RC23939 (W0604-117) April 24, 2006
Computer Science

IBM Research Report

Modeling Differentiated Services of Multi-Tier Web Applications

Yixin Diao, Joseph Hellerstein, Sujay Parekh, Hidayatullah Shaikh, Maheswaran Surendra, Asser Tantawi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Modeling Differentiated Services of Multi-Tier Web Applications

Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Hidayatullah Shaikh,
Maheswaran Surendra, Asser Tantawi

IBM Thomas J. Watson Research Center
Hawthorne, New York 10532
{diao, hellers, sujay, hshaikh, suren, tantawi}@us.ibm.com

Abstract

In this paper we present a hybrid performance model for modeling differentiated service of multi-tier web applications with per-tier concurrency limits, cross-tier interactions, as well as a work-conserving resource allocation model. The service dependencies between multiple tiers are captured first using a layered queueing model. We then show how to model per-tier concurrency limits and service differentiation between multiple classes while maintaining work conservation at each tier. We use a function approximation approach combined with a coupled processor model. Our model is calibrated from an actual multi-tier J2EE testbed, and we show the ability of the model to accurately model common performance metrics. Our proposed (layered) model shows 78% improvement in root mean square error over a single-tier machine repair model as well as a tandem queue model. We also demonstrate one application of the model for model-based resource allocation.

1 Introduction

Many important commercial websites are designed using a component-based approach which divides the website functionality according to a multi-tiered architecture. This architecture, embodied by Sun's J2EE and Microsoft's .NET specifications, partitions the processing of web requests into stages or tiers concerning the presentation logic (HTTP servers), business logic (application servers), and persistent storage (database servers). Although this simplifies the job of the application developer, the interdependencies between the tiers makes it more difficult to manage the overall website performance.

Modeling such sites is challenging because of:

- Cross-tier dependencies: inter-tier interaction is generally synchronous; the threads or processes in upstream tiers are blocked while waiting for the completion of processing in downstream tiers. Hence, the service rate of upstream tiers is intimately tied to the performance of the downstream tiers.
- Concurrency limits per tier. Administrators typically limit the number of threads or processes created in a tier. This is due to memory limitations as well as to avoid thrashing. When combined with the synchronous behavior discussed earlier, it results in situations where a tier may be unutilized, yet requests are queued up because all the allocated processes/threads are busy waiting on downstream tiers.
- Supporting multiple classes. This challenge is further exacerbated when it is important to provide differentiated Quality of Service (QoS) to the users. For the purposes of specifying QoS requirements, it is common to group the users into different *classes* and then specify service level objectives such as desired response time or throughput for each class of users.
- Work conserving behavior. At each tier, multiple classes share the resources available at the tier. The work conserving property refers to the fact that even when per-class resource shares are being enforced, resources unused by one class are used up by other classes when there is demand. This behavior improves the tier utilization, but can be difficult to model since the amount of resource share devoted to a particular class depends on the utilization by the other classes.

Existing literature on modeling Internet services can be grouped as follows.

- **Single-tier**. Much of the research on modeling single-tier applications [1, 2, 3, 4]. do not trivially extend to multiple tiers.

- **Multi-tier, single class**. In the context of multi-tier systems, [5] describe a predictive queueing model that considers resource contention within each server and service dependencies between different tiers. However, they do not consider multiple service classes.

- **Multi-tier, multi-class**. [6] present a closed queueing network model that considers session-based workloads, and can be extended for multiple service classes. However, this paper applies a tandem queue like structure without considering the service dependencies between different tiers; it also does not handle multiple classes when concurrency limits exist at each tier.

To the best of our knowledge, none of the existing literature has modeled service differentiation in a multi-tier system that considers per-tier concurrency limits, cross-tier interactions as well as modeling a work-conserving resource allocation model.

In this paper, we develop a practical model for such multi-tiered sites supporting differentiated services to multiple classes of users. This model can be used for multiple performance-oriented tasks such as QoS prediction, resource control and capacity planning. Our model is a hybrid model that combines a layered queuing approaches with some key approximation-based simplifications for making the model tractable for modeling multi-class service differentiation. Our proposed (layered) model shows 78% improvement in root mean square error over a single-tier machine repair model as well as a multi-class tandem queue model that does not capture inter-tier dependencies. Moreover, we show that our model can accurately model the response time, throughput and utilization levels at each tier – the other models can only capture a subset of these metrics.

The remainder of the paper is organized as follows. Section 2 describes the service differentiation problem for multi-tier systems in more detail. Section 3 describes the multi-tier performance model for single service class and Section 4 extends the model to service differentiation for multiple classes. In Section 5, we assess the model validity against a real multi-tier system and in Section 6, we illustrate one use of the performance model for model-based resource allocation. The conclusions are contained in Section 7.

2 Problem

In a multi-tier system, servicing a request involves several stages. Examples of such stages are accessing static content, server processing, access to enterprise applications, and access to data base systems. Each stage of service is provided by a set of hardware and software resources that constitute a tier. Thus, in a J2EE-based web serving system, we may have the following tiers: Web servers (HTTP), Servlets, Enterprise Java Beans (EJB), and Data Base (DB) servers. As a request flows through the various tiers, it uses physical resources, e.g. CPU and disk, as well as virtual resources, e.g. threads, DB connections and locks. Further, a request may visit a tier multiple times and move forward (downstream) and backward (upstream) several times until its service is complete. Typically, threads at a given tier are held by requests visiting that tier until the request receives its intended service from downstream tiers and returns back to that tier. This phenomenon is referred to as cross-tier dependency [7]. Hence, other requests may have to wait at a given tier until threads are released and become available. Furthermore, requests running in threads may still have to wait for other requests to complete their usage of a given resource in the tier. This delay is referred to as resource contention.

The flow of requests among the various tiers gives rise to a layered structure, where the first (last) tier corresponds to the outer (inner) layer. Due to resource contention and cross-tier dependency, a server (and its server threads) can be in one of the following states: 1) using a resource, e.g. CPU or disk, of this tier, 2) waiting for a resource of this tier to become available, 3) waiting for a response from a lower level tier, or 4) idle after being released by a request upon completion of service at that tier. In a multi-tier system a service request can only be completed after it receives services from all related tiers; this means that the service rate of any tier depends not only on its own tier service capacity (and resource contention) but also on the service rate of downstream tiers. Therefore, one needs a model that captures resource contention within a tier as well as cross-tier dependency. A simple tandem queue model does not capture such dependencies, and as we show later, it therefore fails to accurately model resource utilizations in such environments. On the other hand, a model that captures simultaneous resource possession at all tiers becomes complicated rather quickly, hence inefficient to solve.

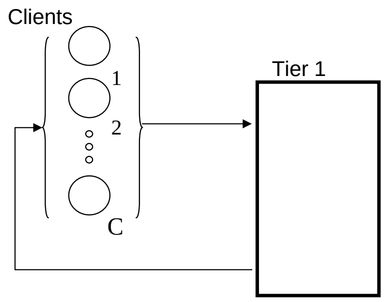
In formulating our solution, we make the following assumptions.

1. There exists a tier with the largest resource usage that constitutes the bottleneck for the system.

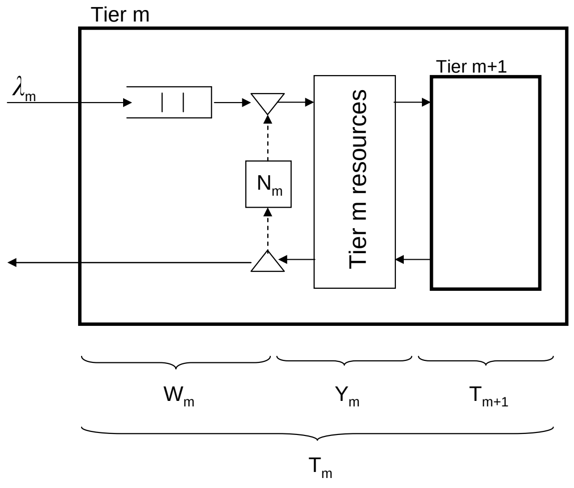
2. The total concurrency level at a given tier does not change, once the model is trained. Our model does not take the total concurrency level as a parameter, but rather it is subsumed in the functional approximation.

3. We assume that there exist monitoring agents that provide statistics on resource usage and flow rates at the various tiers.

For convenience of discussion, we use "bottleneck resource" to indicate the resource with largest usage for a given tier, and "bottleneck tier" for the tier with the largest



(a) System model



(b) Tier model

Figure 1. Layered queueing model for multi-tier web applications (single service class).

resource usage for a given system. Since different service classes may stress each tier differently, the above bottleneck tier is defined by considering the aggregated effect of all the classes. Note that we define bottleneck tier for convenience of discussion, but the system is not required to have just a single bottleneck tier.

3 Modeling Multi-tier Web Applications

This section describes the method of modeling a multi-tier system with single class of requests. This approach integrates a layered queueing model to capture the cross-tier dependencies and a function approximation method to model the per-tier concurrency limits and resource contention. Although the function approximation model is not necessary in modeling single class, it enables modeling service differentiation of multiple service classes, which is presented in Section 4.

3.1 Layered Model for Cross-Tier Dependency

Consider a single class of requests submitted by a finite number of clients to a multi-tier system. Figure 1 shows the layered queueing model for multi-tier web applications.

The Internet workload is modeled as concurrent sessions through a closed queueing network consisting of two service centers [8]. As shown in the left part of Figure 1(a), service center 1 has as many servers, C, as the number of clients (and corresponding sessions). The service time, Z, at each of the C servers corresponds to the think time elapsed between receiving a response and submitting a subsequent request. Service center 2, marked as a block in the right part of Figure 1(a), represents the M-tier system as a whole.

Since tiers have a nested structure, the model for tier $m, m = 1, 2, ..., M-1$, includes the model for the downstream tier $m+1$, as depicted in Figure 1(b). This layered queueing model captures the cross-tier dependency [7] as discussed below. Note that the flow of a request through the various tiers is restricted in the following way. We assume that a request at a given tier may visit its adjacent downstream tier, and hence recursively may visit any tier in the downstream. Further, a request at tier m holds a virtual resource (a thread) at tier m as long as it is either receiving service at tier m or any downstream tier, $m+1, m+2, ..., M$. The model for tier m consists of a pool of N_m tokens, a queue for acquiring a token if none is available, the resources of tier m, and the model for tier $m+1$ if $m < M$. The pool of N_m tokens is used to represent the concurrency limit such as the number of concurrent threads. The value of N_m is not required to be known to the model though. Actually, in certain tiers such as the database server tier, it is not obvious to get N_m because it can be composed of multiple interrelated limits (e.g., the number of database agents and the lock lists).

We denote the arrival rate of requests to tier m by λ_m, the total time spent in the tier m model by T_m, the waiting time for a token by W_m, the residence time while using or waiting for resources at tier m by Y_m, and the total time spent at tier $m+1$ model by $T_{m=1}$. Thus, the tier response time is written as

$$T_m = W_m + Y_m + V_{m+1}T_{m+1} \qquad (1)$$

where

$$V_m = \frac{\lambda_m}{\lambda_{m-1}}, \qquad m > 1, \qquad (2)$$

is the visit ratio. This is to consider the fact that requests processed at one tier may generate more (or less) requests for a lower level tier. Note that when a service request is accepted into Tier m, according to the concurrency limit

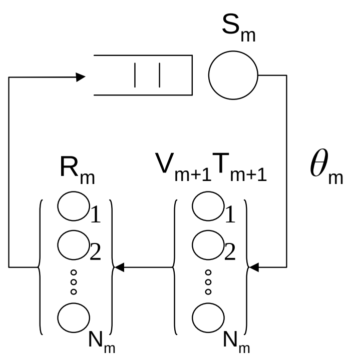


Figure 2. Per-tier queuing network model.

N_m, a process, thread, or agent is allocated to execute this service request on behalf of this client. Due to the layered queueing structure, this process cannot service other requests, even when this request completes its service at tier m and is being serviced by lower tiers ($m+1$, $m+2$, ...). Therefore, the request waiting time W_m not only depends on the residence time Y_m, but also the response time of lower tier T_{m+1}. Due to the dependency of T_m on T_{m+1}, one needs to solve the layered queueing model iteratively as in [7].

3.2 Function Approximation for Per-Tier Queueing

As shown in Figure 2, a closed queueing network model is used to model per-tier queueing. We represent all resources at tier m as two entities: (1) a single server queue of the bottleneck resource and (2) an infinite server queue of a combined resource of all other resources. The assumption is that most of the waiting will be for the bottleneck resource. The bottleneck resource has a service time of S_m; the combined non-bottleneck resource has a service time of R_m. The lower tier response time is also modeled as an infinite server with service time of $V_{m+1}T_{m+1}$. (Even if the lower tier response time may be significant, its computation is considered in its corresponding tier model.) We can solve this closed queueing network model using mean value analysis (MVA) [9]. However, regardless of the computational complexity, although the MVA approach can be extended to multiple classes, we have not seen any work on addressing service differentiation with MVA especially when concurrency limits exist.

In this section we propose a function approximation method that builds an equivalent single server model for each tier. The per-tier service rate is modeled as a function of load, specifically, the number of clients, and the parameters of this function approximator are calibrated using actual performance measurements. Having an equivalent single server model helps to use the coupled processor model

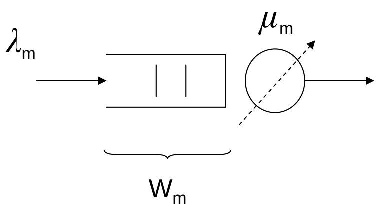


Figure 3. Equivalent per-tier model.

approach to further model service differentiation between multiple service classes, which is presented in Section 4.

As shown in Figure 3, we logically set up an equivalent single server whose service rate is

$$\mu_m = \frac{1}{\frac{1}{X_m} + V_{m+1}T_{m+1}}, \tag{3}$$

Note that μ_m depends on both the service rate at tier m (X_m) and the total response time at tier $m+1$.

Furthermore, since X_m is load dependent, we define it as

$$X_m = \frac{f_m}{S_m} \tag{4}$$

which is invert proportional to the service time of bottleneck resource, and uses a concurrency adjustment function f_m to capture the load dependence property. Specifically, we approximate f_m by an exponential function

$$f_m = a_m \left(1 - e^{-b_m C}\right) \tag{5}$$

where C is the number of clients (i.e., describing the load) and the model parameters a_m and b_m are estimated through curve fitting techniques to match the actual performance measurements. An illustration of f_m is provided in Figure 4. It emulates the effect that the higher the workload, the better utilization of the multiple concurrent threads until all threads are fully utilized.

Given the equivalent service rate μ_m, the tier response time can be computed as

$$T_m = \begin{array}{ll} h_{M/M/1}\left(\lambda_m, \mu_m\right), & m > 1 \\ h_{M/M/1/\infty/C}\left(C, Z, \mu_1\right), & m = 1, \end{array} \tag{6}$$

where $h_{M/M/1}()$ represents the response time of an $M/M/1$ queue and $h_{M/M/1/\infty/C}()$ represents the response time of an closed queue (i.e., machine repair mode).

3.3 Algorithms

In this section we present algorithms to solve the layered queueing model and the function approximation model. We consider two phases: model calibration and model-based prediction (i.e., extrapolate the model for unseen workload).

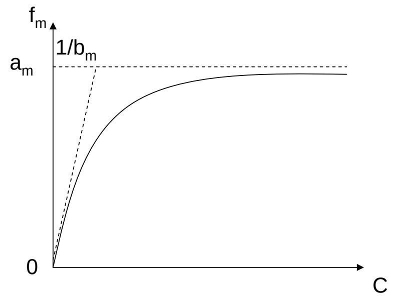


Figure 4. Exponential approximation.

During the model calibration phase, we use measured data to calculate model parameters. Then, we use the calibrated model to predict performance when the load (number of clients) changes.

- Model calibration phase
  - Input and output data: The set $\Im$ represents measurements at epoch $i, i = 1, \ldots, N, N \geq 2$, for tier $m, m = 1, \ldots, M$ that are input to the model calibration algorithms. We add the superscript $*$ to denote a measured quantity, as opposed to a calculated one. The variable ρ_m represents the utilization of the bottleneck resource at tier m. The set $\aleph$ represents the internal states of the algorithm, and the set $\wp$ indicates the output of the calibration. Specifically, we define the following sets.

$$\begin{aligned}\Im &= \{C^*(i), T_1^*(i), \lambda_m^*(i), \rho_m^*(i)\} \\ \aleph &= \{Z(i), S_m(i), V_m(i), T_m(i), \lambda_m(i), X_m(i), \\ &\quad \mu_m(i), f_m(i)\} \\ \wp &= \{Z, S_m, V_m, a_m, b_m\}\end{aligned}$$

  - Algorithms: There are three algorithms used during the calibration phase: CA_0, CA_1, and CA_2. We use the notation, $A : \Im \Rightarrow \aleph \Rightarrow \wp$, to represent algorithm A that uses the input set $\Im$ to produce the internal state set $\aleph$ and the output set $\wp$.
    - CA_0: $\{C^*(i), T_1^*(i), \lambda_m^*(i), \rho_m^*(i)\} \Rightarrow \{Z(i), S_m(i), V_m(i)\} \Rightarrow \{Z, S_m, V_m\}$
    - CA_1: $\{C^*(i), T_1^*(i), \lambda_m^*(i)\} + \{Z, S_m, V_m\} \Rightarrow \{T_m(i), \lambda_m(i), X_m(i), \mu_m(i)\} \Rightarrow \{f(i)\}$
    - CA_2: $\{C^*(i)\} + \{f_m(i)\} \Rightarrow \{a_m, b_m\}$
- Model-based prediction (extrapolation) phase
  - Algorithm EA_0:
    - $C + \wp \Rightarrow \{T_m, \lambda_m, \rho_m\}$

3.3.1 Calibration Algorithm 0 (CA_0)

The think time Z, service time at the bottleneck resource S_m, and visit ratios V_m are directly computed from the measured data using Little's Law and flow conservation, respectively.

$$Z(i) = \frac{C^*(i)}{\lambda_1^*(i)} - T_1^*(i) \quad (7)$$

$$S_m(i) = \frac{\rho_m^*(i)}{\lambda_m^*(i)} \quad (8)$$

$$V_m(i) = \frac{\lambda_m^*(i)}{\lambda_{m-1}^*(i)} \quad (9)$$

$$Z = \frac{1}{N}\sum_{i=1}^{N} Z(i) \quad (10)$$

$$S_m = \frac{1}{N}\sum_{i=1}^{N} S_m(i) \quad (11)$$

$$V_m = \frac{1}{N}\sum_{i=1}^{N} V_m(i) \quad (12)$$

3.3.2 Calibration Algorithm 1 (CA_1)

The purpose of algorithm CA_1 is to compute the value of $f_m(i)$ so that the response time of the calibrated model matches the measured response time, i.e. $T_1(i) = T_1^*(i)$. Using Little's law, this also implies $\lambda_1(i) = \lambda_1^*(i)$. This algorithm will run N times for each $i, i = 1, \ldots, N$. Without loss of generality and for simplicity of presentation, we omit the measurement epoch (i) in this subsection.

- Step 1: Given $\{\rho_m^*\}$, determine the bottleneck tier denoted by m'. This is the one with the largest utilization.
- Step 2: Initialization. Iteration variable $j = 0$ and the concurrency adjustment factor of the bottleneck tier $\Delta^{(0)} = 1$.
- Step 3: Determine the value of f_m. That is, $f_m = 1$ if $m \neq m'$; $f_m = \Delta^{(j)}$ if $m = m'$.
- Step 4: Compute T_1 using the layered queueing model.
  - Step 4.1: Initialization. Iteration variable $k = 0$ and the initial predicted throughput $\lambda_1^{(0)} = \lambda_1^*$.
  - Step 4.2: Determine $\lambda_m^{(k)}, m = 2, \ldots, M$ based on $\lambda_1^{(k)}$ and $V_m, m = 2, \ldots, M$.
  - Step 4.3:

$$\lambda_1^{(k+1)} = SolveModel(\lambda_m^{(k)}, f_m, C, Z, S_m, V_m) \quad (13)$$

– Step 4.4: If $\lambda_1^{(k+1)} = \lambda_1^{(k)}$, go to Step 5; otherwise, $k = k + 1$ and

$$\lambda_1^{(k)} = (1-\alpha)\lambda_1^{(k-1)} + \alpha\lambda_1^{(k)} \qquad (14)$$

and go to Step 4.2.

- Step 5: If $T_1 = T_1^*$, terminate the algorithm with $f_m = \Delta^{(j)}$; otherwise, $j = j + 1$, find $\Delta^{(j)}$, for example, through line search, and go to Step 3.

The details of Step 4.3 are as follows.

1. Initialization. $m = M$.
2. From Equation (4), compute $X_m^{(k)}$ based on f_m, S_m.
3. From Equation (3), compute $\mu_m^{(k)}$ based on $X_m^{(k)}, T_{m+1}^{(k)}, V_{m+1}$. Note that $T_{M+1}^{(k)} = 0$.
4. From Equation (6), compute $T_m^{(k)}$ based on $\lambda_m^{(k)}$ and $\mu_m^{(k)}$ if $m > 1$; otherwise, compute $T_1^{(k)}$ based on C, Z and $\mu_1^{(k)}$.
5. If $m = 1$, terminate with $\lambda_1^{(k+1)} = \frac{C}{Z+T_1^{(k)}}$; otherwise, $m = m - 1$ and go to 2.

Note that Sep 4.4 introduces a digital filter design to improve the convergence. Also note that when the effect of non-bottleneck tier is not negligible (i.e., their utilization is still large), the above algorithm can be iterated to go through different tiers with the descending order of the tier utilization.

3.3.3 Calibration Algorithm 2 (CA_2)

Given the values of f_m, obtained from CA_1 and the number of clients C^*, we obtain the parameters a_m and b_m by solving Equation (5) using a nonlinar regression analysis algorithm [10].

3.3.4 Extrapolation Algorithm 0 (EA_0)

Once the model is calibrated, we can use it to predict performance due to changes in the number of clients C, the bottleneck resource service times S_m, or the visit ratios V_m. Extrapolation algorithm EA_0 obtains T_m given in Equation (6), using the analysis provided in Section 3.2.

4 Handling Service Differentiation

In this section we extend multi-tier modeling to include multiple service classes and introduce QoS actuators for service differentiation. Figure 5 illustrates the architecture

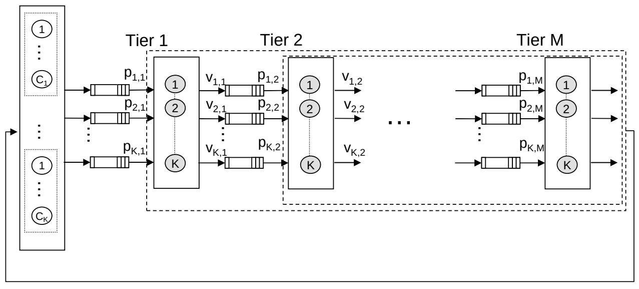


Figure 5. Architecture of a closed queueing model for multi-tier web applications with multiple service classes.

of the closed queueing network model for M tiers of servers with K service classes.

As in Section 3, the behaviors of workload and concurrent sessions are modeled with a machine repair model and the cross-tier dependency is modeled through the layered queueing model. The service center 1, the solid box on the left, contains the served concurrent clients (and their corresponding servers). Grouped by dotted boxes, they are marked by $1, 2, \ldots, C_k$ for service class k and have think time of Z_k. The service center 2, as indicated by the outer dashed box on the right, includes multi-tier servers to service client requests. The QoS control actuators are represented by $p_{k,m}$ for class k on tier m. For Application Server tier, the QoS actuator can be the number of threads. For Database Server tier, the QoS actuator can be the CPU shares. In addition, we use a scaling factor $v_{k,m}$ to model the behavior that the requests processed at one tier may generate multiple requests for the lower tier; the value of $v_{k,m}$ can be obtained through monitoring the throughput of each tier.

As described in Section 3, we use a function approximation approach to build an equivalent model for each class at each tier, where the service rate is approximated using Equation (3)-Equation (5). By doing that, we encapsulate the level of per-class details such as concurrent server threads within the representation of the equivalent server, so that service differentiation between different classes can be studied at a compact level through a coupled processor model [11]. For the coupled processor model (CPM), suppose a system consists of two service classes and two servers–each server serves one service class. If customers of both classes are available, both servers are busy and serve with service rate μ_1 and μ_2. However, if only class 1 is available but class 2 is not, server 2 is idle and server 1 acts with a service rate μ_1'; whereas if only class 2 is available, server 1 is idle and server 2 acts with a service rate μ_2'. The coupled processor model shows work-conserving behavior if $\mu_1' = \mu_2' = \mu_1 + \mu_2$, and can be solved using a birth-death process to determine stationary distributions of Markov chains (e.g., with the power series algorithm [12]).

Specifically, deriving from Equation (3), the service rate of class k at tier m is defined as

$$\mu_{k,m} = \frac{1}{\frac{1}{p_{k,m} X_{k,m}} + V_{k,m+1} T_{k,m+1}} \tag{15}$$

where $p_{k,m}$ indicates the allocated capacity portion from the QoS control actuators, $\sum_{k=1}^{K} p_{k,m} = 1, (p_{k,m} \geq 0)$. If class l at tier m is idle, the service rate for server k at tier m can be defined by

$$\mu_{k,m} = \frac{1}{\frac{1}{\left(p_{k,m} + \frac{p_{l,m}}{K-1}\right) X_{k,m}} + V_{k,m+1} T_{k,m+1}} \tag{16}$$

Given the equivalent service rate $\mu_{k,m}$, the tier response time can be computed using the coupled processor model

$$T_{k,m} = \begin{matrix} h_{CPM}\left(\lambda_{k,m}, \mu_{k,m}\right), & m > 1 \\ h_{CPM}\left(C_k, Z_k, \mu_{k,1}\right), & m = 1 \end{matrix} \tag{17}$$

and the end-to-end response time can be computed using the layered queueing model as in Section 3.

Note that we have also modeled the system as an open queueing model for the situations where the number of clients and think time are difficult to obtain. Instead, we only need to measure the arrival rate of each class. In the interest of brevity, however, we do not discuss the open queueing model in this paper.

5 Model Calibration and Validation

In this section we assess the model quality by conducting experiments on the testbed using the *Trade* [13] benchmark (which simulates a stock trading service). The model is first calibrated using the measurement data (i.e., response time, throughput, and server utilizations) from single class runs. Afterwards, the model is extrapolated to two classes and validated against the measurements. Furthermore, we compare the performance of the proposed model with the machine repair model and the tandem queueing model, respectively.

5.1 Testbed Setup

The testbed is composed of a workload driver, one HTTP proxy node, four application server nodes and one database node connected via 100Mb Ethernet LAN. All nodes use SUSE Linux Enterprise Server v9.3 as the Operating System. The proxy and application servers have Pentium 4 2.4 GHz CPU with 2GB RAM and run IBM WebSphere Application Server (eXtended Deployment) v6.0.1 (WAS XD). The DB2 node has a Pentium 4 1.8 GHz CPU with 1.5GB RAM and several 10,000 RPM EIDE disks, and runs IBM DB2 v8.2.4. In all experiments, the workload driver, proxy node and network were not saturated.

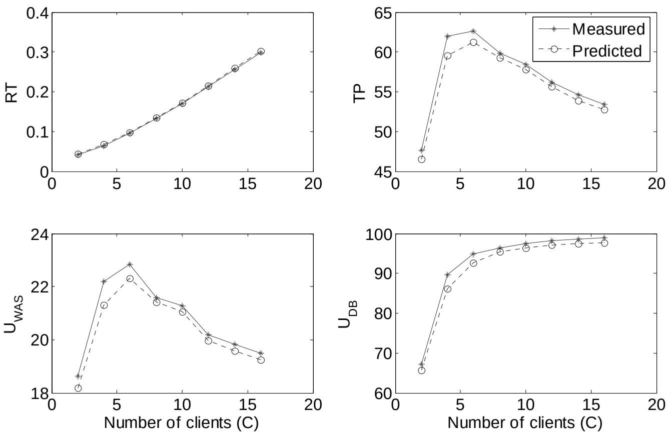


Figure 6. Modeling results for a multi-tier system (single class, model calibration).

In our experiments, the two classes of work (A and B) correspond to separate instances (TradeA and TradeB) of the Trade application. There is one instance of each application deployed on each of the four WAS nodes. Both applications share the same HTTP proxy node. In the experiments shown here, the proxy node utilization is consistently below 30%, and it achieves fairly uniform load balancing across the application servers. Each Trade instance accesses its own database on the DB2 server. The databases are set up with the data files and logs on separate disks. We allocate CPU shares on the DB2 server between the two databases using the CKRM [14] facility provided with the OS.

The workload driver is a closed-loop driver that issues requests by randomly generating query parameters and performs various transactions according to a probability distribution. Response time and throughput information is recorded from the workload driver. CPU utilization statistics are collected using a reporting facility built in to WAS XD.

5.2 Model Calibration

We start from calibrating the model from one class experimental data as shown in Figure 6. It indicates a close fit between the measured data (marked by the asterisks and solid lines) and the predicted values (marked by the circles and dashed lines) of response time (in the unit of seconds), throughput, WAS utilization (in the unit of percentage), and DB utilization (in the unit of percentage) for different numbers of clients. Regardless of only using a small number of clients, since their think time is set to zero, this still results

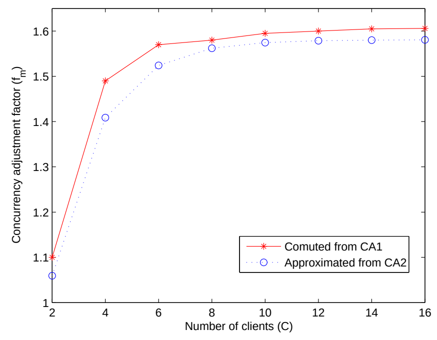


Figure 7. Curve fitting during model calibration.

in a large range of workload intensity (from 65% to 99% CPU utilization for the database tier). Figure 7 also shows the results of Calibration Algorithm 2 ($CA2$) that use the exponential function to approximate the load effect, where the computed values from $CA1$ are marked by the asterisks and solid lines, and the approximated values from $CA2$ are marked by the circles and dashed lines. Note that this is a biased approximation because of nonlinear model regression.

The model accuracy is also assessed through the R-Square metric (R^2) and the Root Mean Squared Error ($RMSE$). The R-Square metric is defined as $R^2 = 1 - \frac{var(y-\hat{y})}{var(y)}$ where y is the response variable (e.g., the measured response time), $\hat{y}$ is the estimated response variable, and $var(.)$ is the variance. The R^2 metric quantifies model accuracy by computing the variability explained by the model. It can take values from 0 to 1. A value of 0 means the response data variability is not captured at all, and a value of 1 may suggest a perfect fit. Note that it is also possible to get a negative R^2 value if the model performs worse than just fitting a constant. The Root Mean Squared Error is defined as $RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y(n) - \hat{y}(n))^2}$, where N is the number of measurements, $y(n)$ is the n-th measurement, and $\hat{y}(n)$ is the n-th estimated value.

For the modeling results shown in Figure 6, the R^2 metrics are shown in Table 1. That is, for example, the model has been calibrated to explain 99% of the variability in the response time data and the $RMSE$ is 0.0028.

5.3 Model Validation

After model calibration, we proceed to the model extrapolation phase by conducting experiments with two classes of work (TradeA and TradeB). The model was used to answer the question–if there are two classes with different number of clients and different CPU shares, what are their

Table 1. Model assessment using the R^2 metric and the Root Mean Squared Error (calibration phase). The model types include the proposed hybrid layered model and two other comparative models, the single-tier machine repair model, and the tandem queueing model.

	Model Type	RT	TP	U_{WAS}	U_{DB}
R^2	Layered	0.9999	0.9831	0.9743	0.9934
R^2	Single-Tier	0.9998	0.9423	N/A	0.4781
R^2	Tandem	0.9999	0.9279	0.8672	0.5471
$RMSE$	Layered	0.0028	1.1944	0.4341	1.8112
$RMSE$	Single-Tier	0.0016	1.7983	N/A	9.4864
$RMSE$	Tandem	0.0007	2.3265	0.8825	8.9595

Table 2. Model assessment using the R^2 metric and the Root Mean Squared Error for the response time (model-based prediction phase).

	Layered	Single-Tier	Tandem
R^2	0.83	-2.55	-2.61
$RMSE$	0.015	0.069	0.070

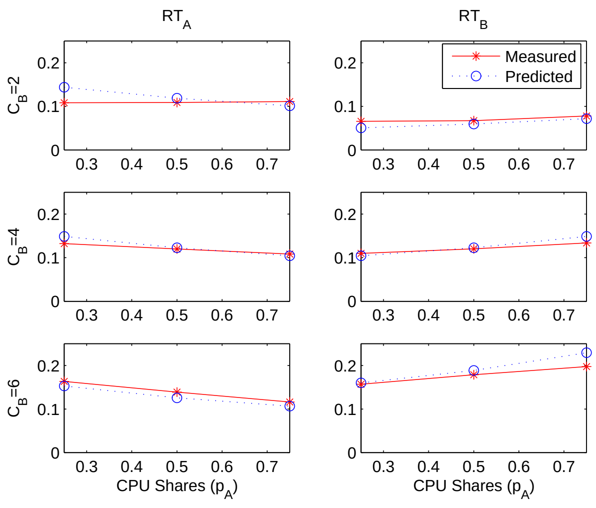


Figure 8. Modeling results for a multi-tier system (two classes, model extrapolation).

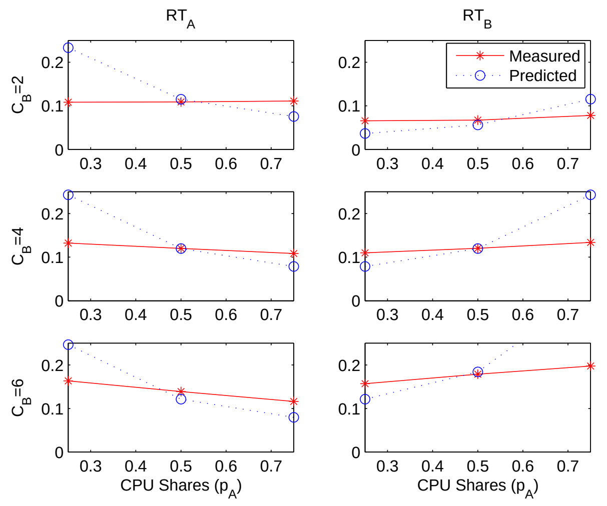


Figure 9. Modeling results for a multi-tier system (two classes, model extrapolation): Single-tier machine repair model.

response time? Then the estimated response time is compared to the measured ones.

As shown in Figure 8, we fix the number of clients for class A at 4 and vary the class B clients (C_B) from 2 to 4 and then to 6. We also vary the CPU shares on the DB2 server from 25% to 50% and then to 75% for class A (p_A). The CPU shares for class B is $p_B = 1 - p_A$. It also indicates a close fit between the measured data (marked by the asterisks and solid lines) and the predicted values (marked by the circles and dashed lines). For the modeling results shown in Figure 8, $R^2 = 0.83$ and $RMSE = 0.015$, as shown in Table 2, where the response variable includes the response time for both class A and class B. The model is able to explains 83% of the variability in the response time data.

5.4 Comparative Studies

By assuming that there is a single bottleneck tier/resource (as in the above example where the CPU in the database tier is the bottleneck), a natural question is whether a layered model is required. We discussed earlier in the paper that the layered queueing model is essential because it models the structure of dependencies between tiers. In this section, we use the experimental data to access the quality of a single tier model. Specifically, we build a machine repair model where the database tier is modeled as service center 2 and its service time is computed using the calibration algorithm 0 (CA_0) as in Section 3.3.1., and the service time of the WAS tier is modeled as part of the think time in service center 1. The function approximation technique (described in Section 3.2) is also used to fit for the response time with Calibration Algorithm 1 (CA_1). The R^2 metrics for model calibration are shown in Table 1. Although, the model is able to be calibrated to get a best fit of the response time, the machine repair model overestimates the database tier utilization, which results in a low R^2 of 0.48. The model extrapolation results are shown in Figure 9. Although the single-tier machine repair model is able to capture the trend of workload and actuator variations, the magnitude of the error is much large. Its variance is even larger than the variance in the measurement, which gives a negative R^2 value as shown in Table 2. With regard to the RMSE error, the layered model shows 78% of improvement over the machine repair model.

We have also compared to a tandem queue model which explicitly models the WAS tier. However, since the WAS tier is a non-bottleneck tier, and tandem queue does not capture the structure of cross-tier dependencies, it does not improve the modeling performance as shown in Table 1 and Table 2.

6 Model Applications

We consider applying the model to model-based resource control. That is, online adjusting the resource shares for each class in response of workload changes. As shown in Figure 10, there are two service classes in our experiments: Class A has a response time SLO of 0.2 sec; class B has a response time SLO of 1 sec. In the first 1200 seconds of the experiment, the number of clients for both classes is 2; after that, the number of clients for class A increase to 8. This results in an increase of response time. At 2700 seconds, the model based control adjusts the CPU share from 50% to 90% for class A and brings down its response time so that its SLO can be achieved.

7 Conclusions

Managing Quality of Service in multi-tiered Internet services is a challenging task. There is a lack of models that capture the important characteristics of real-world software stacks used for delivering such services. In this paper we have presented a hybrid performance model for differentiated services of multi-tier web applications. We use a layered queueing model to capture the service dependencies between multiple tiers, and model the per-tier concurrency limits and resource contention using a function approximation approach. This simplified approach allows the deployment of a coupled processor model to quantify service differentiation between multiple classes.

The validity of our model was assessed by comparison to a real testbed using commercial software products. We

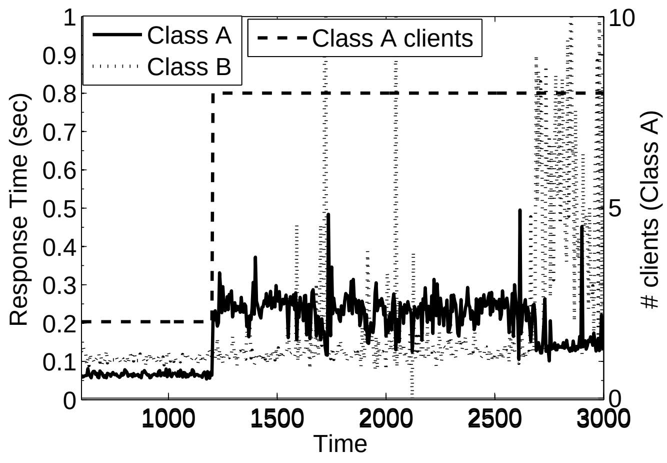


Figure 10. Model-based resource control. (Place holder plot–waiting for new results.)

show that the layered model captures the inter-tier relationships, and provides much better performance then a comparable single-tier machine repair model. Further, we also show the value of such inter-tier modeling against a tandem queue approach that does not reflect the synchronous waiting induced by the multi-tier architecture. Finally, we illustrate how the proposed performance model can be effectively used for model-based resource allocation.

In our future work, we seek to explore further applications of the model. In particular, the model can be used in a closed-loop model-driven realtime resource controller. An interesting issue is to compare control strategies that considers the request interactions between multiple tiers, against the behavior of local control schemes used at individual tiers. The model could also be used for capacity planning and dynamic provisioning scenarios. Our evaluation in this paper has been limited to a single workload – We would also like to conduct experimental assessment for a larger variety of workloads, as well as testing our model's validity when mixing heterogenous workloads.

References

[1] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster based web services," in *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management, (Colorado Springs, CO)*, pp. 247–261, 2003.

[2] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: a mechanism for resource management in cluster-based network servers," in *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, (New Orleans, LA)*, pp. 90–101, 2000.

[3] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated resource management for cluster-based internet services," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (Boston, MA*, 2002.

[4] M. Karlsson, C. Karamanolis, and J. Chase, "Controllable fair queuing for meeting performance goals," in *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation, Juan-les-Pins, France*, pp. 278–294, 2005.

[5] D. Menasce, D. Barbara, and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: A performance model approach," in *Proceedings of 2001 ACM Conference on E-commerce*, 2001.

[6] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Banff, Alberta, Canada), June 6–10 2005.

[7] J. A. Rolia and K. C. Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689–700, 1995.

[8] S. S. Lavenberg, ed., *Computer performance modeling handbook*. Orlando, FL: Academic Press, INC, 1983.

[9] M. Reiser and S. Lavenberg, "Mean-value analysis of closed-multi chain queueing networks," *Journal of the ACM*, vol. 27, no. 2, 1980.

[10] L. Ljung, *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice Hall, second ed., 1999.

[11] J. W. Cohen and O. J. Boxma, *Boundary value problems in queueing system analysis*. North-Holland, 1983.

[12] G. Hooghiemstra and G. Koole, "On the convergence of the power series algorithm," *Performance Evaluation*, vol. 42, pp. 21–39, 2000.

[13] IBM WebSphere Software. http://www.ibm.com/software/webservers/appserv/benchmark3.html.

[14] S. Nagar, R. van Riel, H. Franke, C. Seetharaman, V. Kashyap, and H. Zheng, "Improving linux resource control using ckrm," in *Proceedings of the Linux Symposium*, vol. 2, (Ottawa, ON, Canada), pp. 511–524, 2004.