# Provisioning Multi-tier Cloud Applications Using Statistical Bounds on Sojourn Time

*Upendra Sharma* *Prashant Shenoy* *Don Towsley*
University of Massachusetts Amherst
{*upendra, shenoy, towsley*}@cs.umass.edu

## Abstract

*In this paper we present a simple and effective approach for resource provisioning to achieve a percentile bound on the end to end response time of a multi-tier application. We, at first, model the multi-tier application as an open tandem network of M/G/1-PS queues and develop a method that produces a near optimal application configuration, i.e, number of servers at each tier, to meet the percentile bound in a homogeneous server environment – using a single type of server. We then extend our solution to a K-server case and our technique demonstrates a good accuracy, independent of the variability of service-times. Our approach demonstrates a provisioning error of no more than 11% as compared to a 196% worst case provisioning error obtained by techniques based on an M/M/1-FCFS queue model. In addition, we extend our approach to handle a heterogenous server environment, i.e., with multiple types of servers. We find that fewer high-capacity servers are preferable for high percentile provisioning. Finally, we extend our approach to account for the rental cost of each server-type and compute a cost efficient application configuration with savings of over 80%. We demonstrate the applicability of our approach in a real world system by employing it to provision the two tiers of the java implementation of TPC-W – a multi-tier transactional web benchmark that represents an e-commerce web application, i.e. an online bookstore.*

## 1 Introduction

Cloud computing platforms are becoming increasingly popular for hosting enterprise applications due to their ability to support dynamic provisioning of virtualized resources to handle workload fluctuations and also because of usage based pricing. Enterprise applications are known to observe dynamic workload and provisioning correct capacity for these applications remains an important and challenging problem. High variability in workload is caused by a variety of reasons, such as flash crowds, short term sustained surges, or long-term fluctuations based on change in business or underlying IT infrastructure etc. Predicting these workload fluctuations or the peak workload is challenging. Erroneous predictions often lead to under-utilized systems or in some situations cause temporarily outage of an otherwise well provisioned web-site; for e.g. in November 2000 Amazon.com site suffered a forty-minute outage due to overload. Consequently, rather than provisioning server capacity to handle infrequent (and hard to predict) peak workloads, an alternate approach of dynamically provisioning capacity on-the-fly in response to workload fluctuations has become popular. Dynamic provisioning is especially well suited to the cloud due to the ability of cloud platforms to provision capacity when needed and charge for usage on pay-per-use basis.

There have been numerous efforts that have addressed the issue of dynamic provisioning of server capacity to distributed applications [10, 13, 21, 24, 23] . These efforts fall into two categories - *proactive*, where a model of the application is used to compute the capacity needed to service a particular workload at a certain performance level, or *reactive*, where additional capacity is allocated *after* a workload spike arrives and causes significant performance degradation.

In case of proactive approaches, application models have been derived to predict how much capacity is needed to provide a certain *mean* response time for a given workload [23, 24]. However, typical service level agreement (SLAs) for the application are specified in terms of the worst case (or peak) response times [9] (e.g. 99% of the requests should see no more than a 1-sec response time). Consequently, there is a mismatch between the provisioning models which allocate capacity for a target mean response, time and the SLA, which dictates that the capacity should be allocated based on a high percentile (peak) response time.

Second, many enterprise applications are distributed or replicated with multi-tier architecture. Typically SLAs are specified on an end-to-end basis for the entire application. The few provisioning efforts that focus on allocating capacity for the tail of the work translate it to per-tier SLA metrics [24]; doing provisioning for per-tier SLA metric can result in large errors in the capacity provisioning if the tier response times estimates are incorrect.

Third, most provisioning techniques to-date are cost oblivious – they can determine *how much* server capacity to allocate but do not consider the *cost* of allocating the server capacity. In cloud platform, different types of server are available at different prices. For example, small 1-core server is 8.5c/hr, large 4-core costs 34c/hr. The capacity does not scale linearly across configurations and nor does the price [20]. Thus it may be possible to provision a certain capacity C for an application in many different ways - using different types of server (e.g. 4 small or 2 large) which incur different costs for provisioning the same amount of capacity. A cloud specific provisioning scheme must take the cloud costs into account when making provisioning decisions.

In this work we present a new model driven provisioning approach targeted for cloud platforms. Our approach focuses on i.) allocating capacity based on peak (high percentile) of the workload, ii) takes a holistic view of the entier multi-tier application by considering bounds on on end to end response times while making provisioning decisions and iii) takes cloud server configs and pricing models when determining the most cost effective config to provision a certain amount of capacity.

### 1.1 Research Contributions

The contributions of this paper are the following:

**Cost aware provisioning subject to a percentile response time SLA.** We present an algorithm for resource optimization for a multi-tier cloud application, subject to an SLA expressed in terms of high percentile of end to end response time, that minimizes the total cost of compute resources required by the application. Our formulation models the application as an open tandem queue network of M/G/1-PS queues to estimate the capacity (resource requirement) of the application to honor the SLA expressed as a response time percentile.

**Service time and response-time approximations.** We present an approximation of the response time distribution of the M/G/1 processor sharing queue based on the distribution of conditional expected response times given the service times and show it to be accurate for our purposes. In addition, we present a new service time characterization based on a mixture of shifted exponential distributions, which leads to the efficient solution of a tandem system consisting of M/G/1 processor sharing queues.

**Cost-efficient configuration with heterogeneous servers subject to percentile SLA.** We extend the above approach to account for the presence of multiple types of servers with different costs and computational capabilities. This is achieved by formulating an integer optimization problem with the constraint that per-tier capacity should be at least as much as that computed by the queueing theoretic model.

**Prototype implementation and experimentation.** We have implemented our analytical model in MATLAB and tested it using a multi-tier application, i.e. java implementation of TPC-W, over a private cloud. We created the private cloud over a Linux cluster with Xen hypervisor using OpenNebula [16]. For comparison, we also implemented a baseline case using M/M/K-FCFS queues. Our experimental results show that our approach is able to provision the application to meet the SLA specified on 99 percentile of end-to-end response time, while the baseline techniques provisioned with a provisioning error as large as $140\%$. We show that our approach is able to handle the service time variability with the worst case provisioning error of $10.4\%$ as opposed to a $196\%$ error shown by the baseline case. In the case of heterogeneous provisioning, our approach shows, as high as, $81\%$ savings in server cost as compared to that of the corresponding optimal homogeneous configuration. In case of private cloud experiments we found that heterogeneous approach showed around $11\%$ cost saving (using Amazon EC2 pricing) over homogenous configurations.

The rest of the paper is organized as follows: Section 2 describes the architecture of a multi-tier application, the cloud platform, the SLA metric, and formulates the optimization problem. Section 3 describes our queuing theoretic model of the multi-tier application. Section 4 describes how we fit service time distribution to the given service-time histogram, and proposes an approximation to the response time distribution. Section 5 describes our approach for obtaining a near optimal application configuration with a single server type (i.e., homogenous setup). Section 6 describes our method for obtaining a cost efficient application configuration with multiple server types (i.e., heterogenous setup). Section 8 explains the results of numerical and experimental validation. Section 9 presents related work and we conclude the work in Section 10.

## 2 Background and Problem Formulation

In this section, we present the system model and a high level problem description. We describe the SLA performance metric, and thereafter formulate the provisioning problem that we address in this work.

### 2.1 Multi-tier Application

Modern large scale web applications are developed as multiple tiers for reasons pertaining to scalability. A multi-tier architecture offers flexibility for development as well as deployment of applications. Each application tier, typically, provides a specific functionality and the various tiers form a processing pipeline. In a typical multi-tier application various tiers participate in the processing of an incoming request; each of the participating tiers receives partially processed requests from the previous tier and feeds these requests into the next tier after local processing (see Figure 1). The tiers are replicated to scale according to the processing demand; a load balancer is used to distribute the load over all replicas of such a tier. Figure 1 depicts a two-tier application[1] where both tiers

[1]Such an application is, sometimes, also called a three-tier architecture [4], where the "remote web clients" form the first tier and the other two form the second and third tiers. Since in this work we focus on the provisioning of

are replicated. This is a commonly employed architecture by e-commerce web applications where, both, web-server and database tiers are clustered to scale up according to increase in the incoming workload.

We assume that each tier is placed on a dedicated server and that replicating a tier essentially means replicating the server. Each clustered tier is also assumed to employ a protocol-session aware load balancer responsible for distributing requests to replicas in that tier. It is assumed that the each tier's capacity, i.e., number of servers allocated to it, can be varied dynamically without disturbing the application's normal functioning, and that each tier can be independently provisioned for capacity.

## 2.2 Cloud Platforms

Cloud computing has emerged as a new IT delivery model. Infrastructure as a Service (IaaS) cloud-model is being seriously evaluated by enterprises to deploy their web applications that support dynamic capacity resizing. In this model, an organization/client can rent remote compute and storage resources to host networked applications and resources can be dynamically added or removed on an as-needed basis. We consider a cloud computing platform that allows compute servers to run hosted applications. We assume that the platform offers $N$ heterogeneous server configurations for rent, each with a different rental-cost and configuration.

We assume that the cloud platform has an infinite pool of servers and that servers can be provisioned by invoking server-instance creation APIs; servers may be requested and terminated at any time and billing is based on the amount of time for which each server is used (e.g., based on the number of hours for which each server is used). We also assume that the cloud platform employs virtualization—each physical server is assumed to run a hypervisor that controls the allocation of physical resources on the machine and offers performance isolation to each of its virtual servers.

## 2.3 Problem Formulation

Let $N$ and $M$ denote the number of tiers and server-types respectively. Let tier $j$ be jointly served by $\sum_{i=1}^{M} n_{ij}$ servers, where $n_{ij}$ denotes the number of servers of type $i$ present at tier $j$. Let $\overline{n}_j = [n_{1j}, n_{2j}, \dots n_{Mj}]$ be a vector representing the server configuration of tier $j$ and $\overline{p} = [p_1, p_2, \dots p_M]$, where $p_k$ denotes the cost of a server of type $k$. Let $T$ be the end-to-end response time of requests to the multi-tier application and $F_T$ be its CDF, i.e. $F_T(t) = P(T \leq t)$. Then for a given percentile bound $\theta$, and response-time threshold $T_D$, the cost minimization problem becomes:

$$\text{minimize} \sum_{j=1}^{N} \sum_{i=1}^{M} n_{ij} p_i, \tag{1}$$

the web application capacity, which is the web-server tier (i.e. the Java Tier) and the DB tier; thus we address it as a two-tier setup

subject to the constraint

$$F_T(T_D) \geq \theta. \tag{2}$$

It should be noted that $F_T$, also depends on $n_{ij}$, since $n_{ij}$ specifies the application configuration that determines the end-to-end response time of the application. In the next section we present a model of a multi-tier application which enables us to capture the effect of $n_{ij}$ on $F_T$.

# 3 Application Model

In this section we model the multi-tier application as a network of queues. Our first model of multi-tier application is a chain of tiers where each tier is modeled as single M/G/1-PS queue (see Figure 2). Each tier carries out a specific function, for instance, a web-application server or a database server etc. In this work we assume single customer class.

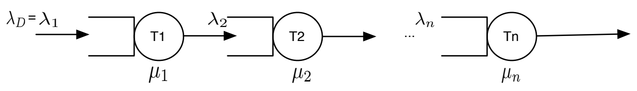


Figure 2: Multi-tier application model

Let $A_i$ denote the $i^{th}$ tier of the application, $\lambda_i$ the average arrival rate of incoming requests at the $i^{th}$ tier, and $\mu_i$ the average service rate $\forall i = 1 \dots N$. We define the total response time of a request as the time between when it enters the first tier and the time when it leaves the last tier. Note that different $\lambda_i$ for each tier handles the case where one tier issues multiple requests to the lower tier.

In our model of multi-tier application, we assume that the response times at all tiers are independent. Let $T_j$ be a random variable representing the response time for tier $j$, then the end-to-end response time of a request is

$$T = \sum_{j=1}^{N} T_j. \tag{3}$$

Let $f_T(t)$ be the probability density function (PDF) of the response time $T$ and $L_T(s) = \mathcal{L}(f_T(t))$ be the Laplace transform of the PDF of response time $T$ then

$$L_T(s) = \prod_{j=1}^{N} L_{T_j}(s), \tag{4}$$

where $L_{T_j}(s)$ is the Laplace transform of the PDF of $T_j$. Thus the PDF of end-to-end response time, $f_T(t)$, can be computed by taking the Laplace inverse of (4)

$$f_T(t) = \mathcal{L}^{-1}\left(\prod_{j=1}^{N} L_{T_j}(s)\right). \tag{5}$$

To solve (5) we require the PDF of the random variable $T_j$. Unfortunately there are no exact formulas for response time distributions of an M/G/1-PS queue. We, therefore, present an approximation for the same in the next section.

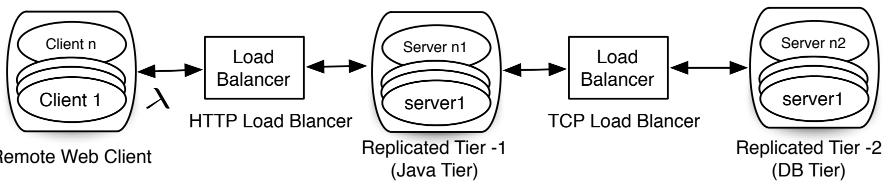


Figure 1: Topological configuration of a typical replicated two-tier web application

## 4 Estimating End-to-end Response Times

In this section we describe our approach to estimate the PDF of end-to-end response time of a chain of M/G/1-PS queues. In order to do that we estimate the PDF of response time of a single M/G/1-PS queue and then leverage (5) to compute the end to end response time.

Section 4.1 describes our method of approximating the response time distribution of a M/G/1-PS queue in. The result depends of the definition of the PDF of service-time distribution of the queue and we describe a mechanism to approximate the same for any real-life system in section 4.2. Section 4.3 provides a closed form equation of the end-to-end response time of the chain of queues.

### 4.1 Approximate Response Time Distribution

The exact form of the response time distribution for the M/G/1-PS is not generally known [27]. Thus we approximate it with the expected conditional response time distribution as described below. Let $T$ denote the job response time, and $X$ its service time; then the expected conditional response time, conditioned on the service time being $x$ is

$$\tau = E[T|X = x] = \frac{x}{1-\rho}, \tag{6}$$

where $\rho = \lambda/\mu$ is the average load.

We approximate $T$ by $\tau$. Since $\tau$ is a function of $X$,

$$\begin{aligned} F_\tau(t) = P[\tau \le t] = P[\tfrac{X}{1-\rho} \le t] = P[X \le t(1-\rho)], \\ F_T(t) \approx F_\tau(t) = F_X(t(1-\rho)), \end{aligned} \tag{7}$$

It has been observed in real-life systems that job service time distributions exhibit heavy tailed behavior [8]. Heavy tailed distributions have very high variance; high variance in service time distribution of jobs makes it a dominant factor in determining the behavior of response time distribution. Approximation proposed in (7) captures the variability of service time and will be particularly useful in such situations. We discuss the impact of variability of service time in section 7 and demonstrate that our approach shows significant improvement.

### 4.2 Approximate Service Time Distribution

In real systems, like computer clusters and web servers, there is a strong evidence that job service times are highly variable [8]. Some heavy tailed distributions do not have a closed-form Laplace transforms, e.g., the Pareto distribution, while those possessing convenient Laplace transforms might lead to an intractable complex function after undergoing an $N^{th}$ order convolution in (4). We, thus, need a distribution function, which can closely approximate a service time distribution observed by a real world application and leads to an easily invertible Laplace transform even after undergoing higher order convolutions. In this section we describe such a distribution function and also present an algorithm to approximate the service time distribution from the service-time histogram; service time histograms can be easily collected from the server either through logs or through off-line profiling.

We express the service time distribution as a mixture of $K$ shifted exponentials, as shown in (8). The motivation behind this is two fold: i.) the web application workload is a mix of different job types [14, 7]. Capturing the service time distribution as sum of shifted exponentials, essentially, means that job-size of each job-type is exponentially distributed but each job-type has a different mean job-size. ii.) The formulation leads to a Laplace transform that is easy to invert.

Formally, we want to fit a mixture of shifted exponentials,

$$f_X(x) = \sum_{k=1}^{K} \alpha_k \mathbf{1}\{x \ge t_k\} \mu_k e^{-\mu_k(x-t_k)}, \quad x \ge 0 \tag{8}$$

to data $x_1, x_2, \ldots, x_n$, where $\mathbf{1}\{P\}$ is one if predicate $P$ is true and zero otherwise. This involves inferring the number of shifted exponentials, $K$, the shifts of each exponential, $\{t_k\}$, the mix proportion of the shifted exponential, $\{\alpha_k\}$, and their average rates $\{\mu_k\}$ from the data. Let us begin by assuming that $K$ and $t_1, \ldots, t_K$ are already known. In other words we want to find the best fit for $\{\mu_k\}$ and $\{\alpha_k\}$; we perform maximum likelihood estimation using the expectation-maximization algorithm (EM).

#### 4.2.1 EM algorithm for estimating mixture parameters

Suppose we know which shifted exponential distribution each observation $x_i$ belongs to, in other words suppose we have $y_i \in \{1, \ldots, K\}$ available to us where $y_i \in \{1 \ldots K\}$ represents the particular shifted exponential distribution. Then the parameter values that maximize the log likelihood function can be computed as:

$$\alpha_k = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{y_i = k\}/n, \quad k = 1,\dots,K \quad (9)$$

$$1/\mu_k = \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i = k\}x_i}{\sum_{i=1}^{n} \mathbf{1}\{y_i = k\}}, \quad k = 1,\dots,K \quad (10)$$

EM is an iterative algorithm that infers $y_i$ as needed. Suppose $\mu_k^j$ and $\alpha_k^j$ are the estimates at the end of the $j$-th iteration. The next iteration consists of an expectation step followed by a maximization step as given below.
*Expectation.* Let $y_{i,k}$ denote the probability (expectation) that sample $x_i$ belongs to the $k$-th shifted exponential. It is given as

$$\begin{aligned} y_{i,k} &= P[Y_i = k | X = x_i] \\ &= \frac{\alpha_k^j \mathbf{1}\{x_i \geq t_k^j\}\mu_k e^{-\mu_k^j (x_i - t_k^j)}}{\sum_{l=1}^{K}(\alpha_l^j \mathbf{1}\{x_i \geq t_l^j\}\mu_l e^{-\mu_l^j (x_i - t_l^j)})} \quad (11) \end{aligned}$$

$\forall i = i,\dots,n$ and $k = 1,\dots K$. Note that $y_{i,k} = 0$ when $x_i < t_k^j$.
*Maximization.* Having computed $y_{i,k}$, we now update our estimates of $\alpha_k$ and $\mu_k$. This is done by using modified versions of (9) and (10).

$$\alpha_k^{j+1} = \frac{1}{n}\sum_{i=1}^{n} y_{i,k}, \quad k = 1,\dots,K \quad (12)$$

$$1/\mu_k^{j+1} = \frac{\sum_{i=1}^{n} y_{i,k}x_i}{\sum_{i=1}^{n} y_{i,k}}, \quad k = 1,\dots,K \quad (13)$$

This is referred to as the maximization step because the above estimates maximize the likelihood given the current values of $\{y_{i,k}\}$.

These steps are repeated until the parameters converge; $\{\alpha_k^0\}$ and $\{\mu_k^0\}$ are the initial values, which can be computed as mentioned in the section below.

#### 4.2.2 Algorithm for approximating service-time distribution

We use an iterative approach to determine the best number of exponentials $K$, and then determine $t_k$, $\mu_k^0$, and $\alpha_k^0$, to initialize the EM algorithm, (11), (12) and (13).

The basic idea underlying the algorithm, as outlined in as mentioned in [14], is to iteratively run a *k-means* clustering algorithm for every value of $k = 1 \dots K_{max}$ and compute the following three metrics[2]: coefficient of variation[3] of intra-cluster distance ($C_{intra}$), coefficient of variation of inter-cluster distance ($C_{inter}$), and ratio of intra-cluster to inter-cluster coefficient of variation ($\beta_{cv}$). The value of $\beta_{cv}$ drops as number of clusters increase and will be minimum (i.e. zero) when number of clusters is equal to the total number of points. We find that $K$, where the rate of decrease of $\beta_{cv}$ falls below a threshold (or the slope goes above a negative threshold value).

Having computed $K$, and the cluster centers $e_k$, we compute initial estimates of the mean service rate $\{\mu_k^0\}$ and mixture fraction $(\alpha_k^0)$ as follows:

$$\mu_k^0 = \frac{1}{e_k - t_k}, \ \alpha_k^0 = \frac{\text{number of points in cluster}}{\text{total number of points}}. \quad (14)$$

We set the shifts to be equidistant from from two neighboring cluster centers, i.e., $t_i = (\mu_{i-1} + \mu_i)/(2\mu_{i-1}\mu_i)$, $\forall i = 2 \dots K$. However, $t_1 = 0$, i.e., the shift for the first exponential is zero (details of the algorithm can be found in Appendix A.1.

### 4.3 Approximate Application Response Time Distribution

The PDF of the end to end response time of $N$-tier application is obtained using (8) and (7) in (5) as

$$f_\tau(t) = \mathcal{L}^{-1}\left(\prod_{j=1}^{N}\sum_{k=1}^{K_j}\frac{\alpha_{jk}\mu'_{jk}e^{-st'_j}}{(s+\mu'_{jk})}\right), \quad (15)$$

where for each tier $j = 1,\dots,N$, service times are modeled as mixtures of $K_j$ shifted exponentials and their density functions are expressed using (8); we rewrite the result for the $j^{th}$ tier for the sake of completeness:

$$f_{X_j}(x) = \sum_{k=1}^{K_j}\alpha_{jk}\mathbf{1}\{x \geq t_{jk}\}\mu_{jk}e^{-\mu_{jk}(x - t_{jk})}. \quad (16)$$

After inverting (15), the final expression of $f_\tau(t)$ takes the following form:

$$f_\tau(t) = \sum_{i_1=1}^{K_1}\cdots\sum_{i_N=1}^{K_N}\Big(\mathbf{1}\{t \geq t'\}\prod_{j=1}^{N}\alpha_{ji_j}\mu'_{ji_j} \times \sum_{l=1}^{N} r_l e^{-\mu'_{li_l}(t-t')}\Big), \quad (17)$$

where $\mu'_{ji_j} = \mu_{ji_j}(1-\rho_j)$, $t' = \sum_{j=1}^{N} t_{j,i_j}/(1-\rho_j)$, and $r_l = 1/\left(\prod_{k\neq l}^{N}(\mu'_{ki_k} - \mu'_{li_l})\right)$.

Note that $\alpha_{ji_j}$ and $\mu_{ji_j}$ are the parameters of the $k^{th}$ shifted exponential of the $j^{th}$-tier (as shown in (16) ); $\rho_j$ is the average utilization of the $j^{th}$ tier, and $r_j$ is the $j^{th}$ residue, where $j = 1,\dots,N$.

Note that the expression in (15) does not involve higher order poles[4] because none of the rates $\mu_{li_l}$ is ever equals any

[2]the metrics are computed as mentioned in [14]

[3]Coefficient of variation or variation coefficient is defined as a ratio of the standard deviation to the mean, i.e. $C_v = \sigma/\mu$;

[4]If for some $l, j$, $\mu_{li_l} = \mu_{ji_j}$, we slightly perturb the starting $\mu_{li_l}^0$ for tier-$l$ by adding a small random number and re-run the EM algorithm for that tier-$l$

of the $\mu_{ji_j}$. This becomes especially helpful in inverting the Laplace transform as absence of higher order terms in denominator leads to a simple computation of partial fractions.

The final expression of $f_\tau(t)$ in (17) is, essentially, a product of sums of the shifted exponentials, which is easily readable in (15). This means that the $f_\tau(t)$ will be expressed, in total, by $\prod_{j=1}^{N} K_j$ terms; for example let $K_j = a, \forall j = 1 \ldots N$, then $f_\tau(t)$ will be expressed as a sum of $a^N$ terms. It is easy to see that number of terms grow exponentially with number of tiers. Fortunately, real life systems do not have more than three or at most four tiers and thus $f_\tau(t)$ is easily computable.

## 5 Approach for Finding Near-optimal Homogeneous Configuration

In this section we present a solution to the the resource optimization problem, as expressed by (1) and (2), but with only one type of server, $M = 1$ (homogeneous setting).

We substitute the approximate response time of an M/G/1-PS queue, i.e. $f_\tau(t)$ as shown in (17), in (2) to obtain:

$$F_\tau(T_D) = \sum_{i_1=1}^{K_1} \cdots \sum_{i_N=1}^{K_N} \Big( \mathbf{1}\{T_D \geq t'\} \prod_{j=1}^{N} \alpha_{ji_j} \mu'_{ji_j} \sum_{l=1}^{N} \frac{r_l (1 - e^{-\mu'_{li_l}(T_D - t')})}{\mu'_{li_l}} \Big) \geq \theta, \quad (18)$$

where $\mu'_{jk_j}$ and $r_j$ are the same as in (17) while $t' = \sum_{j=1}^{N} t_{j,i_j}/(1-\rho_j)$.

Thus the problem of minimizing (1) reduces to the problem of maximizing $\rho_j$ ($\forall j = 1, \ldots, N$) such that $F_\tau(T_D) \geq \theta$, where $F_\tau(T_D)$ is given by (18). As this is an $N$-dimensional non-linear maximization problem, it is not easy to solve. However, the problem complexity is significantly reduced by constraining the loads to be the same at each tier[5], i.e.,

$$\rho_1 = \rho_2 = \ldots = \rho_N = \rho.$$

It should be noted that it is desirable to have a balanced utilization at each tier in real-life systems. In practice, administrators often use a rule of thumb to bound the max utilization of servers of all tiers to avoid performance problems and outages [17].

Consequently, (18) reduces to an inequality in a single variable, namely $\rho$.

$$F_\tau(T_D) = \sum_{i_1=1}^{K_1} \cdots \sum_{i_N=1}^{K_N} \Big( \mathbf{1}\{T_D \geq t'\} \prod_{j=1}^{N} \alpha_{ji_j} \mu_{ji_j} \sum_{l=1}^{N} \frac{r'_l (1 - e^{-\mu'_{li_l}(T_D - t')})}{\mu_{li_l}} \Big) \geq \theta, \quad (19)$$

where, $t' = \sum_{j=1}^{N} t_{j,i_j}/(1-\rho)$, and $r'_l = 1/\prod_{k \neq n}^{N}(\mu_{ki_k} - \mu_{li_l})$. We solve for the maximum value of $\rho$, say $\rho^*$, by numerically solving (19) as an equality.

[5]The constraint reduces the solution search space and thus the final solution is not guaranteed to be an optimal solution as it could result into a slightly over-provisioned system.

### 5.1 Computing the Application Configuration

In practice, large scale applications have each of their tier replicated for scalability as depicted in Figure 3. The idea is to be able to handle increasing number of requests while conforming to the SLA. In an ideal situation an application-tier's ability to process the number of requests increases linearly with number of its replicas, which means that if an application or application-tier with a single replica had a service rate of $\mu$ then $K$ replica version of application-tier will have a request rate of $K\mu$. We have assumed a linear scaling in this work but that is not a limitation and any kind of scaling function can used in the technique to obtain the number of replicas at each application-tier. We have used replicas and servers interchangeably because we have assumed dedicated hosting model.

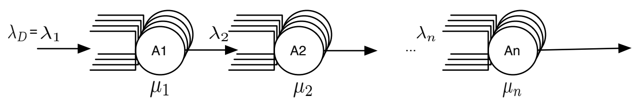


Figure 3: Multi-tier application model

We use $\rho^*$ to compute the number of servers at each tier, i.e. $n_{ij}$. In the homogenous setup $i = 1$ and thus we solve for $n_j, j = 1 \ldots N$. Let $\lambda_j$ and $\mu_j$ be arrival and service rates respectively, at tier $j$ then $n_j = \lceil \lambda_j/(\rho^* \mu_j) \rceil$, and

$$\mu_j = \sum_{i=1}^{K_j} \alpha_{ji} \frac{(1 + \mu_{ji} t_{ji}) e^{-\mu_{ji} t_{ji}}}{\mu_{ji}}. \quad (20)$$

The pseudo code of the algorithm for finding the near optimal application configuration in homogenous setup (i.e. for a single-server-type) is in Appendix A.2.

## 6 Cost Efficient Heterogenous Configuration

We extend the solution approach described in Section 5 to be able to generate a cost efficient configuration in a heterogenous setting.

The basic idea underlying our approach is to greedily search for a low cost configuration which has a high utilization. At a high level the algorithm is iterative involving the following three steps at each iteration: 1.) creating a single hybrid-server from a given hybrid-configuration for each tier, 2.) solve the homogeneous configuration problem for the hybrid-server, 3.) translate the solution for hybrid-server into a heterogenous configuration, and the iterations are used to search for new hybird-configuration with lower cost and higher utilization. Figure 4 shows the block diagrammatic representation of the cost effective heterogeneous configuration algorithm.

*Hybrid server*: Inorder to reuse our methodology for finding the near optimal number of servers in homogeneous setting, it

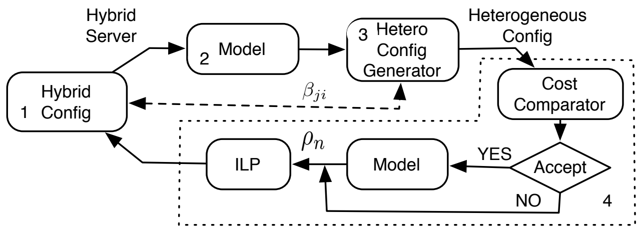


Figure 4: Functional block diagram of heterogeneous configuration algorithm

is imperative that we approximate each hybrid configuration at each tier by a single server; we call it a hybrid-server. We construct the service time distribution of the hybid-server for each tier as a proportional mixture of the service time distributions of the servers involved in the heterogeneous configuration. Let $\overline{n} = \{n_i\}$ denote the hybrid-configuration where $n_i, i = 1, \ldots, M$, is the number of servers of type $i$. Then the hybrid-server's service-time distribution function for tier-$j$ is expressed as

$$f'_j = \sum_{m=1}^{M} \beta_{jm} f_{jm}, \tag{21}$$

where $f_{jm}$ is the service-time probability density function (PDF) of the $m^{th}$-server-type at $j^{th}$-tier and $f'_j$ is the PDF of the hybrid-server for tier-$j$; $\beta_{jm}$ is the mixing proportion of the component server $m$ for tier-$j$ and is computed using the formula

$$\beta_{jm} = \frac{n_m \mu_{jm}}{\sum_{j=1}^{M} n_m \mu_{jm}}. \tag{22}$$

We explain our procedure of creating a hybrid-server with the following example: suppose we have two servers, say $s_1$ and $s_2$, with corresponding average service rates at tier-$j$ as $\mu_{j1} = 50$ and $\mu_{j2} = 100$, respectively. We construct a single hybrid-server, say $s_h$, by proportionally mixing the component shifted-exponentials of each $s_1$ and $s_2$. Let the configuration be one-server of each type, i.e. $\overline{n} = [1, 1]$; then the mixing proportions using (22) is $\beta_{j1} = 1/3$ and $\beta_{j2} = 2/3$, and the final service-time distribution of the hybrid-server for the $j^{th}$ tier is $f'_j = \left(\frac{f_{j1}}{3} + \frac{2f_{j2}}{3}\right)$.

*Heterogeneous configuration*: Once we obtain the optimal configuration for a given hybrid-server, and given workload and percentile, we translate this solution configuration to the corresponding heterogenous server configuration; this is done by reversing the steps of creating the hybrid-server. Let us assume that the servers are indexed in increasing order of their average service rate; i.e. $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_M$; let $n'_j$ be the number of hybrid-servers at tier-$j$, then the number of servers of type-$i$ for tier-$j$ is $n_{ji} = \beta_{ji} n'_j / (\mu_i/\mu_1))$.

*Searching for a new hybrid-configuration*: The cost of the new heterogenous configuration, computed in step-3, is evaluated using the prices of the servers. If the cost is less than that of the current solution configuration, then this new configuration is accepted else it is dropped. The new configuration is again fed to the model, and its utilization $\rho^*$ is evaluated for the desired arrival rate $\lambda_D$. We then try to search for a new hybrid-configuration which has higher utilization than this configuration but has lower cost then the current-configuration; the new utilization $\rho_n = (\rho_{max} + \rho_l)/2$, where $\rho_{max}$ is maximum utilization of the hybrid-server and $\rho_l = \rho^*$. The new hybrid configuration is searched for using the following ILP solved for each tier:

$$\text{minimize} \sum_{i=1}^{M} n_{ji} p_i, \tag{23}$$

subject to the constraint

$$\sum_{i=1}^{M} n_{ji} \mu_{ji} > \lambda_D / \rho_n. \tag{24}$$

Note that if the currently suggested configuration is not accepted we continue to search for higher $\rho^*$. The algorithm stops when $\rho_n - \rho_{max}$ is less than a pre-decided threshold; the pseudo code is given in Appendix A.3.

## 7 Experimental Evaluation

In this section we demonstrate the efficacy of our approach. We have implemented our analytical method using MATLAB®. For solving the ILP, we have used lpsolve version 5.5.2.0 and have used mxlpsolve MATLAB Interface version 5.5.0.6 for calling lpsolve from within the MATLAB environment.

We begin by showing the effectiveness of the service-time approximation algorithm on lognormal[6] distribution with different coefficient of variations ($C_v$). Thereafter we evaluate the goodness of the approximation of the response-time distribution for a 1-tier and a 2-tier system by comparing the response times computed using (17) with those obtained using a multi-tier application-simulator described below. Finally we do a case-study of provisioning of a two-tier application for a SLA specified as a threshold on the $99^{th}$ percentile of response time. We evaluate the effectiveness of our approach by computing the 99 percentile of response times obtained using a two-tier application-simulator configured according to the capacity decisions provided by our approach; note that the simulator depicts an ideal version of a multi-tier application which we analytically model as a chain of M/G/1-PS queues. We also evaluate the effectiveness of our approach, using a metric called provisioning error (described in Section 7.4), by comparing against the two other baseline approaches, which model the multi-tier application as an open tandem network of M/M/K-FCFS queues.

[6]PDF of a log normal distribution is expressed as $f(x, \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-((ln(x)-\mu)/(\sqrt{2}\sigma))^2}$, where mean of the distribution is $e^{\mu+\sigma^2/2}$

## 7.1 Multi-tier Application Simulator

We implemented a simulator for the PS queue in MATLAB®. It takes as input an array of request arrival instants and size of each request (in terms of service time) and outputs the request departure instants. We used this queue simulator to simulate a multi-tiered application by feeding the output of first queue to the input of the next queue.

To simulate an application with replicated tiers, we have implemented a loadbalancer, as shown in Figure 1, which takes the incoming requests from the previous tier and distributes it to the next tier according to a specific load distribution policy. It also does the necessary book-keeping to track each request across various tiers for computing the end-to-end response time. We have implemented a random load-balancing policy, i.e. loadbalancer distributes the requests at random but ensures that each server gets the same load, i.e. $\rho*$ as computed in section 5. We have assumed an ideal loadbalancer, which means that it introduces no queueing and processing delay. Note that this is not a limitation of our approach, as our approach can easily account for loadbalancer by considering it as another tier and its capacity can also be computed, which is often needed in a real setup.

## 7.2 Service Time Approximation

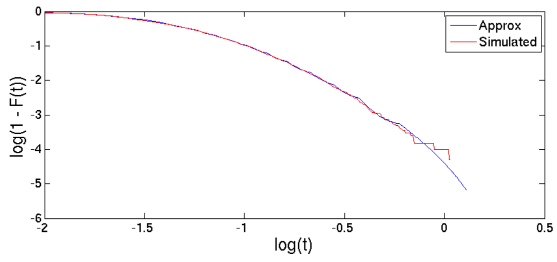
We have implemented the EM algorithm (in MATLAB) for finding the parameters of mixture of shifted exponentials, namely $\alpha_i, \mu_i$ in (8), using the E and M steps mentioned in Section 4.2.1. The shifts and initial values of parameters are estimated using the algorithm outlined in Section 4.2.2. We use MATLAB's implementation of KMeans algorithm with 10 iterations for each clustering and have kept $K_{max} = 20$ in all our experiments, which means that we search for the number of shifted exponents from 3 till 20. We evaluate the accuracy of CDF approximation using relative percentage error defined as $\epsilon(x) = (F_{aprx}(x) - F_{sim}(x))/(F_{sim}(x))$, where $F_{aprx}(x)$ and $F_{sim}(x)$ are the values of approximate and actual CDFs, respectively, evaluated at $x$.

We approximate a log-normal distribution with mean rate of 20 using a mixture of shifted exponentials. The maximum error is less than 8%, which reduces as we approach the tail of the distribution. The coefficient of variation of the original distribution is 1 ($C_v = 1$); it was expressed as a sum of 7 shifted exponentials. The log-log plot of complimentary CDF (CCDF) and approximation done by our approach is shown in Figure 5a.

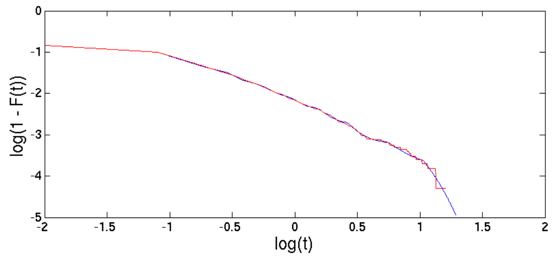
To evaluate the effectiveness of our approach in approximating highly variable distributions, we approximated of a log-normal distribution with same mean rate of 20 but with a coefficient of variation of 100 ($C_v = 100$); it was expressed using 10 shifted exponentials. Figure 5b shows the CCDF of the actual distribution in red while our approximated distribution is shown in blue.

The CCDFs in Figure 5 highlight the approximation of the tail of the distribution by plotting the $1 - F(x)$ in a log log scale. We observed that the approximation shows a relatively high error at low percentiles (as high as 21%) but displays low errors at the tail, with errors less than 1% at 95 percentile. This is because, that at low percentiles the number of exponentials available to approximate the distribution are less but as we approach the tail of the distribution a large number of exponentials contribute towards the approximation of the PDF and thus we observe much greater accuracy.

Another aspect of our algorithm is $K$, i.e. the number of exponentials required to approximate a distribution. We conducted a large number of experiments on various data sets with $C_v$ ranging from 1 to 100. In our experiments we found that $K$ its average values starts at 14 for $V_v = 1$ and slightly decreases to an average value of 10 for $C_v = 100$. It should be noted that we are testing our approximation scheme for a smooth distribution function, but the scheme has been designed keeping a web application in vision, which has only a limited number of request types at each tier and our approach is tuned to estimate this number as $K$.



(a) Lognormal with $C_v = 1$



(b) Lognormal with $C_v = 100$

Figure 5: Figure shows the log log plot of 20,000 data points sampled from lognormal distribution with $C_v = 1$and 100; the original CDF is shown in red and approximate in blue.

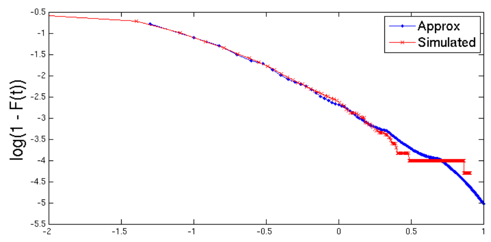
*In summary, the service time approximation approach offers very low errors, i.e. less than 1%, in estimating the tail of a distribution.*

## 7.3 Response Time Approximation

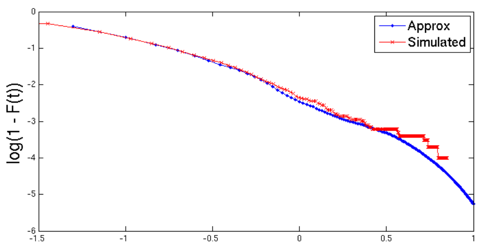
In this section we describe the effectiveness of our approach of approximating the end to end response time of an application modeled as a chain of M/G/1-PS queues.

We evaluated the goodness of the response-time approximation we compare the response time computed by our approach with that obtained from the simulator described above. We show the results for for a 1 and 2-tier setups by plotting the response time CDFs for our approximation and simulation. To evaluate the impact of high variability, we have sampled service times from a lognormal distribution with a $C_v = 10$.

We generate the workload which has exponential interarrival times with $\lambda = 25$ and service times sampled from a lognormal distribution with $\mu = 50$ at each tier. The simulation results are considered exact since the simulation model is an exact representation of the queueing network under study.



(a) CDF of RT 1-tier app



(b) CDF RT of 2-tier app

Figure 6: Figure shows the CDF plot of actual response time distribution in red and approximated using our approach in blue for a heavy-tailed service-time distribution with $\mu = 50$ and $c_v = 10$

*The approximated response time, using our approach, exhibits high accuracy, as can be seen from Figure 6. The jagged tail of simulation result is because of less number of data points.*

## 7.4 Provisioning in a Homogenous Setup

In this section we evaluate the effectiveness of our approach, outlined in section-5, in finding the homogenous configuration for a two-tier application, where each tier is replicated using same type of servers.

For a given SLA, expressed as a cutoff threshold $T_D$ on the 99th percentile of end to en response time, we fix a service time distribution and for different arrival rates compute the number of servers required at each tier of the application. We, then, run the replicated application simulator with these number of servers and obtain the end to end response time distribution for the provisioned application. To evaluate the goodness of provisioning decisions made, we define a metric called *provisioning error*, which essentially calculates the error in the $99^{th}$ percentile response time observed from the simulator, i.e. $T_{scheme}$, and $T_D$. Formally, $\epsilon_{scheme} = (T_{scheme} - T_D) * 100/T_D$. To do a comparative evaluation of our technique, we have implemented two baseline provisioning algorithms based on M/M/1-FCFS queues, namely *per-tier-exp* and *end-to-end-exp*. The schemes are described below:

*per-tier-exp (pte)* : In this scheme we assume the knowledge of average proportion of time spent by a request at each tier. In other words, let $T$ be the total time spent by a request in the system and $T_i$ be the time spent at tier $i$; then, *pte* assumes the knowledge of $E[\delta_i]$, where $\delta_i = T_i/T$. We model each tier as an M/M/K-FCFS queue and again approximate multiple servers by a single server, thus each tier can be approximated by an M/M/1-FCFS queue. For this system the response time is exponentially distributed with parameter $\mu(1-\rho)$. Finally, as in Section 5, for each tier $j$, we solve for $\rho^*$ with $T_D = \delta_j T$ and compute $n_j = \lceil \lambda_j/(\rho^* \mu_j) \rceil$

*end-to-end-exp (ete)*: We developed this scheme completely along the lines of our scheme, however assuming an M/M/K-FCFS queue based model instead of an M/G/K-PS queue based model. The corresponding version of (19) is:

$$F_T(t') = \sum_{j=1}^{N} r_j (1 - e^{-\mu'_j t'}) \geq \theta, \qquad (25)$$

where $t' = T_D$, $r_j = 1/\prod_{k \neq j}^{N} (\mu'_k - \mu'_j)$ and $\mu'_j = \mu_j(1-\rho)$. The provisioning algorithm for homogenous setting remains nearly unchanged[7].

We ran the experiment with $T_D = 0.4s$, $\mu = 50$ and $C_v = 3$. We increased the workload from $\lambda = 40$ rps to 240 rps and for each $\lambda$ we computed application capacity using each of the three algorithms. For *pte* we used $\delta_1 = \delta_2 = 0.5$. The results are shown in Table 1.

| $\lambda$ | %-$\epsilon_{our}$ | %-$\epsilon_{ete}$ | %-$\epsilon_{pte}$ | Config$_{our}$ | Config$_{ete}$ | Config$_{pte}$ |
|---|---|---|---|---|---|---|
| 40 | -3.63 | 16 | 15.2 | [3;3] | [2;2] | [2;2] |
| 80 | -6.17 | 48.1 | 27.9 | [5;5] | [3;3] | [4;3] |
| 120 | -0.235 | 94.3 | 38.5 | [8;7] | [4;4] | [5;5] |
| 160 | -2.25 | 91.6 | 49.9 | [9;9] | [5;5] | [6;6] |
| 200 | -2.17 | 140 | 40.7 | [12;12] | [6;6] | [8;8] |

Table 1: Near optimal configuration suggested by the three schemes and provisioning error of each scheme. Note that, unlike the positive error, negative value of $\epsilon$ is not an SLA violation.

A Positive value of $\epsilon$ means that some or all of the tiers of the application were provisioned with fewer servers than required (we call it under-provisioning); however, a negative value means the opposite ( we call it over-provisioning). Thus a positive $\epsilon$ is an SLA violation, while a negative $\epsilon$ is not. However, a negative $\epsilon$ does suggests a possibility of finding a

[7]Algorithm in Appendix A.2 takes two minor changes: 1.) step-4 is skipped, 2.) step-2 replaces (19) by (25)

more cost efficient solution. Note that our scheme never reports under-provisioning as opposed to the worst case under-provisioning of 140% by *ete* and 53.7 by *pte*.

*In summary: for a single server type scenario (i.e. homogeneous setup), application provisioned by our scheme conforms to SLA, while the baseline approaches show as high as* 140% *provisioning error*

## 7.5 Effect of Variability of Service Time

In this section we evaluate the effect of variability of service-time distribution on three provisioning schemes, namely *ours, pte* and *ete*.

For a fixed $\lambda = 160$, we computed the capacity of the two tier application, in a homogenous setting, using all the three schemes. We obtained the service times for both the tiers by sampling from a lognormal distribution with a fixed $\mu$ of 50 rps, while a varying the standard deviation $\sigma$. We vary $\sigma$ so that we can control $C_v$, ranging from 1 to 10.

The computed capacities, by each of the schemes, were again tested using the application-simulator. Their percentage provisioning errors were computed and plotted in Figure 7.

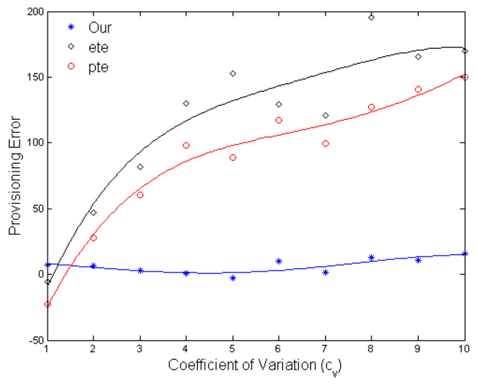


Figure 7: Variation in provisioning error with $c_v$

Figure 7, shows that percentage provisioning error for *ete* and *pte* increases as a function of $C_v$, while maintaining the average service-rate constant, as opposed to our scheme, which shows a worst case provisioning error of 11%. The main reason behind this is that both *ete* and *pte* schemes are unable to capture the tail of the service-time distribution and thus cause severe under-provisioning.

*Thus we conclude that our scheme captures the tail of service-time distribution and is able to provision for the 99-percentile capacity with a max provisioning error less than 11% as opposed to other schemes which severely under-provision the capacity with the max-provisioning error of 196 %*

## 7.6 Cost Efficient Server Configuration in a Multiple Server-type Environment

Here we demonstrate the effectiveness of our heterogenous provisioning algorithm in finding a cost-efficient solution when multiple types of servers are available. We have kept the time threshold $T_D = 0.4$-sec and varied the desired load from $\lambda = 40$-rps to $\lambda = 240$-rps. We have considered four types of servers, namely small (S), medium (M), large (L), and extra-large (XL), with their corresponding average service rates being 50, 100, 150 and 200 rps, respectively. The coefficient of variation of service times for requests at each of the tiers is $C_v = 9$.

| ServerType | Small | Medium | Large | XLarge |
|---|---|---|---|---|
| Price | 0.02 | 0.04 | 0.06 | 0.08 |

(a) server prices

| $\lambda$ | %-$\epsilon_{homo}$ | %-$\epsilon_{hetro}$ | Config$_{homo}$ | Config$_{hetro}$ | %CostSaving |
|---|---|---|---|---|---|
| 40 | 1.63 | -40.9 | [9;9] | [0 1 0 0;0 1 0 0] | 77.78 |
| 80 | 1.01 | -35.7 | [17;15] | [0 0 0 1;0 0 1 0] | 78.13 |
| 120 | 1.16 | -22.9 | [26;23] | [0 0 2 0;0 1 1 0] | 77.55 |
| 160 | 1.06 | -23.5 | [34;30] | [0 0 1 1;0 0 2 0] | 79.69 |
| 200 | 1.09 | -21.5 | [43;47] | [0 0 3 0;0 1 2 0] | 81.11 |
| 240 | 1.04 | -9.82 | [51;45] | [0 0 2 1;0 0 3 0] | 80.21 |

(b) 99-percentile provisioning and cost benefit

Table 2: Near optimal configuration suggested by the three schemes and provisioning error of each scheme. Note that a negative $\epsilon$ only means over-provisioning and is not an SLA violation

We assume linear pricing as depicted in Table 2a. The results of provisioning algorithms in homogenous and heterogeneous settings are shown in Table 2b. We call the computed capacity configurations in the homogenous and heterogeneous settings as $Config_{homo}$ and $Config_{hetro}$, respectively. $Config_{homo}$ uses only the "small" server-type, while $Config_{hetro}$ uses all the available server types. As in previous evaluations, we again test the computed configuration using the multi-tier application simulator.

Each configuration is $N \times M$ dimensional matrix depicting the number of servers of each type; each row $j$ depicts the configuration of the $j^{th}$ tier, while each column tells the number of servers for each type: for e.g. Config$_{homo} = [9; 9]$ means 9-small servers at both the tiers, while Config$_{hetro} =$ [0 1 0 0;0 1 0 0] means 0-small, 1-Medium, 0-large and 0-x-large server at both the tiers. The "%Cost Saving" is computed as a percentage of cost of homogenous configuration, i.e. $\frac{Cost(Config_{homo}) - Cost(Config_{hetro})}{Cost(Config_{homo})}$.

We make following important observations: 1) the percentage provisioning error for the heterogeneous scheme is as large as $-41\%$, which means that not-only is this configuration cost-efficient but it also provides low average response-times (because negative provisioning error means the system is probably over-provisioned). 2) it is better to use larger servers that fit the same cost and average service-rate; in other words its better to use a small number of large servers instead of a large number of small servers.

*In summary, it is better to use a small number of large servers instead of a large number of small servers for high percentile provisioning ii) Cost efficient heterogenous algorithm offers server configurations with cost savings as high as 81% and also offer a configurations with lower average response-times.*

## 8 Evaluation on Private Cloud

In this section we describe an experimental investigation for provisioning for a percentile SLA in a private cloud setup. Our goal is to evaluate our provisioning algorithm under situations which are typical to multi-tier web applications deployed in a datacenter or private/public cloud environment. As in Section 7.4, we do a case-study of server provisioning for the $99^{th}$ percentile of response time threshold of a two-tier system and verify our results using the two-tier java implementation of TPC-W. Similar to Section 7.4 and 7.6, we evaluate the effectiveness of our approach over the multi-tier TPC-W application. Our evaluation metrics are the overall rental cost of the virtual servers supporting the application deployment, and percentage provisioning error (defined in Section 7.4).

### 8.1 Private Cloud Setup

In this section we provide the necessary details of our experimental testbed, i.e private cloud, and necessary steps before we can perform server provisioning.

#### 8.1.1 Experimental Testbed and Workload

**Web Application:** We used TPC-W [1] for our experiments. TPC-W is a multi-tier transactional web benchmark that represents an e-commerce web application – an online bookstore – comprising of a web server tier and a database tier. It simulates the activities of a retail store website using 14 different type of pages for web interactions; each of these pages are created dynamically by the web server using differing amounts of data stored in the database tables. TPC-W benchmark defines three different mixes of web interactions, namely browsing, shopping and ordering, each varying the ratio of inventory browsing related web pages and ordering related web pages. It applies the workload mixes via remote browser emulator (RBE).

We used the Java implementation of TPC-W [4]. The web application has following two-tiers: i.) Web server tier based on Apache Tomcat servlet container 5.5.26 ii.) database tier based on MySQL 5.0.77. We deployed each of the tiers on separate dedicated servers. We performed round robin load balancing between replicas of web server tier using a dedicated loadbalancer server on HAProxy [11] on a server as a dedicated load balancer. We used round robin load balancing at the database tier by setting up a master-slave replication configuration of MySQL servers; we instrumented TPC-W to use the replication aware MySQL JDBC connector version 3.1.12.

**Private Cloud:** We constructed private cloud using OpenNebula [16] on Xen/linux-based cluster consisting of two types of servers: 8-core 2GHz AMD Opteron 2350 servers and 4-core 2.4 GHz Intel Xeon X3220 systems. All machines run Xen 3.3 and Linux 2.6.18 (64bit kernel). Our platform is assumed to support *small* and *large* servers, comprising 1 and 4 cores, respectively. These are constructed by deploying a Xen VM on the above mentioned servers and dedicating the corresponding number of cores to the VM (by pinning the VM's VCPUs to the cores)

**Workload:** We used the browsing mix of the TPC-W specification; that was generated using TPC-W clients. We have tested our approach in each of the settings, namely homogenous and heterogenous, by increasing the workload in large steps to test the resiliency of the provisioned application setup in being able to conform to SLAs.

#### 8.1.2 Server Profiling for Service-time Histograms and Per Tier Arrival Rates

We profiled each server type separately for each tier. The concerned tier would always have only one server of concerned server-type .

**Profiling servers for web server tier:** In order to record the request service times at the first tier, i.e. web-server tier, we instrumented Tomcat[8] such that it reports per-request service times, along with the other default statistics, in the server logs. For each server type (e.g. *small* and *large* ) we provision one instance of the server-type and deploy the first tier of TPC-W (i.e. the web server tier with instrumented version of Tomcat) and attach it to an already installed TPC-W database on an instance of *large* server type. We issue the browsing workload using the TPC-W clients (i.e. RBEs) for a duration of 35 mins and collect the service times from the tomcat server logs.

**Profiling servers for database server tier:** Profiling the servers for the second tier of TPC-W (i.e. the database tier) was in two steps: firstly, we collect the 35-min query logs from MySQL server, executing the TPC-W workload; then for each server type we slowly replay each of the SQL query and record their execution time as service times.

**Estimating $\lambda_j$:** We estimate the per tier arrival rates, i.e. $\lambda_j$, $j = 1 \ldots N$, using the average scaling ratios $V_j$ for each tier $j$.We compute $V_j$ by dividing the number of number of requests at tier-$j$ by those at tier-1. This is carried out after the server profiling step.

[8]for each request processing call in a thread we make a JNI call to a system call, namely getrusage(), for recording the CPU utilization

### 8.2 Percentile Based Capacity Provisioning on Private Cloud

Given $\lambda_D$ and $T_D$, we outline the high level steps required to compute application capacities for both homogenous and heterogenous setup. In both the cases we assume to require an SLA where $99^{th}$ percentile of the end-to-end response time must be less than 0.5 seconds. We follow the following sequence of steps

**Step 1: Estimating service time distributions:** We use the service times collected during the offline profiling step and use the service time approximation algorithm – outlined in Appendix A.1 – based on the results of section 4.2.

**Step 2: Estimating capacity in a homogenous/Heterogenous setup:** We used the single core virtual machines (i.e. *small* ) in our homogenous setup. Load across multiple web-server replicas was distributed using a HAProxy based load-balancer, however, in the case of database tier, we used the master-slave setup. In this setup all the wites are sent to the master, whereas the reads are load-balanced.

We test our approach for both homogenous and heterogeneous environment. For homogenous setup, we choose *small* server type for this case and assume $T_D = 0.5$sec. To test the provisioning setup for large change in workload, we varied $\lambda_D$ from 15 rps to 90 rps. For each $\lambda_D$, using our approach, we computed server capacities for each of the tier of TPC-W. We ran the setup for 35-mins and in the end we collected the end-to-end response times from the first tier (i.e. web-server tier). We ran our heterogeneous provisioning algorithm on the same setup and found that it gave a different configuration, only for $\lambda_D = 90$. Table 3a provides the details of the final configuration and also the $99^{th}$-percentile of the end-to-end response time details of the experiment. We compute the percentage provisioning error, $\epsilon_{our}$, as mentioned in 7.4.

| $\lambda_D$ | $99^{th}$ % | % $\epsilon_{our}$ | Config$_{our}$ |
|---|---|---|---|
| 15 | 0.361 | -27.8 | [1;1] |
| 30 | 0.459 | -8.2 | [1;2] |
| 45 | 0.488 | -2.4 | [1;3] |
| 90 | 0.512 | 2.4 | [2;7] |
| 90 | 0.46 | -8.0 | [2,0;2,1] |

(a) Server Provisioning

| Server Type | *small* | *large* |
|---|---|---|
| Prices ($) | 0.085 | 0.34 |

(b) Server prices

Table 3: Near optimal configuration suggested by our algorithm and percentage provisioning error for both homogenous and heterogeneous setup. Note that -ve $\epsilon_{our}$ only means that the system is over-provisioned and thus SLA will not be violated

We found that server provisioning by our approach keep provisioning error below 3%. The positive 2.4% error at $\lambda = 90$ for homogenous setup could be because the master database server gets over loaded as it has to replicate the updates to each of the 6 slaves. We see that the server provisioning for the heterogenous environment, is not only 11.11% cheaper than the corresponding homogenous server setup but also has a lower $99^{th}$ response time.

*In summary, our algorithm effectively accurately captures the service time distributions and provisions the two-tier implementation of TPC-W with the worst provisioning error of 3%. Also, we, again, find that its better to use bigger server for high percentile provisioning.*

## 9 Related work

A number of efforts have modeled internet applications. Modeling single tier has gotten much of the attention. Doyle *et al.* propose a queuing model for static content [10], Menasce uses a queuing model to model the web servers [13], while Slothouber [21] modeled the HTTP server. Urgaonkar *et al.* use G/G/1 queueing model for replicated tiers and assume the knowledge of per-tier response times. They use the peak session arrival rate to capture the workload and provisioning capacity to service this peak workload. Thus, the approach is of less practical significance as it cannot give a statistical bound on response time.

Abdelzaher *et al.* in [2] use classical feedback control theory to model the bottleneck tier for providing performance guarantees for web applications serving static content. Similarly, Chen *et al.* in [6] use a machine learning technique for provisioning the database tier.

Chandra *et al.* in [5] propose a queuing model based approach, that partitions capacity of a shared server among multiple hosted applications. The approach focuses on the capacity needed to handle high request volumes during overload scenarios. Ranjan *et al.* [18] use a G/G/N queuing model to compute the number of servers necessary to maintain a target utilization level. This strategy is shown to be effective for sudden increases in request arrival rate. Other efforts have employed similar M/G/1 queuing models in conjunction with offline profiling to model service delay and predict performance [22] but they do not provision for response time percentile and neither do they address the problem in heterogenous environment. The approach in [25] formulates the application tier server provisioning as a profit maximization problem and models application servers as M/G/1/PS queuing systems; the approach only considers the impact of different number of end-clients (and thus, request volumes) and does not solve for a solution with a percentile response time.

Benanni *et al.* in [3] employ approximate mean-value analysis (MVA) to develop an online provisioning technique for multiple request classes. Urgaonkar *et al.* in [23] develop a queuing network model for multi-tier Internet applications having request classes with differentiated QoS; the authors use mean response times and do not provision the system for percentile response-times. Zhang et. al. [28] use a multi-class model to capture the dynamics of workload by employing a fixed set of 14 predefined transactions-types and leverage it to predict the performance of a multi-tier system but again plan using the mean response-time.

There has been some work for finding the pdf of response time, for e.g. Muppula *et al.* in [15] derive the response time for a closed queuing network using pteri-nets and sojourn time distribution was calculated for large Markov chains in [12]. The approach leads to an inversion of a complex Laplace transform. Xiong *et al.* in [26] perform the provisioning of a multi-station setup for a given percentile bound. The model the system as a open tandem network of M/M/1-FCFS queues and compute the response time PDF by numerical inversion of its Laplace transform; they assume that each station is serviced by same type of servers.

In contrast to these efforts, our work automatically characterizes service time distribution as a mixture of shifted exponentials and we use this to estimate the response time distribution. This estimated distribution is used to estimate the capacity of the system. This estimated capacity is used to provide a near optimal solution to the provisioning problem. Further, while most of these efforts have focused on a single server type environment (i.e. homogeneous), we extend our approach for the cloud specific heterogenous environment as well. We developed a full prototype implementation and our experiments were conducted on an actual private cloud.

## 10 Conclusion

Multi-tier architecture is a preferred architecture for enterprise web applications and high response time percentile provisioning is the more meaningful than mean response time based ones. We present an approach of optimizing server allocation for a multi-tier application to achieve a percentile bound on the end to end response time. We model the application as an open tandem network of queues and model each tier as an M/G/1-PS queue. We have developed an approximate model to compute the response time distribution and have also developed a technique to estimate the service time distribution from the service time histograms. We have developed an algorithm to compute per tier server allocation of the application and in a homogenous setup. We also have extended the homogenous setup solution to solve the server allocation problem in a heterogenous setup. We have tested the efficacy of our approach using a multi-tier application simulator and also compared it against two other baseline approaches developed using models based on M/M/K-FCFS queue. We have demonstrated superior performance of our approach as compared to the baseline approaches. Our experiments indicated that its better to use small number of large servers than large number of small servers. Finally we tested our approach using the multi-tier implementation of TPC-W benchmark over private cloud created using Xen over Linux.

## References

[1] The tpcw benchmark. `http://www.tpc.org/tpcw/`.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[3] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC '05*, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.

[4] H. W. Cain and R. Rajwar. An architectural evaluation of Java TPC-W. In *In Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 229–240, 2001.

[5] A. Chandra, W. Gong, and P. Shenoy. Dynamic Resource Allocation for Shared Data Centers Using Online Measurements. In *Proceedings of Eleventh International Workshop on Quality of Service (IWQoS 2003)*, June 2003.

[6] J. Chen, G. Soundararajan, and C. Amza. Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers. In *ICAC*, pages 231–242, June 2006.

[7] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51(6), June 2002.

[8] M. Crovella. Performance evaluation with heavy tailed distributions. In *Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, TOOLS '00, pages 1–9, London, UK, 2000. Springer-Verlag.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

[10] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS ' 03), Seattle, WA*, March 2003.

[11] Haproxy the reliable, high performance tcp/http load balancer. `http://haproxy.1wt.eu/`.

[12] P. G. Harrison and W. J. Knottenbelt. Passage time distributions in large markov chains. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 77–85, New York, NY, USA, 2002. ACM.

[13] D. Menasce. Web Server Software Architectures. In *IEEE Internet Computing*, volume 7, November/December 2003.

[14] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 119–128, New York, NY, USA, 1999. ACM.

[15] J. K. Muppala, K. S. Trivedi, V. Mainkar, and V. G. Kulkarni. Numerical computation of response time distributions using stochastic reward nets. In *Annals of Operations Research*, pages 155–184, 1994.

[16] Opennebula. `http://www.opennebula.org`.

[17] G. Pacifici, W. Segmuller, M. Spreitzer, M. Steinder, A. Tantawi, and A. Youssef. Managing the response time for multi-tiered web applications. In *IBM, Technical Report*, January 2005.

[18] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.

[19] S. M. Ross. *Stochastic Processes*. Wiley, 2 edition, Jan. 1995.

[20] U. Sharma, P. J. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *ICDCS*, pages 559–570, 2011.

[21] L. Slothouber. A Model of Web Server Performance. In *Proceedings of the 5th International World Wide Web Conference*, 1996.

[22] C. Stewart and K. Shen. Performance Modeling and System Management for Multi-component Online Services. In *Proc. USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, May 2005.

[23] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An Analytical Model for Multi-tier Internet Services and Its Applications. In *Proc. of the ACM SIGMETRICS Conf.*, Banff, Canada, June 2005.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Adaptive and Autonomous Systems (TAAS), Vol. 3, No. 1*, pages 1–39, March 2008.

[25] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning Servers in the Application Tier for E-commerce Systems. In *Proceedings of the 12th IWQoS*, June 2004.

[26] K. Xiong and H. Perros. Qrp01-6: Resource optimization subject to a percentile response time sla for enterprise computing. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1 –6, 27 2006-dec. 1 2006.

[27] S. F. Yashkov. Processor-sharing queues: some progress in analysis. *Queueing Syst. Theory Appl.*, 2(1):1–17, 1987.

[28] Q. Zhang, L. Cherkasova, N. Mi, and E. Smirni. A regression-based analytic model for capacity planning of multi-tier applications. *Cluster Computing*, 11(3):197–211, 2008.

## A Pseudocodes

This appendix section presents the pseudocode of three algorithms which have been proposed by us in this paper.

### A.1 Pseudocode for service time approximation algorithm

We outilne the service time approximation algorithm mentioned; the analytical results are mentioned in Section 4.2.

```
input : Service-time histogram $\mathbf{x} = \{x_1 \dots x_n\}$ ;
        Maximum number of clusters $K_{max}$;
output: $\boldsymbol{\alpha} = [\alpha_1 \dots \alpha_K], \boldsymbol{\mu} = [\mu_1 \dots \mu_K], \mathbf{t} = [t_1 \dots t_K]$;
1  for k=1:$K_{max}$ do
2      $[\mathbf{e}_k, \beta_k]$= kmeans$(\mathbf{x}, k)$;
3  end
4  $[\beta_{cv}, K]$ = min$(\beta_k)$;$\mathbf{e} = \mathbf{e}_K$ ;
5  $\mathbf{e}^0$ = sort$(\mathbf{e})$;
6  $\alpha_1^0$= (# of points in $1^{st}$ cluster)/$n$; $t_1 = 0$; $\mu_1^0 = 1/e_1^0$;
7  for k=2:K do
8      $\alpha_k^0 = \frac{\text{(\# of points in } k^{th} \text{ cluster)}}{n}$;
9      $t_k = \frac{e_k^0 - e_{k-1}^0}{2}$; $\mu_k^0 = \frac{1}{e_k^0 - t_k}$;
10 end
11 $[\boldsymbol{\alpha}, \boldsymbol{\mu}]$= EM$(\mathbf{x}, K, \boldsymbol{\alpha}^0, \boldsymbol{\mu}^0, \mathbf{t})$;
12 return $\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{t}$
```

**Algorithm 1:** Determining approximate service time distribution from the service-time histogram

### A.2 Pseudocode for finding the tier-wise configuration algorithm

We outilne the near optimal algorithm mentioned in Section 5.

```
input : number of tiers = $N$;
        arrival rate for $j^{th}$-tier $\lambda_j \ \forall j = 1, \dots N$, ;
        the service-time distribution $f_{X_j}(t)$ of each tier$j$ ;
        percentile, i.e. $\theta$ and response time threshold $T_D$ ;
        bound on $\rho$, i.e $\rho_l \leq \rho \leq \rho_u$
output: Configuration $\overline{n} = \{n_1, \dots, n_N\}$;
        The optimal per-tier utilization $\rho^*$;
1 Calculate $\mu_{ji}, t_{ji}, \alpha_{j,i}$ using Alogrithm 1;
2 Search for $\rho_l \leq \rho^* \leq \rho_u$ such that (19) is satisfied;
3 for each tier $j$ do
4     Compute $\mu_j$ using (20) ;
5     Compute the integers $n_j = \lceil \lambda_j / (\rho^* \mu_j) \rceil$;
6 end
7 return $\overline{n} = \{n_1, \dots, n_N\}$ and $\rho^*$
```

**Algorithm 2:** Find number of servers required at each tier in a single server-type setting

## A.3 Pseudocode for cost efficient algorithm

We outilne the pseudo-code mentioned in Section 6.

```
input : $N$; $\boldsymbol{\lambda} = [\lambda_1 \dots \lambda_N]$;
        $f_{ji}, \forall j = 1 \dots N, i = 1 \dots M$; $\theta$; $T_D$;
        $hybConf = [n_{jm}]_{N \times M}$; $\rho_l \leq \rho \leq \rho_u$;
output: Complete configuration $[n_{ji}]_{N \times M}$
1 ITER=true, $\rho_l = 0$; $\rho_{max} = 0$; curConfig=[0];
  $curCost = \infty$; $\delta = 0.001$;
2 while ITER do
3     for each tier-$j$ do
4         for each server type-$m$ do
5             Compute $\beta_{jm}$ using (22);
6         end
7         Compute $f'_j$ using (21);
8     end
9     Get $hybConf' = [n'_j]_{1 \times N}$ and $\rho^*$ using Algorithm 2;
10    $\rho_{max} = (\rho^* > \rho_{max})?\rho^*:\rho_{max}$ ;
11    for each tier-$j$ and server-type $m$ do
12        Compute $curConfig = [n_{jm}]_{N \times M}$, where
          $n_{jm} = \lceil \beta_{jm} n'_j / (\mu_{jm} / min(\mu_{jm})) \rceil$ ;
13    end
14    if cost of $[n_{jm}]_{N \times M} < curCost$ then
15        $curCost$ = cost($[n_{jm}]_{N \times M}$);
16        $\rho_l$ = utilization of $curConfig$;
17        $\rho_n = (\rho_{max} + \rho_l)/2$;
18    else
19        $\rho_n = (\rho_{max} + \rho_n)/2$;
20    end
21    if $|\rho_n - \rho_{max}| \leq \delta$ then
22        ITER=false;
23    end
24    Compute new $hybConf$ after solving (23) and (24).
25 end
26 return $[n_{ji}]_{N \times M}$
```

**Algorithm 3:** Find number of servers required at each tier in a heterogenous server setting

# B Stochastic Ordering of Approximate Response Time

We consider an M/G/1 queue with Poisson arrivals at rate $\lambda$ and service time $X$ with density function $f_X(x)$ and cumulative density function (CDF) $F_X(x) = \mathrm{P}(X < x)$, $x \geq 0$. We assume that customers are served under the processor sharing discipline. Under the assumption of iid service times and $\rho = \lambda \mathrm{E}[X]$,the sytem exhibits steady state behavior. Let $\tau$ denote the response time. We are interested in approximating its density We are interested in approximating the response time distribution. Let $\tau(x) = \mathrm{E}[T|X = x]$ denote the expected response time under PS conditioned on the service time being $x$. It is given as

$$\tau(x) = x/(1-\rho), \quad x \geq 0$$

Hence $\tau(x)$ is a random variable and we drop the dependence on $x$; it has the following cumulative distribution,

$$F_\tau(y) = \int_0^y dF_X(x(1-\rho), \quad y \geq 0$$

Note that $\mathrm{E}[\tau] = \mathrm{E}[T]$. We define the following stochastic ordering relation that we will use to order $T$ and $\tau$. Let $X, Y \in \mathbb{R}^+$ be rvs with cdfs $F_X(x), F_Y(x)$, $x \geq 0$. We say that $X$ is smaller than $Y$ in the convex ordering sense (written $X \leq_{cx} Y$ iff $\mathrm{E}[f(X)] \leq \mathrm{E}[f(Y)]$ for all convex functions $f$. Moreover, (see [19]), $X \leq_{cx} Y$ iff

$$\int_0^y F_X(x)dx \leq \int_0^y F_Y(x)dx, \quad \forall y \geq 0.$$

We have the following result

**Theorem B.1** $\tau \leq_{cx} T$.

**Proof.** Note that $\tau$ conditioned on the service time being $x$ has the following cumulative distribution,

$$F_{\tau|X=x}(u|X=x) = \begin{cases} 0, & u < x/(1-\rho), \\ 1, & u \geq x/(1-\rho) \end{cases}$$

the following relation holds,

$$\int_0^y F_{\tau|X=x}(u|X=x)du \leq \int_0^y F_{T|X=x}(u|X=x)du, \quad y \geq 0$$

because a positive valued random variable taking value $d > 0$ is smaller in the sense of convex order than a positive random variable with arbitrary distribution and mean $d$. Now focus on $\tau$,

$$\begin{aligned}
\int_0^y F_\tau(u)du &= \int_0^y \int_0^\infty F_{\tau|X=x}(u|X=x)dF_X du \\
&= \int_0^\infty \int_0^y F_{\tau|X=x}(u|X=x)dudF_X \\
&\leq \int_0^\infty \int_0^y F_{T|X=x}(u|X=x)dudF_X \\
&= \int_0^y F_T(u)du
\end{aligned}$$

As this holds for all $y \geq 0$, we conclude that $\tau \leq_{cx} T$.