# Chapter 6

# Dynamic Resource Provisioning in Multi-tier Applications

## 6.1 Introduction

Most of the complex internet applications deployed in large cloud data centers are implemented on multi-tier architectures [75, 76]. The multi-tier architecture style has become an industry standard in which each tier provides certain functionality, employs services from the previous tiers and forwards a defined service to the next tiers. In comparison to the single tier applications the problem of resource allocation in multi-tier applications is harder as the tiers are not homogenous and a performance bottleneck in one tier may reduce the overall performance of the system thus violating the service level agreement [9]. For the efficient utilization of resources under variable workload, adaptive self-managing techniques are required to dynamically assign resources [77]. The major issue of holding the QoS is the high variability of the workload, which makes it hard to figure out the resource requirement in advance [78]. In this chapter, we analyze the performance of the VMs by dynamically increasing the mean service rate of the VMs to avoid congestion in the multi-tier environments.

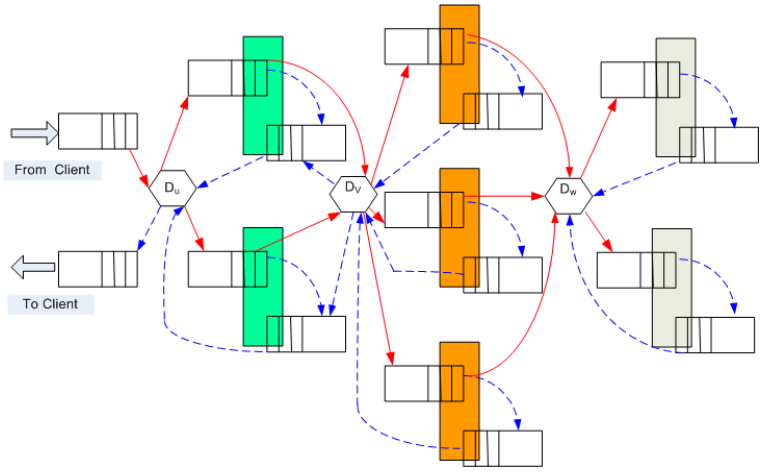


Figure 6.1: A typical 3-tier application in cloud.

## 6.2 Architecture Overview

Figure 6.1 shows a typical 3-tier web application deployed in a cloud. In the figure, $D_*$ stands for the request dispatchers in different tiers. The solid lines represent forward requests while the dash lines represent backward requests.

The architecture of a shared data center is shown in Figure 6.2, 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 [79]. The *Dispatcher* dispatches the requests of different applications to the corresponding environment in a shared data center. In each AE, a *sentry* node obtains requests from the dispatcher, rejects excessive requests when the local AE is found to be overloaded in order to meet the QoS and preserves the system stability. Since the workloads of separate AEs vary with time, the global resource manager may decide to alter the capacity of different VMs to deal with varying workload and to avoid congestion. Each tier holds a VMM behaving as the resource allocation actuator, which assigns variable resource capacity to multiple VMs within the same tier.

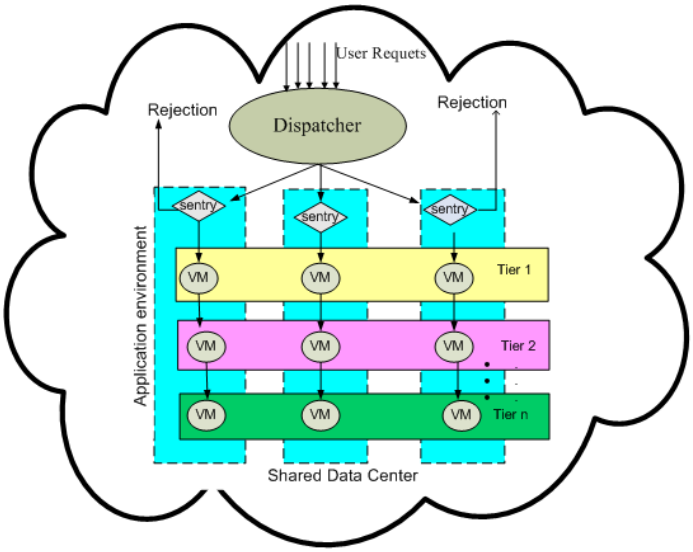


Figure 6.2: Data Center architecture.

### 6.2.1 Cloud Computing Infrastructure

Self-managing techniques, such as Supervise, Examine, Design and Execute (SEDE) [80] command loops are needed to dynamically provision the resources for virtualized multi-tier application execution environments (VAEEs) of different customers. Figure 6.3 provides an adaptive self-controlling architecture, in which a common physical infrastructure is shared by a multiple AEs. Periodically, the whole system checks its own status, judges the overall efficiency, and then adopts itself to the workload variation after each control interval. The components are organized according to the *Supervise -Examine-Design-Execute* pattern reported in the architectural approach to autonomic computing [81], as follows.

**Pool of Resources** contain physical and virtualized resources. A batch of virtual machines hold several VAEEs sharing the physical resources and can insulate various applications from the hardware.

**Self-management community** automates the virtual machines for main-

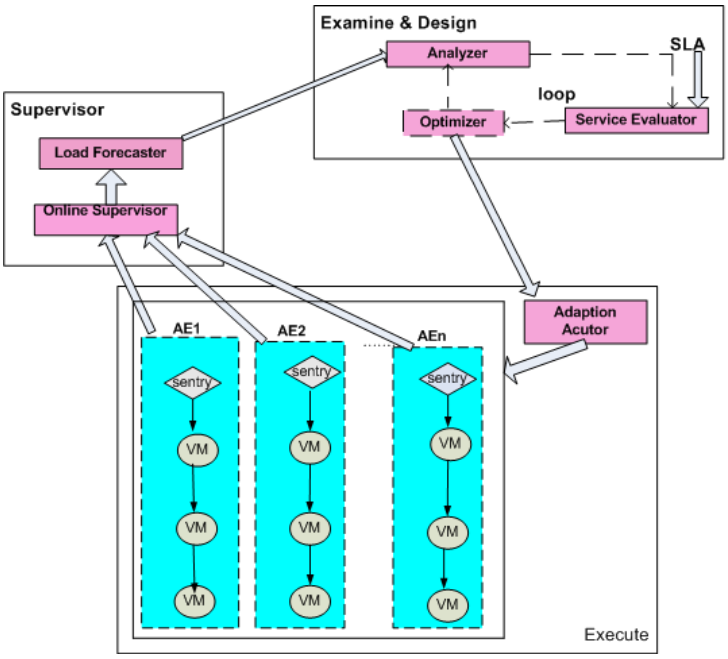


Figure 6.3: The dynamic resource provisioning of cloud data center.

taining the response time requirements of the customers. The components of self-management community are as follows:

- **Supervisor**: The supervisor gathers the relevant information of the running states such as workload intensity, performance levels, current configuration, etc., of the application environments. After receiving the details the load forecaster analyzes the current workload and forecasts the intensity of load in the next control interval.

- **Examine and Design**: The Analyzer analyzes the given load, waiting queue, present configuration and produces estimates of future performance levels for each application environment. 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 6.3 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 updating the allocation of resources of the application environments.

### 6.2.2 Virtualized Multi-tier Application Queueing Model

In cloud computing environment, a virtualized multi-tier application is deployed on multiple virtual machines. Let us assume that each tier of application environment has $c$ parallel identical VMs. Dispatcher of each tier collects the requests processed in the previous tier and distributes them to multiple parallel VMs of the tier to execute. Each tier employs a load-balancing component which is responsible for the client requests processed in the tier. The requests may be forwarded to the succeeding tier for processing. After processing the request in the final tier $T_n$, the results are sent back to the customer in the reverse order through the tier $T_1$. In complex processing scenarios, each request at tier $T_i$ can trigger zero or multiple requests to tier $T_{i+1}$. For instance, a client request for a static web page may be processed by the web tier only and it may not be forwarded to the other tiers for processing. A key word search may spark multiple requests to the next successive tiers. We need to determine the required processing speed of the VMs based on the waiting queue length to improve the processing efficiency. We adopt $GI/M(n)/1/N$ queuing system to model each tier.

### 6.2.3 Active Monitoring Load Balancer

ActiveVMLoadBalancer maintains information about each VMs and the number of requests currently allocated to each VM . When a request to allocate a new VM arrives, it identifies the least loaded VM. If there are more than one VMs with least load, the first identified VM is selected. ActiveVMLoadBalancer returns the VM id to the datacenter controller. The data center

controller sends request to the identified VM and notifies the ActiveVMLoad-Balancer of the new allocation.

## 6.3 Model Description and Analysis

We develop the differential difference equations by relating the state of the system at two consecutive time epochs $t$ and $t + dt$. Using supplementary variable techniques and taking limit as $t \to \infty$, we obtain the steady-state differential difference equations as

$$\begin{aligned} -\frac{d}{du}\pi_{0,0}(u) &= \mu_1\pi_1(u) && (6.1)\\ -\frac{d}{du}\pi_n(u) &= -\mu_n\pi_n(u) + \mu_{n+1}\pi_{n+1}(u) + a(u)\pi_{n-1}(0), \\ &\quad 1 \le n \le N-1 && (6.2)\\ -\frac{d}{du}\pi_N(u) &= -\lambda_N\pi_N(u) + a(u)(\pi_{N-1}(0) + \pi_N(0)), && (6.3) \end{aligned}$$

where $\pi_n(0)$ are the respective rates of arrivals. Multiplying (6.1) to (6.3) by $e^{-\theta u}$ and integrating with respect to $u$ from 0 to $\infty$, yields

$$\begin{aligned} -\theta\pi_0^*(\theta) &= -\mu_1\pi_1^*(\theta) - \pi_0(0), && (6.4)\\ (\mu_n - \theta)\pi_n^*(\theta) &= \mu_{n+1}\pi_{n+1}^*(\theta) + A^*(\theta)\pi_{n-1}(0) \\ &\quad -\pi_n(0),\ 1 \le n \le N-1 && (6.5)\\ (\mu_N - \theta)\pi_N^*(\theta) &= A^*(\theta)(\pi_{N-1}(0) + \pi_{N-1}(0) \\ &\quad +\pi_N(0). && (6.6) \end{aligned}$$

Adding equations (6.4) to (6.6) and simplifying yields

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

Using the normalization condition and taking limit as $\theta \to 0$, we get

$$\sum_{n=0}^{N}\pi_n(0) = \lambda. \quad (6.7)$$

The left hand side denotes the mean number of arrivals into the system per unit time.
Substituting $\theta = \mu_N$ in (6.6), we get

$$\pi_{N-1}(0) = \frac{1 - A^*(\mu_N)}{A^*(\mu_N)}\pi_N(0). \tag{6.8}$$

From (6.6), we have

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

Setting $\theta = \mu_n$ in (6.5), 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 (6.5), we obtain

$$\begin{aligned}\pi_n^*(\theta) &= \frac{\mu_{n+1}\pi_{n+1}^*(\theta) + A^*(\theta)\pi_{n-1}(0) - \pi_n(0)}{(\mu_n - \theta)}, \\ &\quad \theta \neq \mu_N,\ n = N-1, \ldots, 1.\end{aligned} \tag{6.9}$$

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

$$\pi_N^*(\theta) = -A^{*(1)}(\theta)\left(\pi_{N-1}(0) + \pi_N(0)\right), \tag{6.10}$$

$$\begin{aligned}\pi_n^*(\theta) &= -\left(\mu_{n+1}\pi_{n+1}^{*(1)}(\theta) + A^{*(1)}(\theta)\pi_{n-1}(0)\right), \\ &\quad 1 \leq n \leq N-1.\end{aligned} \tag{6.11}$$

We can easily evaluate $\pi_n(0)$ $(0 \leq n \leq N)$ from the above set of expressions [82].

### 6.3.1 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 $n$ customers in the system at an arrival 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 (6.7) in the above expression, we obtain

$$\pi_n^- = \frac{\pi_n(0)}{\lambda},\ 0 \leq n \leq N. \tag{6.12}$$

Setting $\theta = 0$ in the equations (6.5) - (6.6) and using (6.12), after simplification we obtain

$$\pi_n = \frac{\lambda}{\mu_n}\pi_{n-1}^-,\quad 1 \leq n \leq N. \tag{6.13}$$

Using the normalization condition,

$$\pi_0 = 1 - \sum_{n=1}^{N} \pi_n. \tag{6.14}$$

The arbitrary epoch probabilities may be easily computed once the pre-arrival epoch probabilities are known from the above set of expressions.

**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: $\mu_n = \mu,\ \forall\ n = 1, \ldots, N$. The model reduces to $GI/M/1/N$ queue and results match with the results available in literature.

## 6.4 Computational algorithm

This section presents a computational algorithm at steady-state for pre-arrival epoch and arbitrary epoch probabilities. The algorithm is based on the analysis of Section 2, 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 (6.7). After computing the probabilities $\pi_n(0)$, we can evaluate pre-arrival epoch and the arbitrary epoch probabilities. The computational algorithm has computational complexity of order $N^3$, where $N$ is the buffer size.
**Step 1**: For $n = 0, 1, \ldots, N$, calculate $\pi_n(0)$ in terms of $\pi_N(0)$ as follows

$$\pi_n(0) = \psi_n \pi_N(0),\ 0 \leq n \leq N, \tag{6.15}$$

$$\pi_n^*(\theta) = \zeta_{n,\theta}\pi_N(0),\ 1 \leq n \leq N, \tag{6.16}$$

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

- Calculate $\psi_n$ as follows

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

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

**if** $n = N$ **then**
  **if** $\theta = \mu_N$ **then**
    $\zeta_{N,\theta} = -A^{*(1)}(\theta)\left(\psi_{N-1} + \psi_N\right)$
  **else**
    $\zeta_{N,\theta} = \frac{A^*(\theta)(\psi_{N-1}+\psi_N)-\psi_N}{\mu_N-\theta}$
  **end if**
**else 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**
**else 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 (6.7) as
$\pi_N(0) = \lambda \left[\sum_{n=0}^{N} \psi_n\right]^{-1}$.

**Step 3**: Compute pre-arrival epoch probabilities $\pi_n^-$ from equation (6.12) as

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

**Step 4**: The arbitrary epoch probabilities $\pi_n$ are determined by equation (6.13).

The computational complexity of the given algorithm is $O(N^3)$, where $N$ is the maximum capacity of the system.

## 6.5 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, performance measures of the queueing system can be computed. The average number of customers in the system ($L_s$) and the average number of customers in the queue ($L_q$) 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 (blocking) is $P_{loss} = \pi_N^-$. Using Little's rule, the average waiting time of a customer in the system ($W_s$) and the average waiting time of a customer 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 arrival rate.

## 6.6 Numerical Illustrations

Some numerical illustrations are discussed in this section. Using MATLAB we have developed a computational program. Figure 6.4 depicts the average

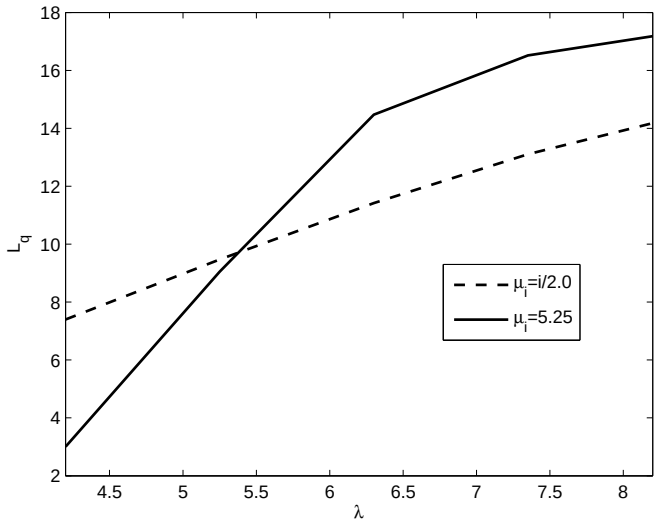


Figure 6.4: Impact of $\lambda$ on $L_q$ for fixed and variable service rate.

number of customers ($L_q$) in the system versus arrival rate $\lambda$ for the exponential distribution. In the darken line the service rate has been fixed to 5.25. It is seen that the average number of customer requests ($L_q$) in the buffer increases with the arrival rate $\lambda$. But when the service rate ($\mu_i$) is fixed the number of customers 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.

Figure 6.5 shows the blocking probability ($P_{loss}$) versus the buffer size $N$ for different distributions. The figure shows the blocking probability decreases as the buffer size of the VM increases. Figure 6.6 is a bar graph for the buffer size $N$ Vs the blocking probability ($P_{loss}$). Figure 6.7 shows the impact of $\lambda$ on the waiting time of the client request $w_q$ in the buffer for different distributions. It is shown from the figure that for all the distributions the waiting time in the buffer increases as $\rho$ increases.

Figure 6.8 shows the impact of $\lambda$ on $L_q$ for the exponential distribution for various service rates.From the figure it is shown that for a fixed arrival rate $\lambda$ the number of client requests waiting in the buffer is higher for a slower

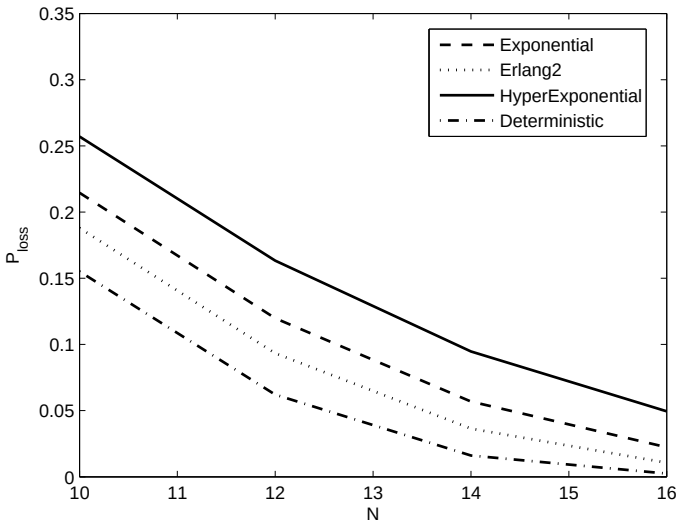


Figure 6.5: Impact of $N$ on $P_{loss}$ for different distributions.

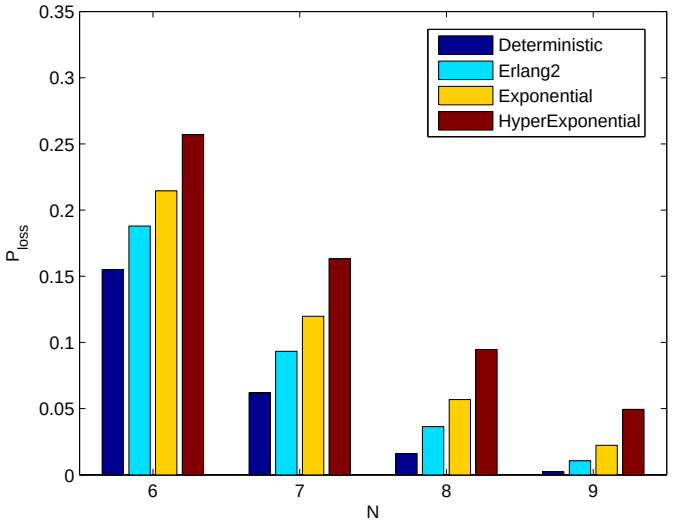


Figure 6.6: Impact of $N$ on $P_{loss}$ for different distributions.

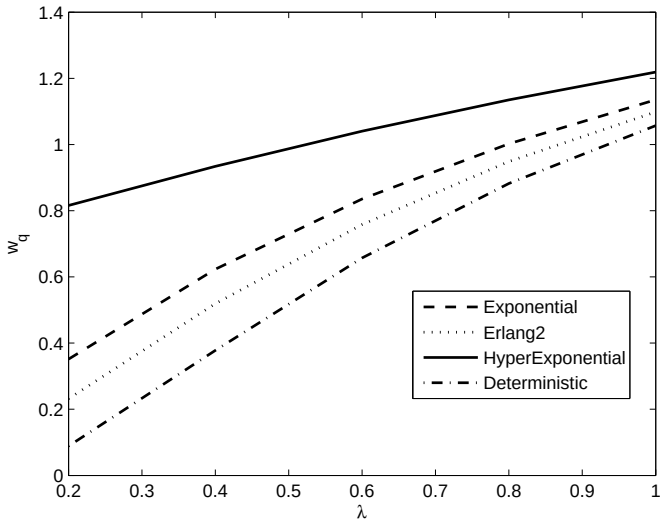


Figure 6.7: Impact of $\lambda$ on $w_q$ for different distributions.

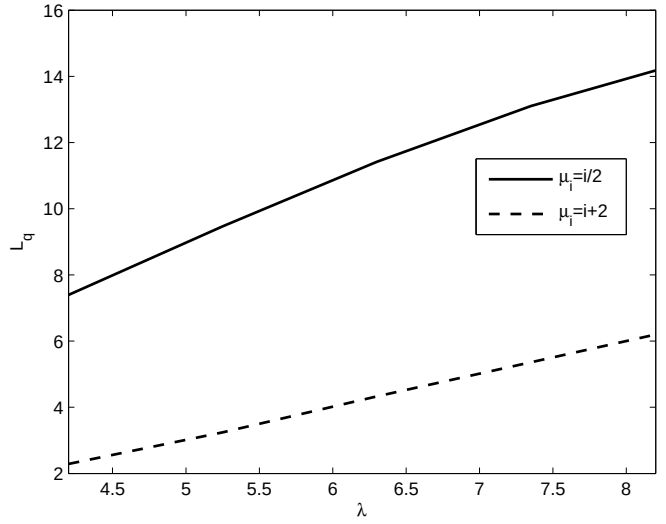


Figure 6.8: Impact of $\lambda$ on $L_q$ for different service rates.

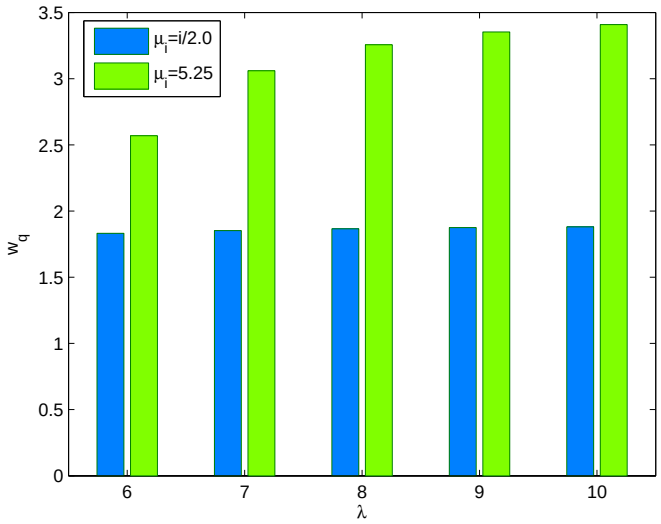


Figure 6.9: $\lambda$ Vs $W_q$ for constant and variable service rate.

service rate $\mu$. 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 6.9 compares the waiting time in the queue ($w_q$) by considering constant service rate $\mu$=5.25 and variable service rate $\mu_i$=i/2.0 for exponential distribution. One can observe that for fixed as well as variable service rate as $\lambda$ increases $w_q$ is also increased. But waiting time in the buffer $w_q$ is always less for variable service rate as compared to fixed service rate. So the cloud controller can adjust the service rate to decrease the waiting time in the system buffer.

The variation in the queueing delay for different values of the arrival rate and the buffer size $N$ is shown in Figure 6.10, when the inter arrival time is exponentially distributed. We varied the arrival rate $\lambda$ 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 arrival 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 arrival rate increases. Therefore, we can define an admissible region in terms of the arrival rate $\lambda$ and buffer size $N$ so

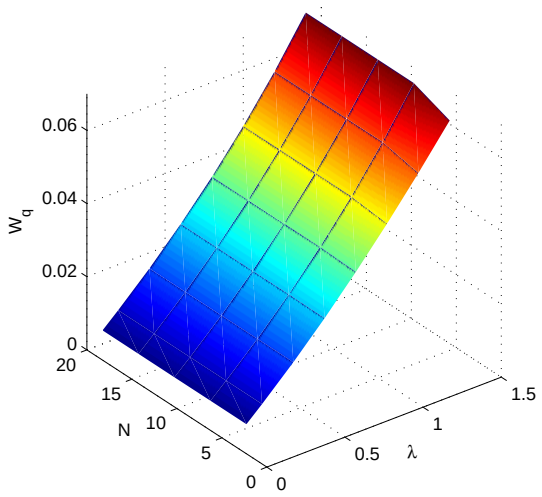


Figure 6.10: The $W_q$ for different values of $N$ and $\lambda$.

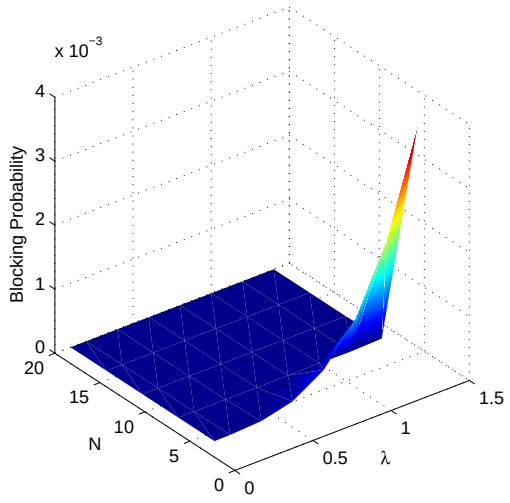


Figure 6.11: The blocking probability for different values of $N$ and $\lambda$.

that an acceptable queueing delay can be guaranteed. Figure 6.11 illustrates dependency of the blocking probability on the buffer size $N$ varying from 4 to 20 and the arrival rate $\lambda$ varying from 0.1 to 1.4. The inter arrival time is assumed to be exponential with $\mu = 0.2$. It is observed that for fixed arrival rate the blocking probability increases when the buffer size $N$ increases. Further with fixed buffer size $N$ the blocking probability increases when the arrival rate increases. To accomplish this, we can carefully setup the arrival rate and the buffer size $N$ in the system in order to ensure the minimum blocking probability.

## 6.7 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 chapter, 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 formulated a recursive method, applying the supplementary variable method and treating the rest inter-arrival time as the supplementary variable, to find the steady-state system length distributions at pre-arrival and arbitrary epochs [83]. 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.