Dynamic control and Resource management for Mission Critical Multi-tier Applications in Cloud Data Center

C. N. Sahoo[*] **and Veena Goswami**[**]
[*]School of Computer Engineering, KIIT University, Bhubaneswar-751024, India
[**]School of Computer Application, KIIT University, Bhubaneswar-751024, India

Article Info

Article history:
Received Feb 5, 2016
Revised May 7, 2016
Accepted May 19, 2016

Keyword:
Cloud computing
Virtual machines
Multi-tier application
Queueing
Dynamic control
Resource management

ABSTRACT

The multi-tier architecture style has become an industry standard in modern data centers with each tier providing certain functionality. To avoid congestion and to adhere the SLA under fluctuating workload and unpredictable failures of Mission Critical Multi-tier applications hosted in the cloud, we need a Dynamic admission control policy, such that the requests must be processed from the first tier to the last without any delay. This paper presents the least strict admission control policy, which will induce the maximal throughput, for a two-tier system with parallel servers. We propose an optimization model to minimize the total number of virtual machines for computing resources in each tier by dynamically varying the mean service rate of the VMs. Some performance indicators and computational results showing the effect of model parameters are presented. This model is also applicable to priority as well as real-time based applications in Cloud based environment.

Copyright © 2016 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:
C. N. Sahoo
School of Computer Engineering, KIIT University
Bhubaneswar-751024, India
Phone: +91-9880149447
Email: nishikant.choudhury@gmail.com

1. INTRODUCTION

Cloud computing greatly lowers the threshold for deploying and maintaining web applications since it provides infrastructure as a service (IaaS) and platform as a service (PaaS) for web applications [1]. Consequently, a number of web applications, particularly the web applications of medium and small enterprises, have been built into a cloud environment. Meanwhile, leading IT companies have established public commercial clouds. For example, Google App Engine enables enterprises to build and host web applications on the same systems that power Google applications. App Engine offers fast development and deployment; simple administration, with no need to worry about hardware, patches or backups; and effortless scalability [2]. IBM also provides cloud options [3]. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers [4]. We even can establish a private cloud with Ubuntu Enterprise Cloud to offer immediacy and elasticity in the infrastructure of web applications [5]. In summary, both of the numbers of cloud applications and providers have kept gradually increasing for a couple of years [6, 7]. As a result, computing resource scheduling and performance managing have been one of the most important aspects of cloud computing [8, 9].

This paper focuses on queueing-based analytical model for performance of web based applications with multi-tiered architecture. It is quite difficult to predict the traffic in web based applications. In case of Real-time or Mission-Critical applications, requests must be processed from the 1st tier to the last without any delay. If the release and processing times of requests are known, the problem for determining the processing order of requests is typically a scheduling problem. However, if requests arrive randomly, in order to prevent any delay of requests currently in the system and ensure that the new request will go through all the tiers successfully, an admission control should be used to decide whether or not to accept the new request. This paper deals with the admission control policies for no-wait tandem queueing systems. We present the least strict admission control policy, which will induce the maximal throughput, for a two-tier system with parallel servers (VMs). This policy can be easily extended for multi-tier systems.

In order to maximize the total profit, the system dynamically decides whether to accept a new request based on his bandwidth requirement and duration time and the system data (the numbers of requests in system and their remaining service times). Based on the known service times at all tiers of the new request and the system information upon arrival, the system decides whether to accept this new request such that all accepted requests will go through the rest of tiers successfully. In this paper, we present a feasible admission control policy, called the new never block the old [10] (NNBO) policy. The main idea of this policy is that the presence of a newly admitted request will not block other existing requests. It can be easily shown that the NNBO policy is the least strict policy. For a controlled system, an important performance measurement is the resulting loss probability of any request or, equivalently, the loss rate of the system. It is a greedy system in the sense that requests try their best to enter all tiers. Therefore, intuitively, the total loss rate from all tiers in a free system may be smaller than the loss rate in a controlled system. However, it is evident that, under the exponential service times, the loss rates of NNBO system and the free system are equal when there is only one server at the 1st tier.

The rest of the paper is organized as follows. Section 2 briefly reviews the related works. Section 3 presents the system description. Model description and its analysis is carried out in Section 4. Various performance measures are evaluated in Section 5. Section 6 contains computational numerical illustrations with a variety of Results and Discussion in the form of graphs to show the effectiveness of the model parameters. Section 7 concludes our paper.

2. RELATED WORK

Jung et al. [11] proposed a generating adoption for multi-tier applications in virtualized consolidated server environments. It provides dynamic management method and optimizes offline resources to generate suitable configurations by evaluating a model consisting of multi-tier $M/M/n$ queues. Urgaonkar et al. [12] proposed a model for multi-tier internet applications to provide the resources to each tier of the application, and combine predictive and reactive methods. The closed system model of muti-tier business applications based on mean value analysis (MVA) algorithm to predict performance of multi-tier applications has been discussed in Chen et al. [13]. A nonlinear integer optimization model for determining the number of machines at each tier in a multi-tier server network has been studied in Zhang et al. [14]. A single queue model for all tiers to prevent overload and maintain absolute client response time has been reported in Kamra et al. [15]. Wang et al. [16] presented a new self-adaptive capacity management framework for multi-tier virtualized environments. It executes periodically and reassigns resources by evaluating a model consisting of multi-tier $M/M/1$ queues and solves an optimization problem [17].

A model for dynamic resource provisioning in multi-tier internet applications captures various characteristics of an arbitrary number of heterogeneous tiers has been reported in Urgaonkar et al. [18]. Ardagna et al. [19] developed a heuristic solution for maximization of profits using a cost model for multi-tier data controller center. Chang et al. [10] proposed a model for Admission control policies for two-stage tandem queues with no waiting spaces to provide the resources to each tier of the just-in-time based production lines and compare the resulting loss rate of the controlled system with the loss rate of a system without any admission control called the free system. This model is also applicable to systems where the system manager must maintain the no-wait privilege for the higher priority customers in order to differentiate the qualities of the services. In our work we propose an optimization model to minimize the total number of virtual machines for computing resources in each tier by dynamically varying the mean service rate of the virtual machines (VMs).

3. SYSTEM DESIGN

This section presents the architecture of the hosting platform required in our work.

3.1. Architecture overview

Model View Controller (MVC) framework comprises of multiple tiers such as web-tier, middle-tier and persistence tier. Web-tier typically consists of web-servers whereas middle-tier consists of app-servers, file-servers to host middleware technologies and persistence-tier consists of Databases or backend systems such as legacy systems. The MVC design pattern is a way of taking an application and breaking it into three distinct parts: the model, the view, and the controller. The advantage of using the MVC pattern is that there is no business or Model-specific processing within the presentation, or view, component itself. The opposite is also true; that is, there is no presentation logic in the model and business layers. This improves component reuse there and also improve the ability to change a tier implementations with minimal effect on the other tiers [20, 21]. Figure 1 shows the request processing flow

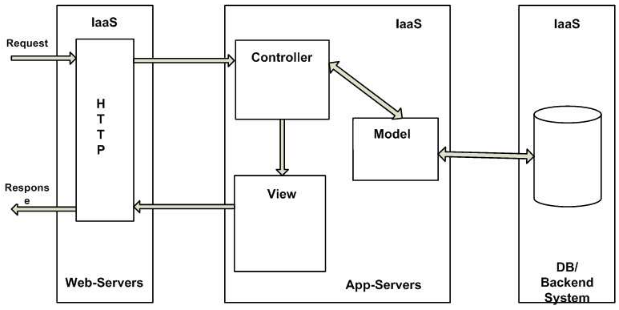

Figure 1. A typical 3-tier application in cloud

of a typical three tier MVC based web application deployed in cloud, in which each rectangle represents a tier. A request moves through the tiers , may visit a tier multiple times and get processed at the visited tier. Finally, the processing completes and returns to request senders from the front tier. Since different tiers are designed to provide different functionalities, tiers could be clustered by a group of servers with similar resource characteristics. For example, a middle-tier business logic server would be better to have fast processing capability, while a backend-tier database server is usually required to provide high I/O operation rate and Web-Tier doesn't have any processing logic rather it works as a request forwarding agent. Therefore, physical servers are clustered into different groups (VMs), serving different tiers of applications. The architecture of a shared data center is shown in Figure 2, which consists

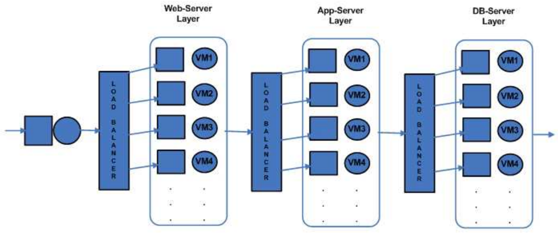

Figure 2. Virtualized Data Center architecture consisting of VMs in IaaS

of heterogeneous physical nodes (IaaS), shared by multiple independent applications, hosting web application from different companies or organizations. Modern transactional web applications are designed using multiple tiers, which are often distributed across different servers. Active VM *Load Balancer* maintains information about each VM along with the number of requests currently allocated to VMs in a intended tier. When a request to allocate a new VM arrives, it identifies an existing free VM.

3.2. Virtualized Multi-tier Application Queueing Model

A virtualized multi-tier application in cloud computing environment is deployed on multiple virtual machines, and each tier provides certain functionality to its preceding tier. Let us consider an online application that consists of n tiers, $T_1, T_2, \ldots, T_n$. We assume that there are c parallel identical VMs in each tier but they are provisioned when needed. The load balancer distributes the load to different parallel VMs queueing models of that tier to execute. Each tier is assumed to employ a perfect load-balancing element for a virtualized application that is responsible for processing requests at that tier, and each request is forwarded to its succeeding tier for further processing. Figure 3 represents Tandem queueing system with zero-buffer and there are multiple nodes with multiple Servers (VMs) at each node.

4. MODEL DESCRIPTION AND ANALYSIS

We discuss the dynamic admission control to a two-tier no-wait tandem queueing system with N_1 VMs at tier 1 and N VMs at tier 2. We consider the epoch when a new request X, whose service time at stage j is denoted

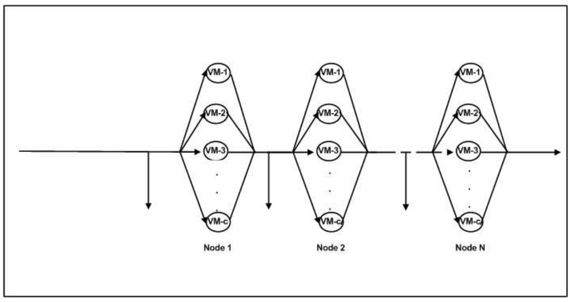

Figure 3. Zero-Buffer Tandem queues

by X_j; j = 1, 2, arrives. Suppose that there are n_1 and n_2 requests already at stage 1 and stage 2, respectively. As per NNBO admission control policy: X is admitted if and only if: (i) When X arrives, there is at least one free VM at tier 1. (ii) Based on the system data observed upon X's arrival, there should be at least one free VM at tier 2 when X enters tier 2. (iii) During the X's sojourn at tier 2, any of those n_1 requests left behind by X at tier 1 can still enter tier 2. Here we consider a no-wait tandem queueing system in which there is only one server $(n_1 = 1)$ at stage 1. Assume that requests arrive according to a Poisson process with rate λ and the service times of each request at stages 1 and 2 are exponentially distributed with rate μ_1 and μ_2, respectively. In this section, we define the states as (n_1, n_2), where n_1 and n_2 are the numbers of requests at stages 1 and 2, respectively. The stationary state balance equations are given as

$$\begin{aligned}
\lambda P_{0,0} &= \mu_2 P_{0,1}, && (1)\\
(\lambda + n_2\mu_2)P_{0,n_2} &= \mu_1 P_{1,n_2-1} + (n_2+1)\mu_2 P_{0,n_2+1},\ 1 \le n_2 \le N-1, && (2)\\
(\mu_1 + n_2\mu_2)P_{1,n_2} &= \lambda P_{0,n_2} + (n_2+1)\mu_2 P_{1,n_2+1},\ 0 \le n_2 \le N-1, && (3)\\
(\lambda + N\mu_2)P_{0,N} &= \mu_1\left(P_{1,N-1} + P_{1,N}\right), && (4)\\
(\mu_1 + N\mu_2)P_{1,N} &= \lambda P_{0,N}. && (5)
\end{aligned}$$

From (5), we get

$$P_{1,N} = \frac{\lambda}{\mu_1 + N\mu_2} P_{0,N}. \tag{6}$$

Using (6) in (4) and simplifying, we have

$$P_{1,N-1} = \frac{N\mu_2(\lambda + \mu_1 + N\mu_2)}{(\mu_1 + N\mu_2)\mu_1} P_{0,N}. \tag{7}$$

Substituting $n_2 = N - 1$ in (3), yield

$$P_{0,N-1} = \left\{ \frac{(\mu_1 + (N-1)\mu_2)(\lambda + \mu_1 + N\mu_2)N\mu_2}{(\mu_1 + N\mu_2)\lambda\mu_1} - \frac{N\mu_2}{\mu_1 + N\mu_2} \right\} P_{0,N}. \tag{8}$$

Solving (2) and (3), recursively, we obtain

$$\begin{aligned}
P_{1,n_2-1} &= \frac{\lambda + n_2\mu_2}{\mu_1} P_{0,n_2} - \frac{n_2+1}{\mu_1} P_{0,n_2+1},\ n_2 = N-1, \ldots, 2, 1, && (9)\\
P_{0,n_2} &= \frac{\mu_1 + n_2\mu_2}{\lambda} P_{1,n_2} - \frac{(n_2+1)\mu_2}{\lambda} P_{1,n_2+1},\ n_2 = N-1, \ldots, 1, 0. && (10)
\end{aligned}$$

We can obtain $P_{0,N}$ by applying Normalizing condition $\sum\limits_{n_2=0}^{N} (P_{0,n_2} + P_{1,n_2}) = 1$.

4.1. Recursive algorithm

In this section, we establish a computational algorithm to compute recursively stationarity state probabilities according to the following algorithm:

Step 1: Assume $P_{0,N} = 1$.
Step 2: Calculate $P_{1,N}$ from (6).
Step 3: Calculate $P_{1,N-1}$ and $P_{0,N-1}$, using (7) and (8).
Step 4: Balance equations for states $(1, N-2), (1, N-3), \ldots, (1, 0)$ yield P_{1,n_2}, $n_2 = N-2, N-3, \ldots, 0$ as function of P_{0,n_2}.
Step 5: Balance equations for states $(0, N-2), (0, N-3), \ldots, (0, 0)$ yield P_{0,n_2}, $n_2 = N-2, N-3, \ldots, 0$ as function of P_{1,n_2}.
Step 6: Repeat Steps 4 and 5 for $n_2 = N-2, \ldots, 0$.
Step 7: Normalization: $\sum\limits_{n_2=0}^{N} (P_{0,n_2} + P_{1,n_2}) = 1$ yields $P_{0,N}$.
Step 8: Compute $P_{n_1,n_2} = P_{n_1,n_2} \times P_{0,N}$ for $n_1 = 0, 1; n_2 = 0, 1, \ldots, N$.

5. PERFORMANCE MEASURES

Performance measures are the means to examine the efficiency of the queueing system under consideration. As the steady-state probabilities at various epochs are known, the main performance measures of the queueing system can be obtained as follows:

- The probability that an arrival finds Node-1 (tier-1) full is given by $L_1 = \sum\limits_{n_2=0}^{N} P_{1,n_2}$.
- Average number of lost customers per unit time at Tier-1 is $L_{T1} = \lambda \sum\limits_{n_2=0}^{N} P_{1,n_2}$.
- Average number of lost customers per unit time at Tier-2 is $L_{T2} = \mu_1 P_{1,N}$.
- Loss rate of the NNBO System is given by $L_{Loss} = \sum\limits_{n_2=0}^{N} P_{1,n_2} + \left(\frac{\mu_1}{\mu_1 + N\mu_2}\right) P_{0,N}$.

5.1. Cost analysis

We develop a total expected cost function per unit time for the tandem queuing system where the number of nodes are represented by n and number of VMs in each node is represented by c. Our objective is to determine the optimum number of VMs c, say c^*, and the optimum number of nodes n, say n^*, as decision variables so that the expected cost function is minimized. Let,
C_h = holding cost per unit time for each client request present in the system.
C_1 = fixed cost per unit time during the busy period for node 1
C_2 = fixed cost per unit time during the busy period for node 2.
C_3 = fixed cost for every lost client.
Let $F(c, n)$ be the expected cost per unit time. Using the definitions of each cost element and its corresponding system characteristics, we have

$$F(c, n) = C_h(N + 1) + C_1 L_{T1} + C_2 L_{T2} + C_3 \lambda L_{Loss} \tag{11}$$

The objective is to determine the optimum number of VMs c and optimum system size (nodes) n to minimize the cost function $F(c, n)$. Here, we are specifically considering 2-Nodes, hence $n = 2$. Hence, the cost function reduces to $F(c)$. We have implemented the numerical searching approach for the cost function using the genetic algorithm. The genetic algorithm is a probabilistic search algorithm that iteratively transforms a set (called a population) of mathematical objects, each with an associated fitness value, into a new population of offspring objects using the natural selection and mutation. Haupt et al. [22]. Genetic algorithms are adaptive search algorithms based on the evolutionary ideas of natural selection and genetics. It represents potential solutions by bit strings of a fixed length. By analogy to genetics, the strings can be rendered as chromosomes with individual bits referring the presence (bit = 1) or absence (bit = 0) of a gene. A genetic algorithm allows a population composed of many individuals to evolve under specified selection rules that minimize the fitness function, that is, the cost function in this paper. A population of alternative possible solutions (chromosomes) is created and allowed to evolve through a number of generations. Old generations beget new generations in a fashion that mimics genetic change in nature. The solution procedure is as follows:
INPUT: $\lambda, \mu_1, \mu_2, C_h, C_1, C_2, C_3, c, n$ and genes, probability of crossover, and probability of mutation.

OUTPUT: approximate solution c^*, n^*, F^*.
Step 1: Population Initialization. An implementation of genetic algorithm initiates with a encoding of each input into a chromosomes. Each gene value either 0 or 1 is randomly generated.
Step 2: Fitness Computation. To determine the optimal expected profit per unit time for optimal virtual machines and system capacity, the fitness of a chromosome is computed using the expected cost function $F(c, n)$ in equation (11).
Step 3: Selection and Crossover. The selection is a process which mimics the natural survival of the fittest creatures. Each chromosome has a fitness value obtained through the fitness function. The chromosomes which perform better fitness values are given more chances and it discards poor quality genes based on their fitness value. Crossover is done by selecting two parents during reproduction and combining their genes to produce offspring. The parent chromosomes are then mated to generate a new set of offspring chromosomes. This mated procedure is also called crossover.
Step 4: Mutation. Mutation is the random changing of one or more bits in a chromosome. It is useful to create new genes that are not in the initial set of population, or ones that have evolved out of the population, but now would be beneficial.
Step 5: Repeat Step 2 - Step 4 until the stopping criterion is met. We use 50 generations as our stopping criterion.

6. RESULTS AND DISCUSSION

In this section some numerical results are discussed. Numerical results for various system performance measures are presented in Table 1. We observe that for fixed μ_1 as μ_2 increases: (i) The optimum cost increases. This is because the number of the VMs in the system also increases. But for fixed μ_2 as μ_1 increases: (i) The optimum cost and other performance measures such as P_{loss} decreases. This is because the number of the VMs in the system remains the same. With the same number of VMs and fixed μ_2, as μ_1 increases both L_{T1} & L_{T2} decreases. Figures 4 and 5 show the effect of μ_2 on the expected number of lost customers per unit time at Tier-1 and Tier-2, respectively. It is seen that as μ_2 increases, L_{T1} and L_{T2} increases monotonically to certain extend then monotonically decreases. From Figure 4, as μ_1 increases, lost customers per unit time at tier-1 L_{T1} decreases. Whereas from Figure 5, lost customers per unit time at tier-2 L_{T2} increases as μ_1 increases. This is because of the admission control policy.
Figure 6 depicts the impact of VMs on the Cost. It can be observed that cost increases as the c and μ_2 increases. For the fixed number of VMs and μ_2, the cost involved remains almost same, that is, the small variation in cost is due to the variation in μ_1. The effect of VMs on the P_{Loss} is represented in Figure 7. It is seen that as μ_2 increases, P_{Loss}

Table 1: Optimal system performance measures

μ_1	μ_2	c	$F(c,n)$	L_{T1}	L_{T2}	P_{Loss}
0.5	1	5	196.00	1.6	0.00001	0.8
	2	10	347.45	1.63516	0.04394	0.81757
	4	12	408.57	1.85527	0.00081	0.92763
	5	14	466.67	1.66667	0.00024	0.83333
0.75	1	5	194.55	1.45455	0.00001	0.72727
	2	10	347.32	1.51703	0.08592	0.75851
	4	12	408.03	1.79648	0.00243	0.89823
	5	14	465.73	1.57143	0.00069	0.78571
1	1	5	193.33	1.33333	0.00000	0.66667
	2	10	347.61	1.42327	0.13490	0.71163
	4	12	407.58	1.74471	0.00518	0.87235
	5	14	465.04	1.5	0.00142	0.74999
1.25	1	5	192.31	1.23077	0.00001	0.61541
	2	10	348.18	1.34683	0.18860	0.67341
	4	12	407.22	1.69883	0.00912	0.84941
	5	14	464.51	1.44444	0.00244	0.72222

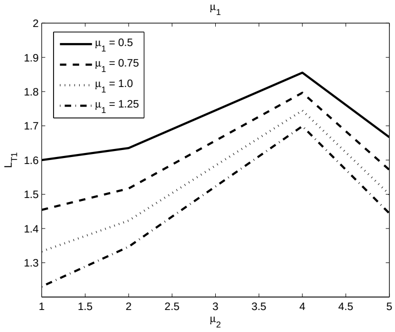

Figure 4. Impact of μ_2 on L_{T1}

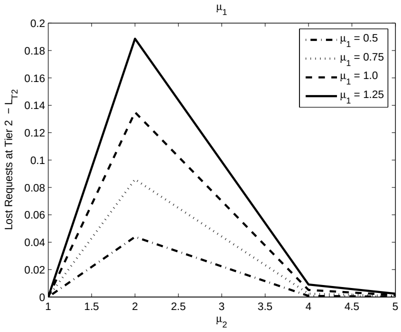

Figure 5. Effect of μ_2 on L_{T2}

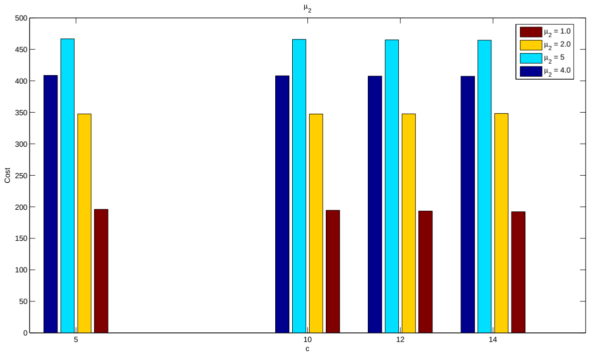

Figure 6. Impact of VMs on Cost

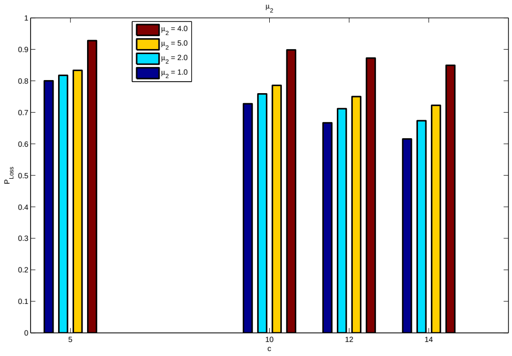

Figure 7. Impact of P_{Loss} on VMs

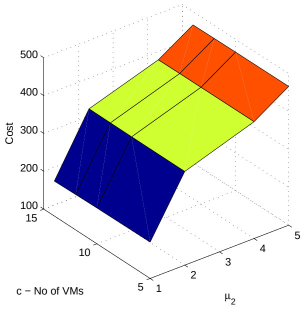

Figure 8. Cost versus c versus μ_2

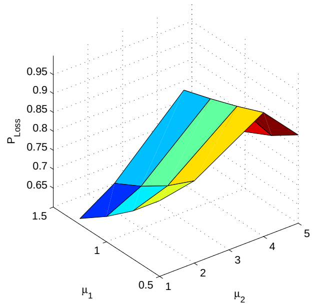

Figure 9. Cost versus c versus μ_2

increases (with a small variation at μ_2 = 4) and hence resulting more loss. This is because of more loss at node-1, that is, L_{T1}.

The impact of different parameters on Cost and P_{Loss} is shown on Figure 8 and Figure 9, respectively. It can be seen from Figure 8, the Cost increases monotonically as the μ_2 and number of VMs increases. But in case of Figure 9, with the increase of μ_2 and μ_1, the P_{Loss} increases monotonically till μ_2 = 4 and then decreases. This shows that due to dynamic admission policy more loss is happening at node-1, that is, L_{T1}.

7. CONCLUSION

Highly performance sensitive mission critical as well as real-time applications are rarely hosted in public Clouds. With a dynamic admission control policy, we can easily address these type of application specific issues where extra cost is incurred to ensure high-availability as well fault tolerance. In this paper, we propose an optimal policy for provisioning of VMs in cloud data center to minimize the congestion in the network by varying the service rate of the virtual machines. An analytical model is developed to fit cloud environment with heterogeneous servers (as required for different tiers) to minimize the total number of VMs and finally cutting down the cost involved. The objective is to improve the efficiency and flexibility in cloud environment for resource provisioning. A variety of numerical results in the form of tables and graphs are discussed to display the effect of the system parameters on the performance measures. Cost analysis has been done to improve the grade of service by selection of appropriate system parameters using genetic algorithm. To achieve significant performance level, we adopted Service Level Agreement based negotiation of prioritized applications to determine the costs and penalties. It is a trade-off that potential applications need to consider in deciding the performance evaluation of server farms as an important aspect of cloud computing which is of crucial interest for both cloud providers and cloud customers. As future work, research will be carried out on useful algorithms for measuring deployment costs of virtual resources in multi-cloud environments.

REFERENCES

[1] Michael Armbrust, et al., “Above the Clouds: A Berkeley View of Cloud Computing. [Online],” *http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf*, 2009.

[2] “Google App Engine [Online],” *http://code.google.com/intl/en/appengine/.*

[3] “IBM Smart Business Cloud Computing [Online],” *http://www.ibm.com/ibm/cloud/.*

[4] “Amazon Elastic Compute Cloud (Amazon EC2),” [Online] *http://aws.amazon.com/ec2/*

[5] “Ubuntu, Private cloud: Ubuntu Enterprise Cloud [Online],” *http://www.ubuntu.com/cloud/private.*

[6] Seo J. and Kim H. K., “A Prototype of Online Dynamic Scaling Scheduler for Real- Time Task based on Virtual Machine,” *International Journal of Electrical and Computer Engineering*, vol. 6, No. 1, pp. 205-211, 2016.

[7] Vijaya A. and Neelanarayanan V., “A Model Driven Framework for Portable Cloud Services,” *International Journal of Electrical and Computer Engineering*, vol. 6, No. 2, pp. 708-716, 2016.

[8] Buyya R., et al., “An architectural approach to autonomic computing,” *Future Generation Computer Systems*, vol. 25, No. 6, pp. 599-616, 2009.

[9] Kundu A., et al., “Introducing New Services in Cloud Computing Environment,” *International Journal of Digital Content Technology and its Applications*, vol. 4, No. 5, pp. 143-152, 2010.

[10] Chang, K.H. et al., “Admission control policies for two-stage tandem queues with no waiting spaces,” *Computers & Operations Research*, vol. 30, No. 4, pp. 589-601, 2003.

[11] Jung G., et al., “Generating adaptation policies for multi-tier applications in consolidated server environments,” *Proceedings of the 5th International Conference on Autonomic Computing* pp. 23-32, 2008.

[12] Urgaonkar B., et al., “Agile dynamic provisioning of multi-tier Internet application,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, No. 1, pp. 1-39, 2008.

[13] Chen Y., et al., “SLA decomposition: Translating service level objectives to system level thresholds,” *Proceedings of the 4th International Conference on Autonomic Computing*, pp. 3-15, 2007.

[14] Zhang A., et al., “Optimal server resource allocation using an open queueing network model of response time,” *HP Labs Technical Report*, pp. 1-17, 2001.

[15] Kamra A., et al., “A self-tuning controller for managing the performance of 3-tiered web sites,” *Proceedings of International Workshop on Quality of Service*, pp. 47-58, 2004.

[16] Wang X., et al., “Energy-efficient datacenters.Computer-Aided Design of Integrated Circuits and Systems,” *The Journal of Systems and Software* , vol. 81, No. 9, pp. 1591-1608, 2006.

[17] White S.R.,et al., “An architectural approach to autonomic computing,” *Proceedings of the First IEEE International Conference on Autonomic Computing*, pp. 2-9, 2004.

[18] Urgaonkar B., et al., “An analyticial model for multi-tier Internet services and its applications,” *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and modeling of computer systems*, pp. 291-302, 2005.

[19] Ardagna D., et al., “SLA based profit optimization in multi-tier systems,” *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications*, pp. 263-266, 2005.

[20] Reddy K.V., et al., “Research Issues in Cloud Computing,” *Global Journal of Computer Science and Technology* vol. 11, No. 11, pp. 59–64, 2011.

[21] Bi J., et al., “Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center,” *Proceedings of the Third IEEE International Conference on Cloud Computing* pp. 370-377, 2010.

[22] Haupt, R. L. and Haupt, S. E., “Practical Genetic Algorithms,” *John Wiley and Sons, Inc.*, Hoboken, New Jersey, Canada, 2004.