

# Translating Service Level Objectives to lower level policies for multi-tier services

Yuan Chen · Subu Iyer · Xue Liu · Dejan Milojevic · Akhil Sahai

Received: 4 June 2008 / Accepted: 16 June 2008 / Published online: 16 July 2008  
© Springer Science+Business Media, LLC 2008

**Abstract** Service providers and their customers agree on certain quality of service guarantees through Service Level Agreements (SLA). An SLA contains one or more Service Level Objectives (SLO)s that describe the agreed-upon quality requirements at the service level. Translating these SLOs into lower-level policies that can then be used for design and monitoring purposes is a difficult problem. Usually domain experts are involved in this translation that often necessitates application of domain knowledge to this problem. In this article, we propose an approach that combines performance modeling with regression analysis to solve this problem. We demonstrate that our approach is practical and that it can be applied to different  $n$ -tier services. Our experiments show that for a typical 3-tier e-commerce application in a virtualized environment, the SLA can be met while improving CPU utilization by up to 3 times.

**Keywords** SLA management · Performance modeling · Multi-tier application · Queueing model

## 1 Introduction

In a typical scenario, a Service Provider agrees on an SLA with a customer and then a service administrator on behalf of the service provider designs the service and then stages it. Staging is typically an iterative process where the system is

observed under the desired workload. During this stage, incremental changes are applied to the initial design to obtain the desired quality levels. Once the system administrator is satisfied with the performance of the service, it is put into production.

Multi-tier services are becoming quite common in today's enterprises. Such services are typically comprised of a large number of components, which interact with one another in a complex manner. Since each sub-system or component potentially affects the overall behavior of the system, any high level goal (e.g., performance, availability, security, etc.) specified for the service potentially relates to many and in some cases all low-level sub-systems or components. One of the key tasks during the design stage of such a service is to undertake SLA Decomposition—translate high-level service level objectives to low level system thresholds. The thresholds can then be used to create efficient designs to meet the SLA. For example, the system thresholds are used to determine how much and how many of the resources should be allocated to satisfy the SLA. With the advent of virtualization and application sharing techniques, opportunities exist for improving overall system performance and resource utilization by allocating optimal resources for the service.

Domain experts bring their knowledge to bear upon the problem of SLA decomposition. Automatically deriving and inferring low level thresholds from high level goals and thus eliminating domain experts from this process is a difficult task due to the complexity and dynamism inherent in building such systems. The range of design choices in terms of operating systems, middleware, shared infrastructures, software structures etc. further complicate the problem. For example, different virtualization technologies (e.g., Xen [1] or VMware [23]) can be used in a utility data center. Applications can use different software structure (e.g., 2-tier PHP,

---

Y. Chen (✉) · S. Iyer · D. Milojevic · A. Sahai  
Hewlett Packard Labs, Palo Alto, CA 94304, USA  
e-mail: yuan.chen@hp.com

X. Liu  
School of Computer Science, McGill University, Montreal,  
H3A 2A7, Canada

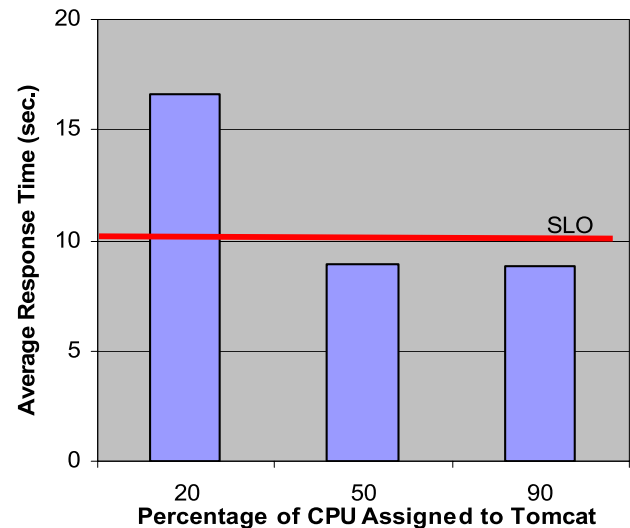
3-tier Servlet, or 3-tier EJB [2]) to implement the same functionality. Different implementations are also available for each tier (e.g., Apache or IIS for web server; WebLogic, WebSphere, or JBoss for EJB server; Microsoft SQL Server, Oracle, or MySQL as database server).

In this article, we have applied our approach to SLOs that relate to performance. Our approach captures the relationship between high level performance goals (e.g., response time of the overall system) and the refined goals for each component (e.g., CPU shares required for the component) through an analytical model. In particular, we present a queueing network model for multi-tier architecture, where each tier is modeled as a multi-station queueing center. Our model is sufficiently general to capture a number of commonly used multi-tier applications with different application topology, configuration, and performance characteristics. Our approach also builds profiles characterizing per-component performance metrics (e.g., average service time) as functions of resource allocations (e.g., CPU, memory) and configuration parameters (e.g., max connections). We use a two-step approach for SLA Decomposition. First the analytical models provide relationship between the high level performance goals and the low level component goals. Second, the component profiles provide component level resource requirements and configuration, which can meet high level goals. The low level goals can then be used to create efficient designs to meet the high level SLA. Some of the thresholds, such as healthy ranges of lower level metrics, are used for monitoring the systems during operation. The developed models including both analytical models and component profiles are archived for future reuse.

The remainder of this paper is organized as follows. Section 2 describes a motivating scenario for SLA decomposition in a virtualized data center. Section 3 provides an overview of our approach. We then describe in detail an analytical performance model for multi-tier applications in Sect. 4. Section 5 presents the implementation of profiling and decomposition of a multi-tier application as the experimental validation of our approach. Related work is discussed in Sect. 6. Section 7 concludes the paper and discusses future work.

## 2 Motivating scenario

Today's enterprise data centers are designed with on-demand computing and resource sharing in mind, where all resources are pooled into a common shared infrastructure [9]. Virtualization technologies such as VMware ESX Server [23] and Xen Virtual Machine Monitor [1] enable applications to share computing resources with performance isolation. Such a model also allows organizations to flex their computing resources based on business needs. These



**Fig. 1** Performance of a multi-tier application in a virtualized data center

data centers host multiple applications (often from different customers).

Consider a typical 3-tier application consisting of a web server, an application server and a database server in the virtualized data center, each tier hosted on a virtual machine. Figure 1 shows the application's average response time with three different CPU shares assigned to the virtual machine hosting the application server tier (i.e. Tomcat). Given the SLO of average response time less than 10 seconds, the configuration with CPU assignment of 20% fails to meet the SLO while the CPU assignment of 90% meets the SLO. However, the system is over-provisioned in this case since CPU assignment of 50% is sufficient to ensure the SLO. One key task of designing such a system is to determine the resource requirement of each tier to meet high level SLA goals while achieving high resource utilization. For the above example, SLA decomposition determines the CPU assignment to Tomcat, e.g., "CPU assignment = 50%" such that if the virtual machine is configured that way, the application will meet the response time requirement with reasonable CPU utilization.

## 3 SLA decomposition

Given high level goals, SLA decomposition translates these goals into bounds on low level system metrics such that the high level goals are met. In other words, the task of SLA decomposition is to find the mapping of overall service level goals (e.g., SLOs) to the state of each individual component involved in providing the service (e.g., resource requirement and configuration). For example, given SLOs of a typical 3-tier e-commerce environment in terms of response time and throughput requirement, the decomposition task is to

find the following mapping  $R \rightarrow (\theta_{\text{http-cpu}}, \theta_{\text{app-cpu}}, \theta_{\text{db-cpu}})$  where  $R$  denote the response time of the service and  $\theta$  is the CPU share required to meet the response time requirement. The SLA decomposition problem is the opposite of a typical performance modeling problem, where the overall system’s performance is predicted based on the configuration and resource consumption of the sub-components.

Multiple system resources (CPU, Memory, Disk, etc.) can become bottlenecks. We focus on CPU resource in this work since CPU is often the key resource in determining the performance of multi-tier applications. The conceptual architecture of our approach is illustrated in Fig. 2. We benchmark the application and generate a detailed performance profile for each component. For example, for a 3-tier application consisting of http web server, application server, and database server, we obtain the service rate as a function of CPU share that is allocated to the server for each tier  $\mu_{\text{http}} = f_1(\text{CPU}_{\text{http}})$ ,  $\mu_{\text{app}} = f_2(\text{CPU}_{\text{app}})$ ,  $\mu_{\text{db}} = f_3(\text{CPU}_{\text{db}})$ . An analytical model is built to capture the relationship between the application’s high level goals (e.g., application’s response time  $R$ ) and lower level goals (e.g., CPU share assigned to http web server  $\text{CPU}_{\text{http}}$ , application server  $\text{CPU}_{\text{app}}$  and database server  $\text{CPU}_{\text{db}}$  and workload  $w$ :  $R = g(\mu_{\text{http}}, \mu_{\text{app}}, \mu_{\text{db}}, w)$ . Given a response time requirement of  $R < r$  and workload  $w$ , we then use the profiles and analytical models to generate CPU share thresholds of each tier and response times at each tier:  $\text{CPU}_{\text{http}} > v_1$ ,  $\text{CPU}_{\text{app}} > v_2$ ,  $\text{CPU}_{\text{db}} > v_3$  where  $v_1, v_2, v_3$  are the minimum CPU shares required for web, app and db tier respectively to meet the response time requirement  $R < r$ .

### 3.1 Component profiling

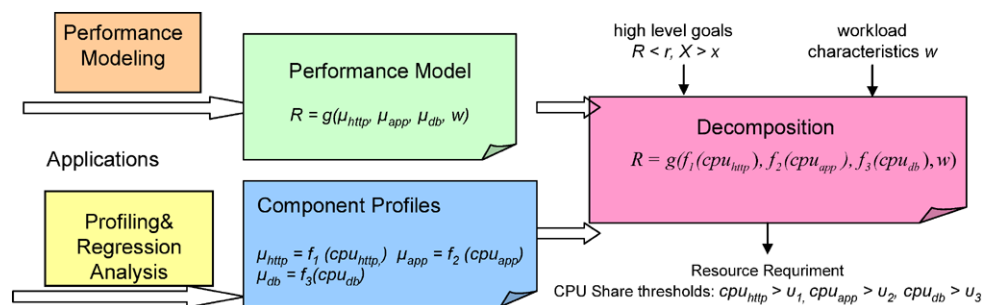
This step creates detailed profiles of each component. A component profile captures the component’s performance characteristics as a function of the resources that are allocated to the component and its configuration. In order to obtain a component profile, we deploy a test environment and change the CPU resources allocated to each component. We then apply a variety of workloads by changing the workload intensity (e.g., number of users) and collect the component’s performance characteristics independent of

other components (e.g., mean service rate  $\mu$ ). After acquiring the measurements, general functional mappings from system metrics to the component’s performance metrics are derived using either a classification or regression analysis based approach. For example, Apache Web server’s profile captures the relationship between an Apache Web server’s mean service rate and the CPU share allocated to it, i.e.  $\mu_{\text{http}} = f(\text{CPU}_{\text{http}})$ . The profiling can be performed either through operating system instrumentation [10] or estimation based on application or middleware’s monitoring information [21] (e.g., service time recorded in Apache and Tomcat log file). The former approach can achieve transparency to the application and component middleware but may involve changes to the system kernel while the latter approach is less intrusive. There are typically multiple classes of users or sessions in realistic applications, representing different SLAs (e.g., Gold customer, Silver customer and Bronze customer) and heterogeneous workloads (e.g., Browsing-heavy transactions, Purchase-heavy transactions). For an application that involves multi-classes of users, we obtain a profile for each user class by varying the workload intensity for each class of user.

### 3.2 Performance modeling

Performance modeling captures the relationship between each single component and the overall system performance. For example, given performance characteristics of each of the components in a 3-tier application,  $\mu_{\text{http}}, \mu_{\text{app}}, \mu_{\text{db}}$ , and the workload characteristics of the overall system  $w$ , model  $R = g(\mu_{\text{http}}, \mu_{\text{app}}, \mu_{\text{db}}, w)$  predicts the response time of the 3-tier application. We propose a queueing network model of multi-tier applications. In this model, the server at each tier is modeled as a multi-station queueing center (i.e.,  $G/G/K$  queue) which represents the multi-threaded architecture commonly structured in modern servers (e.g., Apache, Tomcat, JBoss, and MySQL). An application with  $N$  tiers is then modeled as a closed queueing network of  $N$  queues  $Q_1, Q_2, \dots, Q_N$ . Each queue represents a tier of the application and the underlying server that it runs on. Mean Value Analysis (MVA) [14] is used for evaluating the performance of the queueing network. For applications with

Fig. 2 SLA decomposition



multi-class users, we obtain a profile for each class user and apply a multi-class MVA algorithm. Such a model can handle user sessions-based workloads found in most e-business applications. Since we explicitly capture the concurrent limits in our model (e.g., max number of concurrent threads), this model inherently handles concurrent limits at tiers. The performance model is further discussed in Sect. 4.

### 3.3 Decomposition

Once we have the component profile,  $\mu_{\text{http}} = f_1(\text{CPU}_{\text{http}})$ ,  $\mu_{\text{app}} = f_2(\text{CPU}_{\text{app}})$ ,  $\mu_{\text{db}} = f_3(\text{CPU}_{\text{db}})$ , and the model  $R = g(\mu_{\text{http}}, \mu_{\text{app}}, \mu_{\text{db}}, w)$ , the decomposition of high level goals response time  $R < r$  is to find the set of  $\text{CPU}_{\text{http}}$ ,  $\text{CPU}_{\text{app}}$ ,  $\text{CPU}_{\text{db}}$  satisfying the following constraint:

$$g(f_1(\text{CPU}_{\text{http}}), f_2(\text{CPU}_{\text{app}}), f_3(\text{CPU}_{\text{db}}), w) < r$$

Once the equations are identified, the decomposition problem becomes a constraint satisfaction problem. Various constraint satisfaction algorithms, linear programming and optimization techniques are available to solve such problems [5]. Typically, the solution is non-unique and the solution space is large. However, for the problems we are studying, the search space is relatively small. For example, if we consider assigning CPU shares to virtual machines at a granularity of 5%. We can efficiently enumerate the entire solution space to find the solutions. Also, we are often interested in finding a feasible solution, so we can stop the search once we find one. Other heuristic techniques can also be used during the search. For example, the hint that the service time of the component typically decreases with respect to the increase of resource allocated to it can reduce the search space.

If the high level goals or the application structures change, we only need to change the input parameters of analytical models and generate new low level operational goals. Similarly, if the application is deployed to a new environment, we only need to regenerate a profile for new components in that environment. Further, given high level goals and resource availability, we can apply our decomposition approach for automatic selection of resources and for generation of sizing specifications that could be used during system deployment. The generated thresholds are used for creating an efficient design and by monitoring systems for proactive assessment of SLOs. The detailed implementations of modeling, profiling and decomposition of multi-tier applications in a virtual data center are discussed in the following two sections.

## 4 Modeling multi-tier Web applications

### 4.1 Basic queueing network model

Modern Web applications and e-Business sites are usually structured into multiple logical tiers, responsible for distinct

sets of activities. Each tier provides certain functionality to its preceding tier and uses the functionality provided by its successor to carry out its part of the overall request processing. Consider a multi-tier application consisting of  $M$  tiers,  $T_1, \dots, T_M$ . In the simplest case, each request is processed exactly once by each tier and forwarded to its succeeding tier for further processing. Once the result is processed by the final tier  $T_M$ , the results are sent back by each tier in the reverse order until it reaches  $T_1$ , which then sends the results to the client. In more complex processing scenarios, each request at tier  $T_i$ , can trigger zero or multiple requests to tier  $T_{i+1}$ . For example, a static web page request is processed by the Web tier entirely and will not be forwarded to the following tiers. On the other hand, a keyword search at a Web site may trigger multiple queries to the database tier.

Given an  $M$ -tier application, we model the application using a network of  $M$  queues  $Q_1, Q_2, \dots, Q_M$  (see Fig. 3). Each queue represents an individual tier of the application. Each queue models the request queue on the underlying server where it runs on. A request, after being processed at queue  $Q_i$  either proceeds to  $Q_{i+1}$  or returns to  $Q_{i-1}$ . A transition to the client denotes a request complementation (i.e. response to the client). We use  $V_i$  to denote the average request rate serviced by  $Q_i$ . Our model can handle multiple visits to a tier. Given the mean service time  $S_i$  of queue  $Q_i$ , the average service demand per user request  $D_i$  at  $Q_i$  can be approximated as  $S_i \times V_i / V_0$ , where  $V_0$  is average request rate issued by the users.

### 4.2 Multi-station queueing network model

Modern servers typically utilize a multi-threaded and/or multi-process architecture. The server listens in the main thread for requests. For each request, it allocates a thread to handle it. For example, the flow of servicing a static HTTP request is as follows. A request enters the TCP accept queue where it waits for a worker thread. A worker thread processes a single request to completion before accepting another new request. In the most general case, each of the tiers may involve multiple servers and/or multiple threads. The application server tier for example may involve one or more multi-threaded application servers (e.g., Tomcat) running on multiple processors. A similar notion is applicable to the database tier which may consist of one or more database servers (e.g., MySQL) which in turn may run on a multi-threaded/multi-processor system.

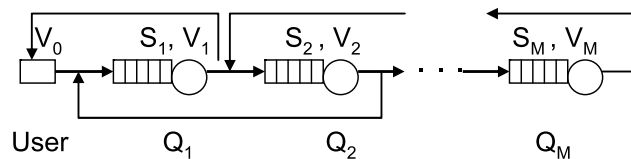
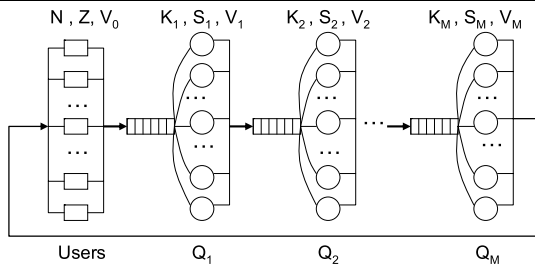


Fig. 3 Basic queueing network model



**Fig. 4** Closed multi-station queueing network

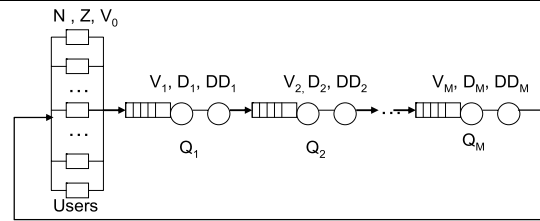
The amount of concurrency may also be determined by the number of processes or concurrent threads/servers the tier supports. In order to capture the multi-threaded/server architecture and the concurrency, we enhance the basic model by using a multi-station queueing center to model each tier. In this model, each worker thread/server in the tier is represented by a station. The multi-station queueing model thus is the general representation of modern server architecture.

### 4.3 Closed multi-tier multi-station queueing network model

The workload on a multi-tier application is typically user session-based, where a user session consists of a succession of requests issued by a user with think time  $Z$  in between. At a time, multiple concurrent user sessions interact with the application. In order to capture the user session workload and the concurrency of multiple sessions, we use a closed queueing network, where we model concurrent sessions by  $N$  users in the queueing system. Figure 4 shows the closed multi-station queueing network model (QNM) of a multi-tier application. Each tier is modeled by a multi-station queueing center as discussed earlier, with the number of stations being the server’s total number of worker threads. We use  $K_i$  to denote the number of worker threads at tier  $i$ . Similarly, the mean service time at tier  $i$  is denoted by  $S_i$ .  $V_i$  denotes the average request rate serviced by  $Q_i$  triggered by a user request. A user typically waits until the previous request’s response returns to send the following request. The average time elapsed between the response from a previous request and the submission of a new request by the same user is called the “think time”, denoted by  $Z$ .

### 4.4 Deriving queueing network performance

Given the parameters  $\{N, Z, V_i, M_i, S_i\}$ , the proposed closed queueing network model can be solved analytically to predict the performance of the underlying system. For example, an efficient algorithm such as the Mean-value analysis (MVA) can be used to evaluate the closed queueing network models with exact solutions [14]. MVA algorithm is iterative. It begins from the initial conditions when the system



**Fig. 5** Approximate model for MVA

**Fig. 6** Modified MVA algorithm

```

Input:  $N, Z, M, K_i, S_i, V_i (i = 1.. M)$ 
Output:  $R, X$ 
//initialization
 $R_0 = Z; D_0 = Z; Q_0 = 0;$ 
for  $i = 1$  to  $M$  {
  // Tandem approximations for each tier
   $Q_i = 0; D_i = (S_i * V_i) / V_0;$ 
   $qrD_i = D_i / K_i; drD_i = D_i * (K_i - 1) / K_i;$ 
  //introduce  $N$  users one by one
  for  $i = 1$  to  $N$  {
    for  $j = 1$  to  $M$  {
       $R_j = qrD_j * (1 + Q_j);$  // queueing resource
    }
     $RR_j = drD_j;$  //delay resource
  }
   $X = \frac{i}{R_0 + \sum_{j=1}^m (R_j + RR_j)}$ 
  for  $j = 1$  to  $M$ 
     $Q_j = X * R_j;$ 
  }
   $R = \sum_{i=1}^M (R_i + RR_i)$ 
}
    
```

population is 1 and derives the performance when the population is  $i$  from the performance with system population of  $(i - 1)$ , as follows

$$R_k(i) = \begin{cases} D_k & \text{delay resource} \\ D_k \times (1 + Q_k(i - 1)) & \text{queueing resource,} \end{cases}$$

$$X(i) = \frac{i}{\sum_{k=1}^K R_k(i)},$$

$$Q_k(i) = X \times R_k(i)$$

where  $R_k(i)$  is the mean response time (mean residence time) at server  $k$  when system population is  $i$ ;  $R_k(i)$  includes both the queueing time and service time;  $X(i)$  the total system throughput when system population is  $i$ ; and  $Q_k(i)$  is the average number of customers at server  $k$  when system population is  $i$ .

Traditional MVA has a limitation that it can only be applied to single-station queues. In our model, each tier is modeled with a multi-station queueing center. To solve this problem, we adopt an approximation method proposed by Seidmann et al. [16] to get the approximate solution of performance variables. In this approximation, a queueing center that has  $m$  stations and service demand  $D^1$  at each station

<sup>1</sup> $D_i$  represents the average service demand per user request at  $Q_i$ . It can be approximated as  $S_i \times V_i / V_0$ .

is replaced with two tandem queues. The first queue being a single-station queue with service demand  $D/m$ , and the second queue is a pure delay center, with delay  $D \times (m - 1)/m$ . It has been shown that the error introduced by this approximation is small [13]. By using this approximation, the final queueing network model is shown in Fig. 5 where  $D_i$  and  $DD_i$  are average demands of the regular queueing resource and the delay resource in the tandem queue respectively.

The modified MVA algorithm used to solve our queueing network is presented in Fig. 6. The algorithm takes the following set of parameters of a multi-tier application as inputs:

- $N$ : number of users
- $Z$ : think time
- $M$ : number of tiers
- $K_i$ : number of stations at tier  $i$  ( $i = 1, \dots, M$ )
- $S_i$ : service time at tier  $i$  ( $i = 1, \dots, M$ )
- $V_i$ : mean request rate at tier  $i$  ( $i = 1, \dots, M$ )

The MVA algorithm computes the average response time  $R$  and throughput  $X$  of the application.

#### 4.5 Handling multiple classes of users

There are typically multiple classes of users or sessions in realistic applications, representing different SLAs (e.g., Gold customer, Silver customer and Bronze customer) and heterogeneous workloads (e.g., Browsing-heavy transactions, Purchase-heavy transactions). Constructing multiple-class models for heterogeneous workload can accurately model heterogeneous workloads and differentiate SLA requirements of different classes. We extend our model to handle multiple classes of users. Let  $C$  be the number of classes. Each class  $C$  has a fixed number of users  $N_c$  with think time  $Z_c$ . Let  $S_{c,i}$  denote the service time of class  $C_i$  at tier  $i$  and  $V_{c,i}$  denote request rate of class  $C_i$  at tier  $i$ . The multi classes closed queueing network can be analytically solved by using an extension of the single class MVA algorithm [7]. The extended algorithm is presented in Fig. 7. The algorithm takes the following set of parameters of as inputs and computes the per-class response time  $R_c$  and throughput  $X_c$ .

- $C$ : number of classes
- $N_c$ : number of users of class  $c$  ( $c = 1, \dots, C$ )
- $Z_c$ : think time of class  $c$  ( $c = 1, \dots, C$ )
- $M$ : number of tiers
- $K_i$ : number of stations at tier  $i$  ( $i = 1, \dots, M$ )
- $S_{c,i}$ : service time of class  $c$  at tier  $i$  ( $c = 1, \dots, C, i = 1, \dots, M$ )
- $V_{c,i}$ : mean request rate of tier  $i$  ( $c = 1, \dots, C, i = 1, \dots, M$ )

The time complexity of the algorithm is  $CM \prod_{c=1}^C (N_c + 1)$  where  $CM$  is the time complexity of the computations for

```

Input:  $N_c, Z_c, M, K_i, S_{c,i}, V_{c,i}$  ( $i = 1, \dots, M, c = 1, \dots, C$ )
Output:  $R_c, X_c$  ( $c = 1, \dots, C$ )

//initialization
 $Q_0 = 0$ ;
 $N = \sum_{c=1}^C N_c$ ;
for  $i = 1$  to  $M$   $Q_i = 0$ ;
for  $c = 1$  to  $C$   $\{R_{c,0} = Z_c; D_{c,0} = Z_{c,i}\}$ 

for  $c = 1$  to  $C$ 
  for  $i = 1$  to  $M$   $\{$ 
    // Tandem approximations for each tier
     $D_{c,i} = (S_{c,i} * V_{c,i}) / V_{i,0}$ ;
     $qrD_{c,i} = D_{c,i}/K_i$ ;  $drD_{c,i} = D_{c,i} \times (K_i - 1)/K_i$ ;
   $\}$ 

for  $n = 1$  to  $N$ 
  for each feasible population with total number of  $n =$ 
    ( $n_1, \dots, n_C$ )
   $\{$ 
    for  $c = 1$  to  $C$   $\{$ 
      for  $i = 1$  to  $M$   $\{$ 
         $R_{c,i} = qrD_{c,i} \times (1 + Q_i)$ ; // queueing resource
         $RR_{c,i} = drD_{c,i}$ ; //delay resource
       $\}$ 
     $\}$ 
    for  $c = 1$  to  $C$ 
       $X_c = \frac{n_c}{R_{c,0} + \sum_{i=1}^M (R_{c,i} + RR_{c,i})}$ 
    for  $i = 1$  to  $M$ 
       $Q_i = \sum_{c=1}^C X_c R_{c,i}$ 
     $\}$ 
    for  $c = 1$  to  $C$ 
       $R_c = \sum_{i=1}^M (R_{c,i} + RR_{c,i})$ 
   $\}$ 

```

Fig. 7 Multi-class MVA algorithm

one feasible population, and the product term is the total number of feasible populations [7]. The space complexity is  $M \prod_{c=1}^C (N_c + 1)$ . The time and space complexities are proportional to the number of feasible populations and hence it can require excessive time and space for the large number of classes or large number of users. Approximate algorithms are often used in practice [7]. It has been demonstrated that approximate algorithms are quite accurate and require much less storage than the exact algorithm. The saving in time is considerable empirically though it is harder to quantify because of the iterative nature of the approximate algorithms. Parallel MVA algorithms have been proposed in [8, 24].

#### 4.6 Discussion

There are several limitations in our performance model. First, our queueing model captures system operation under steady-state conditions, and hence may not work well if the workload characteristics change very fast. Another limitation is that our model is we only calculate the mean response time which could be misleading. An enhancement of the model for handling bursty workloads and estimating probability distributions of response time is the subject of future work. Finally, one of assumption of queueing network model is resources held at exactly one tier. So our approach won't apply to certain internet applications where

a request could hold multiple resources simultaneously at multiple tiers/servers such as a video streaming application.

## 5 Profiling and SLA decomposition

### 5.1 Profiling

One of the key objectives of profiling is to accurately estimate the service time of each component since the accuracy of a model depends directly on the quality of its input parameters. When the scheduling discipline is processor sharing (PS) that is representative of scheduling policies in commodity operating systems such as Linux, the MVA algorithm works without making any assumptions about the service time distributions [7], hence it is sufficient to just capture mean service times without considering the variance. To effectively measure service time for each component, the time stamp is recorded either when a new thread is created or an idle thread is assigned. Similarly, we record the timestamp when the thread is returned to the thread pool or destroyed. The time interval between the two time stamps is the time spent in each component. This time also includes the waiting time for its neighbor's reply. The time spent on waiting from next tier is measured in a similar way. The difference between the two time intervals is the actual service time. This approach works for both lightly loaded as well as overloaded systems. Details of measurement implementation can be found in [12].

During profiling, we also collect the workload characteristics, including the average visit rate on each tier  $V_i$ . This number is used to derive the average service demand  $D_i$  per user request such as  $D_i = V_i/V_0 * S_i$ , where  $S_i$  is the mean service time and  $V_0$  is the average user request rate.

For the purpose of profiling, we change the configurations of each virtual machine hosting the application including assignment of CPU share to each virtual machine hosting a tier from 10% to 100% in 5% increments. We then apply workloads with different intensity (i.e., number of users) to each tier and measure the service time of that tier. When we profile a tier, we configure other tiers at their maximum capacity to prevent them from becoming performance bottlenecks. This ensures that interdependencies do not affect the accuracy of the profile. After collecting the service times for different CPU share, in general, we can apply statistical analysis techniques, such as regression analysis, to derive the relationship between the service times of a tier and its respective configuration. In our current implementation, we record (service time, CPU share) pair in a table as the tier's profile. For an application with multi-class users, the above process is repeated for each class of user and a profile is generated to store resource demand for each class of user.

A guideline regarding resource utilization for practitioners in real world is to keep peak utilizations of resources,

such as CPU, below 70% [4]. Actually, in practice, enterprise system operators are typically even more cautious than this conservative guideline. As shown in [19], service times remain relatively stable when the workload is not very intense and the CPU is not saturated though they become more sensitive to workload intensity when the server is heavily loaded, for example, when CPU utilization reaches 75% on the application server. If we assume that the system is not overloaded, then service times won't change with the change of the workload intensity. As a result, the profile we obtained is valid for different workload intensity.

Our current profiling takes into account the change in the volume of demand only (i.e., number of users), and assumes a fixed or stationary transaction mix. So the service time in a profile is actually an aggregated value for certain transaction mix and is only valid for that type of transaction mix. A more practical approach must handle workload changes in both the volume and transaction mix. To address this limitation, T. Kelly et al. proposed an approach to capture resource demand per transaction type [15]. We plan to apply this idea to estimate service times for non-stationary workloads. Finally, in order to apply our approach to SLA management, it is necessary to integrate automated profiling/benchmarking tools into our approach to simplify and automate the profiling process.

### 5.2 Decomposition

Our performance model can be represented as follows:

$$R = g(N, Z, M, K_i, S_i, V_i) \quad i = 1, \dots, M$$

where variables  $R$  and  $N$  denote response time and the number of concurrent users respectively. Please see Sect. 4 for the definitions of the other variables. We also obtain the service time profile  $S_i = f_i(\text{CPU}_i)$  profile at tier  $i$  that captures the relationship between service time and CPU share assigned to tier  $i$ , the number of stations  $K_i$  for each tier and the workload characteristics such as average visiting rate  $V_i$  and think time  $Z$  via profiling.

Given high level goals of  $R < r$  and  $N$  of a  $M$ -tier application, the decomposition problem is to find a set of  $\text{CPU}_i$  ( $i = 1, \dots, M$ ) that satisfies the following constraint:

$$g(N, Z, M, K_i, f_i(\text{CPU}_i), V_i) < r \quad (i = 1, \dots, M)$$

To find the solution to the above equation, we simply enumerate all combinations of CPU assignments,  $\text{CPU}_i = 10\%$  to 100%, in 5% unit increments that satisfy the constraints.  $f_i(\text{CPU}_i)$  can be obtained by looking up the profile table created during profiling phase. We then choose the combinations of  $\text{CPU}_i$  such as the sum of  $\text{CPU}_i$  ( $i = 1, \dots, M$ ) is minimized. For complex decomposition problem that involves large number of variables, a more advanced algorithm can be applied [25].

For an application with multi-classes users and different SLOs, we use multi-class MVA to construct the following constraints:

$$g_1(N, M, C, Z, K_i, f_{j,i}(\text{CPU}_i), V_{j,i}) < r_1,$$

$$i = 1 \dots M, j = 1 \dots C$$

$$g_2(N, M, C, Z, K_i, f_{j,i}(\text{CPU}_i), V_{j,i}) < r_2,$$

$$i = 1 \dots M, j = 1 \dots C$$

...

$$g_C(N, M, C, Z, K_i, f_{j,i}(\text{CPU}_i), V_{j,i}) < r_C,$$

$$i = 1 \dots M, j = 1 \dots C$$

where  $C$  is the number of classes and  $f_{j,i}(\text{CPU}_i)$  is the class  $j$ 's profile at tier  $i$  and  $V_{j,i}$  is the visit rate of class  $j$  at tier  $i$ , and  $r_i$  denotes the response time requirement of class  $i$ . We apply a similar enumeration algorithm to find a feasible CPU assignment that can meet all classes' response time.

## 6 Experiment evaluation

Our experimental testbed consists of a virtual data center where multiple applications share a common pool of resources. We use a cluster of dual processor x86 based servers with Xen virtual machines (VMs) to simulate such a virtual data center. The testbed consists of multiple HP Proliant servers, each running Fedora 4, kernel 2.6.12, and Xen 3.0-testing. Each of the server nodes has two processors, 4 GB of RAM, and 1G Ethernet interfaces. Each server has a set of VM images, database images, and swap images. These hardware resources are shared between the virtual machines that host the application.

### 6.1 Performance model

To validate the correctness and accuracy of our model, we experimented with two open-source 3-tier applications running on a virtualized Linux-based server testbed. The testbed is composed of four machines. One of them is used as client workload generator and the other three machines are used as Apache 2.07 web server, Tomcat 5.5 Servlet server and MySQL 5.0 database server respectively. Although the grouping of application tiers on each physical server can be arbitrary in principle, we specifically chose the design where different servers host different application tiers for two reasons. First, this is a natural choice for many consolidated data centers for potential savings in software licensing costs. Second, we want to evaluate our performance model by isolating possible interferences between virtual machines and minimizing the overhead introduced by Xen.

We measure the service time for each tier by computing the elapsed time when a thread is dispatched to process a new request at that tier and when it finishes the task. Another required parameter for our model is the number of stations for each queue or tier. For Apache and Tomcat, the

total number of stations is determined by the size of thread pool (i.e., maxClients in Apache and maxThreads in Tomcat). MySQL manages threads in a more dynamic fashion. Depending on the server configuration settings and current status, the thread may be either created new, or dispatched from the thread cache. The average number of all worker threads during a run is used to approximate the number of stations. This approximate model enables us to use load-independent multi-station queueing to model thread cache based server. Average visit rate of each tier is obtained from log files.

The first application we use is TPC-W [5], an industry standard e-commerce application. TPC-W specifies 14 unique Web interactions. The database is configured for 10,000 items and 288,000 customers. Session based workload is generated from a client program to emulate concurrent users. The think time is exponential distribution with a mean of 0.035 seconds. The max clients of Apache and max threads of Tomcat are set as 50. We change workload by varying the number of concurrent sessions generated by the workload generator. Each run lasts 200 seconds after 60 seconds of warm-up period. We measure different model input and output parameters during each run. We then apply MVA algorithm described in Sect. 4.4 to derive the response time and throughput. Figure 8 shows the results of the response time predicted by the model and directly measured for sessions varying from 1 to 200. From the figures, we can see that the analytic model does predict the performance of TPC-W accurately. The results predicted by our model are close to the measurement under different workloads.

To further validate the effectiveness of our performance model, we experimented with RUBiS. RUBiS defines 26 interactions and has 1,000,000 users and 60,000 items. The think time is exponential distribution with a mean of 3.5 seconds. We vary the workload from 100 to 300 and each run lasts 300 seconds with a 120 seconds warm up. Unlike the

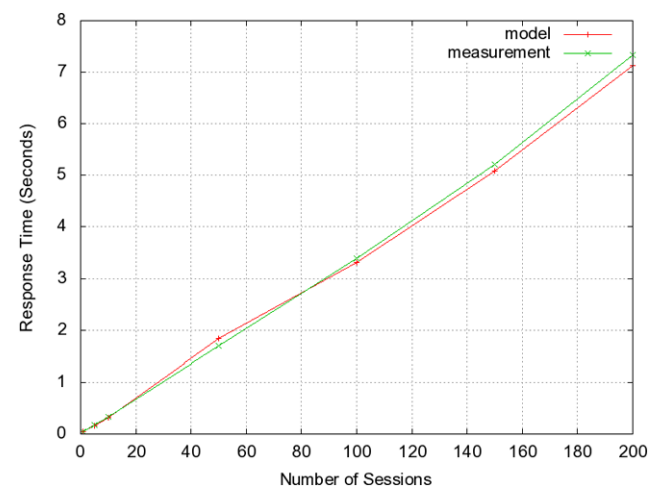


Fig. 8 TPC-W performance

TPC-W experiments, we use the same set of input parameters obtained during profiling to predict the performance for different workloads. The results of response time are depicted in Fig. 9. Even using the same set of model input parameters, the model can still predict the performance for different workloads.

### 6.2 Multi-class performance model

In this set of experiments, we create two classes of RUBiS users: Browse user and Bid user. A browse user session is browse transactions intensive while a bid user session mainly involves bidding-related transactions. In the first experiment, we fix the number of bid users at 50 and vary the number of browse users from 20 to 100. We calculate the average response time of browse users using the extended performance model and compare the results with the actual measurement. The results are shown in Fig. 10. The predicted response times match the measurements well. In the second experiment, we fix the number of browse users at 50 and change the number of bid users from 20 to 100. Similarly, we compare the predicted response times with the observed values for bid users. The results in Fig. 10 show that

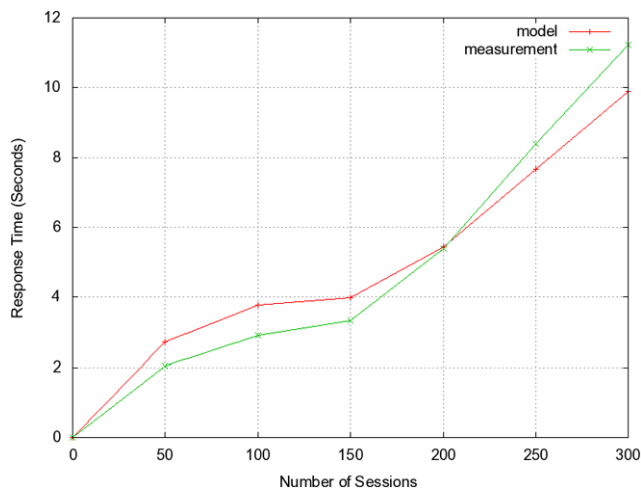
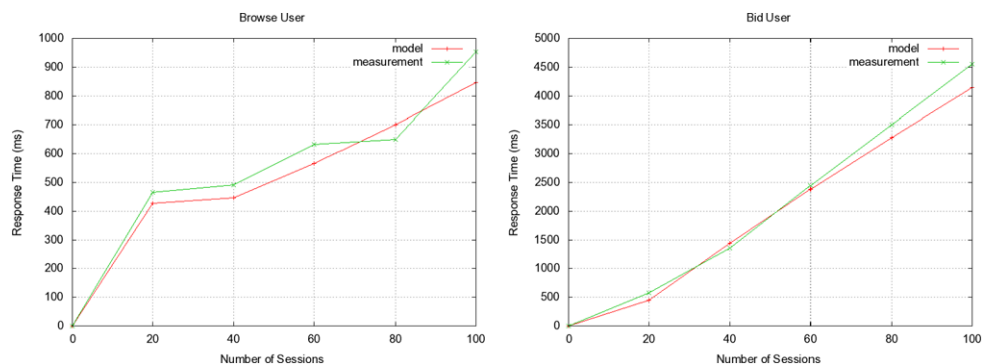


Fig. 9 RUBiS performance

Fig. 10 Multi-class RUBiS



our model can accurately predicate the performance of bid users as well.

### 6.3 Profile

For capturing profiling information, we use a 3-tier Servlet based implementation of RUBiS consisting of an Apache Web server 2.0, a Tomcat 5.5 Servlet container, and a MySQL 5.0 database server, running on virtual machines hosted on different servers. A synthetic workload generator runs on the fourth server. To isolate performance interference, we restrict the management domain to use one CPU and virtual machines to use the other CPU.

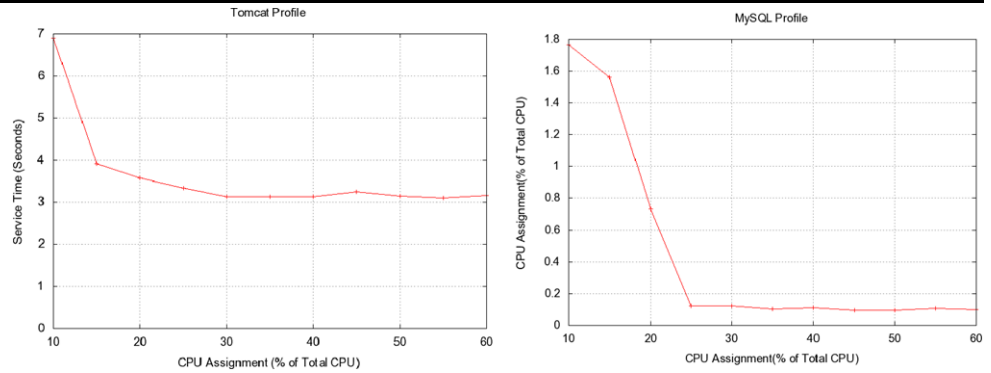
Using the steps mentioned above, we have built profiles for Apache, Tomcat and MySQL with different CPU assignments to each tier. We used SEDF (Simple Earliest Deadline First) algorithm for controlling the percentage of total CPU assigned to a virtual machine. We used the capped mode of SEDF to enforce the fact that a virtual machine cannot use more than its share of the total CPU. We changed the CPU assignment from 10% to 100% to measure the service times with different CPU assignments.

Figure 11 shows the service time of Tomcat and MySQL as a function of the percentage of CPU assignment to the virtual machine hosting them. As shown in the figure, both Tomcat and MySQL demonstrate similar behavior. As the CPU assignment increases, the service time drops initially and remains constant after getting enough CPU. The results are then saved as Tomcat and MySQL's profiles.

### 6.4 SLA decomposition

We apply SLA decomposition to design RUBiS, an eBay like auction site developed at Rice University [15]. We also experiment with various configurations of RUBiS systems with different SLA goals and software architectures. Given high level SLA goals, we generate low level CPU requirements through SLA decomposition and then configure the VMs based on the derived low level CPU requirements. We then validate our design by measuring the actual performance of the system and comparing the results with the

**Fig. 11** Service time profiles



**Table 1** Decomposition results of 3-tier RUBiS

SLOs	Design	CPU assignment to VMs		Actual performance resp. (sec)	VM CPU utilization	
		Tomcat	MySQL		Tomcat	MySQL
Users = 300	Optimal	40%	25%	9.79	67%	70%
Resp. < 10 sec	<b>System0</b>	<b>45%</b>	<b>30%</b>	<b>9.89</b>	<b>61%</b>	<b>67%</b>
	System1	20%	20%	15.93	99%	92%
	System2	90%	90%	8.86	23%	21%
Users = 100	<b>System0</b>	<b>15%</b>	<b>15%</b>	<b>4.83</b>	<b>71%</b>	<b>69%</b>
Resp. < 5 sec						

**Table 2** Decomposition results of 2-tier RUBiS

SLOs	Design	CPU assignment to VMs		Actual performance resp. (sec)	VM CPU utilization	
		Apache	MySQL		Apache	MySQL
Users = 100	System0	10%	15%	4.83	65%	53%
Resp. < 5 s						
Users = 500	System0	35%	30%	8.2	68%	61%
Resp. < 10 s						

**Table 3** Decomposition results of RUBiS with two classes of users

SLOs		CPU assignment to VMs		Actual response (sec)	VM CPU utilization	
		Tomcat	MySQL		Tomcat	MySQL
Class 1	Users = 100 Resp. < 1 s			Class 1 0.42 s	52%	47%
Class 2	Users = 20 Resp. < 2 s	30%	415%	Class 2 1.60 s		

SLA goals. In the experiments, we consider the high level SLA goals defined as number of concurrent users, average response time. We use 5% of the total CPU capacity as a unit for CPU assignments.

In the first experiment, we use a 3-tier implementation of RUBiS, an Apache Web server, a Tomcat server and a MySQL database server hosted on VMs on different servers. Table 1 summarizes the results of different CPU assign-

ments for two different SLA goals. The first column shows the SLA goals in terms of number of users and response time requirements. The column of *CPU assignment* describes the system design parameters in terms of the CPU share (in percentage of total physical CPU capacity) allocated to a virtual machine hosting an application tier. The column of *Performance* show the actually measured response time and the column of *VM CPU utilization* denotes the virtual machine's CPU utilization of the actual system.

For the SLA goal of *300 users, response time < 5 seconds*, optimal system ensures the SLA using the minimum CPU resource (i.e., assigned 40% CPU share to the VM hosting Tomcat and 25% to the VM hosting MySQL). System0 is the system designed based on the proposed SLA decomposition approach and the assignment of CPU share 45% to Tomcat VM and 30% CPU share to MySQL VM is very close to the optimal solution. System0 meets the SLAs with reasonable CPU utilization, i.e., the utilization of Tomcat VM and MySQL VM are 61% and 67% respectively. Two other systems (system1 and system2) are used for the purpose of comparisons, too. System1 is under-provisioned while system2 is over-provisioned. From the table, we observe that system1 fails to meet the SLA since the virtual machine hosting Tomcat is completely overloaded with CPU utilization 99% while system2 meets the SLAs but both virtual machines hosting Tomcat and MySQL are highly under-utilized with less than 25% CPU utilization. We also experimented with a less demanding SLA of *100 users and response time < 5 seconds*. These results are also summarized in Table 1, too. From the results, we can see that System0, which is designed based on the low level system thresholds derived by our approach, can meet the high level SLA with efficient CPU usage.

In order to further check the applicability of our approach, we applied the decomposition to design a 2-tier RUBiS implementation consisting of an Apache Web server and a MySQL database server. The 2-tier application runs PHP script at Web server tier and puts much higher load on Web tier than 3-tier. We evaluated our approach with two different SLAs. The results in Table 2 show that our approach can be effectively applied to design such a 2-tier system with different SLA requirements.

Finally, we experiment with RUBiS that involves two classes of users with different workloads and SLOs. The results are summarized in Table 3. As shown in the results, the design based on the SLA decomposition approach can meet both users' SLOs while maintaining reasonable CPU usage.

## 7 Related work

Previous studies have utilized performance models to guide resource provisioning and capacity planning [3, 22, 25]. Urgaonkar et al. propose a dynamic provisioning technique for

multi-tier applications [22]. Our work is different from theirs in several aspects. First, their model only takes into account the request rate and number of servers at each tier while our model can estimate how performance is affected by different workloads, resource allocations, and system configurations and can handle general SLAs, such as response time, throughput and the number of concurrent users. Second, they assume an open queueing network for request-based transactions whereas we assume a closed network for user session based interactions. Third, our approach has been applied in virtualized environments, managing resource assignment in a more fine-grained manner than just determining the number of servers for each tier. Zhang et al. present a nonlinear integer optimization model for determining the number of machines at each tier in a multi-tier server network [25]. The techniques to determine the bounds can be applied to solve our general decomposition problem.

Stewart et al. present a profile-driven performance model for multi-component online service [18]. Similar to ours, their approach builds profiles per component and uses the model to predict average response time and throughput. However, the basic assumption and the focus are different. They use the model to discover component placement and replication that achieve high performance in a cluster-based computing environment while our work is focused on ensuring SLAs are met with optimized resource usage. They profile component resource consumption as a function of different workloads while we profile the component performance characteristics as a function of low level goals such as resource assignments and configurations. As a result, our approach can support a more fine grained resource share and management. Additionally, their approach uses a simple  $M/G/1$  queue to model service delay at each server, which is less accurate than general  $G/G/1$  closed queueing network we used. Though our approach can incorporate different resources, we have not taken into account the I/O and memory profile as they did. Their model explicitly captures communication overhead which is not included in our current model.

A lot of research efforts have been undertaken to develop queueing models for multi-tier business applications. Many such models concern single-tier Internet applications, e.g., single-tier web servers [6, 11, 17, 20]. A few recent efforts have extended single-tier models to multi-tier applications [12, 21, 22]. The most recent and accurate performance model for multi-tier applications is proposed by Urgaonkar [21]. Similar to our model, their model uses a closed queueing network model and mean value analysis (MVA) algorithm for predicating performance of multi-tier applications. Despite the similarities, the two models are different in the following aspects. First, our model uses multi-station queues to capture the multi-thread architecture, hence explicitly handling the concurrency limits. Use of multi-station

queues also enables us to model multi-server tier the same way as single server tier. The approximate MVA algorithm for multi-station queue is more accurate than simply adjusting the total workload. Second, our measurement methodology can work well for both light load as well as heavily load conditions. Finally, we systematically study the performance and validate our models in a virtualized environment. These unique features enable us to model the application in a more fine-grained manner and handle various workload and conditions in a consistent way. Though our model can be adjusted to handle imbalance across tier replicas based on queueing theory, we have not explored these areas yet. An early result of our performance model was presented at [12]. The model used in this work makes numerous enhancements to our earlier model, including general model for any tier applications, an integrated MVA analysis with SLA decomposition and additional new experiments for model validation.

Kelley et al. present an approach to predicting response times as a function of workload [10]. The model does not require knowledge of internal application component structure and uses only transaction type information instead. It has been shown that the approach works well for realistic workload under normal system load, but it's not clear how well it will perform under high system load, which is crucial for our work. The specific performance model used in this paper is based on queueing model. Conceptually, any model which can help determine the performance (e.g., response time) of applications can be incorporated into our solution. It would be interesting to investigate how to integrate other models into our solution.

## 8 Conclusion and future work

It is a prerequisite for next generation data centers that computing resources are available on-demand and that they are utilized in an optimum fashion. One of the most important steps towards building such systems is to automate the process of designing and thereafter monitoring systems for meeting higher level business goals. These are intriguing but difficult tasks in IT automation. In this paper, we propose an SLA decomposition approach that combines performance modeling with performance profiling to solve this problem by translating high level goals to more manageable low-level sub-goals. These sub-goals feature several low level system and application level attributes and metrics which are used for creating an efficient design to meet high level SLAs. We have built a testbed to validate our methodology using a number of multi-tier business applications. The evaluation results show the efficacy of our approach.

In the future, we plan to address some limitations of our approach by extending the performance model to handle non-stationary dynamic workload, and multiple-resources in

addition to CPU usage and by deriving probability distribution of response time other than mean response time. Another interesting research topic is to apply the proposed approach to online SLA management by dynamically translating SLOs to resource requirements at runtime. We are also planning to incorporate deployment/benchmarking tool into our approach to simplify/automate the profiling process.

## References

1. Barham, P., et al.: Xen and the art of virtualization. In: Proc. of the Nineteenth ACM SOSP, 2003
2. Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., Zwaenepoel, W.: A comparison of software architectures for E-business applications. In: Proc. of 4th Middleware Conference, Rio de Janeiro, Brazil, June 2003
3. Chandra, A., Gong, W., Shenoy, P.: Dynamic resource allocation for shared data centers using online measurements. In: Proc. of International Workshop on Quality of Service, June 2003
4. Cockcroft, A., Walker, B.: Capacity Planning for Internet Services. Sun Press, Cleveland (2001)
5. Council, T.P.C.: TPC-W. <http://www.tpc.org/tpcw>
6. Doyle, R., Chase, J., Asad, O., Jin, W., Vahdat, A.: Model-based resource provisioning in a Web service utility. In: Proc. of the 4th USENIX USITS, Mar. 2003
7. Edward, D., Lazowska, J., Zahorjan, G., Graham, S., Sevcik, K.C.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall, Englewood Cliffs (1984)
8. Gennaro, C., King, P.J.B.: Parallelising the mean value analysis algorithm. *Simulation* **72**(3), 148 (1999). doi:10.1177/003754979907200304
9. Graupner, S., Kotov, V., Trinks, H.: Resource-sharing and service deployment in virtual data centers. In: Proc. of the 22nd ICDCS, pp. 666–674, July 2002
10. Kelley, T.: Detecting performance anomalies in global applications. In: Proc. of Second USENIX Workshop on Real, Large Distributed Systems (WORLDS 2005), 2005
11. Levy, R., Nagarajarao, J., Pacifici, G., Spreitzer, M., Tantawi, A., Youssef, A.: Performance management for cluster based Web services. In: Proc. of IFIP/IEEE 8th IM, 2003
12. Liu, X., Heo, J., Sha, L.: Modeling 3-tiered Web applications. In: Proc. of 13th IEEE MASCOTS, Atlanta, Georgia, 2005
13. Menasce, D., Almeida, V.: Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice-Hall PTR, Englewood Cliffs (2001)
14. Reiser, M., Lavenberg, S.S.: Mean-value analysis of closed multichain queueing networks. *J. ACM* **27**, 313–322 (1980). doi:10.1145/322186.322195
15. RUBiS: Rice University Bidding System. <http://www.cs.rice.edu/CS/Systems/DynaServer/rubis>
16. Seidmann, A., Schweitzer, P.J., Shalev-Oren, S.: Computerized closed queueing network models of flexible manufacturing systems. *Large Scale Syst.* **12**, 91–107 (1987)
17. Slothouber, L.: A model of Web server performance. In: Proc. of Int'l World Wide Web Conference, 1996
18. Stewart, C., Shen, K.: Performance modeling and system management for multi-component online services. In: Proc. of USENIX NSDI, 2005
19. Stewart, C., Kelly, T., Zhang, A.: Exploiting nonstationarity for performance prediction. In: Proc. of EuroSys, 2007
20. Urgaonkar, B., Shenoy, P.: Cataclysm. Handling extreme overloads in Internet services. In: Proc. of ACM SIGACT-SIGOPS PODC, July 2004

21. Uргаonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An analytical model for multi-tier Internet services and its applications. In: Proc. of ACM SIGMETRICS, June 2005
22. Uргаonkar, B., Shenoy, P., Chandra, A., Goyal, O.P.: Dynamic provisioning of multi-tier Internet applications. In: Proc. of IEEE ICAC, June 2005
23. VMware, Inc., VMware ESX Server User's Manual Version 1.5, Palo Alto, CA, April 2002
24. Yaikhom, G., Cole, M., Gilmore, S.: Combining measurement and stochastic modelling to enhance scheduling decisions for a parallel mean value analysis algorithm. In: Proc. of International Conference on Computational Science (ICCS 2006), LNCS. Springer, Berlin (2006)
25. Zhang, A., Santos, P., Beyer, D., Tang, H.: Optimal server resource allocation using an open queueing network model of response time. HP Labs Technical Report, HPL-2002-301



**Yuan Chen** is a researcher at HP Labs in Palo Alto, CA and has over 10 years of research and development experience in the design, implementation and management of distributed and enterprise systems. Yuan received a B.S. from University of Science and Technology of China, a M.S. from Chinese Academy of Sciences, and a Ph.D. from Georgia Institute of Technology, all in computer science. Yuan joined HP Labs in 2005. Yuan's current research has been in the area of automated management of large scale, complex enterprise systems and services with a focus on SLA based management. His past work includes policy-based management, adaptive monitoring, distributed event systems and middleware, Internet storage systems and high performance computing. Yuan has published over 30 technical papers and filed 3 patents.



**Subu Iyer** is a senior researcher at HP Labs, Palo Alto, CA. Subu engages in research associated with enterprise systems and in particular automation of large distributed systems including data centers. His recent interests are in the areas of distributed systems, operating systems, and computer networks with a current focus on model based automation. Subu joined HP in 2002 from Compaq Computer Corp where he was a research engineer at Compaq's Western Research Laboratory (WRL) in Palo Alto, CA. Prior to that, Subu worked at DEC's Network Systems Laboratory (NSL) since 1997. During his tenure at HP, Subu has contributed to research and development in the areas of performance monitoring, virtualization, power management, multimedia and automation. He has authored more than 25 peer reviewed papers in the area of multimedia and enterprise systems and has been granted one U.S. patent, with 15 pending. He is a member of ACM.



**Xue Liu** received the B.S. degree in applied mathematics and the M.Eng. degree in control theory and applications from Tsinghua University in 1996 and 1999, respectively. He received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2006. He is currently an assistant professor in the School of Computer Science at McGill University. He is also affiliated with the McGill Center for Intelligent Machines. His research interests include real-time and embedded computing, performance and power management of server systems and data centers, sensor networks, fault tolerance, and control. He has authored/coauthored more than 40 refereed publications in leading conferences and journals in these fields. He is a member of the ACM and the IEEE.



**Dejan Milojicic** is a senior researcher manager at HP Labs. He has worked in the area of operating systems and distributed systems for more than 20 years. He has been the program chair of the IEEE Agent Systems and Applications Symposium (ASA/MA'99) and of the first USENIX Workshop on Industrial Experiences with System Software (WIESS'2000). Dr. Milojicic published in many journals and at various events. He is currently editor IEEE Computing Now portal. He has been engaged in various standardization bodies, such as OMG and Global Grid Forum. He is a member of the ACM, IEEE, and USENIX. He received his B.Sc. and M.Sc. from University of Belgrade and his PhD from University of Kaiserslautern. Prior to HP Labs, Dejan worked at Institute "Mihajlo Pupin", Belgrade and at OSF Research Institute, Cambridge, MA.



**Akhil Sahai** is a senior product manager at VMware. He manages the Virtual Appliance product offerings. He was a senior researcher at HP Laboratories from 2000–2007. He has a Ph.D. from INRIA-IRISA and an MBA from Wharton School of Business. He has over 75 refereed publications, 4 book chapters, and has authored a book "Web Services in the Enterprise: Concepts, Standards, Solutions and Management". He has 3 granted patents with 19 pending. He has been in the Technical program Committees and co-chairs of large number of IEEE Conferences.