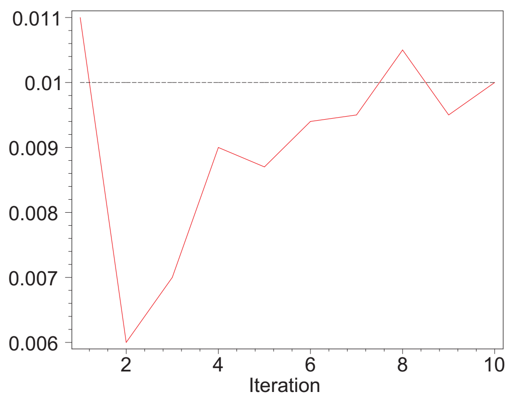
Overall, the threshold-based policy allows the cloud operator to fine-tune the perfor-



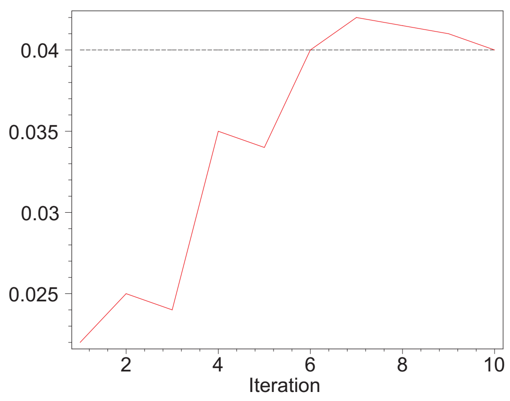
(a) Task Blocking Probability of new arriving tasks in case of full priority.



(b) Task Blocking Probability of new arriving tasks in case of threshold-based priority ($T_r = 30$).



(c) Task Blocking Probability of overflow tasks in case of full priority.



(d) Task Blocking Probability of overflow tasks in case of threshold-based priority ($T_r = 30$).

Figure 5.16: Task Blocking Probability through iterations in case of giving full and threshold-based priorities to overflow tasks. Arrival rate = 1300 tasks/min. Service time = 1 min. $L = 4$.

mance of the cloud, as there are a number of parameters which can be adjusted to provide the desired values, or ranges thereof, for critical performance indicators such as mean delay and task blocking.

5.4 Chapter Summary

We have proposed a solution for resource allocation of on-demand job requests in mobile cloud computing. We have developed two priority schemes for resource allocation in a server pool based on giving different priorities to the overflow tasks including full priority of overflow tasks and the threshold-based priority of overflow tasks. Unlike most of existing works that either rely on a linear programming formulation or on intuitively derived heuristics that offer no theoretical performance guarantees, our model does not sacrifice the complexity of offloading problem just to make it solvable. Instead, complexity is addressed through the use of two interacting stochastic models which are solved through fixed point iteration to achieve any desired error level. We have investigated the impact of task arriving rate, service time and the size of offloaded job on the performance metrics for both priority schemes. Also, we have evaluated the effect of threshold location on the threshold-based priority scheme. Our results confirm that threshold-based priority presents better system performance than full priority of overflow tasks.

Also, we have observed that the performance metrics do not change linearly with regard to the offered load, which indicates that finding the settings of parameter values that would lead to optimal values of performance metrics is non-trivial.

A continuation to the work presented in this chapter can be the modification of threshold-based priority scheme in order to adjust the location of threshold in the overflow queue.

Chapter 6

Conclusion

In this thesis we have examined the behavior of an IaaS cloud data center in which servers are partitioned into pools of hot, warm, and cold machines. This evaluation has been performed in different operating areas of linear, transition to saturation and saturation. We have observed that the manner in which servers are partitioned in the pools affects the performance, sometimes even more than the variations of offered load. We have also demonstrated that the task arrival rate is more critical parameter affecting the performance of the cloud datacenter than mean task service time, due to the overhead incurred in resource allocation and provisioning. We have also shown that the energy expenditure is highly dependent on the manner in which servers are partitioned into pools.

Also, in other attempt where PMs are partitioned into a hot and a cold pool, we have analyzed the effects of pool threshold and mean look up time in hot and cold pools on the performance. Our results confirm that the system should operate well below saturation in order to provide acceptable level of performance. Moreover, we have shown that the performance is more sensitive to the mean look up time for the PMs in the cold pool.

Next, we have proposed two task admission control algorithms to prevent the cloud system getting into the saturation region. These algorithms are based upon establishing thresholds for task arrival rate and task blocking probability. Our simulation results confirm that the proposed admission control algorithms improve system performance.

Furthermore, we have proposed two task admission control algorithms utilizing two controlling parameters, full rate task acceptance threshold and filtering coefficient. The first algorithm is lightweight and it is appropriate for systems which experience small steps of change in task arrival rate. This algorithm is based on long-term estimation of average utilization and offered load. The second algorithm is efficient for the systems with unexpected arrival rate, is more computationally-intensive and is based on instantaneous utilization. Our results confirm that both of the task admission mechanisms maintain the cloud's stability. Also, we have observed that in different ranges of incoming task rate and mean service time, the average offered load will result in the similar average performance in the cloud system.

We have proposed a solution for resource allocation of on-demand offloaded jobs in mobile clouds. This solution can be also used in other clouds which their users' requests fork new tasks. We have developed two priority schemes for resource allocation in a server pool based on giving different priorities to the overflow tasks. The overflow tasks are the forked tasks that may not find sufficient resources in the designated PM. The two priority schemes include full priority of overflow tasks and the threshold-based priority of overflow tasks. We have evaluated the impact of task arriving rate, service time and the size of offloaded job on the performance metrics for both priority schemes. Also, we have examined the effect of threshold location on the threshold-based priority scheme. Our results confirm that threshold-based priority presents better system performance than full priority of overflow tasks.

Our next step in providing resource allocation for mobile clouds will be the modification of threshold-based priority scheme in order to adjust the location of threshold in the overflow queue. Finding the best location of threshold is an optimization problem and the position can change according to the performance metrics, different policies adopted by cloud computing providers or cloud system's requirements. Also, we have observed that the performance metrics do not change linearly with regard to the offered load. Therefore, finding the optimal position of the performance metrics is difficult. One approach can be the segmentation of the performance metrics into linear regions and solve the optimization problem according to these linear regions.

Appendix A

Abbreviations and Symbols

A.1 Abbreviations

ADP	Approximate Dynamic Programming
AP	Access Point
BMHA	Batch Mode Heuristic Scheduling Algorithm
CPA	Cloud Personal Assistant
CTMC	Continuous Time Markov Chain
DES	Discrete Event Simulation
EDF	Earliest Deadline First
EWMA	Exponentially Weighted Moving Average
FCFS	First Come First Served
FIFO	First-In, First-Out
FPTAS	Fully Polynomial Time Approximation Scheme

GAMS	General Algebraic Modeling System
GPRS	General Packet Radio Service
IaaS	Infrastructure-as-a-Service
IEEE	Institute of Electrical and Electronics Engineers
LMMF	Lexicographically Max- Min Fair
LP	Linear Programming
MAC	Medium Access Control
MFTF	Most Fit Task First
MDP	Markov Decision Process
NIST	National Institute of Standards and Technology
PaaS	Platform-as-a-Service
PGF	Probability Generating Function
PHY	Physical Layer
PM	Physical Machine
PMM	Pool Management Module
PMSM	Pool Management Sub-Model
QoS	Quality of Service
RAM	Resource Allocation Module
RASM	Resource Allocation Sub-Model
RED	Random Early Detection
RR	Round Robin

SaaS	Service-as-a-Service
SJF	shortest-Job-First
SLA	Service Level Agreements
SMDP	Semi-Markov Decision Process
WLAN	Wireless LAN
VM	Virtual Machine
VMM	VM Provisioning Module
VMPM	Virtual Machine Provisioning Module
VMSM	VM Provisioning Sub-Model
VNE	Virtualized Network Embedding

A.2 Symbols and Corresponding Descriptions

N	Number of servers
m	Number of available VMs on the PM
λ, λ_t	Incoming task rate
γ	Partitioning coefficient of hot, warm and cold servers
ρ	Offered load
μ_{tot}	Overall service rate
P_{blk}	Blocking probability
P_k	Probability of occurrence of state k

λ_{org}	Full rate of incoming task rate
λ_f	Accepted rate of task arrivals
F_{cf}	Filtering coefficient
ρ_{av}	Average utilization of the system
T	Threshold of full rate task acceptance rate
R	System capacity / Thresholds of full rate task rejection
N_{ov}	Number of waiting overflow tasks in RAM
U_{thr}	Utilization threshold
u'_{av}	Estimated average utilization
u_c	Instantaneous utilization
M_{thr}	Threshold margin
ρ_{thr}	Offered load threshold
ρ_{av}	Average offered load
$1/\beta$	Mean look up time to find a PM
$1/\eta$	Mean clean up time in RAM
λ_i	Primary task arrival rate into the PMs
λ_{ci}	Secondary task generation rate in a job
μ	Mean service rate of primary tasks
d	Mean service rate of secondary tasks
L_q	Size of queue/queues in RAM
L	Maximum number of secondary tasks in a job

c	Minimum number of jobs which can be accommodated on a PM
O_i	Incoming overflow rate from the RAM to VMM
O_o	Outgoing overflow rate from the VMM to RAM
T_r	Threshold of number of waiting overflow tasks in RAM
P_o	Probability of giving service to the overflow tasks
P_N	Probability of giving service to the new incoming tasks
P_s	Successful provisioning probability
P_{bq}	Blocking probability due to full RAM
P_{br}	Rejection probability due insufficient resources
P_{rj}	Total rejection probability
φ	Basic instantiation/ Full transition rate
φ_x	Partial transition rate

Appendix B

Publications

B.1 Journal Papers

- *Submitted, under review:* H. Khojasteh, J. Misic, and V. B. Misic. Prioritization of Overflow Tasks to Improve Performance of Mobile Cloud, In *IEEE Transactions on Cloud Computing (TCC) 2016*.
- *To appear in:* H. Khojasteh and J. Misic. Task Admission Control Policy in Cloud Server Pools Based on Task Arrival Dynamics, In *Wireless Communications and Mobile Computing (WCMC) 2016*.
- V. B. Misic, M. S. I. Khan, Md. M. Rahman, H. Khojasteh and J. Misic. Simple Solutions May Still Be Best: On the Selection of Working Channels in a Channel-Hopping Cognitive Network, In *Wireless Communications and Mobile Computing (WCMC) 2013*.
- H. Khojasteh, J. Misic, and V. B. Misic. Integration of an IEEE 802.15.4 RFID

Network with Mobile Readers with a 802.11 WLAN, In *Wireless Communications and Mobile Computing (WCMC) 2012*.

B.2 Conference Papers

- H. Khojasteh, J. Misic and V. B. Misic. Task Filtering as a Task Admission Control Policy in Cloud Server Pools, In *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2015)*, Dubrovnik, Croatia, August 2015.
- H. Khojasteh, J. Misic and V. B. Misic. Task Admission Control for Cloud Server Pools, In *IEEE Wireless Communications and Networking Conference (WCNC 2015)*, New Orleans, LA, United States, March 2015.
- H. Khojasteh, J. Misic and V. B. Misic. Analyzing the Impact of Provisioning Overhead Time in Cloud Computing Centers, In *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE2014)*, Toronto, ON, Canada, May 2014.
- H. Khojasteh, J. Misic and V. B. Misic. Characterizing Energy Consumption of IaaS Clouds in Non-saturated Operation, In *IEEE INFOCOM 2014 Workshop on Mobile Cloud Computing*, Toronto, ON, Canada, April 2014.
- H. Khojasteh, J. Misic and V. B. Misic. A Two-Tier Integrated RFID/Sensor Network with a WiFi WLAN, In *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC 2012)*, Limassol, Cyprus, August 2012.
- J. Misic, H. Khojasteh, N. Khan and V. B. Misic. CSCD: A Simple Channel Scan Protocol to Discover and Join a Cognitive PAN, In *IEEE Wireless Communications*

and Networking Conference (WCNC 2012), Paris, France, June 2012.

- J. Misic, N. Khan, H. Khojasteh and V. B. Misic. Towards an Efficient Rendezvous Protocol for a Cognitive PAN, In *IEEE International Conference on Communications (ICC 2012)*, Ottawa, ON, Canada, June 2012.

- H. Khojasteh and A. Abhari, Mobile Mash-up Model Based on Hybrid P2P Using Ajax Technology. In *Spring Simulation Multiconference (SCS/SpringSim10 Poster Workshop in Collaboration with ACM/SIGSIM)*, Orlando, FL, United States, April 2010.

B.3 Book Chapters

- H. Khojasteh, J. Misic and V. B. Misic. Task Admission Control for Cloud Server Pools, A chapter in *Advances in Mobile Cloud Computing Systems, F. R. Yu and V. Leung, editors*, CRC Press (Taylor & Francis Group), December 2015.

Appendix C

Related Scripts

C.1 Resource Allocation Model in a Pooled IaaS cloud

```
#Algorithm 1 Successive Substitution Method
#Input: Initial success probabilities in pools: Ph0; Pw0; Pc0
#Input: Initial idle probability of a hot PM: Pi0
#Output: Blocking probability in Global Queue: BPq
#counter=0; max=30; diff=1
Digits:=20:
differc:=1:
maxim:=30:
Lq:=50;
lambdast:=1000/3600:
alphah:=0.1736:
alphaw:=0.2291:
alphac:=0.2291:
phih:=0.021:
```

```
phiw:=0.027:
phic:=0.03:
Ph0:= 0.5:
Pw0:= 0.8:
Pc0:= 0.9:
Pi0:= 0.4:
Nh:=15: Nw:=10: Nc:=10:
energyfact:= 0.5:
Ntotal:=Nh+Nw+Nc:
mu:=1/2400:
# BPq0 <-RASM(Ph0,Pw0,Pc0) (Fig2: Resource Allocation Sub-model)
rho1:=(alphac+lambdast+(Ph0*alphah)+(Pw0*alphaw))/(alphah+lambdast);
rho2:=(alphah*(1-Ph0)+lambdast)/(alphaw+lambdast);
rho3:=(alphaw*(1-Pw0)+lambdast)/(alphac+lambdast);
lastrho1:=(((lambdast+(Ph0*alphah)+(Pw0*alphaw))/(alphaw))^Lq)/Lq!;
lastrho2:= (((alphah*(1-Ph0)+lambdast)/alphaw)^Lq)/Lq!;
lastrho3:= (((alphaw*(1-Pw0)+lambdast)/alphac)^Lq)/Lq!;
Gtrunc1:=sum((rho1^k1)/k1!, k1=0..Lq-1);
k1='k1':
Gtrunc2:=sum((rho2^k1)/k1!, k1=0..Lq-1);
k1='k1':
Gtrunc3:=sum((rho3^k1)/k1!, k1=0..Lq-1);
k1='k1':
Gtrunc:=Gtrunc1+Gtrunc2+Gtrunc3+lastrho1+lastrho2+lastrho3;
i:='i':
for i from 0 to 2 do
  for j from 0 to Lq-1 do
   if (i=0) then
```

```
     resiv[i][j]:= ((rho1^j)/j!)/Gtrunc;
   elif (i=1) then
     resiv[i][j]:= ((rho2^j)/j!)/Gtrunc;
   else
     resiv[i][j]:= ((rho3^j)/j!)/Gtrunc;
   end if;
  od;
od;
resiv[0][Lq]:=lastrho1/Gtrunc;
resiv[1][Lq]:=lastrho2/Gtrunc;
resiv[2][Lq]:=lastrho3/Gtrunc;
BPq0:= resiv[0][Lq]+resiv[1][Lq]+resiv[2][Lq];
# [Nh,Nw,Nc]<-PMM(Ph0, Pi0)  (Fig5: Pool Management Sub-model)
Gtrunc:=0: Gtrunc1:=0: Gtrunc2:=0: Gtrunc3:=0: rho1:=0: rho2:=0: rho3:=0:
i:'i': j:='j': k:='k': k1:='k1':
i:=Nh; j:=Nw; k:=Nc;
totalp:=0:
rho1:=FRw/RPi;
rho2:=RPi/FRw;
rho3:=(FRc+FRw)/(FRc+SU+RPi);
rho4:=(SU+FRw+RPi)/(SU+FRw+RPi);
rho5:=(SU+2*RPi)/(FRw+RPi);
SUw:=3: SUc:=2: RPi:=1: SU:=2:
FRw:=1/(lambdast*BPq0*(1-Ph0)+(1/SUw)):
FRc:=1/(lambdast*BPq0*(1-Ph0)+(1/SUc)):
Gtrunc1:=sum((rho1^k1)/k1!, k1=0..i+j-1);
k1:='k1':
Gtrunc2:=sum((rho2^k1)/k1!, k1=0..j-1);
```

```
k1:='k1':
Gtrunc3:=sum((rho3^k1)/k1!, k1=j..j+k);
k1:='k1':
Gtrunc4:=sum((rho4^k1)/k1!, k1=j..j+k-1);
k1:='k1':
Gtrunc5:=sum((rho5^k1)/k1!, k1=j-1..j+k-2);
k1:='k1':
Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+Gtrunc5;
mp:='mp':
Ncalculatori:=0:
Ncalculatorj:=0:
#Ncalculatork:=0:
for mp from 0 to i do
  pmiv[i-mp][j][k+mp]:= ((rho1^mp)/mp!)/Gtrunc;
  Ncalculatori := Ncalculatori+(i-mp)*pmiv[i-mp][j][k+mp];
  Ncalculatorj := Ncalculatorj+j*pmiv[i-mp][j][k+mp];
  #Ncalculatork := Ncalculatork+(k+mp)*pmiv[i-mp][j][k+mp];
od;
mp:='mp':
for mp from 0 to j-2 do
  pmiv[0][j-mp][k+i+mp]:= ((rho1^(mp+i))/(mp+i)!)/Gtrunc;
  Ncalculatorj := Ncalculatorj+(j-mp)*pmiv[0][j-mp][k+i+mp];
  #Ncalculatork := Ncalculatork+(k+i+mp)*pmiv[0][j-mp][k+i+mp];
od;
mp:='mp':
for mp from 0 to j-1 do
  pmiv[i+mp][j-mp][k]:= ((rho2^mp)/mp!)/Gtrunc;
  Ncalculatori := Ncalculatori+(i+mp)*pmiv[i+mp][j-mp][k];
  Ncalculatorj := Ncalculatorj+(j-mp)*pmiv[i+mp][j-mp][k];
  #Ncalculatork := Ncalculatork+k*pmiv[i+mp][j-mp][k];
od;
mp:='mp':
for mp from 0 to k do
  pmiv[i+j+mp][0][k-mp]:= ((rho3^(mp+j))/(mp+j)!)/Gtrunc;
  Ncalculatori := Ncalculatori+(i+j+mp)*pmiv[i+j+mp][0][k-mp];
  #Ncalculatork := Ncalculatork+(k-mp)*pmiv[i+j+mp][0][k-mp];
od;
mp:='mp':
for mp from 0 to k-1 do
  pmiv[i+j+mp][1][k-mp-1]:= ((rho4^(mp+j))/(mp+j)!)/Gtrunc;
  Ncalculatori := Ncalculatori+(i+j+mp)*pmiv[i+j+mp][1][k-mp-1];
  Ncalculatorj := Ncalculatorj+pmiv[i+j+mp][1][k-mp-1];
  #Ncalculatork := Ncalculatork+(k-mp-1)*pmiv[i+j+mp][1][k-mp-1];
od;
mp:='mp':
for mp from 0 to k-1 do
  pmiv[i+j+mp-1][2][k-mp-1]:= ((rho5^(mp+j-1))/(mp+j-1)!)/Gtrunc;
  Ncalculatori := Ncalculatori+(i+j+mp-1)*pmiv[i+j+mp-1][2][k-mp-1];
  Ncalculatorj := Ncalculatorj+2*pmiv[i+j+mp-1][2][k-mp-1];
  #Ncalculatork := Ncalculatork+(k-mp-1)*pmiv[i+j+mp-1][2][k-mp-1];
od;
mp:='mp':
Nh:=Ncalculatori;
Nh:= ceil(Nh);
Ncalculatori:=0:
if (Nh=0) then
```

```
  Nh:=1:
end if;
Nw:=Ncalculatorj;
Nw:= ceil(Nw);
Ncalculatorj:=0:
if (Nw=0) then
  Nw:=1:
end if;
Nc:= Ntotal-(Nh+Nw);
#while diff >= 10^-6 do
Nhvaraverage := 0:
counter:='counter':
for counter from 1 to maxim while differc > 0.000001 do
# [Ph, Pi] <-VMPSM-hot(BPq0,Nh)
i:'i': j:='j': k:='k': k1:='k1': k2:='k2': m:=10:
lambdah:=lambdast*(1-BPq0)/Nh:
Gtrunc1:=sum((((mu+lambdah)/(phih+lambdah))^k1)/k1!,k1=0..1);
k1:='k1':
Gtrunc2:=sum((((mu+lambdah)/(2*phih+lambdah))^k1)/k1!, k1=2..Lq);
k1:='k1':
Gtrunc3:=sum((((2*mu+lambdah+phih)/(phih+lambdah+mu))^k1)/k1!,k1=0..1);
k1:='k1':
Gtrunc4:=sum(sum((((k2*mu+lambdah+2*phih)/(2*phih+lambdah+(k2-1)*mu))^k1)
/k1!, k1=2..Lq),k2=2..m-1);
k1:='k1': k2:='k2':
Gtrunc5:=sum((((m*mu+lambdah+2*phih)/(phih+lambdah+(m-1)*mu))^k1)
/k1!,k1=0..Lq+1);
k1:='k1':
```

```
Gtrunc6:=sum((((phih+lambdah)/(m*mu+lambdah))^k1)/k1!, k1=0..Lq);

k1:='k1':

Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+Gtrunc5+Gtrunc6;

totalp:=0;

for t from 0 to 1 do

  vmivh[t][1][0]:= ((((mu+lambdah)/(phih+lambdah))^t)/t!)/Gtrunc;

od;

t:='t':

for t from 2 to Lq do

   vmivh[t][2][0]:= ((((mu+lambdah)/(2*phih+lambdah))^t)/t!)/Gtrunc;

od;

t:='t':

for t from 0 to 1 do

  vmivh[0][t][1]:= ((((2*mu+lambdah+phih)/(phih+lambdah+mu))^t)/t!)/Gtrunc;

od;

t:='t':

for t from 2 to Lq do

  for n from 1 to m-1 do

    vmivh[t][2][n]:= ((((n*mu+lambdah+2*phih)/

    (2*phih+lambdah+(n-1)*mu))^t)/t!)/Gtrunc;

  od;

od;

t:='t': n:='n':

vmivh[0][0][m-1]:= 1/Gtrunc;

for t from 0 to Lq do

  vmivh[t][1][m-1]:= ((((m*mu+lambdah+2*phih)/

  (phih+lambdah+(m-1)*mu))^t)/t!)/Gtrunc;

od;
```

```
t:='t':

for t from 0 to Lq do

  vmivh[t][0][m]:= ((((phih+lambdah)/(m*mu+lambdah))^t)/t!)/Gtrunc;

od;

t:='t':

Bh:= sum(vmivh[Lq][2][t],t=0..m-2)+vmivh[Lq][1][m-1]+vmivh[Lq][0][m];

t:='t':

Ph:= 1-(Bh^Nh);

# Pw <-VMPSM-warm(BPq0,Ph,Nw) (Similar to Fig4: VMPSM for a PM in hot pool)

i:'i': j:='j': k:='k': k1:='k1': k2:='k2': m:=10:

lambdaw:=lambdast*(1-BPq0)*(1-Ph)/Nw:

Gtrunc1:=sum((((mu+lambdaw)/(phiw+lambdaw))^k1)/k1!,k1=0..1);

k1:='k1':

Gtrunc2:=sum((((mu+lambdaw)/(2*phiw+lambdaw))^k1)/k1!, k1=2..Lq);

k1:='k1':

Gtrunc3:=sum((((2*mu+lambdaw+phiw)/(phiw+lambdaw+mu))^k1)/k1!,k1=0..1);

k1:='k1':

Gtrunc4:=sum(sum((((k2*mu+lambdaw+2*phiw)/

(2*phiw+lambdaw+(k2-1)*mu))^k1)/k1!, k1=2..Lq),k2=2..m-1);

k1:='k1': k2:='k2':

Gtrunc5:=sum((((m*mu+lambdaw+2*phiw)/(phiw+lambdaw+(m-1)*mu))^k1)

/k1!,k1=0..Lq+1);

k1:='k1':

Gtrunc6:=sum((((phiw+lambdaw)/(m*mu+lambdaw))^k1)/k1!, k1=0..Lq);

k1:='k1':

Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+Gtrunc5+Gtrunc6;

totalp:=0;

for t from 0 to 1 do
```

```
  vmivw[t][1][0]:= ((((mu+lambdaw)/(phiw+lambdaw))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 2 to Lq do
  vmivw[t][2][0]:= ((((mu+lambdaw)/(2*phiw+lambdaw))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 0 to 1 do
  vmivw[0][t][1]:= ((((2*mu+lambdaw+phiw)/(phiw+lambdaw+mu))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 2 to Lq do
  for n from 1 to m-1 do
    vmivw[t][2][n]:= ((((n*mu+lambdaw+2*phiw)/
    (2*phiw+lambdaw+(n-1)*mu))^t)/t!)/Gtrunc;
  od;
od;
t:='t': n:='n':
vmivw[0][0][m-1]:= 1/Gtrunc;
for t from 0 to Lq do
  vmivw[t][1][m-1]:= ((((m*mu+lambdaw+2*phiw)/
  (phiw+lambdaw+(m-1)*mu))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 0 to Lq do
  vmivw[t][0][m]:= ((((phiw+lambdaw)/(m*mu+lambdaw))^t)/t!)/Gtrunc;
od;
t:='t':
```

```
Bw:= sum(vmivw[Lq][2][t],t=0..m-2)+vmivw[Lq][1][m-1]+vmivw[Lq][0][m];
t:='t':
Pw:= 1-(Bw^Nw);
# Pc <-VMPSM-cold(BPq0,Ph,Pw,Nc)
i:'i': j:='j': k:='k': k1:='k1': k2:='k2': m:=10:
lambdac:=lambdast*(1-BPq0)*(1-Ph)*(1-Pw)/Nc:
Gtrunc1:=sum((((mu+lambdac)/(phic+lambdac))^k1)/k1!,k1=0..1);
k1:='k1':
Gtrunc2:=sum((((mu+lambdac)/(2*phic+lambdac))^k1)/k1!, k1=2..Lq);
k1:='k1':
Gtrunc3:=sum((((2*mu+lambdac+phic)/(phic+lambdac+mu))^k1)/k1!,k1=0..1);
k1:='k1':
Gtrunc4:=sum(sum((((k2*mu+lambdac+2*phic)/(2*phic+lambdac+(k2-1)*mu))^k1)
/k1!, k1=2..Lq),k2=2..m-1);
k1:='k1': k2:='k2':
Gtrunc5:=sum((((m*mu+lambdac+2*phic)/(phic+lambdac+(m-1)*mu))^k1)
/k1!,k1=0..Lq+1);
k1:='k1':
Gtrunc6:=sum((((phic+lambdac)/(m*mu+lambdac))^k1)/k1!, k1=0..Lq);
k1:='k1':
Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+Gtrunc5+Gtrunc6;
totalp:=0;
for t from 0 to 1 do
  vmivc[t][1][0]:= ((((mu+lambdac)/(phic+lambdac))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 2 to Lq do
  vmivc[t][2][0]:= ((((mu+lambdac)/(2*phic+lambdac))^t)/t!)/Gtrunc;
```

```
od;
t:='t':
for t from 0 to 1 do
  vmivc[0][t][1]:= ((((2*mu+lambdac+phic)/(phic+lambdac+mu))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 2 to Lq do
  for n from 1 to m-1 do
    vmivc[t][2][n]:= ((((n*mu+lambdac+2*phic)/
    (2*phic+lambdac+(n-1)*mu))^t)/t!)/Gtrunc;
  od;
od;
t:='t': n:='n':
vmivc[0][0][m-1]:= 1/Gtrunc;
for t from 0 to Lq do
  vmivc[t][1][m-1]:= ((((m*mu+lambdac+2*phic)/
  (phic+lambdac+(m-1)*mu))^t)/t!)/Gtrunc;
od;
t:='t':
for t from 0 to Lq do
  vmivc[t][0][m]:= ((((phic+lambdac)/(m*mu+lambdac))^t)/t!)/Gtrunc;
od;
t:='t':
Bc:= sum(vmivc[Lq][2][t],t=0..m-2)+vmivc[Lq][1][m-1]+vmivc[Lq][0][m];
t:='t':
Pc:= 1-(Bc^Nc);
# [Nh,Nw,Nc] <-PMM(Ph, Pi)  (Fig5: Pool Management Sub-model)
# Not in use Nh:=10: Nw:=10: Nc:=10:
```

```
Gtrunc:=0: Gtrunc1:=0: Gtrunc2:=0: Gtrunc3:=0: rho1:=0: rho2:=0: rho3:=0:

i:'i': j:='j': k:='k': k1:='k1':

i:=Nh; j:=Nw; k:=Nc;

totalp:=0:

rho1:=FRw/RPi;

rho2:=RPi/FRw;

rho3:=(FRc+FRw)/(FRc+SU+RPi);

rho4:=(SU+FRw+RPi)/(SU+FRw+RPi);

rho5:=(SU+2*RPi)/(FRw+RPi);

totalpower:= 0;

SUw:=3: SUc:=2: RPi:=1: SU:=2:

FRw:=1/(lambdast*BPq0*(1-Ph)+(1/SUw)):

FRc:=1/(lambdast*BPq0*(1-Ph)+(1/SUc)):

Gtrunc1:=sum((rho1^k1)/k1!, k1=0..i+j-1);

k1:='k1':

Gtrunc2:=sum((rho2^k1)/k1!, k1=0..j-1);

k1:='k1':

Gtrunc3:=sum((rho3^k1)/k1!, k1=j..j+k);

k1:='k1':

Gtrunc4:=sum((rho4^k1)/k1!, k1=j..j+k-1);

k1:='k1':

Gtrunc5:=sum((rho5^k1)/k1!, k1=j-1..j+k-2);

k1:='k1':

Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+Gtrunc5;

mp:='mp':

Ncalculatori:=0:

Ncalculatorj:=0:

#Ncalculatork:=0:
```

```
for mp from 0 to i do
  pmiv[i-mp][j][k+mp]:= ((rho1^mp)/mp!)/Gtrunc;
  timeiv[i-mp][j][k+mp]:= (1/RPi)*((i-mp)*POWER_UNIT+j*energyfact*POWER_UNIT);
  totalpower:= totalpower+timeiv[i-mp][j][k+mp];
  Ncalculatori := Ncalculatori+(i-mp)*pmiv[i-mp][j][k+mp];
  Ncalculatorj := Ncalculatorj+j*pmiv[i-mp][j][k+mp];
  #Ncalculatork := Ncalculatork+(k+mp)*pmiv[i-mp][j][k+mp];
od;
mp:='mp':
for mp from 0 to j-2 do
  pmiv[0][j-mp][k+i+mp]:= ((rho1^(mp+i))/(mp+i)!)/Gtrunc;
  timeiv[0][j-mp][k+i+mp]:= (1/RPi)*((j-mp)*energyfact*POWER_UNIT);
  totalpower:= totalpower+timeiv[0][j-mp][k+i+mp];
  Ncalculatorj := Ncalculatorj+(j-mp)*pmiv[0][j-mp][k+i+mp];
  #Ncalculatork := Ncalculatork+(k+i+mp)*pmiv[0][j-mp][k+i+mp];
od;
mp:='mp':
for mp from 0 to j-1 do
  pmiv[i+mp][j-mp][k]:= ((rho2^mp)/mp!)/Gtrunc;
  timeiv[i+mp][j-mp][k]:= (1/FRw)*((i+mp)*POWER_UNIT+
  (j-mp)*energyfact*POWER_UNIT);
  totalpower:= totalpower+timeiv[i+mp][j-mp][k];
  Ncalculatori := Ncalculatori+(i+mp)*pmiv[i+mp][j-mp][k];
  Ncalculatorj := Ncalculatorj+(j-mp)*pmiv[i+mp][j-mp][k];
  #Ncalculatork := Ncalculatork+k*pmiv[i+mp][j-mp][k];
od;
mp:='mp':
for mp from 0 to k do
```

```
  pmiv[i+j+mp][0][k-mp]:= ((rho3^(mp+j))/(mp+j)!)/Gtrunc;
  timeiv[i+j+mp][0][k-mp]:= (1/(FRc+SU+RPi))*((i+j+mp)*POWER_UNIT);
  totalpower:= totalpower+timeiv[i+j+mp][0][k-mp];
  Ncalculatori := Ncalculatori+(i+j+mp)*pmiv[i+j+mp][0][k-mp];
  #Ncalculatork := Ncalculatork+(k-mp)*pmiv[i+j+mp][0][k-mp];
od;
mp:='mp':
for mp from 0 to k-1 do
  pmiv[i+j+mp][1][k-mp-1]:= ((rho4^(mp+j))/(mp+j)!)/Gtrunc;
  timeiv[i+j+mp][1][k-mp-1]:= (1/(SU+FRw+RPi))*((i+j+mp+energyfact)
  *POWER_UNIT);
  totalpower:= totalpower+timeiv[i+j+mp][1][k-mp-1];
  Ncalculatori := Ncalculatori+(i+j+mp)*pmiv[i+j+mp][1][k-mp-1];
  Ncalculatorj := Ncalculatorj+pmiv[i+j+mp][1][k-mp-1];
  #Ncalculatork := Ncalculatork+(k-mp-1)*pmiv[i+j+mp][1][k-mp-1];
od;
mp:='mp':
for mp from 0 to k-1 do
  pmiv[i+j+mp-1][2][k-mp-1]:= ((rho5^(mp+j-1))/(mp+j-1)!)/Gtrunc;
  timeiv[i+j+mp-1][2][k-mp-1]:= (1/(FRw+RPi))*((i+j+mp-1+(2*energyfact))
  *POWER_UNIT);
  totalpower:= totalpower+timeiv[i+j+mp-1][2][k-mp-1];
  Ncalculatori := Ncalculatori+(i+j+mp-1)*pmiv[i+j+mp-1][2][k-mp-1];
  Ncalculatorj := Ncalculatorj+2*pmiv[i+j+mp-1][2][k-mp-1];
  #Ncalculatork := Ncalculatork+(k-mp-1)*pmiv[i+j+mp-1][2][k-mp-1];
od;
mp:='mp':
Nh:=Ncalculatori;
```

```
Nh:= ceil(Nh);
Ncalculatori:=0:
if (Nh=0) then
 Nh:=1:
end if;
Nw:=Ncalculatorj;
Nw:= ceil(Nw);
Ncalculatorj:=0:
if (Nw=0) then
 Nw:=1:
end if;
Nc:= Ntotal-(Nh+Nw);
Nhvar:=0: Nhmean:=0:
mp:='mp':
for mp from 0 to i do
  Nhvar := Nhvar+((i-mp-Nh)^2)*pmiv[i-mp][j][k+mp]:
  Nhmean:= Nhmean + (i-mp)*pmiv[i-mp][j][k+mp]:
od;
mp:='mp':
for mp from 0 to j-2 do
  Nhvar := Nhvar+((0-Nh)^2)*pmiv[0][j-mp][k+i+mp]:
od;
mp:='mp':
for mp from 0 to j-1 do
  Nhvar := Nhvar+((i+mp-Nh)^2)*pmiv[i+mp][j-mp][k]:
  Nhmean:= Nhmean + (i+mp)*pmiv[i+mp][j-mp][k]:
  #Ncalculatorj := Ncalculatorj+(j-mp)*pmiv[i+mp][j-mp][k];
od;
```

```
mp:='mp':
for mp from 0 to k do
  Nhvar := Nhvar+((i+j+mp-Nh)^2)*pmiv[i+j+mp][0][k-mp]:
  Nhmean:= Nhmean + (i+j+mp)*pmiv[i+j+mp][0][k-mp]:
od;
mp:='mp':
for mp from 0 to k-1 do
  Nhvar := Nhvar+((i+j+mp-Nh)^2)*pmiv[i+j+mp][1][k-mp-1]:
  Nhmean:= Nhmean + (i+j+mp)*pmiv[i+j+mp][1][k-mp-1]:
od;
mp:='mp':
for mp from 0 to k-1 do
  Nhvar := Nhvar+((i+j+mp-1-Nh)^2)*pmiv[i+j+mp-1][2][k-mp-1]:
  Nhmean:= Nhmean + (i+j+mp-1)*pmiv[i+j+mp-1][2][k-mp-1]:
od;
mp:='mp':
Nhvaraverage := Nhvaraverage+Nhvar;
# BPq1 <-RASM(Ph,Pw,Pc)
rho1:=(alphac+lambdast+(Ph*alphah)+(Pw*alphaw))/(alphah+lambdast);
rho2:=(alphah*(1-Ph)+lambdast)/(alphaw+lambdast);
rho3:=(alphaw*(1-Pw)+lambdast)/(alphac+lambdast);
#lastrho1:= ((lambdast/alphah)^Lq)/Lq!;
#lastrho1:=(((alphac+lambdast+(Ph*alphah)+(Pw*alphaw))/alphaw)^Lq)/Lq!;
lastrho1:=(((lambdast+(Ph*alphah)+(Pw*alphaw))/(alphaw))^Lq)/Lq!;
lastrho2:= (((alphah*(1-Ph)+lambdast)/alphaw)^Lq)/Lq!;
lastrho3:= (((alphaw*(1-Pw)+lambdast)/alphac)^Lq)/Lq!;
Gtrunc1:=sum((rho1^k1)/k1!, k1=0..Lq-1);
k1='k1':
```

```
Gtrunc2:=sum((rho2^k1)/k1!, k1=0..Lq-1);

k1='k1':

Gtrunc3:=sum((rho3^k1)/k1!, k1=0..Lq-1);

k1='k1':

Gtrunc:=Gtrunc1+Gtrunc2+Gtrunc3+lastrho1+lastrho2+lastrho3;

i:='i': j:='j':

for i from 0 to 2 do

  for j from 0 to Lq-1 do

    if (i=0) then

      resiv[i][j]:= ((rho1^j)/j!)/Gtrunc;

    elif (i=1) then

      resiv[i][j]:= ((rho2^j)/j!)/Gtrunc;

    else

      resiv[i][j]:= ((rho3^j)/j!)/Gtrunc;

    end if;

  od;

od;

resiv[0][Lq]:=lastrho1/Gtrunc;

resiv[1][Lq]:=lastrho2/Gtrunc;

resiv[2][Lq]:=lastrho3/Gtrunc;

BPq1:= resiv[0][Lq]+resiv[1][Lq]+resiv[2][Lq];

#diff <-|(BPq1 - BPq0)|

differc := abs(BPq1-BPq0);

#BPq0 <-BPq1

BPq0:= BPq1:

i:='i': j:='j':

# Calculate BPr according to Equation (3) in the paper.

BPr:= sum((alphac*(1-Pc)*resiv[2][k1])/(alphac+lambdast), k1=0..Lq);
```

```
k1='k1':
# End of the main loop
end do:
sumresiv:= 0:
Q_res_average:= 0:
for j from 0 to Lq do
  for i from 0 to 2 do
    sumresiv:= sumresiv + resiv[i][j]:
  od:
Q_res_average:= Q_res_average + j * sumresiv:
sumresiv:= 0:
od:
i:='i': j:='j':
wt:= Q_res_average / (lambdast *(1-BPq0));
lut:= ((1/alphah)+((1-Ph)*(1/alphaw))+((1-Pw)*(1/alphac)))/(1-BPq0);
if (counter=maxim) then
 BPq0:= -1;
end if;
#Return BPq0 and Preject.
BPq0;
BPr;
Preject:= BPq0+BPr;
Nhvaraverage:= Nhvaraverage/counter;
Nhdev := sqrt(Nhvaraverage);
Q_res_average;
Nhmean;
PMwt := Nhmean/FRw;
t:='t':
```

```
vprovision_total :=0:
for t from 0 to 1 do
  vprovision_total := vprovision_total + t* vmivh[t][1][0]:
od:
t:='t':
for t from 2 to Lq do
  vprovision_total := vprovision_total + t* vmivh[t][2][0]:
od:
t:='t':
for t from 0 to 1 do
  vprovision_total := vprovision_total + t* vmivh[0][t][1]:
od:
t:='t':
for t from 2 to Lq do
  for n from 1 to m-1 do
    vprovision_total := vprovision_total + t*vmivh[t][2][n]:
  od;
od;
t:='t': n:='n':
for t from 0 to Lq do
  vprovision_total := vprovision_total + t*vmivh[t][1][m-1]:
od:
t:='t':
for t from 0 to Lq do
  vprovision_total := vprovision_total + t*vmivh[t][0][m]:
od:
t:='t':
vprov_delay := vprovision_total/lambdah;
```

```
total_delay := vprov_delay + wt + lut + PMwt;
coeffi := Nhdev/Nhmean;
```

C.2 Admission Control Algorithms

C.2.1 Admission Control Model Based on Offered Load

```
mu:= 1/2400;
R:=1000:
utilT:=0.75:
lambda:='lambda';
dis := proc ( k, ind,  T) options operator, arrow;
    local tmp, discourage, disca:
    description "produces discourage coefficient beyond the threshold";
# discourage:= 1- (k-T)/(R-T):
# discourage:= 1/(k+1):
# in this case  after threshold T we admit only T/R of the tasks
# discourage:= (T)/(R):
# here discouragement depends on offered load above some threshold
# we relate rho_T= T/R so in this script T should be T=R*rho_t
# below the threshold disc is equal to 0 but we do not call it
# discourage := max(0, (rho[ind]- T/R) );
# here max function is not needed since we start admission when
# util[lind]>utilT
    discourage := 1 -  ( rho[ind] - utilT) ;
    tmp:= discourage:
```

```
    return(tmp);
end proc:
## this procedure returns offered load into state k beyond the threshold
offload := proc ( k, ind, T) options operator, arrow;
    local tmp:
    description "produces offered load for state k in Markov chain";


    tmp:= dis(k, ind, T)* lambda / (mu*k) :
        return(tmp);
end proc:
#### this procedure returns probability state k beyond the threshold
PT := proc ( k, lind, T) options operator, arrow;
          local tmp, li:
    description "produces probability of  state k beyond the threshold";
    tmp:= Pt[0]*(lambda/mu)^T /T!  *
                product(offload(li,lind, T), li=T+1..k) :
          return(tmp);
end proc:
#### this procedure returns color of plot, to be used in plot command
## for a distribution of MC
colr := proc ( k) options operator, arrow;
    local tmp::string:
    description "produces color of plot";
    if k=1 then tmp:="red": elif k=2 then tmp:="blue"; elif k=3 then
                tmp:="brown"; elif k=4 then tmp:="green"; elif k=5 then
                tmp:="black": end if:
    return(tmp);
end proc:
```

```
for lind from  12 to 28 do
  print(‘lind=‘, lind);
  lambda := (lind*100)/(2*3600):
  rho[lind]:= lambda/(R*mu):
# Probability that system is empty comes from normalization condition
  P[0]:= 1/ (1 + sum(  (lambda/mu)^i /i!   , i=1..R) ):
  Pp[0][lind]:= P[0]:
#individual state probabilities when we do not have the theshold
  for l from 1 to R do
    P[l]:= P[0]*(lambda/mu)^l /l!
  od:
  l:=’l’:
# we plot the distribution
  for i from 0 to R do
          Point_p[i]:= [i, P[i]]:
  od:
  i:=’i’:
  seqP[lind]:= [seq(Point_p[i], i=0..R)]:
  plotP[lind]:= plot(seqP[lind], axes=boxed, style=point,
        symbol=box,symbolsize=16, color=colr(1), labels=[k,""]):
# test for normalization must be 1 always
  l:=’l’:
  test[lind] :=        add(P[l], l=0..R);
  Pb[lind]:= P[R]:
  util[lind]:= sum( P[jj]*jj, jj=0..R):
### now we check if admission control should be triggered, by comparing
# to threshold normalized utilization for preparation we set threshold
# to be R this can be changed in following if construct
```

```
  T[lind]:=R:
  myT[lind]:=R:
  if util[lind]/R > utilT then
# we need to introduce the threshold and to try to find it under given
# discouragement threshold has to be an integer but solution will be
# a real number so we need rounding
    disc[lind]:= dis(1, lind, T):   Tf:= floor(Tt):
    Pt[0]:= 1/ (1 + sum(  (lambda/mu)^i /i!   , i=1..Tf )
    + sum( (lambda/mu)^Tt /Tt! * product( offload(i,lind,Tf),
    i=Tf+1..k ), k=Tf +1..R)):
    utiln:= sum( (Pt[0]/jj!)*((lambda/mu)^jj)*jj, jj=0..Tf)
          + sum(  (Pt[0]/ko!)* ((lambda/mu)^Tt)*
          ((offload(ko, lind,Tf))^(ko-Tt))* ko, ko=Tf +1..R):
# we want utilization to be equal or as close to limiting util T
# exact solutions would not work since utilization can not be exactly
# equal to utilT but we need approximations  in the form floor(Tt)
    eqn1:= utiln/R = utilT :
    sols:= fsolve(eqn1, Tt, Tt=utilT*R-10..R):
## watch out this T[lind] is not an integer
    Ti[lind]:= subs(Tt=sols, Tt);
    T[lind]:= floor(Ti[lind]);
    util[lind]:= subs(Tt=sols, utiln);
    Pth[0]:= evalf(subs(Tt=T[lind], Pt[0]));
    for j from floor(utilT*R) to R-1 do
      xutil[j]:= evalf(subs(Tt=j, utiln));
      delta[j]:= abs(utilT- xutil[j]/R)  :
    od:
    j:='j':
```

```
    deltaseq[lind]:=  seq(delta[j], j=floor(utilT*R)..R-1):
    mydelta[lind]:= min(deltaseq[lind]);
    for j from floor(utilT*R) to R-1 do
      if delta[j]=mydelta[lind] then
                          mythresh[lind]:= j; myutil[lind]:= xutil[j];
                        fi:
    od:
# this value is for the reference, we use solutions from the equation
    myT[lind]:= mythresh[lind];
    for l from 1 to T[lind] do
      Pt[l]:= Pth[0]*(lambda/mu)^l /l!
    od:
    l:='l':
    for l from T[lind]+1 to R do
      Pt[l]:=  Pth[0]*(lambda/mu)^myT[lind] /myT[lind]! *
                        product(offload(li,lind, myT[lind]), li=myT[lind]+1..l):
    od:
# we plot the distribution
    for i from 0 to R do
            Point_pt[i]:= [i, Pt[i]]:
    od:
    i:='i':
    seqPt[lind]:= [seq(Point_pt[i], i=0..R)]:
    plotPt[lind]:= plot(seqPt[lind], axes=boxed, style=point,
                symbol=box,symbolsize=16, color=colr(1), labels=[k, ""]):
 # test for normalization must be 1 always;
    l:='l':
    testt[lind] := Pth[0]+add(Pt[l], l=1..R);
    Pb[lind]:= sum(  Pt[ko], ko=T[lind]+1..R )   :
  else
    disc[lind]:=1:
        end if: # utilT
  final_lambda[lind]:=disc[lind]*lambda;
        Point_finlamb[lind]:=[rho[lind], final_lambda[lind]];
  PointPt0[lind]:= [rho[lind], Pth[0]];
  PointPbt[lind]:= [rho[lind], Pb[lind]];
  Point_util[lind]:= [rho[lind], util[lind]/10];
  Point_dis[lind]:= [rho[lind], disc[lind]];
  Point_thrsh[lind]:= [rho[lind], T[lind]];
  Point_mythrsh[lind]:= [rho[lind], myT[lind]];
  Point_test[lind]:= [rho[lind], test[lind]];
  Point_zero[lind]:= [rho[lind], 0];
od;
```

C.2.2 Admission Control Model Based on Current Utilization

```
mu:= 1/2400;
R:=1000:
utilT:=0.75:
lambda:='lambda';
Fcr:='Fcr':
Tf:='Tf':
Tt:='Tt':
offload := proc ( k, ind, T) options operator, arrow;
```

```
        local tmp:
        description "produces offered load for state k in Markov
                         chain";
        tmp:= Fcr * lambda / (mu*k) :
        return(tmp);
end proc:

#### this procedure returns probability state k beyond the threshold
PT := proc ( k, lind, T) options operator, arrow;
        local tmp, li:
               description "produces probability of  state k beyond the
                          threshold";
        tmp:= Pt[0]*(lambda/mu)^T /T!  *
                                product(offload(li,lind, T), li=T+1..k):
              return(tmp);
end proc:

#### this procedure returns color of plot, to be used in plot command
#for a distribution of MC
colr := proc ( k) options operator, arrow;
        local tmp::string:
        description "produces color of plot";
        if k=1 then tmp:="red": elif k=2 then tmp:="blue"; elif k=3
                                then tmp:="brown";
              elif k=4 then tmp:="green"; elif k=5 then tmp:="black": end if:
        return(tmp);
end proc:
for lind from  12 to 28 do
```

```
  print(‘lind=‘, lind);
  lambda := (lind*100)/3600:
  rho[lind]:= lambda/(R*mu):
# Probability that system is empty comes from normalization condition
  P[0]:= 1/ (1 + sum((lambda/mu)^i /i! , i=1..R) ):
  Pp[0][lind]:= P[0]:
#individual state probabilities when we do not have the theshold
  for l from 1 to R do
    P[l]:= P[0]*(lambda/mu)^l /l!
  od:
  l:='l':
# we plot the distribution
  for i from 0 to R do
    Point_p[i]:= [i, P[i]]:
  od:
  i:='i':
  seqP[lind]:= [seq(Point_p[i], i=0..R)]:
  plotP[lind]:= plot(seqP[lind], axes=boxed, style=point,
        symbol=box,symbolsize=16, color=colr(1), labels=[k, ""]):
# test for normalization must be 1 always
  l:='l':
  test[lind] :=        add(P[l], l=0..R);
  Pb[lind]:=   P[R]  :
  myutil[lind]:= sum( P[jj]*jj, jj=0..R):
  myT[lind]:=R:
        if util[lind]/R > utilT then
    Tf:= floor(Tt):
    i:='i':  k:='k':  jj:='jj':  ko:='ko':  ii:='ii':  kl:='kl':
```

```
    Pt[0]:= 1/ (1 + sum(  (lambda/mu)^i /i!   , i=1..Tf )
    + sum( (lambda/mu)^Tt /Tt! * product( offload(i,lind,Tf),
    i=Tf+1..k ), k=Tf +1..R)):
    utiln:= sum( (Pt[0]/jj!)*((lambda/mu)^jj)*jj, jj=0..Tf )
    + sum(  (Pt[0]/ko!)* ((lambda/mu)^Tt)*
    ((offload(ko, lind,Tf) )^(ko-Tt))* ko, ko=Tf +1..R ):
    norml:= sum( (Pt[0]/ii!)*((lambda/mu)^ii), ii=0..Tf)
    + sum(  (Pt[0]/kl!)* ((lambda/mu)^Tt)*
    ((offload(kl, lind,Tf) )^(kl-Tt)), kl=Tf+1..R ):
# we want utilization to be equal or as close to limiting util T
# exact solutions would not work since utilization can not be exactly
# equal to utilT but we need approximations in the form floor(Tt)
    for j from floor(utilT*R) to R-1 do
      i:='i':  k:='k':  jj:='jj':  ko:='ko':  ii:='ii':  kl:='kl':
      xutil[j]:= evalf(subs(Tt=j, utiln));
      normx[j]:=evalf(subs(Tt=j, norml));
      eqn:= normx[j]/R =1:
      sols:='sols':
      sols:= fsolve(eqn, Fcr, Fcr=0..1);
      filt[j]:= subs(Fcr=sols, Fcr);
      delta[j]:= abs(utilT- xutil[j]/R)  :
    od:
    j:='j':
    deltaseq[lind]:=  seq(delta[j], j=floor(utilT*R)..R-1):
    mydelta[lind]:= min(deltaseq[lind]);
    for j from floor(utilT*R) to R-1 do
      if delta[j]=mydelta[lind] then
              mythresh[lind]:= j; myutil[lind]:= xutil[j];
```

```
                              disc[lind]:= filt[j];
                        fi :
    od:
# this value is for the reference, we use solutions from the equation
    myT[lind]:= mythresh[lind];
    for l from 1 to T[lind] do
      Pt[l]:= Pth[0]*(lambda/mu)^l /l!
    od:
    l:='l':
    for l from T[lind]+1 to R do
      Pt[l]:=  Pth[0]*(lambda/mu)^myT[lind] /myT[lind]!*
                  product(offload(li,lind, myT[lind]), li=myT[lind]+1..l) :
    od:
# we plot the distribution
    for i from 0 to R do
            Point_pt[i]:= [i, Pt[i]]:
    od:
    i:='i':
    seqPt[lind]:= [seq(Point_pt[i], i=0..R)]:
    plotPt[lind]:= plot(seqPt[lind], axes=boxed, style=point,
                symbol=box,symbolsize=16,
    color=colr(1),    labels=[k, ""]):
    l:='l':
    testt[lind] := Pth[0]+add(Pt[l], l=1..R);
    Pb[lind]:= sum(  Pt[ko], ko=T[lind]+1..R )   :
  else
    disc[lind]:=1:
        end if: # utilT
  final_lambda[lind]:=disc[lind]*lambda:
        Point_finlamb[lind]:=[rho[lind], final_lambda[lind]];
  PointPt0[lind]:= [rho[lind],  Pth[0]];
  PointPbt[lind]:= [rho[lind],  Pb[lind]];
  Point_util[lind]:= [rho[lind], util[lind]/10];
  Point_dis[lind]:= [rho[lind],  disc[lind]];
  Point_thrsh[lind]:= [rho[lind], T[lind]];
  Point_mythrsh[lind]:= [rho[lind], myT[lind]];
  Point_test[lind]:= [rho[lind],  test[lind]];
  Point_zero[lind]:= [rho[lind],  0];
od;
```

C.3 Resource Allocation Solution for Clouds with Mobile Devices

```
# Algorithm 8 The Integrated model Algorithm
# (The full priority of overflow task Scheme)
# Input: Initial successful provisioning probability
# and overflow rate: Ps0,Oo0
# Output: Blocking probability in the RAM: BPq0
# count = 0; maximum = 30; diff = 1;
Digits:=20:
differc:=1:
maxim:=30:
Lq:=50;
N:=100:
```

```
lambdast:=600/3600:

Ps0:=0.8:

Oo0:= 1/60;

A:= 1:

Ro:=0:

Rn:=0:

mu:=1/2400:

betav:=1/0.1;

etav:=10*betav:

flag:=0:

#Pbq0 <- RAM (Ps0,Oo0)

rho1:=(lambdast+(2*Ps0*betav))/(Oo0+lambdast+(Ps0*betav));

rho2:=(lambdast+Oo0+(Ps0*betav))/(Oo0+lambdast+(Ps0*betav));

rho3:=(lambdast+Oo0+etav)/(lambdast+betav);

rho4:= ((1-Ps0)*betav)/etav;

lastrho3:=(((1-Ps0)*betav)/etav)^Lq)/Lq!;

Gtrunc1:=sum((rho1^k1)/k1!, k1=0..Lq);

k1='k1':

Gtrunc2:=sum((rho2^k1)/k1!, k1=0..Lq);

k1='k1':

Gtrunc3:=sum((rho3^k1)/k1!, k1=0..Lq);

k1='k1':

Gtrunc4:=sum((rho4^k1)/k1!, k1=0..Lq);

k1='k1':

Gtrunc:=Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+lastrho3;

i:='i':

for i from 0 to Lq do

  for j from 0 to Lq do
```

```
   if (i=0) then
     resiv[i][j][A]:= ((rho1^j)/j!)/Gtrunc;
   elif (i>0 and i<Lq) then
     resiv[i][j][A]:= ((rho2^j)/j!)/Gtrunc;
   else
     resiv[i][j][A]:= ((rho3^j)/j!)/Gtrunc;
   end if;
  od;
od;
resiv[0][Lq][Rn]:=lastrho3/Gtrunc;
BPq0:=resiv[0][Lq][Rn]:
j:='j':
for j from 0 to Lq do
  resiv[Lq][j][Ro]:= ((rho4^j)/j!)/Gtrunc;
  BPq0:= BPq0 + resiv[Lq][j][R0];
od;
i:'i': j:='j':
for i from 0 to Lq do
BPq0:= BPq0 + resiv[i][Lq][A];
od;
i:'i':
bphi:=1:
#while diff >= 10^-6 do
Nhvaraverage := 0:
counter:='counter':
for counter from 1 to maxim while differc > 0.000001 do
# [Ps, Oo] <-VMM(BPq0)
  i:'i': j:='j': k:='k': k1:='k1': k2:='k2':
```

```
m:=10: L:=4: d:=1/2400:
lambdai:=lambdast*(1-BPq0)/N:
lambdaci:= 0.3*lambdai:
if (flag=0) then
  cphi:= bphi:
  Oo:= 0.1 * lambdai:
  Oi:= 0.1 * lambdai:
  flag:= 1:
else
  cphi:= 0.5*bphi:
end if:
Gtrunc1:='Grtunc1': Gtrunc2:='Grtunc2': Gtrunc3:='Grtunc3':
Gtrunc4:='Grtunc4': Gtrunc:='Gtrunc':
rho1:=(lambdai+mu+Oo)/(lambdai+Oi+bphi);
rho2:=(lambdai+bphi+Oi+2*mu+Oo)/(lambdai+cphi+Oi+mu+Oo);
rho3:=(lambdai+cphi+d+Oi+3*mu+Oo)/(lambdai+Oi+2*mu+Oo+cphi+L*lambdaci);
rho4:=(lambdai+cphi+d+Oi)/(lambdai+Oi+cphi+L*lambdaci+3*mu+Oo));
Gtrunc1:=sum((rho1^k1)/k1!, k1=0..Lq);
k1='k1':
Gtrunc2:=sum(sum((rho2^k1)/k1!, k1=0..Lq)*((L-k2+1)
      *lambdaci+(k2+1)*d)/((L-k2)*lambdaci+k2*d),k2=0..L);
k1:='k1': k2:='k2':
Gtrunc3:=sum(sum((rho3^k1)/k1!, k1=0..Lq)*((L-k2+1)
      *lambdaci+(k2+1)*d)/((L-k2)*lambdaci+k2*d),k2=0..L);
k1:='k1': k2:='k2':
Gtrunc4:=sum(sum((rho4^k1)/k1!, k1=0..Lq)*((L-k2+1)
      *lambdaci+(k2+1)*d)/((L-k2)*lambdaci+k2*d),k2=0..L);
k1:='k1': k2:='k2':
```

```
Gtrunc:= Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4;

for i from 0 to Lq do

  for j from 0 to 3 do

    if (j=0) then

      vmm[i][j][0]:= ((rho1^j)/j!)/Gtrunc;

    elif (j=1) then

      for k from 0 to L do

        vmm[i][j][k]:= ((rho2^j)/j!)*((L-k+1)

      *(lambdaci+(k+1)*d)/((L-k)*lambdaci+k*d))/Gtrunc;

      od;

    elif (j=2) then

      for k from 0 to L do

        vmm[i][j][k]:= ((rho3^j)/j!)*((L-k+1)

      *(lambdaci+(k+1)*d)/((L-k)*lambdaci+k*d))/Gtrunc;

      od;

    else

      for k from 0 to L do

        vmm[i][j][k]:= ((rho4^j)/j!)*((L-k+1)

      *(lambdaci+(k+1)*d)/((L-k)*lambdaci+k*d))/Gtrunc;

      od;

    end if;

  od;

od;

i:'i': j:='j': k:='k':

capphi:=0:

for i from 0 to Lq do

  for j from 0 to 3 do

    for k from 3 to 5 do
```

```
       capphi := capphi+vmm[i][j][k];
      od;
    od;
  od;
  i:'i': j:='j': k:='k':
  Pna:=vmm[Lq][0][0]+ sum(sum(vmm[Lq][k1][k2], k1=1..3),k2=0..L)+capphi;
  k1:='k1': k2:='k2':
  Ps:=1-(Pna^N);
#Pbq1 <- RAM (Ps,Oo)
  Gtrunc1:='Grtunc1': Gtrunc2:='Grtunc2': Gtrunc3:='Grtunc3':
  Gtrunc4:='Grtunc4':Gtrunc:='Gtrunc':
  rho1:='rho1':
  rho1:=(lambdast+(2*Ps*betav))/(Oo+lambdast+(Ps*betav));
  rho2:=(lambdast+Oo+(Ps*betav))/(Oo+lambdast+(Ps*betav));
  rho3:=(lambdast+Oo+etav)/(lambdast+betav);
  rho4:= ((1-Ps)*betav)/etav;
  lastrho3:=(((1-Ps)*betav)/etav)^Lq)/Lq!;
  Gtrunc1:=sum((rho1^k1)/k1!, k1=0..Lq);
  k1='k1':
  Gtrunc2:=sum((rho2^k1)/k1!, k1=0..Lq);
  k1='k1':
  Gtrunc3:=sum((rho3^k1)/k1!, k1=0..Lq);
  k1='k1':
  Gtrunc4:=sum((rho4^k1)/k1!, k1=0..Lq);
  k1='k1':
  Gtrunc:=Gtrunc1+Gtrunc2+Gtrunc3+Gtrunc4+lastrho3;
  i:='i':
  for i from 0 to Lq do
```

```
    for j from 0 to Lq do
      if (i=0) then
        resiv[i][j][A]:= ((rho1^j)/j!)/Gtrunc;
      elif (i>0 and i<Lq) then
        resiv[i][j][A]:= ((rho2^j)/j!)/Gtrunc;
      else
        resiv[i][j][A]:= ((rho3^j)/j!)/Gtrunc;
      end if;
    od;
  od;
  resiv[0][Lq][Rn]:=lastrho3/Gtrunc;
  BPq1:=resiv[0][Lq][Rn]:
  j:='j':
  for j from 0 to Lq do
    resiv[Lq][j][Ro]:= ((rho4^j)/j!)/Gtrunc;
    BPq1:= BPq1 + resiv[Lq][j][R0];
  od;
  i:'i': j:='j':
  for i from 0 to Lq do
    BPq1:= BPq1 + resiv[i][Lq][A];
  od;
  i:'i':
  for j from 0 to Lq do
    resiv[Lq][j][Ro]:= ((rho4^j)/j!)/Gtrunc;
  od;
  j:='j':
#diff <-|(BPq1 - BPq0)|
  differc := abs(BPq1-BPq0);
```

```
#BPq0 <-BPq1
  BPq0:= BPq1:
  i:='i': j:='j':
  if (counter=maxim) then
    BPq0:= -1;
  end if;
od; #end of main loop
```

Bibliography

[1] B. Abbasov. Cloud Computing: State Of The Art Reseach Issues. In *IEEE 8th International Conference onApplication of Information and Communication Technologies (AICT)*, pages 1–4, Astana, Kazakhstan, October 2014.

[2] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. In *IEEE Network Operations and Management Symposium (NOMS)*, pages 204–212, Maui, HI, USA, April 2012.

[3] M. Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *IEEE INFOCOM*, pages 963–971, Orlando, FL, USA, March 2012.

[4] A. Ashraf, B. Byholm, and I. Porres. A Session-Based Adaptive Admission Control Approach for Virtualized Application Servers. In *Fifth IEEE International Conference on Utility and Cloud Computing (UCC)*, pages 65–72, Chicago, IL, USA, November 2012.

[5] J. Baliga, R. W. A. Jayant, K. Hinton, and R. S. Tucker. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167, August 2011.

[6] D. Bertsekas and R. Gallager. *Data Networks (2nd Edition)*. Prentice Hall, 1991.

[7] R. G. Brown. *Statistical Forecasting for Inventory Control*. McGraw-Hill, New York, 1959.

[8] W. Caraballo and T. G. Robertazzi. Markov Chain Flow Decomposition for a Two Class Priority Queue. In *Conference on Information Sciences and Systems*, page 1, Baltimore, MD, USA, March 2003.

[9] D. Chae, J. Kim, J. Kim, J. Kim, S. Yang, Y. Cho, Y. Kwon, and Y. Paek. CMcloud: Cloud Platform for Cost-Effective Offloading of Mobile Applications. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 434–444, Chicago, IL, USA, May 2014.

[10] F. Chang, J. Ren, and R. Viswanathan. Optimal resource allocation in clouds. In *3rd IEEE International Conference on Cloud Computing (CLOUD)*, pages 418–425, Miami, FL, USA, July 2010.

[11] X. Chen. Decentralized Computation Offloading Game For Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):974–983, March 2015.

[12] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *6th conference on Computer systems (ACM EuroSys)*, pages 301–314, Salzburg, Austria, April 2011.

[13] R. B. Cooper. *Introduction to Queuing theory: Second Edition*. North Hollad, 1981.

[14] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. In *8th International ACM Conference on Mobile*, pages 41–48, Honolulu, HI, USA, June 2012.

[15] D. Dechouniotis, N. Leontiou, N. Athanasopoulos, G. Bitsoris, and S. Denazis. ACRA: A Unified Admission Control and Resource Allocation Framework for Virtualized Environments. In *8th International Conference on Network and Service Management (CNSM) and Workshop on Systems Virtualiztion Management (SVM)*, pages 145–149, Las Vegas, NV, USA, October 2012.

[16] A. Demars, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queuing Algorithm. In *ACM SIGCOMM '89*, pages 3–26, Austin, TX, USA, September 1989.

[17] Z. Feldman, M. Masin, A. N. Tantawi, D. Arroyo, and M. Steinder. Using approximate dynamic programming to optimize admission control in cloud computing environment. In *Winter Simulation Conference 2011 (WSC)*, pages 3153–3164, Phoenix, AZ, USA, December 2011.

[18] S. Floyd and V. Jacobson. Random Early Detection (RED) gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[19] B. Gao, L. He, L. Liu, K. Li, and S. A. Javis. From Mobiles to Clouds: Developing Energy-aware Offloading Strategies for Workflows. In *ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pages 139–146, Beijing, China, September 2012.

[20] L. Guan, Y. Wang, and Y. Li. A Dynamic Resource Allocation Method in IaaS Based

on Deadline Time. In *14th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, Seoul, Korea, September 2012.

[21] L. Guan, M. Song X. Ke, and J. Song. A Survey of Research on Mobile Cloud Computing. In *10th IEEE/ACIS International Conference on Computer and Information Science*, pages 387–392, Sanya, China, May 2011.

[22] M. A. Hassan, K. Bhattarai, Q. Wei, and S. Chen. POMAC: Properly Offloading Mobile Applications to Clouds. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, pages 1–6, Philadelphia, PA, USA, June 2014.

[23] Y. He, J. Huang, Q. Duan, Z. Xiong, J. Lv, and Y. Liu. A Novel Admission Control Model in Cloud Computing. Januaury 2014. Available from `http://arxiv.org/abs/1401.4716v2`. Last visited: September 1, 2014.

[24] D. T. Hoang, D. Niyato, and P. Wang. Optimal Admission Control Policy for Mobile Cloud Computing Hotspot with Cloudlet. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3145–3149, Shanghai, China, April 2012.

[25] T. Huu and C. Tham. An Auction-based Resource Allocation Model for Green Cloud Computing. In *IEEE International Conference on Cloud Engineering*, pages 269–278, Santa Clara, CA, USA, March 2013.

[26] Maplesoft Inc. Maple 15. `http://www.maplesoft.com/products/maple/`. Last visited: July 15, 2015.

[27] MathWorks Inc. Simulink. `http://www.mathworks.com/products/simulink/`. Last visited: September 1, 2014.

[28] P. T. Joy and K. P. Jacob. Cooperative Caching Framework for Mobile Cloud Computing. *Global Journal of Computer Science and Technology Network, Web & Security*, 13(8):1–7, July 2013.

[29] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai. Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1894–1902, Hong Kong, China, April 2015.

[30] N. Kaushik and J. Kumar. A Computation Offloading Framework to Optimize Energy Utilisation in Mobile Cloud Computing Environment. *International Journal of Computer Applications & Information Technology*, 5(2):61–69, April 2014.

[31] A. Khalifa and M. Eltoweissy. Collaborative Autonomic Resource Management System for Mobile Cloud Computing. In *Fourth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING)*, pages 115–121, Valencia, Spain, May 2013.

[32] H. Khazaei, J. Mišić, and V.B. Mišić. A Fine-Grained Performance Model of Cloud Computing Centers. *IEEE Transaction on Parallel and Distributed Systems*, 24(11):2138–2147, November 2013.

[33] H. Khazaei, J. Mišić, and V.B. Mišić. Analysis of a Pool Management Scheme for Cloud Computing Centers. *IEEE Transaction on Parallel and Distributed Systems*, 24(5):849–861, May 2013.

[34] H. Khojasteh, J. Mišić, and V.B. Mišić. Characterizing energy consumption of IaaS

clouds in non-saturated operation. In *IEEE INFOCOM 2014 Workshop on Mobile Cloud Computing*, pages 398–403, Toronto, Canada, April 2014.

[35] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley-Interscience, 1975.

[36] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. Admission Control for Elastic Cloud Services. In *Fifth IEEE International Conference on Cloud Computing (CLOUD)*, pages 41–48, Honolulu, HI, USA, June 2012.

[37] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. Elastic Admission Control for Federated Cloud Services. *IEEE Transactions on Cloud Computing*, 2(3):348–361, July 2014.

[38] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 945–953, Orlando, FL, USA, March 2012.

[39] K. Kumar, J. Feng, Y. Nimmagadda, and Y.-H. Lu. Resource allocation for real-time tasks using cloud computing. In *20th Int. Conf. Computer Communications and Networks (ICCCN 2011)*, pages 1–7, Maui, HI, USA, July 2011.

[40] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *ACM EuroSys*, pages 1–12, Nuremberg, Germany, April 2009.

[41] N. Leontiou, D. Dechouniotis, and S. Denazis. Adaptive Admission Control of Dis-

tributed Cloud Services. In *International Conference on Network and Service Management (CNSM)*, pages 318–321, Niagara Falls, Canada, October 2010.

[42] J. Li, M. Qiu, J. Niu, Y. Chen, and Z. Ming. Adaptive Resource Allocation for Preemptable Jobs in Cloud Systems. In *10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 31–36, Cairo, Egypt, November 2010.

[43] A. Scheller-Wolf M. Harchol-Balter, T. Osogami and A. Wierman. Multi-server queueing systems with multiple priority classes. *Queueing Systems: Theory and Applications*, 31(3):331–360, December 2005.

[44] V. Mainkar and K. S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *IEEE Transactions on Software Engineering*, 22(9):640–653, September 1996.

[45] D. C. Marinescu. *Cloud Computing: Theory and Practice: First Edition*. Morgan Kaufmann, 2013.

[46] P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology, Information Technology Laboratory*, September 2011.

[47] S. P. Mirashe and N. V. Kalyankar. Cloud Computing. *Journal of Computing*, 2(3):78–82, March 2010.

[48] Y. Niu, b. Luo, F. Liu, J. Liu, and B. Li. When Hybrid Cloud Meets Flash Crowd: Towards Cost-Effective Service Provisioning. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1044–1052, Hong Kong, China, April 2015.

[49] K. Al Nuaimi, N. Mohamed, M. Al Nuaimi, and J. Al Jaroodi. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. In *IEEE Second Symposium on Network Cloud Computing and Applications (NCCA)*, pages 137–142, London, UK, December 2012.

[50] A. C. Olteanu and N. Tapus. Offloading for mobile devices: A survey. *University Politehnica of Bucharest (UPB) Scientific Bulletin*, 76(1):1–14, XX 2014.

[51] M. J. O'Sullivan and D. Grigoras. Integrating mobile and cloud resources management using the cloud personal assistant. *Simulation Modelling Practice and Theory: Special Issue on Resource Management in Mobile Clouds*, 50:20–41, January 2015.

[52] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain. Cloud Computing Features, Issues and Challenges: A Big Picture. In *International Conference on Computational Intelligence & Networks (CINE)*, pages 116–123, Odisha, India, January 2015.

[53] G. Rastogi and R. Sushil. Cloud Computing Implementation: Key Issues and Solutions. In *2nd International Conference on Computing for Sustainable Global Development*, pages 320–324, New Delhi, India, March 2015.

[54] P. Salot. A Survey of Various Scheduling Algorithm in Cloud Computing Environment. *International Journal of Research in Engineering and Technology*, 2(2):131–135, February 2013.

[55] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, January 2014.

[56] S. T. Selvi, C. Valliyammai, and V. N. Dhatchayani. Resource Allocation Issues and Challenges in Cloud Computing. In *2014 International Conference on Recent Trends in Information Technology*, pages 1–6, Chennai, India, April 2014.

[57] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. COSMOS: computation offloading as a service for mobile devices. In *15th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, pages 287–296, Philadelphia, PA, USA, August 2014.

[58] M. Shiraz, S. Abolfazli, Z. Sanaei, and A. Gan. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, 63(3):946–964, March 2013.

[59] M. Shiraz and A. Gani. Mobile Cloud Computing: Critical Analysis of Application Deployment in Virtual Machines. *International Conference on Information and Computer Networks (ICICN)*, 27:11–16, February 2012.

[60] R. Steele. *Delta Modulation Systems*. Pentech Press, London, 1975.

[61] H. Takagi. *Queueing Analysis, volume 1: Vacation and Priority Systems*. North-Holland, 1991.

[62] A. S. Tam, D. Chiu, J. C. S. Lui, and Y. C. Tay. A Case for TCP-Friendly Admission Control. In *14th IEEE International Workshop on Quality of Service (IWQoS)*, pages 229–238, New Haven, CT, USA, June 2006.

[63] D. Tammaro, E. Doumith, S. Zahr, J. Smets, and M. Gagnaire. Dynamic Resource Allocation in Cloud Environment Under Time-variant Job Requests. In *Third IEEE*

International Conference on Cloud Computing Technology and Science, pages 592–598, Athens, Greece, November 2011.

[64] L. Tomas and J. Tordsson. Improving Cloud Infrastructure Utilization through Overbooking. In *ACM Cloud and Autonomic Computing Conference (CAC)*, pages 1–10, Miami, FL, USA, August 2013.

[65] T. Tomita and S. Kuribayashi. Congestion control method with fair resource allocation for cloud computing environments. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pages 1–6, Victoria, Canada, August 2011.

[66] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely. Dynamic Resource Allocation and Power Management in Virtualized Data Centers. In *IEEE Network Operations and Management Symposium (NOMS)*, pages 479–486, Osaka, Japan, April 2010.

[67] S. Vakilinia, D. Qiu, and M. Mehmet Ali. Optimal multi-dimensional dynamic resource allocation in mobile cloud computing. *EURASIP Journal on Wireless Communications and Networking*, 201:1–14, November 2014.

[68] D. Warneke and O. Kao. Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):985–997, June 2011.

[69] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li. Ready, Set, Go: Coalesced Offloading from

Mobile Devices to the Cloud. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 2373–2381, Toronto, Canada, April 2014.

[70] L. Xu, Z. Zeng, and X. Y. Multi-objective Optimization Based Virtual Resource Allocation Strategy for Cloud Computing. In *IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 56–61, Shanghai, China, May 2012.

[71] H. Yuan, C. C. J. Kuo, and I. Ahmad. Energy efficiency in data centers and cloud-based multimedia services: An overview and future directions. In *IEEE Int. Green Computing Conf.*, pages 375–382, Chicago, IL, USA, August 2010.

[72] D. Zarchy, D. Hay, and M. Schapira. Capturing Resource Tradeoffs in Fair Multi-Resource Allocation. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1062–1070, Hong Kong, China, April 2015.

[73] Q .Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, April 2010.

[74] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau. A Truthful $(1 - \epsilon)$-Optimal Mechanism for On-demand Cloud Resource Provisioning. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1053–1061, Hong Kong, China, April 2015.

[75] J. Zhao, X. Chu, H. Liu, Y. Leung, and Z. Li. Online Procurement Auctions for Resource Pooling in Client-Assisted Cloud Storage Systems. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 576–584, Hong Kong, China, April 2015.