FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 38 (2013) No. 3

DOI: 10.2478/fcds-2013-0008 ISSN 0867-6356
e-ISSN 2300-3405

DYNAMIC PROVISIONING AND RESOURCE MANAGEMENT FOR MULTI-TIER CLOUD BASED APPLICATIONS

Veena GOSWAMI *, S. S. PATRA †, G. B. MUND ‡

Abstract. Dynamic capacity provisioning is a useful technique for handling the workload variations seen in cloud environment. In this paper, we propose a dynamic provisioning technique for multi-tier applications to allocate resources efficiently using queueing model. It dynamically increases the mean service rate of the virtual machines to avoid congestion in the multi-tier environments. An optimization model to minimize the total number of virtual machines for computing resources in each tier has been presented. Using the supplementary variable and the recursive techniques, we obtain the system-length distributions at pre-arrival and arbitrary epochs. Some important performance indicators such as blocking probability, request waiting time and number of tasks in the system and in the queue have also been investigated. Finally, computational results showing the effect of model parameters on key performance indicators are presented.

Keywords: cloud computing, virtual machines, multi-tier web application, queuing, performance modelling.

1 Introduction

Large data centers host many third party web applications providing comprehensive services. In such systems, there is a decisive need to provide quality-of-service (QoS) performance guarantees for each class of differentiated services [3]. The QoS of the hosted applications plays a crucial role in attracting and retaining customers, directly

*School of Computer Application, KIIT University, Bhubaneswar - 751024, India; email: veena_goswami@yahoo.com

†School of Computer Application, KIIT University, Bhubaneswar - 751024, India; email: sudhanshupatra@gmail.com

‡School of Computer Engineering, KIIT University, Bhubaneswar - 751024, India; email: mundgb@yahoo.com

impacting on providers profits. Hence, the service providers guarantee a certain level of QoS for each application. In return, the clients agree to pay the service provider based on the specified level of QoS. Such issues of QoS requirements are often negotiated based on Service Level Agreements (SLAs), in which the expected performance level and the cost model involving both revenue and penalty will be clearly defined [7]. The main issue of preserving the QoS is the high variability of the workload, which makes it difficult to estimate the resource requirement in advance [8].

Enterprise IT infrastructure customers with virtualized applications require lesser resource cost, and thus save resource by distributing workload requests to virtualized multi-tier applications in cloud environment. This creates the need for establishing a computing atmosphere for dynamically provisioning cloud resources from multi-tier domains within and across enterprises. Furthermore, there are many open challenges involved in on-demand resources dynamic provisioning for cloud data centers, such as the CPU, memory, disk and network bandwidth to be partitioned among the resident VMs, and optimal configuration for VMs.

In order to efficiently utilize resources while satisfying the SLA under fluctuating workload and unpredictable failures, adaptive self-managing techniques are required to dynamically assign resources among applications of different clients on the base of short-term demand estimates. Since the multi-tier architecture style has become an industry standard in modern data centers with each tier providing certain functionality, this paper analyzes the performance of the virtual machines (VMs) and dynamically increases the mean service rate of the VMs to avoid congestion in the multi-tier environments. The main contribution of this paper is that it proposes 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.

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. Computational algorithm to compute the stationary system length distribution is presented in Section 5. Various performance measures are evaluated in Section 6. Section 7 contains computational experiences with a variety of numerical results in the form of graphs to show the effectiveness of the model parameters. Section 8 concludes our paper.

2 Related Work

Urgaonkar et al. [10] 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. [4]. A single queue model for all tiers to prevent overload and maintain absolute client response time has been reported in Kamra et al. [6]. Jung et al. [5] 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. 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. [13]. Wang et al. [11] 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.

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. [9]. Ardagna et al. [1] developed a heuristic solution for maximization of profits using a cost model for multi-tier data controller center. They did not distinguish servers in different tiers and allocated physical resources instead of virtual machines. At the same time they adopted a closed queueing network performance model for the automated system. The above approaches are commonly based on the provisioning of identical servers as unit, while in our work we adopt full virtual machines based on an open queueing network model, which supports sharing of physical infrastructure as well as guarantees the performance isolation of different virtualized application environments by deploying them on separate virtual machines.

3 System Design

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

3.1 Architecture Overview

Figure 1 shows the request processing flow of a typical three tier web application deployed in cloud, in which each circle represents the resource being consumed at that 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. Therefore, physical servers are clustered into different groups, serving different tiers of applications, as represented by the solid rectangles in Figure 2.

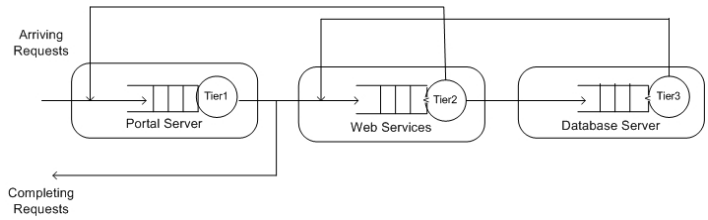

Figure 1. A typical 3-tier application in cloud

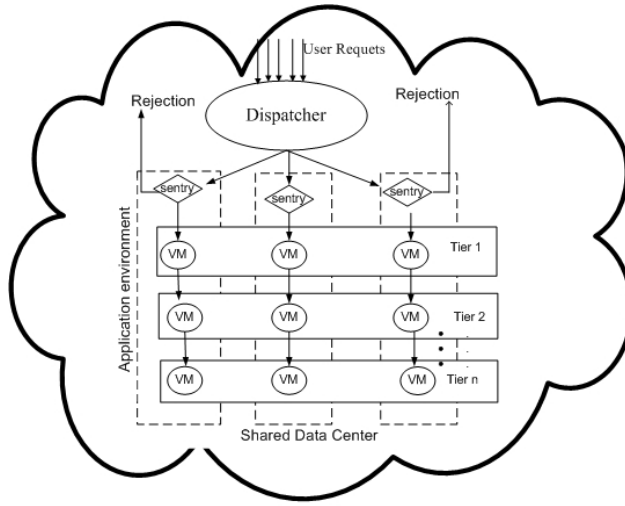

Figure 2. Data center architecture

The architecture of a shared data center is shown in Figure 2, which consists of heterogeneous physical nodes, shared by multiple independent application environments (AE), hosting web applications from different companies or organizations. Each AE may execute several classes of transactions. Modern transactional web applications are designed using multiple tiers, which are often distributed across different servers.

The *Dispatcher* dispatches the requests of different applications to the corresponding environment in a shared data center. In each AE, a *sentry* node receives requests from the dispatcher, and it will reject excessive requests when the local AE is detected to be overloaded, in order to meet the desired QoS requirement and to preserve system stability. Since the workloads of separate AEs are varying as time goes on, the global resource manager may decide to change the capacity of different VMs to capture the varying workload. Each tier holds a virtual machine monitor acting as the resource allocation actuator, which can assign different amounts of resource capacity to multiple VMs within the same tier.

3.2 Cloud Computing Infrastructure

The self-managing techniques, such as Supervise, Analyze, Plan and Execute (SAPE) control loops architecture is needed to dynamically provision the resources for virtualized multi-tier application execution environments (VAEEs) of different customers [2]. The goal is to meet the virtualized application requirements while adapting IT architecture to workload variations.

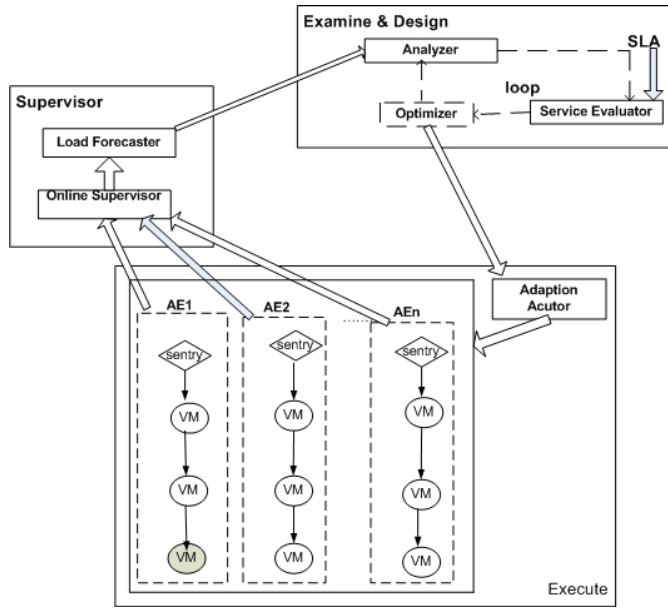

Figure 3. The dynamic resource provisioning of cloud data center

Figure 3 provides an adaptive self-controlling architecture, in which a common physical infrastructure is shared by a multiple AEs. After a certain time (i.e. control interval) elapsed, the whole system will check its own status, judge the overall efficiency, and then adopt itself to the workload variation. The components are organized according to the Supervise Analyze Plan Execute pattern described in the architectural approach to autonomic computing [12] as follows.
Pool of Resources contains physical resources and virtualized resources. A batch of VMs hold several VAEEs sharing the capacity of physical resources and can insulate multiple applications from the hardware.

Self-management community automates the VMs so as to maintain the response time requirements of the different customers. The components of self-management community are as follows:

- **Supervisor**: The supervisor gathers the information about the running states of all AEs which includes workload intensity, performance levels, current configuration, and so on. After receiving the details the load forecaster analyzes the current workload and forecasts the possible load intensity in the next control interval.

- **Analyze and Plan**: The Analyzer analyzes the given load,waiting queue, present configuration and produces estimates of future performance levels for each AE. The Service Evaluator estimates the performance level of each AE, resource usage cost, and then it evaluates the global utility value based on the SLA. The Optimizer generates a candidate configuration, sends it to the Model

Analyzer, and waits for the evaluated global utility result. Through the optimizing loop, as shown in dashed lines in Figure 3, the Optimizer determines the best configuration with the highest utility value.

- **Virtualized Application Executor**: After getting the best configuration from the Optimizer, the Adaptation Actuator starts to change the allocation of resources such as the service rate of the AEs.

3.3 Virtualized Multi-tier Application Queueing Model

A virtualized multi-tier application in cloud computing environment is deployed on multiple virtual machines (VMs), 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 of AEs but they are provisioned when needed. The load balancer distributes the load to different VMs. Through dynamic capacity provisioning proposed in this paper the cloud system handles the workload variations seen in the cloud environment. Dispatcher of each tier is used for collecting requests processed in the pre-tier and distributing them to multiple 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. Once the result is processed by the final tier T_n, the results are sent back by each tier in the reverse order until it reaches T_1, which then sends the results to the customer. In complex processing scenarios, each request at tier T_i can trigger zero, one or multiple requests to tier $T_i + 1$. For example, a static web page request is processed by the web tier entirely and will not be forwarded to the following tiers. On the other hand, a key word search at a web tier may trigger multiple requests to the next tier. In order to capture the virtualized multi-tier application for dynamic resources provisioning to improve the processing efficiency of the request we need to determine the processing speed of the VMs depending on the waiting queue length. Each tier can be modeled as multiple $GI/M(n)/1/N$ queuing systems with renewal arrivals in which the interarrival and service times are respectively, arbitrarily and exponentially distributed. The general uncorrelated arrival process appears to be more appropriate than the exponential distribution, as the memoryless property of the arrival process does not always meet the need of the application and also it can include the special cases of deterministic, hyperexponential, Erlang, etc.

3.4 Active Monitoring Load Balancer

Active VM Load Balancer maintains an 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 a least loaded VM. Active VM Load Balancer returns the VM id to the Data Center Controller. The data Center Controller sends

the request to the VM identified by that id. Data Center Controller notifies the Active VM Load Balancer of the new allocation.

4 Model Description and Analysis

We consider a finite buffer renewal input state dependent queue. We assume that the inter-arrival times of successive arrivals are independent and identically distributed (i.i.d.) random variables with cumulative distribution function $A(u)$, probability density function $a(u)$, $u \geq 0$, Laplace-Stieltjes transform (LST) $A^*(\theta)$ and mean inter-arrival time $1/\lambda = -A^{*(1)}(0)$, where $h^{(1)}(0)$ denotes the first derivative of $h(\theta)$ evaluated at $\theta = 0$. Service times are assumed to be exponentially distributed random variables with service rates μ_n, $1 \leq n \leq N$, when there are n number of client requests in the system. When the number of client requests in the system are relatively few, we set a lower speed operation period in order to economize operation cost. The traffic intensity is given by $\rho = \lambda/\mu$, where $\mu = \sum_{n=1}^{N} \mu_n/N$ is the mean service rate.

- $N_s(t)$ = Number of client requests present in the system including the one who is in service,
- $U(t)$ = Remaining inter-arrival time for the next request,

In the steady-state, let us define

$$\pi_n(u)du = \lim_{t\to\infty} P\Big\{N_s(t) = n, u \leq U(t) < u + du\Big\}, \quad u \geq 0, \qquad 0 \leq n \leq N.$$

We introduce the following Laplace-Stieltjes transforms:

$$\pi_n^*(\theta) = \int_0^\infty e^{-\theta u}\pi_n(u)du, \quad 0 \leq n \leq N.$$

Let $\pi_n \equiv \pi_n^*(0)$, $0 \leq n \leq N$, where π_n is the probability that there are n client requests in the system when the server is at an arbitrary epoch.
To obtain the system length distributions at arbitrary epochs, we develop the differential difference equations that relate the distribution of number of client requests in the system at the end of service period. For this we use supplementary variable technique and relate the state of the system at two consecutive time epochs t and $t + dt$. Using probabilistic arguments and taking limit as $t \to \infty$, the steady-state differential difference equations can be written as

$$-\frac{d}{du}\pi_0(u) = \mu_1\pi_1(u) \tag{1}$$

$$-\frac{d}{du}\pi_n(u) = -\mu_n\pi_n(u) + \mu_{n+1}\pi_{n+1}(u) + a(u)\pi_{n-1}(0), \ 1 \leq n \leq N-1 \tag{2}$$

$$-\frac{d}{du}\pi_N(u) = -\mu_N\pi_N(u) + a(u)\left(\pi_{N-1}(0) + \pi_N(0)\right), \tag{3}$$

where $\pi_n(0)$ are the respective rates of requests. Multiplying (1) to (3) by $e^{-\theta u}$ and integrating with respect to u from 0 to ∞, yields

$$\begin{aligned} -\theta\pi_0^*(\theta) &= \mu_1\pi_1^*(\theta) - \pi_0(0), && (4)\\ (\mu_n - \theta)\pi_n^*(\theta) &= \mu_{n+1}\pi_{n+1}^*(\theta) + A^*(\theta)\pi_{n-1}(0) - \pi_n(0),\ 1 \le n \le N-1 && (5)\\ (\mu_N - \theta)\pi_N^*(\theta) &= A^*(\theta)\left(\pi_{N-1}(0) + \pi_N(0)\right) - \pi_N(0). && (6) \end{aligned}$$

Adding equations (4) to (6) and simplifying yields

$$\sum_{n=0}^{N}\pi_n^*(\theta) = \frac{1 - A^*(\theta)}{\theta}\sum_{n=0}^{N}\pi_n(0).$$

Taking limit as $\theta \to 0$ and using the normalization condition, we get

$$\sum_{n=0}^{N}\pi_n(0) = \lambda. \quad (7)$$

The left hand side denotes the mean number of entrances into the system per unit time and is equal to mean request rate λ.
Substituting $\theta = \mu_N$ in (6), we get

$$\pi_{N-1}(0) = \frac{1 - A^*(\mu_N)}{A^*(\mu_N)}\pi_N(0). \quad (8)$$

From (6), we have

$$\pi_N^*(\theta) = \left(\frac{A^*(\theta) - A^*(\mu_N)}{A^*(\mu_N)(\mu_N - \theta)}\right)\pi_N(0),\ \theta \neq \mu_N.$$

Setting $\theta = \mu_n$ in (5), we get

$$\pi_{n-1}(0) = \frac{\pi_n(0) - \mu_{n+1}\pi_{n+1}^*(\mu_n)}{A^*(\mu_n)}, n = N-1, \ldots, 1$$

From (5), we obtain

$$\pi_n^*(\theta) = \frac{\mu_{n+1}\pi_{n+1}^*(\theta) + A^*(\theta)\pi_{n-1}(0) - \pi_n(0)}{(\mu_n - \theta)},\ \theta \neq \mu_N,\ n = N-1, \ldots, 1. \quad (9)$$

For $\theta = \mu_n$, $\pi_n^*(\theta)$ are given by

$$\begin{aligned} \pi_N^*(\theta) &= -A^{*(1)}(\theta)\left(\pi_{N-1}(0) + \pi_N(0)\right), && (10)\\ \pi_n^*(\theta) &= -\left(\mu_{n+1}\pi_{n+1}^{*(1)}(\theta) + A^{*(1)}(\theta)\pi_{n-1}(0)\right),\ 1 \le n \le N-1. && (11) \end{aligned}$$

It can be seen from the above set of expressions that we can easily evaluate $\pi_n(0)$ $(0 \le n \le N)$.

4.1 Relation between steady-state distribution at arbitrary and pre-arrival epochs

Let $\pi_n^-,\ 0 \leq n \leq N$ denote the pre-arrival epoch probability, that is, an arrival sees n client requests in the system at an request epoch. Applying Bayes' theorem, we have

$$\pi_n^- = \lim_{t\to\infty} \frac{P[N_s(t) = n, U(t) = 0]}{P[U(t) = 0]}.$$

Further, using (7) in the above expression, we obtain

$$\pi_n^- \quad = \quad \frac{\pi_n(0)}{\lambda},\ 0 \leq n \leq N. \tag{12}$$

Setting $\theta = 0$ in the equations (5) - (6) and using (12), after simplification we obtain

$$\pi_n \quad = \quad \frac{\lambda}{\mu_n}\pi_{n-1}^-,\quad 1 \leq n \leq N. \tag{13}$$

Using the normalization condition,

$$\pi_0 \quad = \quad 1 - \sum_{n=1}^{N} \pi_n. \tag{14}$$

It can be seen from the above set of expressions that once we know the pre-arrival epoch probabilities, the arbitrary epoch probabilities can be easily computed.

Remark 1: Results for $GI/M/c/N$ from $GI/M(n)/1/N$ can be obtained by taking $\mu_n = n\mu, 1 \leq n \leq c-1$ and $\mu_n = c\mu, n \geq c$.
Remark 2: Setting $\mu_n = \mu,\ \forall\ n = 1, \ldots, N$, the model reduces to $GI/M/1/N$ queue and results match with the results available in literature.

5 Computation of state probabilities

In this section, we present a computational algorithm to compute pre-arrival epoch and the arbitrary epoch probabilities at steady-state. The algorithm is based on the analysis of Section 4, that is, we compute all probabilities $\pi_n(0),\ 0 \leq n \leq N$ in terms of $\pi_N(0)$. We determine $\pi_N(0)$ using equation (7). After computing the probabilities $\pi_n(0)$, we can evaluate pre-arrival epoch and the arbitrary epoch probabilities.
Step 1: For $n = 0, 1, \ldots, N$, calculate $\pi_n(0)$ in terms of $\pi_N(0)$ as follows

$$\pi_n(0) \quad = \quad \psi_n \pi_N(0),\ 0 \leq n \leq N, \tag{15}$$
$$\pi_n^*(\theta) \quad = \quad \zeta_{n,\theta}\pi_N(0),\ 1 \leq n \leq N, \tag{16}$$

where ψ_n and $\zeta_{n,\theta}$ are computed as follows.

- Calculate ψ_n as follows

$$\begin{aligned} \psi_N &= 1,\ \psi_{N-1} = \frac{1 - A^*(\mu_N)}{A^*(\mu_N)}, \\ \psi_{n-1} &= \frac{\psi_n - \mu_{n+1}\zeta_{n+1,\mu_n}}{A^*(\mu_n)},\quad n = N-1, \ldots, 1. \end{aligned} \tag{17}$$

- Calculate $\zeta_{n,\theta}$ as follows

if $n = N$ **then**
  if $\theta = \mu_N$ **then**
    $\zeta_{N,\theta} = -A^{*(1)}(\theta)\,(\psi_{N-1} + \psi_N)$
  else
    $\zeta_{N,\theta} = \frac{A^*(\theta)(\psi_{N-1}+\psi_N)-\psi_N}{\mu_N-\theta}$
  end if
end if
if $1 \le n \le N-1$ **then**
  if $\theta = \mu_n$ **then**
    $\zeta_{n,\theta} = -\left(\mu_{n+1}\zeta^{*(1)}_{n+1,\theta} + A^{*(1)}(\theta)\psi_{n-1}\right)$
  else
    $\zeta_{n,\theta} = \frac{\mu_{n+1}\zeta_{n+1,\theta}+A^*(\theta)\psi_{n-1}-\psi_n}{\mu_n-\theta}$
  end if
end if

- Calculate $\zeta^{(l)}_{n,\theta}$ as follows

if $n = N$ **then**
  if $\theta = \mu_N$ **then**
    $\zeta^{(l)}_{N,\theta} = -\frac{A^{*(l+1)}(\theta)(\psi_{N-1}+\psi_N)}{l+1}$
  else
    $\zeta^{(l)}_{N,\theta} = \frac{A^{*(l)}(\theta)(\psi_{N-1}+\psi_N)+l\zeta^{(l-1)}_{N,\theta}}{\mu_N-\theta}$
  end if
end if
if $1 \le n \le N-1$ **then**
  if $\theta = \mu_n$ **then**
    $\zeta^{(l)}_{n,\theta} = -\frac{\mu_{n+1}\zeta^{(l+1)}_{n+1,\theta}+A^{*(l+1)}(\theta)\psi_{n-1}}{l+1}$
  else
    $\zeta^{(l)}_{n,\theta} = \frac{\mu_{n+1}\zeta^{(l)}_{n+1,\theta}+A^{*(l)}(\theta)\psi_{n-1}+l\zeta^{(l-1)}_{n,\theta}}{\mu_n-\theta}$
  end if
end if

Step 2: Determine $\pi_N(0)$ from equation (7) as
$\pi_N(0) = \lambda\left[\sum_{n=0}^{N}\psi_n\right]^{-1}$.

Step 3: Compute pre-arrival epoch probabilities π_n^- from equation (12) as

$$\pi_n^- = \frac{1}{\lambda}\pi_n(0),\ 0 \le n \le N.$$

Step 4: The arbitrary epoch probabilities π_n are determined by equation (13). The computational complexity of the given algorithm is $O(N^3)$, where N is the maximum capacity of the system.

6 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, performance measures of the queueing system can be computed. The average number of requests in the system (L_s) and the average number of requests in the queue (L_q) respectively, are given by

$$L_s = \sum_{n=1}^{N} n\pi_n; \quad L_q = \sum_{n=2}^{N}(n-1)\pi_n.$$

The probability of loss or blocking is P_{loss}=π_N^-. Using Little's rule, the average waiting time of a task in the system (W_s) and the average waiting time of a task in the queue (W_q) respectively, are given by

$$W_s = L_s/\lambda^{'}, \quad W_q = L_q/\lambda^{'},$$

where $\lambda^{'} = \lambda(1 - P_{loss})$ is the effective request rate.

Waiting time

We obtain the Laplace-Stieltjes transform (LST) of waiting time distribution of a task who is accepted in the system. If $W(x)$ be the actual waiting time distribution (in the system) of a task which is accepted in the system and let $W^*(\theta)$ be its LST then considering various possible cases, we have

$$W^*(\theta) \quad = \quad \frac{1}{1-P_{loss}} \sum_{n=0}^{N-1} \pi_n^- \left(\frac{\mu_{n+1}}{\mu_{n+1}+\theta}\right)^{n+1}$$

From this expression one can easily obtain mean waiting time in the system which is given by

$$W_s = -W^{*(1)}(0) = \frac{1}{1-P_{loss}} \sum_{n=0}^{N-1} \pi_n^- \Big(\frac{n+1}{\mu_{n+1}}\Big).$$

Remark 3: The average number of requests in the system (L_s) is given by

$$L_s = \sum_{n=1}^{N} n\pi_n = \sum_{n=1}^{N} n\frac{\lambda}{\mu_n}\pi_{n-1}^- = \lambda \sum_{n=0}^{N-1} \frac{(n+1)}{\mu_{n+1}}\pi_n^- = \frac{\lambda^{'}}{1-P_{loss}} \sum_{n=0}^{N-1} \frac{(n+1)}{\mu_{n+1}}\pi_n^- = \lambda^{'} W_s$$

Thus, the Little's formula $L_s = \lambda^{'} W_s$ is established.

7 Numerical Illustrations

In this section some numerical results are presented. A computational program is developed by using MATLAB. Figure 4 depicts the mean number of tasks in the queue (L_q) on request rate λ for the exponential distribution. As can be seen mean number of tasks in the queue increases smoothly when the buffer size increases. When the service rate (μ_i) is fixed the number of tasks in the buffer (L_q) increases more rapidly as compared to the variable service rate. So one can tune up the service rate to minimize the waiting of client requests in the finite buffer. We see that it would be better off by adopting cloud service if its client requests is smaller, thus cloud service is generally attractive to small to medium businesses.

The impact of buffer size N of the virtual machines on blocking probability (P_{loss}) is shown in Figure 5 for various inter-arrival time distributions with same mean. We observe that for all distributions considered here, the blocking probability decreases rapidly when the buffer size increases. As can be seen, the blocking probability in the case of Hyperexponential (HE_2) distribution is higher as compared to deterministic, Erlang-2 (E_2) and exponential distributions. Again one may observe that the deterministic distribution yields the lowest blocking probability. From the figure, we can estimate the minimum buffer size required to keep the blocking probability below a given threshold value.

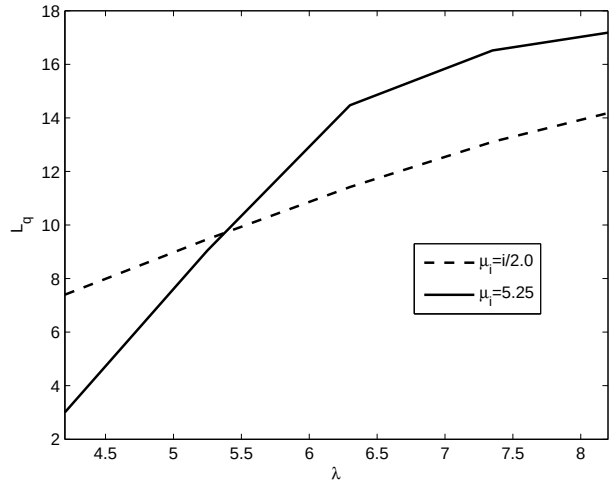

Figure 4. Impact of λ on L_q.

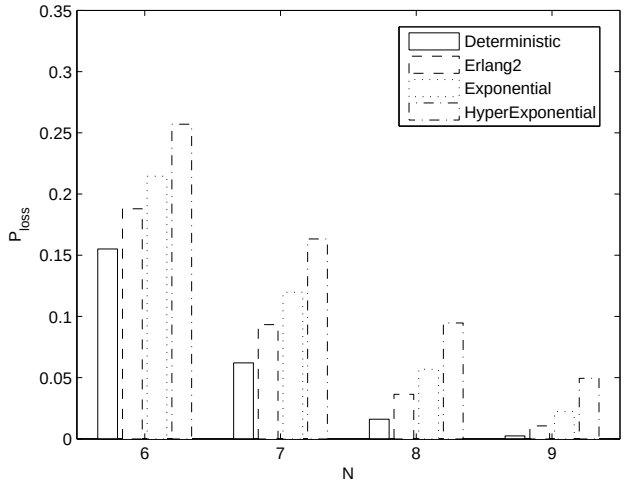

Figure 5. Effect of N on blocking probability.

Figure 6 depicts the impact of client request λ on the average waiting time of the client request W_q in the buffer for various inter-arrival time distributions with same mean. As one would intuitively expect, it is observed that waiting time in the queue increases as client request increases. Because the more client request will remain in the queue until all the required servers become idle, consequently the more the client request is resulted in the longer the response time. It can be seen that the average waiting time of the client request in the case of Hyperexponential (HE_2) distribution is higher as compared to deterministic, Erlang-2 (E_2) and exponential distributions.

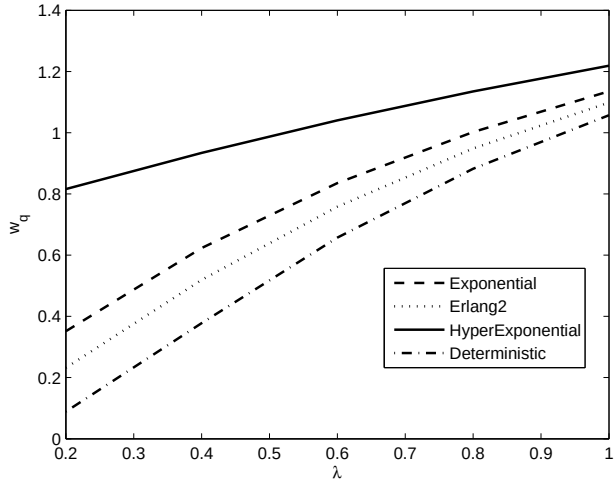

Figure 6. Impact of λ on W_q.

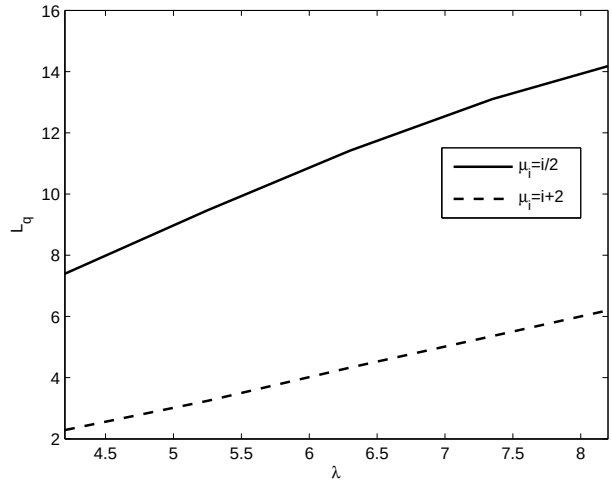

Figure 7. Impact of λ on L_q.

Figure 7 shows the impact of λ on L_q for various service rates when inter-arrival time is exponentially distributed. It can be seen that as request rate λ increases the average number of client requests in the buffer increases. When $\mu_i = i+2$ the number of client requests waiting in the buffer is very less as compared to the the number of client requests waiting in the buffer when the service rate is $\mu_i = i/2$.

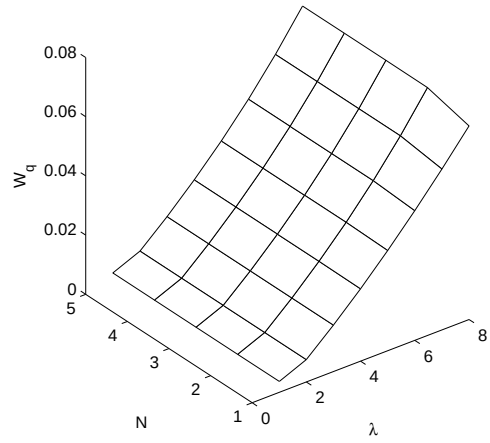

Figure 8. The W_q for different values of N and λ.

The variation in the queueing delay for different values of the request rate and the buffer size N is shown in Figure 8, when the interarrival time is exponentially distributed. We varied the request rate λ from 0.1 to 1.4, while the buffer size N is

varied from 4 to 20 and $\mu = 0.2$. It is observed that for fixed request rate the average waiting-time increases when the buffer size N increases. Further with fixed buffer size N, the average waiting-time increases when the request rate increases. Therefore, we can define an admissible region in terms of the request rate λ and buffer size N so that an acceptable queueing delay can be guaranteed.

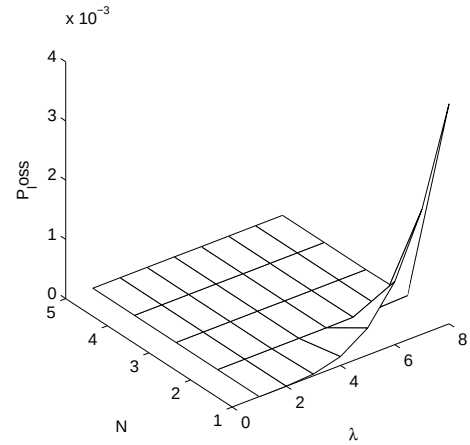

Figure 9. The blocking probability for different values of N and λ.

Figure 9 illustrates dependence of the blocking probability on the buffer size N varying from 4 to 20 and the request rate λ varying from 0.1 to 1.4. The interarrival time is assumed to be exponential with $\mu = 0.2$. It is observed that for fixed request rate the blocking probability increases when the buffer size N increases. Further with fixed buffer size N the blocking probability increases when the request rate increases. To accomplish this, we can carefully setup the request rate and the buffer size N in the system in order to ensure the minimum blocking probability.

8 Conclusion

The dynamic provisioning of virtualized multi-tier applications for cloud environment is a new challenge which has not been addressed by prior work on provisioning techniques. In this paper, we proposed an optimal autonomic virtual machine provisioning architecture for 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 produced by different manufacturers to minimize the total number of VMs for the requirement of requests. The objective is to improve the efficiency and flexibility in cloud environment for resource provisioning. We further integrated load prediction method technique to fit our workload characteristics. To achieve significant performance level, we adopted Service Level

Agreement (SLA) based negotiation of prioritized applications to determine the costs and penalties.

We have developed a recursive method, using the supplementary variable technique and treating the remaining inter-arrival time as the supplementary variable, to find the steady-state system length distributions at pre-arrival and arbitrary epochs. The recursive method is powerful and easy to implement. Various performance indicators such as blocking probability, request waiting time and number of tasks in the system and in the queue have been obtained.

References

[1] Ardagna D., Trubian M., Zhang L., SLA based profit optimization in multi-tier systems, *Proceedings of the 4th IEEE Inyternational Symposium on Network Computing and Applications*, 2005, 263-266.

[2] Bi J.,Zhu Z., Tian R., Wang Q., Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center,*Proceedings of the Third IEEE International Conference on Cloud Computing*, 2010, 370-377.

[3] Buyya R., Yeo C. S.,Venugopal S., An architectural approach to autonomic computing, *Future Generation Computer Systems*, 25, 6, 2009, 599-616.

[4] Chen Y., Iyer S., Liu X., SLA decomposition: Translating service level objectives to system level thresholds, *Proceedings of the 4th International Conference on Autonomic Computing*, 2007, 3-15.

[5] Jung G., Joshi K.R., Hiltunen M.A., Generating adaptation policies for multi-tier applications in consolidated server environments,*Proceedings of the 5th International Conference on Autonomic Computing*, 2008, 23-32.

[6] Kamra A.,Misra V., Nahum E., Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites, *Proceedings of International Workshop on Quality of Service*, 2004, 47-58.

[7] Kundu A., Banerjee A.D., Saha P., Introducing New Services in Cloud Computing Environment, *International Journal of Digital Content Technology and its Applications*, 4, 5, 2010, 143-152.

[8] Reddy K.V., Rao B., Reddy L.S.S.,Kiran P.S., Research Issues in Cloud Computing, *Global Journal of Computer Science and Technology*, 11, 11, 2011, 59-64.

[9] Urgaonkar B., Pacifici G.,Shenoy P., 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*, 2005, 291-302.

[10] Urgaonkar B., Shenoy P., Chandra A., Agile dynamic provisioning of multi-tier Internet application, *ACM Transactions on Autonomous and Adaptive Systems*, 3, 1, 2008, 1-39.

[11] Wang X., Du Z., Chen Y., Li S., Virtualization based autonomic resource management for multi-tier Web applications in shared data center, *The Journal of Systems and Software* 81, 9, 2006, 1591-1608.

[12] White S.R.,Hanson J.E., Whalley I., An architectural approach to autonomic computing, *Proceedings of the First IEEE International Conference on Autonomic Computing*, 2004, 2-9.

[13] Zhang A., Santos P., Beyer D., Optimal server resource allocation using an open queueing network model of response time, *HP Labs Technical Report*, 2001, 1-17.

Received December, 2012