A Systematic and Practical Approach to Generating Policies from Service Level Objectives

Yuan Chen, Subu Iyer, Dejan Milojicic
Hewlett Packard Laboratory
Palo Alto, CA, USA
{firstname.lastname}@hp.com

Akhil Sahai
VMware Inc.
Palo Alto, CA, USA
asahai@vmware.com

***Abstract*—In order to manage a service to meet the agreed upon SLA, it is important to design a service of the required capacity and to monitor the service thereafter for violations at runtime. This objective can be achieved by translating SLOs specified in the SLA into lower-level policies that can then be used for design and enforcement purposes. Such design and operational policies are often constraints on thresholds of lower level metrics. In this paper, we propose a systematic and practical approach that combines fine-grained performance modeling with regression analysis to translate service level objectives into design and operational policies for multi-tier applications. We demonstrate that our approach can handle both request-based and session-based workloads and deal with workload changes in terms of both request volume and transaction mix. We validate our approach using both the RUBiS e-commerce benchmark and a trace-driven simulation of a business-critical enterprise application. These results show the effectiveness of our approach.**

I. INTRODUCTION

A Service Level Agreement (SLA) captures the agreed upon guarantees between a service provider and its customer. The ability to deliver according to a pre-defined SLA is an increasingly critical need in today's highly complex and dynamic IT environments. In order to manage a service to meet the agreed upon SLA, it is important to design a service of the required capacity and to monitor the service thereafter for violations at runtime.

In the past, researchers have made many efforts to address this problem using techniques such as automated provisioning, capacity planning, and monitoring [4, 7, 8, 9, 10, 15, 16, 17]. In particular, we proposed an approach to decompose SLOs to system thresholds on virtualized platforms in [12]. Results from these research efforts are encouraging. However, these works made several simplifying assumptions. As a result, the practicality and effectiveness of these approaches pose major challenges to their applicability. We have identified four main problems associated with existing solutions that are described below.

First, workloads in real applications are dynamic and vary over time. Unfortunately, most existing solutions take into account the change in the volume of demand only, and assume a fixed or stationary transaction mix [12, 16, 17]. Changes in the volume of transactions (e.g., request rate) or the mixture of transaction types can dramatically alter an application's performance and resource. Hence, a practical approach must handle workload changes in both the volume and transaction mix.

Second, existing solutions model enterprise application workloads as either request-based (open workload) or session-based (closed workload) [4, 12, 13, 16, 17]. In reality, workloads are typically semi-open, which is significantly different than either an open or a closed model [3]. Hence, a single model approach in most existing solutions is not sufficient to handle the diversity in realistic workloads. A practical approach should support multiple models and choose an appropriate model that is based on the properties of the real workload.

Third, building accurate performance models typically requires input parameters such as resource demand. However, most existing solutions cannot provide the needed model parameters directly. Instead, such information must be obtained through application or system instrumentation. In practice, instrumentation of production applications is rarely done, as it is difficult, costly, and may introduce overhead that degrades the application's performance [2]. Hence, a practical approach should be non-intrusive and passively utilize data that is already available on most systems.

Lastly, most existing solutions are not applicable to the diverse range of design and implementation choices. Many of them make simplifying assumptions about the application infrastructure, such as considering only one server per tier [12,13] or uniformly distributing the requests across the different tiers [17]. To cope with the diversity and complexity in real applications, a model must be sufficiently general to capture the behavior of applications with different configurations, workloads and performance characteristics.

In this paper, we propose a systematic, non-intrusive and practical approach to address the above issues. Our approach combines a fine-grain performance model and a regression-based profiling technique to translate low-level operational policies from high-level objectives for multi-tier applications. We formalize this translation as a constraint optimization problem, and develop a constraint solver to solve it. Compared with previous work [12], our approach provides the following four key contributions. First, it formally characterizes both request-based and session-based workloads. This enables us to choose an appropriate model based on the workload characteristics of the application. Second, our approach models workload as a transaction mix, and systematically creates a resource profile for each transaction type. This fine-grained model enables us to deal with dynamic and non-stationary workloads. Third, we use regression analysis to estimate the model parameters. The data used in our approach is readily available from regular system and application monitoring and requires no additional instrumentation. It is hence practical to apply our approach to production environments. Finally, the proposed modeling technique can model multi-tier applications with different topologies (i.e., any number of tiers and any number of servers at each tier), and different workloads (open and/or closed). As a result, our performance model and approach can be applied to a vast variety of common multi-tier applications.

The remainder of this paper is organized as follows. Section 2 provides an overview of our approach and our workload model. In Section 3, we describe profiling in detail. We present an analytical performance model for multi-tier applications in Section 4 and our approach in Section 5. The experimental validation of our approach is presented in Section 6. Related work is discussed in Section 7. Section 8 concludes the paper and discusses future work.

II. OVERVIEW OF THE APPROACH

A. Workload Model

Multi-tier applications are common in modern enterprises. Such applications are comprised of a large number of components, which interact with one another in complex patterns. Typically, multi-tier applications are structured into multiple logical tiers. 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.

Table 1. Workload Definition

Type	Workload Parameters
Open	*N*: number of transaction types $(\lambda_1, \lambda_2, \ldots \lambda_N)$: transaction mix where λ_i $(i = 1 \ldots N)$ is the arrival rate of requests of transaction type i during certain time interval
Closed	*N*: number of transaction types *C*: number of users *Z*: think time $\pi (p_1, p_2, \ldots p_i, \ldots p_N)$: transaction mix distribution where p_i $(i = 1, \ldots N)$ is the percentage of requests of transaction type *i*

There are typically a number of transaction types in any multi-tier application. For example, an online auction application has transaction types such as login, browse, bid, etc. In most cases, different transaction types have different service demands on resources. For example, bid transactions in an auction site typically require more CPU time than browse transactions. As previously discussed, empirical workloads tend to be partially-open, which means a user arrives and stays for a certain amount of time (and issues a number of requests) before they leave. Previous work has shown that partly-open workloads can be approximated using an open workload if the number of requests in a session is small, and a closed workload otherwise [3]. We consider these two types of workloads in our workload model.

Open Workload. In an open (request-based) workload, a new request to the application is only triggered by a new user arrival. The requests are independent of each other and the arrival rate is not influenced by the number of requests that have already arrived and are being processed. The number of users who interact with the application at any time may range from zero to infinity. An open workload is characterized by an average arrival rate of requests or more generally by an arrival distribution. A typical open workload is a transaction mix of different transaction types. Assume the total number of transaction types is *N*. We define an open workload during a certain interval (e.g., 5 minutes) as a vector $(\lambda_1, \lambda_2, \ldots \lambda_N)$ where λ_i is the arrival rate of transaction type i during that interval.

Closed Workload. In a closed (session-based) workload, a fixed number of users interact with the application and each of these users issues a succession of requests. A new request from a user is only triggered after the completion of a previous request by the same user. A user submits a request, waits for the response to that request, thinks for a certain time and then sends a new 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*. The next request sent by a user is usually determined by a state transition matrix that specifies the probability to go from one transaction type to another. Assume the number of transaction types is *N*. The state transition matrix has *N* rows and *N* columns where p_{ij} represents the transition probability from transaction type i to transaction type *j*. Let *P* denote a state transition matrix of a closed workload and $\pi = (p_i, p_2 \ldots p_N)$ denote the stationary transaction distribution in a user session where p_i presents the percentage of requests of transaction type i sent by the user based on *P*. We have $\pi P = \pi$ and $\sum_{i=1}^{N} p_i = 1$. We can use the workload with a stationary transaction mix π to approximate the behavior of a closed workload with state transition matrix *P* [4]. A closed workload is characterized by the number of concurrent users *C*, the stationary distribution of transaction mix π, and the think time *Z*.

The open and closed workload models are summarized in Table 1. Unlike many open workload models that assume a static transaction mix and hence use an aggregate request rate to characterize the workload, our transaction vector model captures request rate per transaction type and hence can characterize dynamic transaction mixes. This is important because transaction mixes in real production systems change over time [13].

B. The Approach

An SLA is comprised of multiple Service Level Objectives (SLOs). Our aim is to translate SLOs into design parameters and bounds on low-level system resources such that the high-level SLOs are met. Given a high-level performance SLO and a workload for a multi-tier application (in terms of either a transaction mix for an open workload or a transaction distribution for a closed workload), our approach provides the resource requirements (e.g., number of servers) to handle the workload and meet the specified SLO. It also finds the healthy state of each component involved in providing the services (e.g., resource utilization).

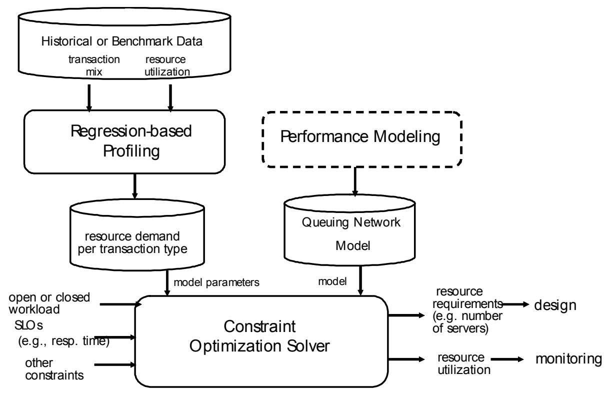

Figure 1. Conceptual Architecture

Our approach is illustrated in Figure 1. First, we use analytical performance models to capture the relationship between high-level performance goals (e.g., response time of the overall system), the application topology, and the resource usage of each component (e.g., CPU utilization). In particular, we develop two queueing network models for a multi-tier architecture, where each tier is modeled as a multi-station queueing center. One of the two models is chosen based on the properties of the real workload. Second, we profile the applications and generate the resource demand of each transaction type at each resource. This is obtained by performing a statistical regression analysis on the historical or benchmark data. The profiling results are stored as the application resource profile in a repository. Finally, we combine the performance model and the application resource profile to formulate a constraint satisfaction problem.

Given a performance goal (e.g., response time), a workload (open or closed in terms of transaction mix) and any other constraints (e.g., CPU utilization < 50%), the solver takes the application resource profile and the analytical model as inputs and generates a low-level policy setting. The output includes the resource requirements, such as how many servers are required at each tier to meet the SLO and the healthy bounds of resource utilization for each component. The resource requirement is then used for the design and reconfiguration of the application accordingly, while the healthy range is used for monitoring the systems during operation. The developed analytical models and application resource profiles are archived for future reuse. If the workload or response times change, we only need to re-solve the constraint satisfaction problem with new parameters to generate a new policy setting.

III. PROFILING

Profiling creates detailed resource profiles of each component in the application. Two key objectives of profiling are to accurately estimate the resource demands for the application, and to identify the input parameters for the performance model. Our regression-based profiling is based on the following observations.

(1) The resource demands of different transaction types are usually different but the resource demand of a transaction type is relatively fixed irrespective of the transaction mix. Hence, it is better to create a profile for each transaction type (e.g., CPU demand for browse transaction, bid transaction, etc.) instead of creating an aggregated profile for the entire workload. The per-transaction type profile remains stable across different transaction mix while the aggregated resource demand of all obtained for a workload only holds for that particular workload with the same transaction mix.

(2) The average resource demand of a request in a workload is determined by the distribution of different transaction types in the workload and the service demand of each transaction type. Hence, once we have per-transaction type profiles, given a new transaction mix, the aggregated resource demand can be derived from per-transaction resource demand.

(3)Few applications are currently instrumented to measure fine-grained transaction resource information. Hence, accurately measuring the service demand of each component requires significant instrumentation of the original application. This is unrealistic in practice. Since the resource demand of each transaction type is relatively static across different transaction mixes, we can derive the parameter of per transaction types using regression-based approaches described below.

During a certain interval, a resource's usage is the sum of all transaction types' demand at that resource, plus a base utilization to account for background activities that are present in real systems (even when the application is completely idle). Hence, a resource's utilization can be obtained as follows

$$U = U_0 + \sum_{i=1}^{N} D_i \bullet \lambda_i \qquad (1)$$

where U is the resource utilization, N denotes the number of transaction types, U_0 represents the background utilization of the resource, D_i represents the resource demand of a request of transaction type i at that resource, and λ_i is average request rate of transaction type i.

In order to obtain U_0 and the demand D_i $(i=1,\ldots N)$ at each resource (e.g., CPU, I/O, network), we first collect utilization data from each resource as well as the arrival rates of different transaction types λ_i $(i=1, \ldots N)$ in different transaction mixes over multiple time intervals (e.g., 5 minutes, 1 hour) either from the historical data or through benchmarking. The latter is used for new applications where no system and application logs are available. We then apply regression analysis to the data on a set of Equations (1) at multiple intervals, to derive the per transaction-type resource demands [4, 13]. We repeat the above steps for each resource and generate the application resource profile. The resulting resource profile for the application is then stored in a repository.

The data required by the regression analysis includes system resource utilization, such as CPU usage, and the application workload information, such as transaction mix. The data is readily available from system and application monitoring logs. This way, our profiling does not require changes to existing applications and systems for instrumentation purposes and hence avoids most of the disadvantages of instrumenting the application. Further, our fine-grained profiles capture the resource demand at per-transaction level and can be used to drive resource demand for different transaction mixes. Hence this approach can be applied to realistic workloads where the percentage of transaction types changes over time.

IV. Performance Model

We utilize a queuing network model of multi-tier applications. An M/M/1 queuing network model is used to evaluate the performance for open workloads, while closed queueing network model is used for closed workloads. Our model is sufficiently general to model any commonly used multi-tier e-commerce application with different application topologies and workloads. The performance model is discussed in detail next.

A. Performance Model for Open Workload

An application with *M* tiers is modeled as a network of M queues Q_1, Q_2, ...Q_M. Each queue represents an individual tier of the application and the underlying server 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 completion (i.e., response to the client). We use V_i to denote the average number of visits to queue Q_i by a request. Given the user request arrival rate λ, the request arrival rate at tier *i* can be approximated as $V_i*\lambda_i$. Given the service demand D_i of a request per visit to tier i, the average service demand per user request at tier i can be approximated as V_i*D_i. Realistic multi-tier applications typically utilize a multi-server/processor architecture to handle a large number of requests. The application server tier for example may involve one or more application servers (e.g., JBoss). A similar notion is applicable to the database tier which may consist of one or more database servers (e.g., MySQL). In order to capture the multi-server/processor architecture, we use a multi-queue center to model each tier. In this model, each server/processor in the tier is represented by a queue. The multi-queue model thus is a general representation of a tier. We use K_i to denote the number of servers at tier *i*. This model shown in Figure 2 represents the multi-server architecture commonly utilized by multi-tier applications.

Consider the following notations.

i: index of transaction type	$(i = 1, \dots N)$
j: index of resource type	$(j = 1, \dots R)$
k: index of tier	$(k = 1, \dots M)$

M: number of tiers (e.g., Web, APP, DB)
N: number of transaction types (e.g., Browse, Bid)
R: number of resources types (e.g., CPU, DISK)
η_k: number of servers at tier k $(k = 1, \dots M)$
$(\lambda_1, \lambda_2 \dots \lambda_N)$: workload where λ_i is the average request rate of transactions type *i*
D_{ik}: service demand of transaction type *i* at a server of tier *k*
U_{kj}: utilization of resource type *j* at tier *k*

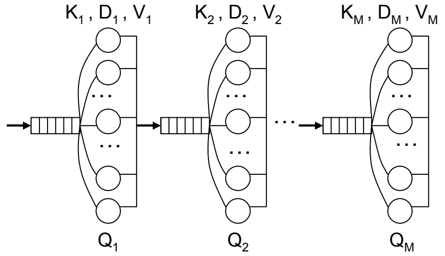

Figure 2. Open Queuing Network

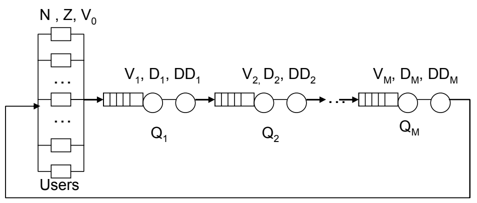

Figure 3. Closed Queueing Network Model

Assume we have a perfect load balancer that evenly distributes the load among all servers of each tier. We model a tier with K servers as K M/M/1 queues. The total service time of a request at tier k is the weighted sum of each transaction type's service time $\sum_{i=1}^{N} \frac{\lambda_i}{\eta_k} \bullet D_{ik}$. The waiting time on a resource type k at tier j is $\sum_{j=1}^{R} \frac{U_{jk}^2}{1-U_{jk}}$. The total residence time of all requests at tier k is the service time plus the waiting time $\sum_{i=1}^{N} \frac{\lambda_i}{\eta_k} \bullet D_{ik} + \sum_{j=1}^{R} \frac{U_{jk}{}^2}{1-U_{jk}}$. The average response time is the sum of the residence times at each tier divided by the overall request rate.

$$RT = \sum_{k=1}^{m} \frac{\sum_{i=1}^{N} \frac{\lambda_i}{\eta_k} \bullet D_{ik} + \sum_{j=1}^{R} \frac{U_{jk}{}^2}{1-U_{jk}}}{\sum_{i=1}^{N} \frac{\lambda_i}{\eta_k}} \quad (2)$$

Given the application profile, the utilization U of each resource can be obtained as follows

$$U = U_0 + \sum_{i=1}^{N} D_i \bullet \lambda_i / \eta \quad (3)$$

The overall resource demand *D* of a transaction type at a server is the sum of all resource demands (e.g., CPU, DISK) at that server. This model is sufficiently general to capture typical multi-tier applications with multiple transaction types and multiple servers at each tier. Given the parameters of the applications, the application resource profile and an open workload *M, N, R,* η_k D_{ik} and $(\lambda_1, \lambda_2 \dots \lambda_N)$ $(k = 1, \dots M, j=1,\dots R)$, Equation (2) is used to predict the response time and Equation (3) is used to derive the resource utilization. Unlike most performance models, our model takes into account the multi-server structure and represents multi-tier applications at a fine-granular level (i.e., per transaction type per resource characterization). As a result, our performance model can be applied to general multi-tier applications with different application topology and open workload with dynamic transaction mix.

B. Performance Model for Closed Workloads

Consider a closed workload with *C* users and think time *Z*. In order to capture the closed workload and the concurrency of multiple users, we use a closed queueing network, where we model *C* concurrent users as *C* delay resources with each of them exhibiting a service demand *Z*. Each tier is modeled as a multi-station queueing center, with the number of stations being the tier's total number of servers and each user is a delay center with service time equaling think time *Z*. We use K_i to denote the number of servers at tier i. Similarly, the service demand at a server of tier i is denoted by D_i.

Given any closed workload in terms of number of users C and the transaction mix percentage $\pi = (p_i, p_2 \ldots p_K)$, the average service demand of the workload D can be computed from the application profile as the weight average of the service demand of each individual transaction $D = \sum_{i=1}^{N} p_i \bullet D_i$.

Given the parameters $\{C, Z, M, K_i, D_i\}$, the proposed closed queueing network model can be solved analytically by using the Mean-Value Analysis (MVA) [5, 14]. MVA algorithm is iterative. It begins from the initial conditions when the system 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 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 proposed by Seidmann et al. [5] to get the approximate solution of performance variables. In this approximation, a queueing center that has K stations and service demand D at each station is replaced by two tandem queues. The first queue being a single-station queue with service demand D/K, and the second queue is a pure delay center, with delay $D \times (K-1)/K$. It has been shown that the error introduced by this approximation is small. By using this approximation, the queueing network model is shown in Figure 3. The MVA algorithm is used to solve our queueing network takes the following set of parameters of a multi-tier application as inputs and computes the average response time R and throughput X of the application.

C: number of users
Z: think time
M: number of tiers
K_i: number of stations at tier i ($i = 1, \ldots, M$)
D_i: service demand of a server at tier i ($i = 1, \ldots, M$)

V. DECOMPOSITION

Given an SLO (e.g., response time) and a workload, the goal of decomposition is to determine the design parameters (e.g., number of servers at each tier) to guarantee that the system has enough capacity for processing the specified workload and meeting the proposed SLO. The output of decomposition contains operational policy settings such as

- How many servers are required for each tier?
- What's the CPU, Memory, IO utilization of each server?

As we discussed before, we generate the profile based on the historical data or benchmarking data with varying

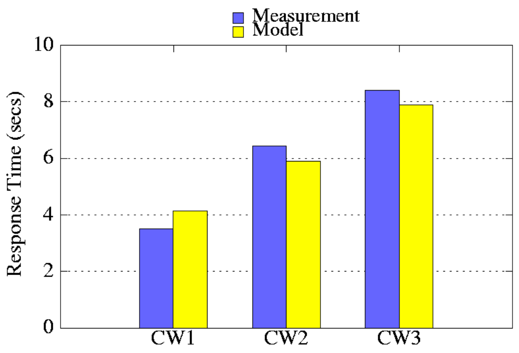

Figure 4. Performance Prediction for Closed Workloads

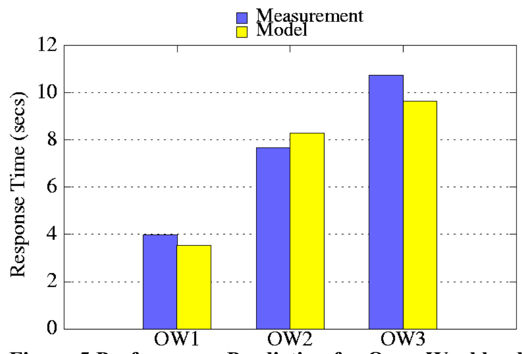

Figure 5.Performance Prediction for Open Workloads

workloads. The service demand of each individual transaction type is retrieved from the archive, as shown in Figure 1. Given any workload and a response time requirement, the task of decomposition is then to find the set of model input parameters such as number of servers that satisfy the response time requirement and further derive the resource utilization. Decomposition thus becomes a constraint satisfaction problem. We have developed a simple constraint satisfaction solver to solve this problem. The solver takes the performance goal, workload, resource profiles and performance model as inputs and constructs a set of constraint equations. Various constraint satisfaction algorithms, such as linear programming and optimization techniques, are available to solve such problems [15]. Typically, the solution is non-deterministic 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 the number of servers at each tier, we can efficiently enumerate the entire solution space to find a solution. Also, we are often interested in finding a single feasible solution (rather than the optimal solution), so we can stop the search once one is found. Other heuristic techniques can also be used during the search. For example, the hint that the response time typically decreases with respect to the increase of allocated resources can also reduce the search space.

One advantage of our approach is that once the profile and model are created, they can be repeatedly used to perform decomposition for different SLOs and workloads. That is, if the response time or workload changes, we only need to resolve the constraint satisfaction problem with the new parameters. Similarly, if the application is deployed to a new environment, we only need to regenerate the profile in that environment using regression analysis.

Table 2. Decomposition Results for Closed Workloads

Input	Output			Measurement	
Workloads and SLOs	**Num. of App. Servers**	**Resp. Time**	**CPU Utili.**	**Resp. Time**	**CPU Utili.**
User=100 Browse Intensive Response time<5 sec	1	3.49 s	21.8%	3.03 s	24.5%
User=100 Bidding Intensive Response time< 5 sec	1	4.03 s	35.6%	4.36 s	33.2%
User=200 Browse Intensive Response time < 5 sec.	1	4.77s	43.2%	4.67 s	47.8%
User=200 Bidding Intensive Response time< 5 sec.	2	4.24 s	37.4%	4.43 s	32.9%

Table 3. Decomposition Results for Open Workloads

Input	Output			Measurement	
Workloads and SLOs	**Num. of App. Servers**	**Resp. Time**	**CPU Utili.**	**Resp. Time**	**CPU Utili.**
30 reqs/s Browse Intensive Response time<5 sec	1	3.88 s	23.4%	3.67 s	25.1%
30 reqs/s Bidding Intensive Response time< 5 sec	1	4.53 s	37.3%	4.75 s	42.0%
40 reqs/s Browse Intensive Response time < 5 sec.	1	4.47s	40.1%	4.81 s	44.5%
40 reqs/s Bidding Intensive Response time< 5 sec.	2	3.94 s	32.3%	4.33 s	36.7%

VI. EXPERIMENT EVALUATION

We evaluated our approach with two applications, the popular RUBiS e-commerce application with synthetic workloads and a real business-critical service with real traces.

A. RUBiS Testbed

RUBiS is an eBay-like online auction site developed at Rice University [1]. We use a 3-tier EJB-based implementation of RUBiS consisting of an Apache Web server 2.0, a JBOSS 4.0.2 application server, and a MySQL 5.0 database server, each running on different servers. The RUBiS implementation defines 26 interactions, has 1,000,000 users and 60,000 items. The testbed includes multiple Linux servers. Each server has 2.4 GHz CPU, 4 GB of RAM, and a 1Gb/s Ethernet interface. We developed a workload generator that can produce both open and closed workloads. For open workloads, the workload generator sends requests according to a specified request rate and transaction mix. For closed workloads, the workload follows a given transition matrix to simulate multiple concurrent users interactions with RUBiS. The workload generator runs on a separate server node from any of the RUBiS systems.

B. Performance Prediction

To validate the correctness and accuracy of our model, we compare the response times predicted by our model and actual measurements with different workloads under different configurations.

We use the workload generator to produce variable workloads with fluctuations in request rate and transaction mix. Application data is obtained from the Apache and JBoss log. System utilization is collected every one minute using the SAR monitor. The data set records two kinds of data about RUBiS, application-level data such as transaction request rate of each transaction type, and system level resource utilization (e.g., CPU utilization). We then apply the regression analysis described in Section 3 to generate the application's resource profile. Given any open or closed workload, we use the resource demand information obtained during profiling as model input parameters, and apply the performance model described in section 4 to derive the response time.

In the experiment, we evaluate the effectiveness of our approach for different mixes of browse and bid transactions. First, we define three typical closed workloads of 200 users with different transaction mixes: CW1: browse dominant, CW2: balanced and CW3: bid dominant. For each workload, we use the profile and model to predict the average response time, and then compare the results with the actual performance. The results are depicted in Figure 4. The results show that our model can accurately predict the performance for different closed workloads with different transaction mixes. Similarly, we define three typical open workloads with different transaction mixes: OW1, OW2 and OW3 and compare the accuracy of response time predicted by our model for each workloads. The results depicted in Figure 5 indicate that our model can also work well with open workloads. These results clearly demonstrate that our model can use the same profiling results (i.e., model input parameters) obtained during profiling to predict the performance of any unforeseen transaction mixes.

We also conducted a similar evaluation of the RUBiS configuration with 2 JBOSS servers. We obtained similar results, and thus do not include the figures here.

C. Decomposition Effectiveness

In this section, we evaluate the effectiveness of our approach. Given any workload and SLOs, our decomposition module constructs a set of constraints and then solves the corresponding constraint satisfaction problem. The output of decomposition contains the number of servers required at each tier to meet the response time requirements, as well as the resource utilization of the configuration. In the following experiments, given any SLO in terms of workload and a response time requirement, we generate the number of application servers needed, and predict the average CPU utilization. We then configure RUBiS based on these derived settings. We validate our design by applying the workload, and measuring the actual performance of RUBiS and comparing the results with the SLO. We also compare the predicted CPU utilization with the actual CPU utilization. A guideline regarding resource utilization is to keep peak utilizations of resources, such as CPU, below 70% [14]. In practice, enterprise system operators are typically even more cautious than this conservative guideline. Hence, we also put additional constraints of CPU utilization to be less than 60% in our evaluation.

In these experiments, we first consider the high level SLOs in terms of the number of concurrent users, the transaction mix and the average response time. Table 2 summarizes the input and output of decomposition for four different SLOs. The first column shows the input to our decomposition and the second column describes the output of decomposition such as the system design parameter (i.e., number of JBOSS servers) and the healthy range of CPU utilization under the proposed

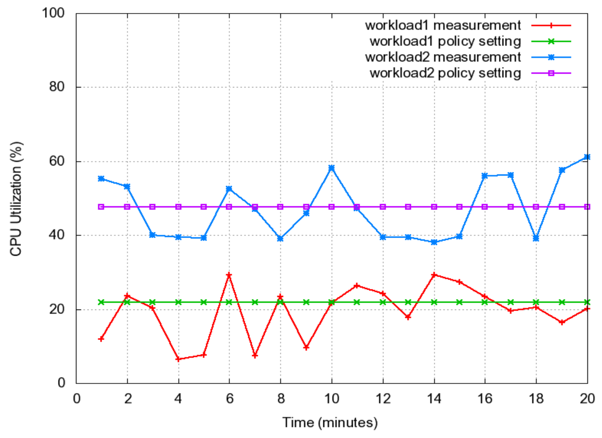

Figure 6. CPU Utilization

configuration. The measurement column shows the actual measurement of response time and the CPU utilization of the system with the design.

As shown in the first row, for the SLO of 100 users with browse-intensive transaction mix and response time < 5 seconds, decomposition determines that only one server is required to ensure the SLO and the response time and CPU utilization are 3.49 seconds and 21.8% respectively. The actual measurements of response time and CPU utilization are 3.03 seconds and 24.5%. This shows that the design can meet SLOs and the utilization prediction is close to real system measurement. The second SLO has 100 users but with a different transaction mix (i.e., bidding intensive). We can see from this experiment that the decomposition results are close to the actual measurements. The third input involves 200 concurrent users and browsing intensive transactions, the decomposition result shows that only one server is needed to meet the SLO. The fourth input has 200 users with bidding intensive workload, which is more resource demanding. The decomposition module determines that 2 servers are required to handle the workload and meet the response time requirement. The actual performance shows that the design can meet the requirement and the prediction of response time and CPU utilization are relatively accurate. From the above results, we can see that our decomposition approach can be effectively applied to design and monitor such multi-tier applications with different SLOs.

In order to further check the applicability of our approach, we also apply the decomposition to SLOs involving open workloads. We experimented with four different SLOs. In the experiment, the workload is specified in terms of request rate and transaction mix. These results are summarized in Table 3. The results show that our approach can also work well with open workloads.

E. Production Application

We also evaluate the ability of our decomposition approach to generate low-level resource utilization policies for a real business-critical enterprise application. This service consists of roughly 20 servers and processes tens of millions of application-level transactions per day. The service is CPU and network intensive and its performance is crucial to many other services. In the evaluation, we run the service with a 24 hour request trace from one of the production servers. As described

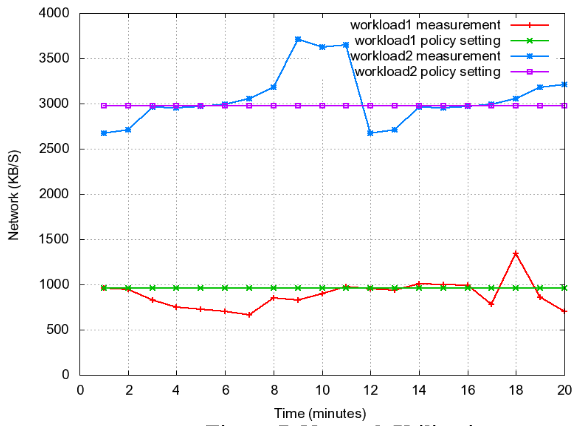

Figure 7. Network Utilization

in Section 3, the profile captures the CPU and network demand for each transaction type. We then extract two typical workloads with different transaction mixes: a lightweight one and a heavyweight one. Given these two workloads, we apply the decomposition approach to generate the CPU and network bounds and further create monitoring policies based on the derived CPU and network bounds. The monitoring policies are according to the utilization predicted by the decomposition model. We apply the workloads and measure the actual CPU and network utilization. The actual CPU and network resource utilization and the monitoring policies are shown in Figure 6 and Figure 7 respectively. As shown in these figures, the monitoring policies accurately capture the healthy range of the application for different workloads. These policies can be continuously used to assess how the system is performing and evaluate whether it will violate any of the goals it was designed for. For example, for the first workload, it should warrant CPU Utilization to be around 20%. This metric has to be monitored to make sure that the higher level SLA is met. For example, appropriate actions can be taken when the threshold is violated. This can be defined in an operational policy. In addition, such monitoring policies will also provide a mechanism using which we could predict or even avoid future SLA violations by provisioning the system accordingly in design or capacity planning phase.

VII. RELATED WORK

Our previous work proposes an SLA decomposition approach based on profiling and a queueing network model [12]. Although it shares some common features with the approach presented in this paper, the basic assumption and modeling techniques are quite different. Our early work focused on managing resource assignment of virtual machines, and the profiling and modeling were relatively simple. The approach presented in this paper aims to develop a practical and advanced model that can be applied to real multi-tier applications with complex, dynamic, non-stationary workloads. Compared to our earlier work, the novel contributions and significant enhancements can be summarized as follows. First, the new performance model is much more advanced and it models multi-tier applications in a much finer-grain manner. Through explicitly modeling per-transaction resource demand, the new approach can handle any unforeseen workloads with different transaction mixes.

Second, our new approach is non-intrusive. The profiling in our early work requires significant efforts to instrument the system and conduct controlled benchmarking in order to collect monitoring data, while our new approach can perform profiling from readily available monitoring data. Third, by defining the workload as transaction mix and introducing open queueing and closed queuing models, our approach can handle both open and closed workloads in a consistent manner. Fourth, our model can directly derive the healthy range of low level system metrics and further develop monitoring policies from the profiles. We also increased the number of system metrics considered. Finally, we evaluate our approach with a real production application. It has been shown that the approach works well for realistic workload under normal system load.

Stewart et al. present an approach to predicting performance as a function of workload [13]. Their model explicitly models non-stationary transaction mix and shares some features with our open workload model. The main difference is that they model the aggregated service time across all tiers, while our model associates service times at a per-tier level. Their model works well for the purpose of performance prediction, but for the purpose of decomposition, it's necessary to model the application in a finer-grained manner. Another improvement from our open workload model is that we generalize M/M/1 queue to K M/M/1 queues and hence can handle general multi-server configuration in typical multi-tier applications. This extension is required by the decomposition. In addition, our model explicitly models the visit rate at each tier, and hence can handle non-uniform requests distribution across tiers. The regression-based profiling presented by Zhang et al. [4] is similar to our profiling, but we model multi-tier applications at a much finer-granularity. Schroeder et al. considered open and closed workloads as part of a separate study [3], but their focus is on the workloads themselves.

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 [9, 10, 11]. A few recent efforts have extended single-tier models to multi-tier applications [6, 16, 17]. The most recent and accurate performance model for multi-tier applications is proposed by Urgaonkar et al. [17]. Similar to our closed workload model, their model uses a closed queueing network model and Mean Value Analysis (MVA) algorithm for predicating performance of multi-tier applications. The main differences are that we explicitly model the service demand or resource demand per transaction type. and use a multi-station queue while their model assumes a stationary workland and single-station queue. Their model takes into account congestion effects, but we have not addressed that yet partly because it is undesirable for a production application to operate under high system load.

VIII. CONCLUSION AND FUTURE WORK

One of the most important tasks towards SLA management is to automate the process of designing and monitoring systems for meeting higher level business goals. It is an intriguing but difficult task due to the complexity and ynamism inherent in today's multi-tier applications. In this paper, we propose a systematic and practical 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 metrics and application level attributes which are used for creating, designing, and monitoring the application to meet high level SLAs. We believe that our approach has several desirable advantages over prior solutions. Our approach can deal with dynamically changing workloads in terms of change in both request volume and transaction mix. It is non-intrusive in the sense that it requires no instrumentation and the data used in our approach is readily available from standard system and application monitoring. It can also process both request-based and session-based workloads. In the future, we are also interested in investigating the dynamic selection of an appropriate workload and performance model for real semi-open workloads.

REFERENCE

1. Rice University Bidding System, http://www.cs.rice.edu/CS/Systems/DynaServer/rubis.
2. S. Agarwala, et al., "QMON: QoS- and Utility-Aware Monitoring in Enterprise Systems". In Proc. of the 3rd ICAC (ICAC 2006), 2006.
3. B. Schroeder, et al., "Open Versus Closed: A Cautionary Tale". Proc. of the 3rd NSDI (NSDI 2006), 2006.
4. Q. Zhang, et al., "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications". In Proc. of the 4th ICAC (ICAC 2007), 2007.
5. M. Reiser and S. S. Lavenberg, "Mean-Value Analysis of Closed Multichain Queueing Networks". J. ACM, vol. 27, pp 313-322, 1980.
6. X. Liu, et al., "Modeling 3-Tiered Web Applications". In Proc. of 13th IEEE MASCOTS, Atlanta, Georgia, 2005.
7. D. Menasce, et al., "Capacity Planning for Web Services: Metrics, Models, and Methods". Prentice Hall PTR, 2001.
8. W. Gong Chandra, et al., "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements". In Proc. of International Workshop on Quality of Service, June 2003.
9. R. Doyle, et al., "Model-Based Resource Provisioning in a Web Service Utility". In Proc. of the 4th USENIX USITS, Mar. 2003.
10. R. Levy, et al., "Performance Management for Cluster Based Web Services". in Proc. of IFIP/IEEE 8th IM, 2003.
11. L. Slothouber, "A Model of Web Server Performance". In Proc. of Int'l World Wide Web Conference, 1996.
12. Y. Chen, et al., SLA Decomposition: Translating Service Level Objectives (SLOs) to Low-level System Thresholds. In Proc. of the 4th ICAC (ICAC 2007), June 2007.
13. C. Stewart, et al., "Exploiting Nonstationarity for Performance Prediction". In Proc. of EuroSys 2007.
14. E. D. Lazowska, et al., "Quantitative System Performance: Computer System Analysis Using Queueing Network Models". Prentice-Hall, Inc., 1984.
15. A. Zhang, et al., "Optimal Server Resource Allocation Using an Open Queueing Network Model of Response Time". HP Labs Technical Report, HPL-2002-301.
16. B. Urgaonkar,et al., "Dynamic Provisioning of Multi-tier Internet Applications". In Proc. of IEEE ICAC, June 2005.
17. B. Urgaonkar, et al., "An Analytical Model for Multi-tier Internet Services and its Applications". In Proc. of ACM SIGMETRICS, June 2005.