Modeling and Optimal resource allocation for IaaS based Cloud Datacenter

A THESIS

submitted by

MANISH GHISULAL HEDA

in partial fulfilment of the requirements

for the award of the degree

of

Master of Technology

in

Electrical Engineering

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2015

THESIS CERTIFICATE

This is to certify that the thesis titled **MODELING OF IAAS BASED CLOUD**, submitted by **MANISH GHISULAL HEDA**, to the Indian Institute of Technology Madras, Chennai for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Research Guide
Dr. Ramakrishna Pasumarthy
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place : Chennai
Date : 26th May 2015

ACKNOWLEDGEMENTS

I would like to express my deep sense of gratefulness towards Dr. Ramakrishna Pasumarthy for giving me the opportunity to work on a project of my interest and for his understanding, patience and constant encouragement and guidance, which was very essential in completing this work. I am thankful to Mr. Sai Krishna who was always there to clear my doubts and kept pushing me to achieve bigger goals.

I am also thankful to Dr. Shanti Swaroop, Dr. Nitin Chandrachoodan, Dr. Arunkumar Mahindrakar, Dr. Venkatesh Ramaiyan and the entire faculty and staff of Department of Electrical Engineering for constantly helping me develop and progress throughout my past five years at IIT Madras.

I would also like to thank my friends Mr. Arejeet Nag, Ms. Niharika Challapalli, Mr. Shubham Agarwal for always being there to help me at every stage of my project.

I am also grateful to my parents and my brothers for their support and patience.

ABSTRACT

KEYWORDS: Data Centers, analytic modeling, Continuous Time Markov Chain(CTMC).

Performance and usability of Cloud services are in general stochastic in nature and they are affected by many factors such as location of Data Centers, number of physical machines in Data Centers, Virtual Machine(VM) characteristics, nature of workload, infrastructure characteristics and management policies. Due to this, developing a scalable analytic model for Cloud becomes difficult. Data Centers, pricing structure and percentage usage of Data centers and Physical Machines are usually not taken into account while modeling resource allocation in an IaaS based Cloud. We have used these factors in modeling the Cloud in this thesis.

In this thesis, we have tried to understand the complexity of the cloud computing systems by modeling the Infrastructure-as-a-Service(IAAS) based Cloud. We have built an analytical model for Cloud using Continuous Time Markov Chain(CTMC) modeling. The model is built as two interacting sub-models for simplification and better understanding. We also have simulated the model to understand the working of the cloud more clearly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology Madras
OS	Operating System
NIST	National Institute of Standards and Technology
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
EC2	Eucalyptus
DC	Data Center
VM	Virtual Machine
KVM	Kernel-based Virtual Machine
HVM	Hardware Virtual Machine
PVM	ParaVirtual Machine
SLA	Service Level Agreement
CTMC	Continuous Time Markov Chain
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
RPDE	Resource Provision Decision Engine
FCFS	First Come First Serve
RAM	Random Access Memory

NOTATIONS

λ	Input rate of job requests
β	Rate of allocation of VM
δ	Rate of finding a PM in a given pool
μ	Service time of a PM
π	Steady state probability of a given state

CHAPTER 1
INTRODUCTION

Cloud computing is mostly based on sharing of resources in order to reduces the wastage of resources on the consumer side and also reduce the usage of power and economy. It is a model of network based computing where the consumers submit their requests for computing resources such as operating systems(OS), software packages, storage, programming environment etc. Once the requests are evaluated and provisioned to the users, the users can use these computing resources without having any knowledge of the resource locations and execution environment.

1.1. Definition for Cloud Computing

National Institute of Standards and Technology (NIST) provides the following definition for cloud computing [1] :

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.

1.2. Properties of Cloud Services

Cloud computing is a computing model where various computing resources such as infrastructure, platforms, software are made available to the users across the globe as services over internet. The Cloud services are different from the traditional hosting mainly in four aspects.

1. It is an on-demand service.
2. It is highly elastic since the resources used by the client can vary at any given time and in general the pricing is modeled on pay-per-use basis, so users have to pay for the resources they use at a given time.
3. It enables the users to access the systems using the internet regardless of their location.

4. The service is fully managed and maintained by the provider. There are different types of services that the supplier provides. One can use the service suitable for ones needs.

1.3. Major Cloud Services

The three main services provided by the cloud computing are:

1 Infrastructure as a Service (IaaS),
2 Platform as a Service (PaaS),
3 Software as a Service (SaaS)

In IaaS, physical or virtual machines and other resources are provided to the consumers on-demand from the large pools installed in the data centers. Examples include OpenStack, Amazon EC2 [4].

In the PaaS models, the consumers are provided with the computing platform, which typically includes operating system, database, web server, development tools. Consumers can create softwares using the tools available through the platform. Example of PaaS is Microsoft Azure.

In SaaS, the consumers are provided access to the applications, softwares and databases, while the providers manage the infrastructure and the platform. Example of SaaS is Gmail.

1.4. Components of an Open Source Cloud

An Open Source Cloud comprises of the following components [3], [14]:

1. Hardware and the Operating System which resides over a physical machine.
2. Hypervisor for example Xen, KVM. It provides a platform to initiate multiple Virtual Machines(VMs) on the same physical machine [12].
3. VM disk images.

4. Networking (includes DHCP and DNS)
5. Interface (for resource configuration and VM allocation)
6. Cloud Framework (like OpenStack, Eucalyptus)

The rest of the thesis is organized as follows. In Chapter 2, we give a generic end-to-end model to IaaS based cloud. In Chapter 3, we model the first sub model of IaaS based Cloud, the Resource Provision Decision Engine(RPDE) using Continuous Time Markov Chains(CTMC) approach, write equations to find the probable time spent in each state of CTMC and hence can be used to compute the delay cause by RPDE. We also discuss the optimizer which can be used to improvised decision making for selecting a PM for the job allocation. In Chapter 4, we model the provisioning of VMs for a job and the service execution after a PM is selected for the particular job. In Chapter 5, we simulate the end-to-end model of IaaS based cloud and present our results. In Chapter 6, we conclude the thesis.

CHAPTER 2
MODELING OF IaaS BASED CLOUD

On receiving a request, an IaaS Cloud provisions on-demand Operating System instance with computational resources in the form of Virtual Machines on one or multiple Physical Machines which are deployed in Datacenter's provided by the supplier. Performance and dependability of such cloud based services are very critical [7]. Providers of IaaS cloud offer Service Level Agreements(SLA) to the clients as a guarantee for their services which are not to be violated. The performance of cloud however depends on various factors such as Physical Machine characteristics(cores, RAM, memory,etc), hypervisor, request types and their rate, location of DataCentres, management of resources by the provider etc. [2]. Hence, systematic assessment of cloud properties is difficult and non-trivial. In this thesis, we have tried to model the IaaS based cloud, which is scalable, so that we get better insight to the structure of the cloud and its functioning and hence one can assess the performance and other properties of the cloud [10].

We have used stochastic approach to model the IaaS cloud. Since the cloud architecture is complicated, we have tried to break the model into submodels, model different parts of the cloud using Continuous Time Markov Chain(CTMC) and combine it altogether to explain the cloud architecture [8].

In the rest of this thesis, first, a general end to end model of IaaS cloud is discussed, then we model various parts of the general model using CTMC approach, quantifying the effects of various parameters like request rate, service rate, location of DataCentres, number of PMs, number of VMs per PM, etc. Then we simulate the end-to-end model and conclude.

2.1. General end to end model of IaaS Cloud

When a job is processed, an instance of OS is deployed on a VM. The VM can be customised by the user giving them access to deciding the size of memory, cores, RAM, etc. These VMs are deployed on a PM which can host multiple VMs. These PMs reside in DataCentres which are managed and maintained by the provider. Each stage in deployment of a VM on a PM causes a delay which can be optimized by various techniques. This optimisation can be done by splitting PMs at each datacentre into three pools namely, hot, warm and cold pool. PMs in hot pool are turned on and are ready to use; in warm pool are turned on but in standby mode; in cold pool are turned off.

Deploying a VM image on a PM of hot pool causes minimum delay, while it requires more time to provision a VM on PM of warm pool as it takes time for the PM to be ready to use for deploying the VM. PMs in cold pool need to be turned on to use and hence causes maximum delay. But this arrangement helps in reducing resource utilisation and power consumption.

2.1.1. Factors considered to decide best PM for a job

There are many factors that are considered before selecting the PM and not just selecting the PM based on the minimum delay time. The metrics used for deciding the best PM for a particular job are :

1. Start up time in the hot, warm and cold pools.
2. Distance of the datacentre from the location of origin of request.
3. Pricing of PMs in datacentre
4. Percent utilisation of PM at a given time due to ongoing requests

It is possible that a PM from hot pool is available but is too far compared to available PM from warm pool from a closer datacentre. In this case the PM from hot pool will cause lesser startup delay but can take much more time to complete the request life cycle. Hence, a warm pool PM can at times be better than hot pool PM. Pricing of PMs vary with datacentres. There are consumers who give priority to pricing over response time. Overloading a PM with multiple requests which slows down the response time is also not

desirable. Hence a proper balance between the above factors depending on the user preferences is necessary for selecting a suitable PM for the job.

The optimal size of these pools can be different across different datacentres and can be determined based on number of jobs a datacentre at a certain location is expected to receive at any given time. Forecasting techniques can be used to determine the size of such pools. This thesis does not concentrate on the size of such pools.

Here, it is assumed that all the PMs and all the VMs are identical, each job requests for one VM.

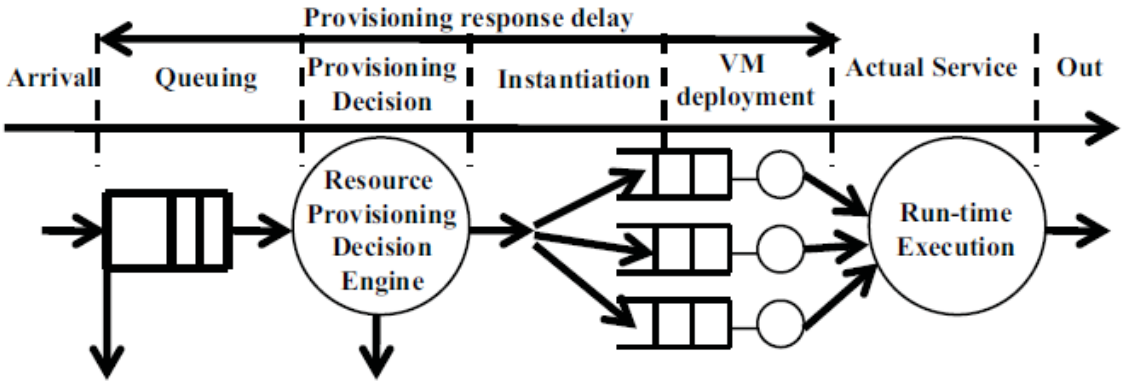

Figure 2.1 : End-to-end model of IaaS based Cloud

The above figure 2.1 shows the generic model of the IaaS based cloud for how the request preceeds through the architecture till it is resolved and results are provided back to the user [11].

Requests when submitted passed through a first come first serve(FCFS) queue to a Resource Provisioning Decision Engine(RPDE). It is assumed that no requests come simultaneously. There is a limit on the number of requests the queue can have at a given time. It rejects any new request if the queue is full. The RPDE takes one request at a time and determines the best possible PM available for it across the datacentres that which is found by optimising the metrics discussed in part **2.1.1.** It is possible that a PM from a warm or cold pool is selected though there is availability of PMs in hot pool in different datacentre. But within a datacentre a PM from hot pool is always given priority over that

from warm pool and similarly PM from warm pool is given over that from cold pool. The algorithm that we have used is such designed that it looks at the K nearest datacentres from the location of origin of request. If no PM is available in any of these datacentres that can satisfy the user requirements then the request is rejected.

Once the PM is selected, the next part is assigning and instantiating a VM in the selected PM. It has been assumed that all the VMs are homogenous that is, they have same storage, cores and RAM. Only one VM can be instantiated at a given time. Hence, in case of multiple requests for VMs in the same PM, the request are stalled in a FCFS queue. Once the VM is instantiated, it is allocated to the user to attend to its job. Once the job is completed, the VM is released and it is available again to be used to attend to another job request.

The entire model has been broken into two sub models.

1. Resource Provisioning Decision Engine(RPDE)
2. VM instantiation and deployment and run time execution of the job

Each of these models are discussed in detail in the next 2 chapters of the thesis.

CHAPTER 3

RESOURCE PROVISION DECISION ENGINE

3.1. Major factors affecting PM selection

The function of RPDE is to allocate the best possible PM for the job. The factors affecting the selection of the best PM are :

1. Start up time in the hot, warm and cold pools.
2. Distance of the datacentre from the location of origin of request.
3. Pricing of PMs in datacentre [6]
4. Percent utilisation of PM at a given time due to ongoing requests

CTMC and queueing theory is used to model RPDE [5]. The model for RPDE is shown in two parts for simplicity. Model showed in figure 4.1 portrays how a PM is found across multiple datacentres for a single job request. Figure 4.2 shows how queue of requests are resolved across multiple datacentres.

3.2. CTMC Model of RPDE

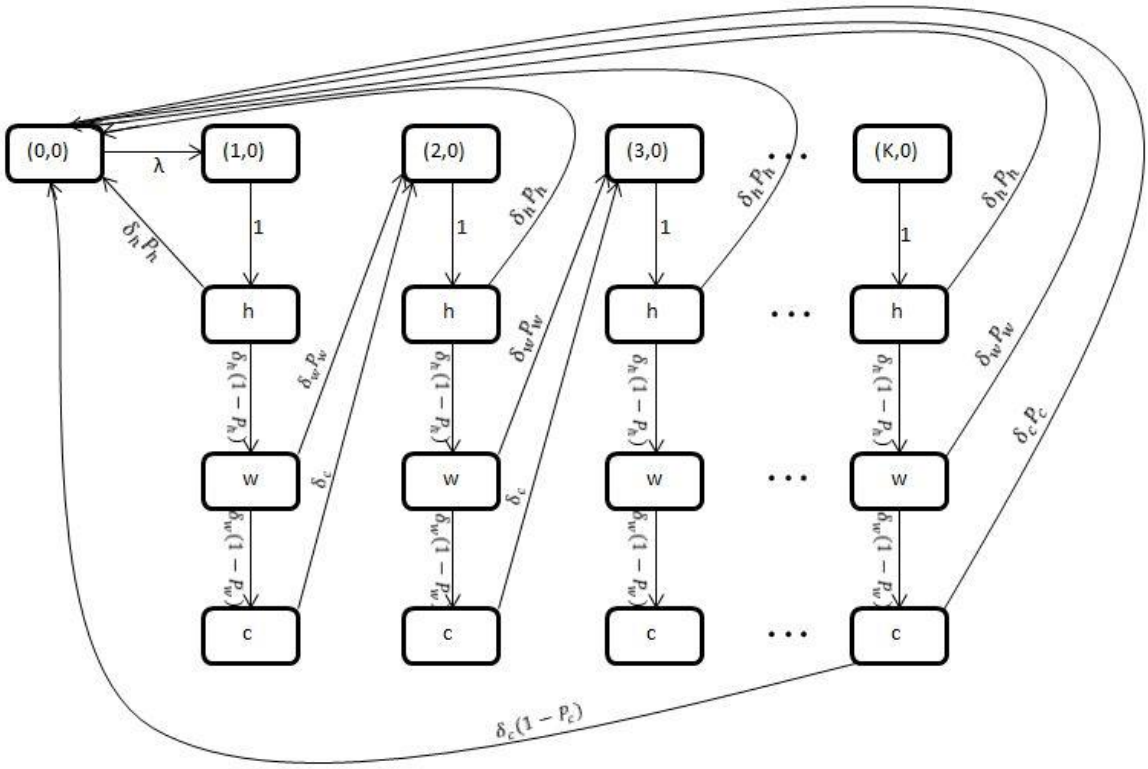

Figure 3.1 : Allocation of PM across multiple DCs for one job request

In figure 3.1, all the states in first row are labeled as *(k,n)* where *k* denotes the k^{th} closest datacentre to the location of origin of the request, and *n* denotes the number of jobs waiting in the queue. Below each of these states, there are three states labeled *'h', 'w and 'c'*. These denote the PMs in hot, warm and cold pool respectively for the given datacentre.

Figure 3.2 also follows the same notation of (k,n). But here, state (k,n) denotes the complete datacentre k that is, it denotes the hot, warm and cold pool of the k^{th} datacentre.

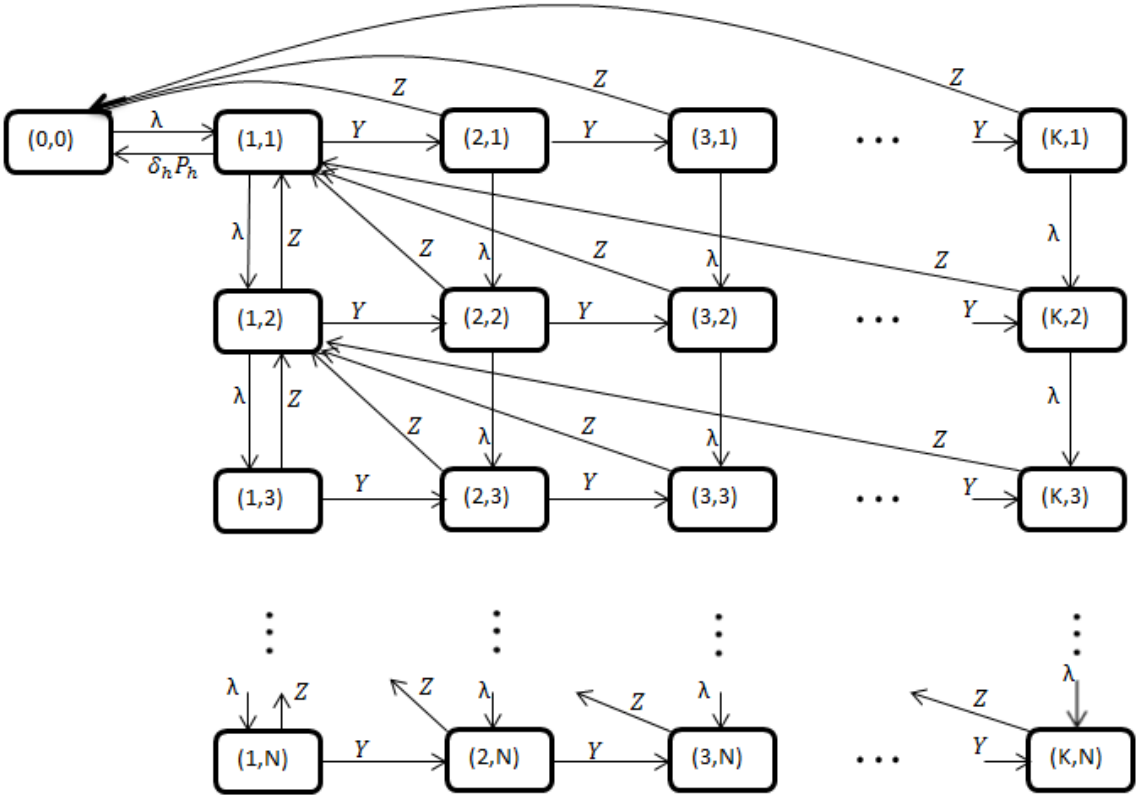

Figure 3.2 : Resolution of multiple job requests across multiple DCs to allocate best possible PMs.

3.3. Input Parameters

Input parameters used are

λ – it denotes mean job arrival rate, where job arrival is assumed to follow poisson function.

$\delta_h, \delta_w, \delta_c$ – rate of finding a PM in hot, warm and cold pool respectively

P_h, P_w, P_c – Probability that the PM of hot/warm/cold pool can accept the job

N – maximum number of jobs allowed in the buffer

K – maximum number of datacentres that the algorithm will look for to find the PM

Y and *Z* – they denote the response rate which is function of the hot, warm and cold states of the corresponding DC and it can be derived by solving the model 4.1. But these

are redundant variables and the complete model can be solved first finding these variables.

P_h, P_w, P_c are the probabilities that a PM in the corresponding pool satisfies the metrics and is eligible to accept the job request and are not the probabilities that the PM is allocated to cater to the request. N and K can be determined by the provider, while λ, $\delta_h, \delta_w, \delta_c$ can be forecasted or estimated for the PMs.

3.4. Working of RPDE

In order to find the K nearest datacentres, bucket sort can be used to identify datacentres in the vicinity of location of request. Then for the nearby datacentres, one can bubble up the nearest datacentre(partly carrying out bubble sort), identify a PM that can cater to the request. If there is a need to find PM from other datacentre that can be better than the current one, then bubble up the next nearest datacentre and follow the same procedure. This model considers the pricing over different datacentres to be constant. Even if pricing varies across different DCs, one can sort on basis of price also as a factor and then carry out the same procedure as explained above.

As shown in figure 3.1, when a request arrives, the algorithm first finds the nearby datacentres to the location of origin of request using bucket sort. Then among the nearby datacentres, it finds the nearest datacentre by bubbling up the nearest one. Now it looks for availability of PM in the hot pool of this datacentre. The mean lookup time is given by $1/\delta_h$, mean probability to find a PM is given by P_h. High priority is given to finding PM in the hot pool hence if it finds a PM in the hot pool of the nearest datacentre, it will allocate the PM for the job. Else if no PM isavailable to cater to the needs of the user, then it will look into the warm pool for PM. Two things can occur in this state,

(i) if it doesn't find a PM in this pool, it will move to the cold pool or
(ii) if it finds the PM in the warm pool then it will generate a metric for the PM considering the factors discussed in part 3.1 , such that this metric can later be used to to compare with other such eligible PMs to identify the best PM available for the job. It

stores the metric along with the other useful details of the PM(like location of datacentre, identity of the PM, etc). Then it will move on to finding the next closet datacentre to find such eligible PM. The next closest datacentre is found by bubbling it up over the nearby datacentres.

When it looks for PM in the cold pool, then no matter if it finds a PM or not, it will move on to the next closest datacentre. If it finds a PM, it will generate a metric and store the details of PM before moving further.

The same procedure is followed over the remaining datacentres also. Whenever it finds a PM in hot pool for any datacentre, it will compare the stored metrics of the eligible PMs that it found for all the nearer datacentres and it finds the best possible PM for the job and this PM is allocated for the job by sending the details of this PM to the next stage where a VM is allocated and instantiated.

If the algorithm doesn't find any PM for the job over the K datacentres then the user request is rejected.

Figure 3.2 explains the queuing in the buffer when a new request comes in while a request is already being carried out. The notation (k,n) denotes k^{th} datacentre and n requests in the buffer. When the system receives its first request, it goes to state (1,1) which is closest datacentre to origin of request and it keeps moving further to states (2,1), (3,1), … untill it finds the optimal PM for the job. Suppose the algorithm was in state (k,1) and during the process it receives another request, the it will put this request on hold in the buffer, which is, it will move to state (k,2). State (k,n) implies that the system is attending to a request and the are *n* requests in the buffer which needs to be attended. While in state (k,n), suppose the system finds the right PM for the current request, then it moves to state (1,n-1) which is it starts attending to the request at the head of the FCFS based queue and there are *n-1* requests in the buffer. The maximum size of the buffer is N, so any more requests while the are N in the buffer are rejected.

3.5. Mathematical Formulation of the Model

We try solving both the given Markov Chain models simultaneously to find the steady state probability of each state which is denoted by $\pi_{(k,n,j)}$, where *k* is the k^{th} datacentre, *n* is the number of pending requests in the buffer and *j* denotes the current pool the system is in and it takes values *h ,w, c* corresponding to hot, warm and cold pools respectively. These probabilities shows the percentage distribution of time the system spends in each state. This information can be used to find the total delay caused by the system per request and also gives insight to what is the optimal number of datacentres should the provider look into to determine the best possible PM such that the rate of rejection is under control and the quality of service is improved.

On solving the above two models simultaneously, we get the following equations :

For the state (0,0),

$$\lambda\pi_{(0,0)} = \delta_h P_h\left(\pi_{(1,1,h)} + \pi_{(2,1,h)} + \cdots + \pi_{(K,1,h)}\right) + \delta_w P_w \pi_{(K,1,w)} + \delta_c \pi_{(K,1,c)}$$

For state (1,1), equations for the hot, warm and cold pool respectively are :

$$(\lambda + \delta_h)\pi_{(1,1,h)} = \lambda\pi_{(0,0)} + \delta_h P_h\left(\pi_{(1,2,h)} + \pi_{(2,2,h)} + \cdots + \pi_{(K,2,h)}\right) + \delta_w P_w \pi_{(K,2,w)} + \delta_c \pi_{(K,2,c)}$$

$$(\lambda + \delta_w)\pi_{(1,1,w)} = \delta_h(1 - P_h)\pi_{(1,1,h)}$$

$$(\lambda + \delta_c)\pi_{(1,1,c)} = \delta_w(1 - P_w)\pi_{(1,1,w)}$$

For states (k,1), where *k* ranges from 2 to K , equations for hot, warm and cold pools are :

$$(\lambda + \delta_h)\pi_{(k,1,h)} = \delta_w P_w \pi_{(1,1,w)} + \delta_c \pi_{(k-1,1,c)}$$

$$(\lambda + \delta_w)\pi_{(k,1,w)} = \delta_h(1 - P_h)\pi_{(k,1,h)}$$

$$(\lambda + \delta_c)\pi_{(k,1,c)} = \delta_w(1 - P_w)\pi_{(k,1,w)}$$

For states (*1*,*n*), where *n* ranges from 2 to *N-1* :

$$(\lambda + \delta_h)\pi_{(1,n,h)} = \lambda\pi_{(1,n-1,h)} + \delta_h P_h\left(\pi_{(1,n+1,h)} + \pi_{(2,n+1,h)} + \cdots + \pi_{(K,n+1,h)}\right) + \delta_w P_w \pi_{(K,n+1,w)} + \delta_c \pi_{(K,n+1,c)}$$

$$(\lambda + \delta_w)\pi_{(1,n,w)} = \delta_h(1 - P_h)\pi_{(1,n,h)} + \lambda\pi_{(1,n-1,w)}$$

$$(\lambda + \delta_c)\pi_{(1,n,c)} = \delta_w(1 - P_w)\pi_{(1,n,w)} + \lambda\pi_{(1,n-1,c)}$$

For states (*k*,*n*) where *n* ranges from 2 to N-1 and k ranges from 2 to K :

$$(\lambda + \delta_h)\pi_{(k,n,h)} = \delta_w P_w \pi_{(k-1,n,w)} + \delta_c \pi_{(k-1,n,c)} + \lambda\pi_{(k,n-1,h)}$$

$$(\lambda + \delta_w)\pi_{(k,n,w)} = \delta_h(1 - P_h)\pi_{(k,n,h)} + \lambda\pi_{(k,n-1,w)}$$

$$(\lambda + \delta_c)\pi_{(k,n,c)} = \delta_w(1 - P_w)\pi_{(k,n,w)} + \lambda\pi_{(k,n-1,c)}$$

For state (1,*N*) :

$$(\lambda + \delta_h)\pi_{(1,N,h)} = \lambda\pi_{(1,N-1,h)}$$

$$(\lambda + \delta_w)\pi_{(1,N,w)} = \delta_h(1 - P_h)\pi_{(1,N,h)} + \lambda\pi_{(1,N-1,w)}$$

$$(\lambda + \delta_c)\pi_{(1,N,c)} = \delta_w(1 - P_w)\pi_{(1,N,w)} + \lambda\pi_{(1,N-1,c)}$$

For states (k,N), where k ranges from 2 to K :

$$(\lambda + \delta_h)\pi_{(k,N,h)} = \delta_w P_w \pi_{(k-1,N,w)} + \delta_c \pi_{(k-1,N,c)} + \lambda\pi_{(k,N-1,h)}$$

$$(\lambda + \delta_w)\pi_{(k,N,w)} = \delta_h(1 - P_h)\pi_{(k,N,h)} + \lambda\pi_{(k,N-1,w)}$$

$$(\lambda + \delta_c)\pi_{(k,N,c)} = \delta_w(1 - P_w)\pi_{(k,N,w)} + \lambda\pi_{(k,N-1,c)}$$

Also we know that,

$$\pi_{(0,0)} + \sum_{k=1}^{K} \sum_{n=1}^{N} \sum_{j=\{h,w,c\}} \pi_{(k,n,j)} = 1$$

In all, the total number of equations are : $\boldsymbol{3NK + 2.}$

3.6. Simulation

A code has been written to solve the above equations and hence to get probability of each state for given input parameters. We have plotted results for the parameters given in table 3.1.

Table 3.1 : Parameters used to simulate RPDE

Description	**Value**
Look up rate in each pool	$\delta_h = 0.5, \delta_w = 0.5, \delta_c = 0.5$ per second
Probability of accepting job	$P_h = 0.5, P_w = 0.6, P_c = 0.8$
Rate of job requests	1 per second
Maximum size of buffer	N = 10
Number of nearest DCs looked to find optimal PM	K = 6

Figure 3.1 shows the percentage time spent in the nearest DC looking for an optimal PM to allocate the job, when there are certain number of requests already in the queue. We have plotted for all the 3 pools. If we add the percentages for a give pool across all the number of requests in the pool, we get the percentage time spent in the given pool of the nearest DC (nearest DC depends on the location of origin of the request).

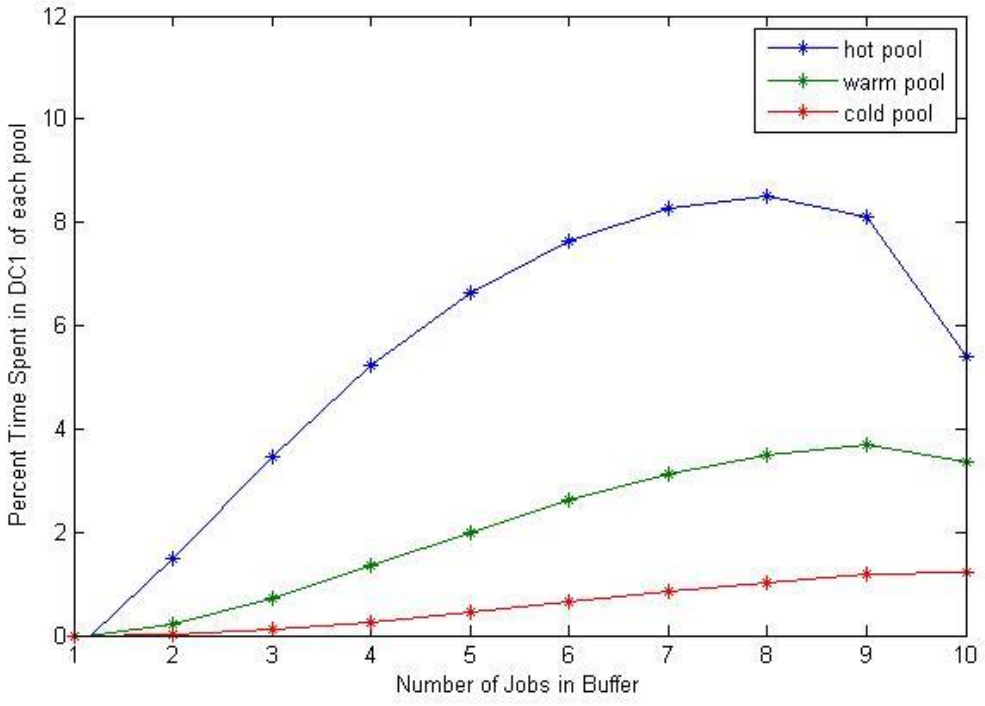

Figure 3.1 : Percentage time spent in nearest DC when given number of jobs are in buffer which needs attention for each type of pool.

Figure 3.2 shows the percent time spent in DCs in the order of distance of it from the location of origin of request in all the pools together. It shows that maximum time is spents in the nearest DC followed by the second nearest DC. The contribution of the rest of DCs is pretty low for the parameters considered for this simulation. Similarly, depending on the forecast of the load, the provider can decide the optimal number of DCs to look at for desired level of consumer satisfaction and low job rejection rate.

Figure 3.3 shows the percent time spent in each of hot, warm and cold pools respectively over all the DCs no matter how many requests are pending to be attended.

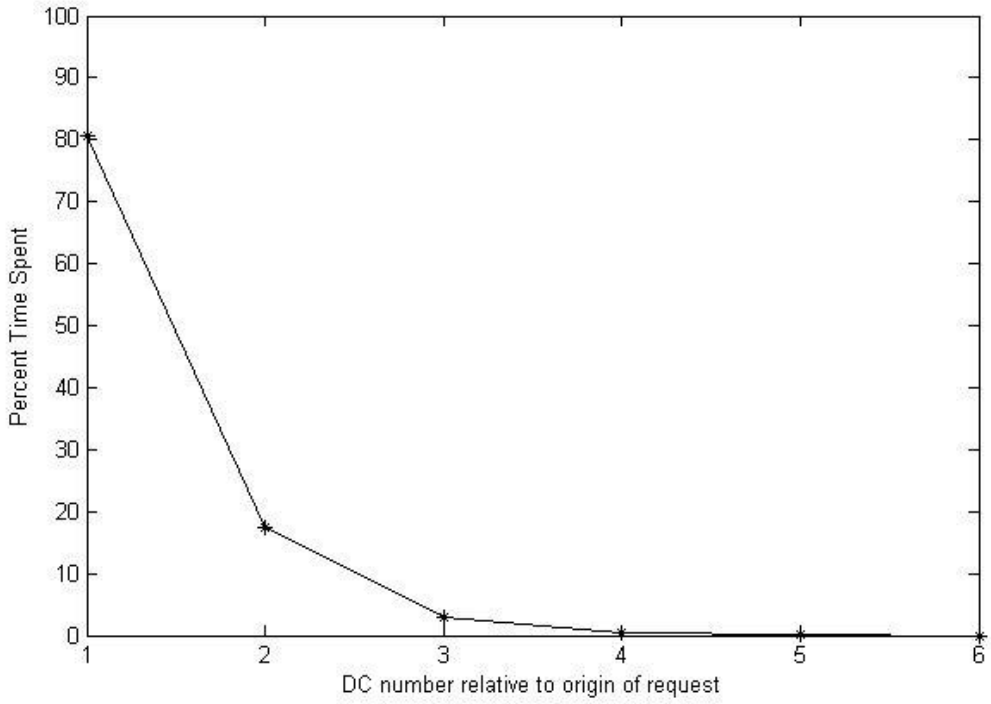

Figure 3.2 : Average percentage of time spent in each of nearest 6 DCs for given job request

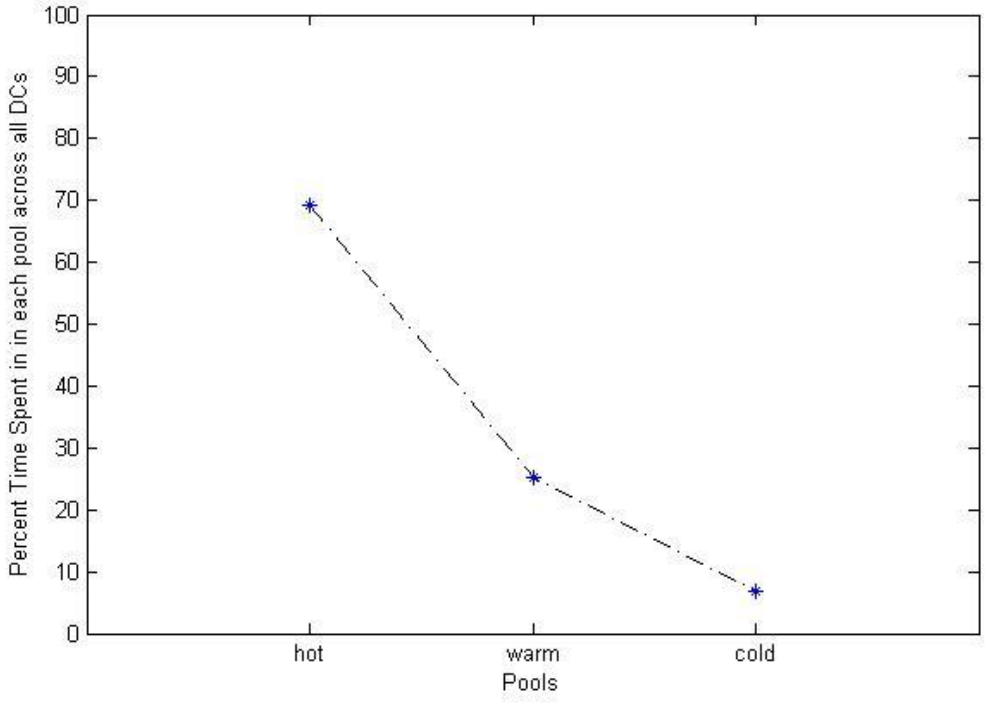

Figure 3.3 : Percent time spent looking for optimal PM in each type of pool.

CHAPTER 4
PROVISIONING OF VMs

4.1. Allocation of VM for given job request

Once an appropriate PM is selected to carry out the job request, the next task is to instantiate and allocate VM/VMs for the job. Its includes configuration of VM, its instantiation and finally allocation of the VM to the user to resolve the job and provide the desirable outputs to the user. The time taken for instantiation of a VM is different for the physical machines belonging to different pools. Instantiation of VM is fastest in the hot pool, followed by warm and cold pool respectively.

We model the hot pool for VM allocation for a given PM using CTMC modeling and see what happens when there are multiple requests to the same PM for VMs.

The notation used in figure 4.3 is of the form (n,m), where n denotes the number of job requests in the buffer, while m denotes the m^{th} VM being utilized for the job. The maximum number of VMs that a PM can hold can be determined by the configurations of the PM and each VM individually. Since it is assumed that all the VMs and PMs are homogeneous, so the configurations of each VM is same and so is the configuration of each PM. Configuration of VM and PM include the available RAM, cores and storage.

Assuming R_p, C_p and S_p denote RAM, cores and storage of a PM respectively and similarly R_v, C_v and S_v denote RAM, cores and storage of a VM respectively. Then the maximum number of VMs, a PM can accommodate is given by

$$m = \min\{floor(\frac{R_p}{R_v}),\ \ floor(\frac{C_p}{C_v}),\ \ floor(\frac{S_p}{S_v})\}$$

The parameters used in the model are as follows :

λ_h, λ_w, λ_c - denote the job arrival rate for PMs in hot pool, warm pool and cold pool respectively. Where,

$$\lambda_h = P_h \times \lambda$$

$\lambda_w = P_w \times \lambda$

$\lambda_c = P_c \times \lambda$

Here, P_h, P_w, P_c are the probabilities of hot pool, warm pool and cold pool getting selected for VM allocation. Note that,

$P_h + P_w + P_c + P_{rejection} = 1$

Where, $P_{rejection}$ is the probability of job rejection.

β – denotes the rate at which the VM is allocated, which includes VM configuration, instantiation and provisioning.

μ - denotes the rate of completion of job.

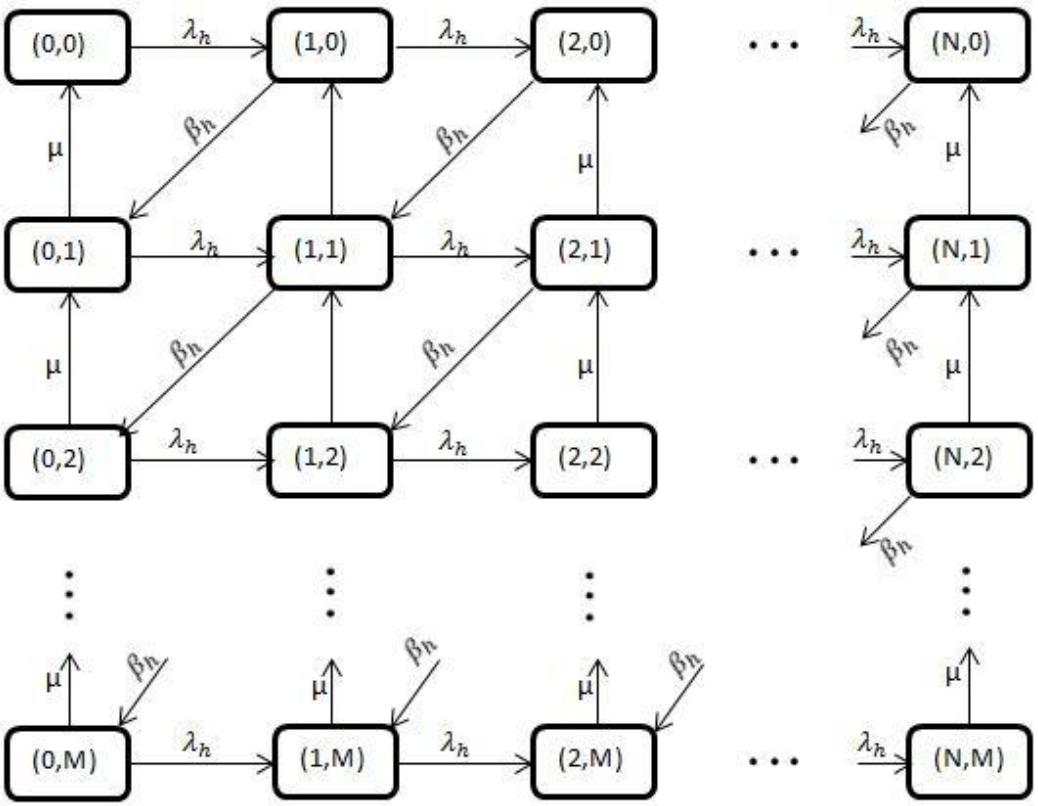

Figure 4.1 : First Come First Serve basis allocation of VMs for a given job when a PM has already been selected.

Figure 4.3 shows state transitions that occur for a given PM, which belongs to the hot pool, when it receives multiple job requests and how VMs are allocated to the users on first come first serve(FCFS) basis. Also average service time for a request is assumed to be μ after which the VM configuration is dissolved and the hardware is now available to cater to new requests which are waiting in the queue.

In the figure, state (0,0) denotes no VM has been allocated for a job and there is no request waiting in the buffer. State (n,0) denotes that there are *n* requests waiting in the buffer while the system is trying to configure and allocate a VM for the first request that came in which is at the head of FCFS queue. When there is a new job request which comes at the rate of λ_h, it goes in a FCFS buffer and there is a state transition to the next state *(n+1,0).* The state (n,m) denotes n requests in the buffer and m VMs have already been allocated for previous jobs and they are still running, while the system is configuring a new VM for the least recent job request. As soon as the system configures a new VM for the job, it is allocated for the corresponding user and the transition of state occurs from (n,m) to (n-1,m+1) which is one less job requests in the buffer and one more VM is now up and running. If the current state is (n,m) and one of the VMs completes its service to the user and now is available to take up another job, then the transition is to the new state (n,m-1).

Similar transitions take place for PMs belonging to warm pool and cold pool, but once a PM is used to allocate a VM for a request, it is turned ON and is now part of the hot pool and hence the dynamics of this PM are no longer those for a warm/cold pool but it follows hot pool dynamics. Then depending on the load of requests that a DC is receiving, it can decide to put some other machines on standby or turn it off, when a machine completes its service to the user/users and is free.

Once the VM is configured and allocated for the job, this VM uses the resources that it gets from the PM of the particular DC to complete the job that it receives from the user. The response time for the completion of job varies depending on various factors such as :

1. The type of work which the user has asked to be finished.
2. The type of hypervisor that is being used (like Xen, KVM)

3. Type of VM used (like HVM - Hardware Virtual Machine, PVM – ParaVirtual Machine)
4. The configuration of the given VM like the number of cores used, RAM and storage allocated for the job
5. Percentage utilisation of the given PM due to multiple requests. The more are the PM resources under usage by multiple requests, the slower is the response time.

We have assumed homogenous PMs, so the hypervisors and the type of VMs used, VM configuration are same. Similar kind of work is asked for completion to the DCs. Hence we can assume the fluctuations of the response time around the mean value to be minimal and hence can be represented by mean response time given by $1/\mu$ (that is the rate of service response is given by μ).

Once the service is rendered, the VM dissolves and it frees all the cores, RAM and storage that it was using towards the service to the user. The hardware is then available to take up a new job.

CHAPTER 5
SIMULATION OF THE MODEL

We have written a code to present the working of the optimizer. For requests originating from different locations, the code determines the appropriate PM from appropriate DC which will best cater the job. The optimizer takes the following factors into account while determining the best PM for the job :

i. Location of DCs with respect to the location of origin of request.
ii. Availability of PMs in different pools of each DC namely hot, warm and cold pools.
iii. Variable pricing structure at each DC.
iv. Percentage usage of each DC.

We have assumed the requests to be coming at random rate with mean λ, which is greater than the rate of completion the job, so that we get insight into the usage of all kinds of pools. In general, the average rate of requests is lesser than the rate of job completion which takes care that the rate of job rejection in under control. The simulation is performed for countable number of DCs and locations for job requests input. We have assumed the pricing for PM per usage to vary linearly with the percentage usage of DCs. Also its is assumed that there is no PM failure and once a request is allotted certain PM, the job does not migrate to another PM [9].

The parameters used for the simulation are as follows :

Table 5.1 : Parameters for simulation of IaaS based cloud model

Description	**Value**
Rate of requests	> 1 per unit (λ)
Number of job requests per iteration	100 (scalable)
Number of iterations	200

Points of request origin	3
Number of DCs	3
Base price for each DC	10,9,8 resp. per VM per hour
Number of PMs per DC	24
Number of PMs per pool	8
Number of VMs per PM	1
Start up time	Hot – 10 units, warm – 20 units , cold – 50 units
Response time	(1/μ) 30 units

The pricing in each DC is different and varies linearly with the number of PMs utilized at any given time. Although the pricing does not vary once a PM is allotted to the user. The variation of pricing with VM is as shown in figure 5.1.

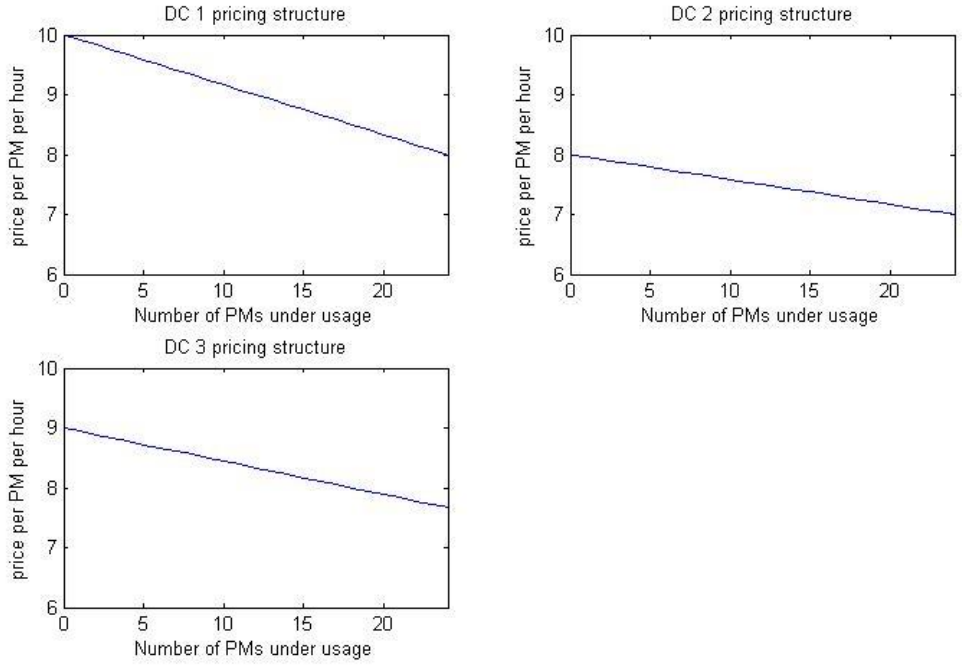

Figure 5.1 : Pricing Structure assumed for each of the Datacenters

In the simulation, one iteration involves 100 job requests for VMs. As we have assumed 1 VM per PM, it is safe to say that each job requests for a PM. The number of job

requests over the process can be scaled to any higher number and the algorithm will generate similar results, thus showing that the model is scalable.

This process is iterated 200 times. In each of the plots 5.2, 5.3 and 5.4, the first 100 iterations denote the results when the optimizer suggested in this thesis is not used to allocate a PM for the given job request whereas the next 100 iterations depict the results when we use the optimizer as discussed in this thesis for PM allocation.

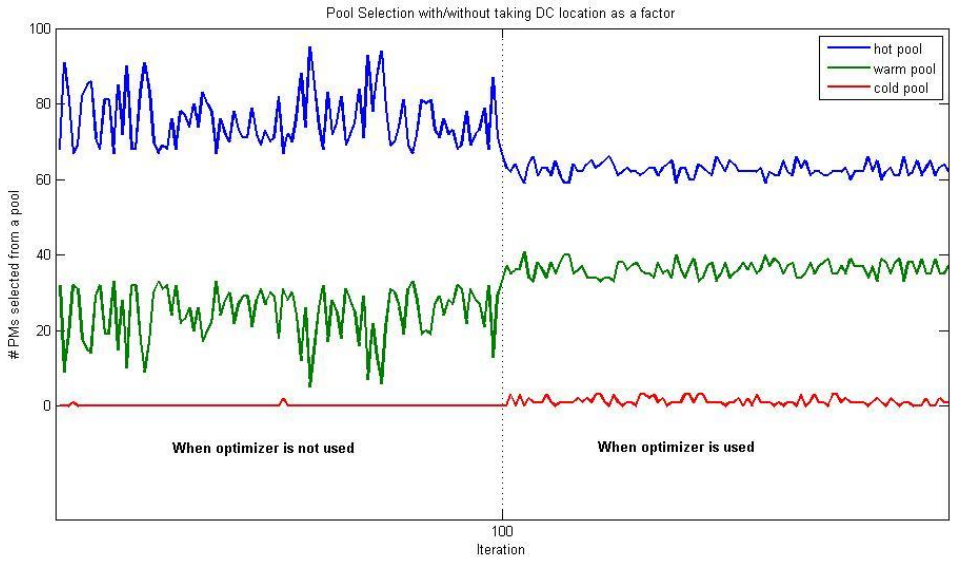

Figure 5.2 : Selection of pool with/without taking location of DC as a factor for deciding the optimal PM.

In the figure 5.2, the first 100 iteration show the allocation of pool when we do not consider the optimizer and allocate a PM just depending on availability of it in the best of the pools. It does not take the distance of the DC from point of request origination, nor does it take pricing structure which can be different in each DC. The next 100 iteration show the allocation of pool when we consider DC location, pricing structure on basis of percent utilisation and availability of PMs in different pools as factors affecting the selection of a particular PM. For each iteration(process) including 100 job requests, the 3 corresponding points on Y axis show the number of PMs selected from each of hot pool, warm pool and cold pool respectively.

We see that on an average, the number of PMs selected from hot pool are larger in case of first 100 iteration where only availability in different pools is taken into account compared to next 100 iteration where we use the optimizer. This is because, a PM from warm pool which is from a closer DC and/or costs lesser to the user can be better compared to a PM from hot pool from a farther DC with higher costs. Such better PM from warm pool is chosen over hot pool PM if we use the optimizer. Hence, we see reduction in number of PMs selected from hot pool while rise in those from warm pool. Same logic can be used to explain the increment in number of cold pool PMs as well.

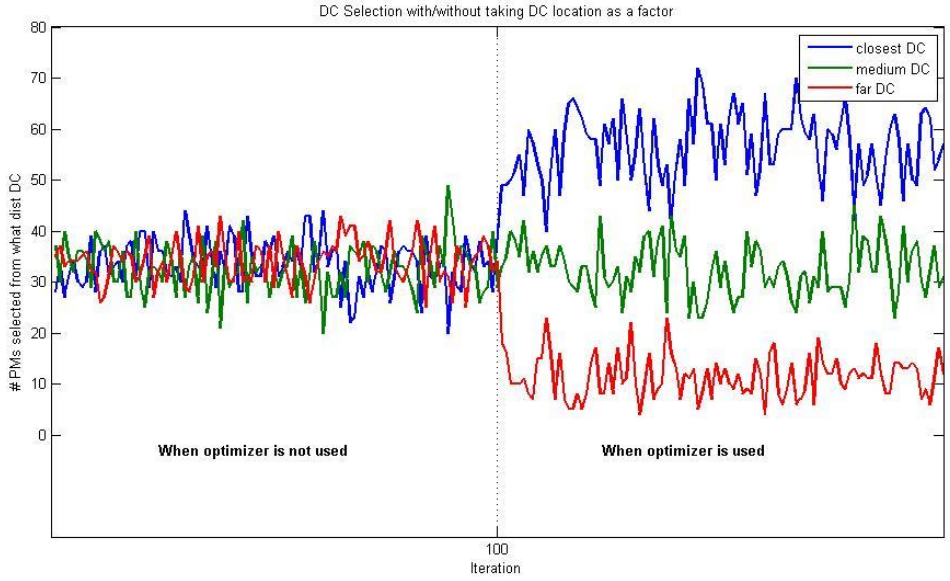

Figure 5.3 : Number of PMs selected from close/medium/far distance DC when the optimizer is/isn't used.

Figure 5.3 shows that over first 100 iterations, for a request originating from any particular point, it is equally likely for any DC to be selected to allot a PM for the job no matter how close or far it is from the location of request origination. However, when the optimizer is used (in the next 100 iterations), the likeliness of the closest DC to get the job increases, while it decreases for the farthest DC.

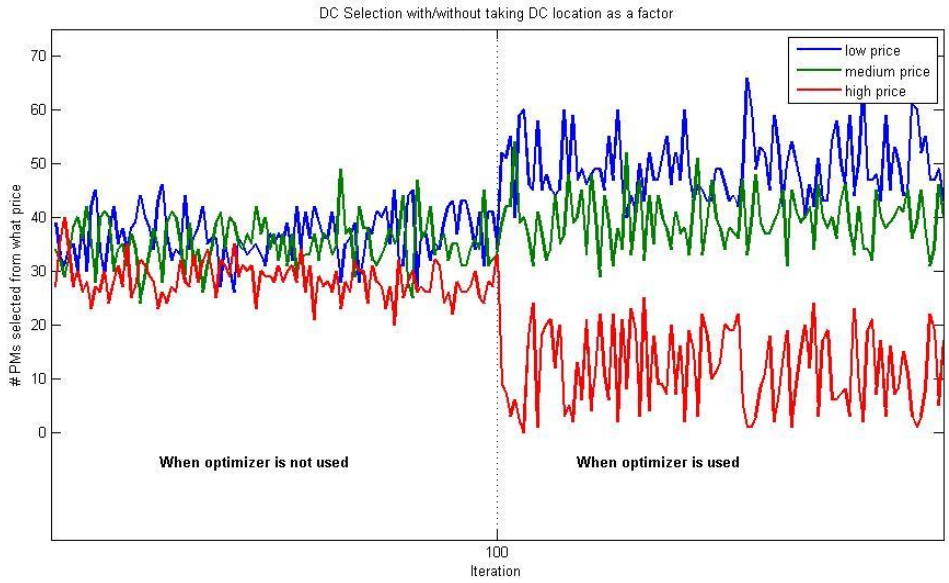

Figure 5.4 : Number of PMs selected from DC which has low/medium/high pricing at that given time when the optimizer is/isn't used.

Figure 5.4 shows that when the optimizer is not used, it is equally likely that the PM selected can belong to any DC no matter what the pricing structure for the particular DC is, at the given time. However, when the optimizer is used, the probability that the job is allotted to a PM from a DC which has lesser price compared to other DCs increases compared to the model which does not use the optimizer. Similarly, job allocation to higher price DC PMs is relatively lesser.

CHAPTER 6
CONCLUSION

In this thesis, we have quantified the effects of rate of job requests, Data Center locations, various pools in each DC, number of PMs in each DC, pricing structure in each DC, service time for a job and percentage usage of each DC on the working and performance of IaaS based Cloud. We modeled the Cloud as two sub models using CTMC and queuing theory and have successfully demonstrated the effects on various factors mentioned above on resource allocation by using an optimizer. These models give better insight into the working of IaaS based Cloud services and the providers of Cloud services can use these models to determine optimum size of various pools in a given DC, number of DCs should it look for to determine the PM which suits most for a job, all while ensuring avoidance of any SLA violations, maintaining a low job rejection rate and saving power consumption.

APPENDIX A

A.1. For simulation of RPDE model

Following is a part of code to simulate RPDE model. It shows how the equations can be converted to matrix form which can then be used to get the desired results.

```
% notation : PI(k,n,j) denotes steady state probability of k^th DC, n jobs in buffer j={hot,warm,cold} pool. PI is denoted as a vector of size (1 x (3NK + 1)).

%Job(2->N-1)DC(2->K)
for k = 2:K
    for n = 2:N-1
        PI(3*k+(n-1)*(3*K)+h-2, 3*k+(n-1)*(3*K)+h-2) = -(lamda + delh);
        PI(3*k+(n-1)*(3*K)+h-2, 3*(k-1)+(n-1)*(3*K)+w-2) = delw*Pw;
        PI(3*k+(n-1)*(3*K)+h-2, 3*(k-1)+(n-1)*(3*K)+c-2) = delc;
        PI(3*k+(n-1)*(3*K)+h-2, 3*(k)+(n-2)*(3*K)+h-2) = lamda;

        PI(3*k+(n-1)*(3*K)+w-2, 3*k+(n-1)*(3*K)+w-2) = -(lamda + delw);
        PI(3*k+(n-1)*(3*K)+w-2, 3*k+(n-1)*(3*K)+h-2) = delh* (1-Ph);
        PI(3*k+(n-1)*(3*K)+w-2, 3*k+(n-2)*(3*K)+w-2) = lamda;

        PI(3*k+(n-1)*(3*K)+c-2, 3*k+(n-1)*(3*K)+c-2) = -(lamda + delc);
        PI(3*k+(n-1)*(3*K)+c-2, 3*k+(n-1)*(3*K)+w-2) = delw* (1-Pw);
        PI(3*k+(n-1)*(3*K)+c-2, 3*k+(n-2)*(3*K)+c-2) = lamda;
    end
end
```

A.2. For simulation of complete model

Defining PMs as cells in MATLAB for each DC and each pool. We have considered 3 DCs and 3 points of origination of requests. We store informations like total number of PMs in the pool, total PMs currently under use and also current status of each PM in the pool. This can be extended to multiple VMs in each PM in a given pool.

```
PMs = cell(3,3);
% cell {ii,jj} denotes ii^th DC and jj^th pool of it
% in PMs{ii,jj}, the first element is number of PMs available to
take up a new job while the last element denotes the total number
of PMs in that pool.
for ii = 1:3
    for jj = 1:3
        PMs{ii,jj} = [8,0,0,0,0,0,0,0,0,0,0,8];
    end
end
```

We can define startup times and pricing structure for different pools and DCs as shown below.

```
% We have defined pricing structure to be a linear function of
the number of PMs available in the given pool.

start_time = [10,20,50];
price = [10-(24 - (PMs{1,1}(1)+PMs{1,2}(1)+PMs{1,3}(1)))/12, 8-(24 -
        (PMs{2,1}(1) + PMs{2,2}(1)+PMs{2,3}(1)))/24,         9-(24 -
        (PMs{3,1}(1)+PMs{3,2}(1)+PMs{3,3}(1)))/18];
```

We define a metric for each pool of each DC, and the pool of a DC with minimum of this metric value can be chosen to take up the job.

```
% dc denotes DC and hwc denotes that it can be either of hot,
warm and cold pools.

for dc = 1:3
    for hwc = 1:3
        metric(dc,hwc)    =    a*start_time(hwc)    +    b*price(dc)    +
                          c*DC_dist(loc,dc);
    end
```

```
end
```

We constantly keep on updating the status of each PM of each pool across various DCs.

```
% We have assumed that as soon as a PM is allocated a job, we
have the information about how much time it will  take to
complete the client services.

% Instead one can also release the PM as soon as the job is
completed. In this case as soon as the job is completed, the PM
sends a status signal and hence the PM can be released to take up
new job.

for aa = 1:3
    for bb = 1:3
        for cc = 1:PMs{aa,bb}(12)
            if PMs{aa,bb}(cc+1) ~= 0
                PMs{aa,bb}(cc+1) = PMs{aa,bb}(cc+1) - 1;
            end
        end
        PMs{aa,bb}(1) = sum(PMs{aa,bb}(2:(PMs{aa,bb}(12)+1))==0);
    end
end
```

REFERENCES

[1] Joe Weinman, “Axiomatic Cloud Theory,” 2011.

[2] S. Chuob, M. Pokharel, J. S. Park, “Modeling and Analysis of Cloud Computing Availability based on Eucalyptus Platform for E-Government Data Center,” Proc. Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitious Computing, 2011.

[3] P. Sempolinski and D. Thain, “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus,” Proc. IEEE Second Int’l Conf. Cloud Computing Technology and Science (CloudCom ’10), pp. 417-426, 2010.

[4] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus Open-Source Cloud-Computing System,” Proc. IEEE Int’l Symp. Cluster Computing and the Grid, pp. 124-131, 2009.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. Queuing Networks and Markov Chains, Second edition. John Wiley, 2006.

[6] D. Gmach, J. Rolia, and L. Cherkasova. Resource and virtualization costs up in the cloud: Models and design choices. In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 395{402, Hong Kong, China, June 2011.

[7] H. Goudarzi and M. Pedram. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In IEEE International Conference on Cloud Computing (CLOUD), pages 324{331, Washington, DC, july 2011.

[8] N. Sato and K. S. Trivedi. Accurate and efficient stochastic reliability analysis of composite services using their compact Markov reward model representations. In IEEE International Conference on Services Computing (SCC), pages 114 { 121, Salt Lake City, UT, 2007.

[9] Y. Wu and M. Zhao. Performance modeling of virtual machine live migration. In IEEE International Conference on Cloud Computing (CLOUD), pages 492{ 499, Washington, DC, July 2011.

[10] F. Longo, R. Ghosh, V.K. Naik, and K.S. Trivedi, “A Scalable Availability Model for Infrastructure-as-a-Service Cloud,” Proc. Int’l Conf. Dependable Systems and Networks, pp. 335-346, 2011.

[11] R. Ghosh, K.S. Trivedi, V.K. Naik, and D.S. Kim, "End-to-End Performability Analysis for Infrastructure-as-a-Service Cloud: An Interacting Stochastic Models Approach," Proc. IEEE 16th Pacific Rim Int'l Symp. Dependable Computing, pp. 125-132, 2010.

[12] J. Fu, W. Hao, M. Tu, B. Ma, J. Baldwin, and F. Bastani, "Virtual Services in Cloud Computing," Proc. IEEE Sixth World Congress Services, pp. 467-472, 2010.

[13] S. Kieffer, W. Spencer, A. Schmidt, and S. Lyszyk, "Planning a Data Center," white paper, http://www.nsai.net/White_Paper- Planning_A_Data_Center.pdf. Feb. 2003.

[14] Saif U. R. Malik, S. U. Khan and S. K. Srinivasan, "Modeling and Analysis of State-of-the-Art VM-Based Cloud Management Platforms," IEEE Transactions on Cloud Computing, Vol. 1, No. 1, 2013.