

# Modeling 3-Tiered Web Applications

Xue Liu  
University of Illinois at  
Urbana-Champaign  
xueliu@cs.uiuc.edu

Jin Heo  
University of Illinois at  
Urbana-Champaign  
jinheo@cs.uiuc.edu

Lui Sha  
University of Illinois at  
Urbana-Champaign  
lrs@cs.uiuc.edu

## Abstract

*The rapid advancement and deployment of Web services call for a precise yet simple model for capacity planning and analysis purposes. The most widely deployed Web services architecture is the 3-tiered system, which is composed of a front-end Web server, an application server and a backend database server. In this paper, we present an analytical model of the 3-tiered Web services architecture. We show by using queueing network theory, we can model the 3-tiered Web services architecture accurately. A testbed is built to measure model parameters based on industry standard server components and TPC-W benchmark. Validation results show the proposed model predicts performance measures such as response time and throughput accurately.*

## 1. Introduction and Related Work

As E-Commerce is keeping rapid pace of growth [1], the underlying architecture providing the service of E-Commerce is becoming more and more important. The architecture is generally called as “Web services”. The term “Web services” describes specific business functionality exposed by a company through Internet connection, for the purpose of providing a way for another entity to use the services it provides [2]. Web services are the building blocks for the future generation of applications and solutions on Internet.

One of the most pressing problems faced by Web services designers and deployment companies is the adequate sizing of their infrastructure so that they can provide the quality-of-service (QoS) required by their clients. This is known as the capacity planning problem. As Web keeps evolving, it is critical for a service provider to provide service level assurances and agreements to its customers. A validated model of Web services architecture is the basis of capacity performance in different settings [3, 4].

In addition to capacity planning, a simple yet accurate model of Web services is also instrumental for other purposes such as overload control [5, 6] or resource provisioning [7-9]. Overload control and performance management are research areas on their own, but their effectiveness are based on the accuracy of good performance models.

Most of the existing work of Web services performance modeling is confined to the front-end Web server. Wells et al. [10] made a performance analysis of Web servers using colored Petri nets. Their model is divided into three layers, where each layer models a certain aspect of the system. The model has several parameters, some of which are known and the remaining unknown parameters are determined by simulations. Dilley et al. [11] used a layered queuing model for analysis of Web server. Recently, Cao et al. [12] provides a  $M/G/1/K$  processor sharing queue model for Web server which assumes Poisson arrival, general service time and bounded accepted request number of  $K$ . An improved version is given in [13] using Markov Modulated Poisson Process (MMPP) to model bursty arrival. Cherkasova and Phaal [14] used a similar model to that in [12] but assumed deterministic service time distribution and session-based workload.

Few work has been done to model the integrated Web services architecture. In [15], Karma et al. use an abstraction of  $M/G/1$  processor sharing queue to represent the whole 3-tiered architecture. This is in essence a single tier modeling for the bottlenecked server in the Web services architecture. Doyle et al. [7] presents a simplified analytical model to predict the response time of Web services. Their model is a combination model of server CPU and storage I/O. Still, the model only applies to single tier and is valid only for static content requests.

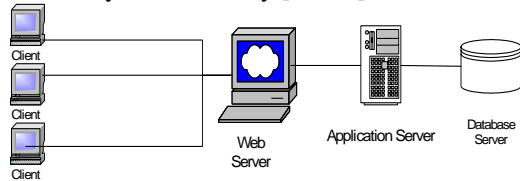
In this paper, we propose a new and effective model of the 3-tiered Web services architecture. The model provides a quantitative way to analyze the effectiveness of the Web services. In contrast to most of the previous work, which has only addressed one

tier or much simpler workload (static content), our approach is fundamentally concerned with multi-tiered Web sites that include dynamic content and back-end databases. As a result, the proposed model provides higher accuracy yet still remains simple. A simple model is important in performance modeling because it renders a smaller parameter space thus easier to estimate; while a complicated model usually contains parameters that are difficult to obtain and validate. Hence a simple model tends to be more useful in real-world applications [4].

The rest of the paper is organized as follows. Section 2 provides the background of Web services architecture. In Section 3, we first give an individual model of each tier in the Web services architecture, then the analytical model of 3-tiered Web services is presented. Solution techniques for the proposed model are discussed in Section 4. The design and implementation of our Web services testbed are discussed in Section 5. We present the validation procedure and results of the proposed model in Section 6. Finally, we conclude the paper with directions of future work in Section 7.

## 2. Background

Modern Web services providers use a multi-tiered architecture to provide required services. While some Web applications use two tiers—Web server and database server—high volume sites typically add a third tier: application server to support complex business logic. As a consequence, the most widely deployed infrastructure of Web services is the 3-tiered architecture which is shown in Figure 1. This 3-tiered architecture provides both high level of scalability and reliability [16, 17].



**Figure 1. 3-Tiered Web services architecture**

In this 3-tiered architecture, on the front line of a typical Web site is the Web server that acts as the presentation layer. This tier has three functionalities: (1) Web server receives requests from the clients, service static Web requests; (2) at the same time forwards complex dynamic content requests to the 2<sup>nd</sup> tier; (3) receives responses from the 2<sup>nd</sup> tier and sends them back to the clients. Typical Web server includes Apache and Microsoft Internet Information Server (IIS).

All the business logic for a Web site resides in the 2<sup>nd</sup> tier -- application server. Application server receives requests from Web server, looks up information in database (3<sup>rd</sup> tier) and processes the information. The processed information is then passed back to the Web server where it is formatted to be displayed on clients' machines. Typical application server includes Apache Tomcat, Sun Java System Application Server, BEA WebLogic, IBM WebSphere, and JBOSS.

The 3<sup>rd</sup> tier -- database server is the storehouse of a Web site's information. Everything from user accounts and catalogs to reports and customer orders is stored in database. Typical database server used in Web services architecture includes Oracle, Microsoft SQL Server, Sybase, IBM DB2, MySQL, and PostgreSQL.

We built a testbed to emulate an online Web services provider in our Lab. The front-end is using Apache Web server [18], application server is running Tomcat [19], and the backend database is running MySQL [20]. The reasons that we select this combination are: 1) all of them are open source projects and freely available; 2) their performances are among the highest of all individual components; 3) they are widely used on the Internet, even in commercial sites such as *Amazon.com*, *Mp3.com* and *Yahoo Finance* to provide Web services. Hence this combination is quite representative to the current technology. It is worth noting that the analytical model proposed in this paper (Section 3) does not depend on this specific combination and is general enough for modeling other Web services deployments.

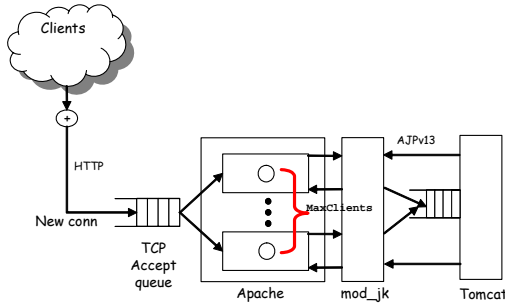
## 3. Modeling of 3-Tiered Web Services Architecture

In this section, we present an analytical model for the 3-tiered Web services architecture. We use the typical deployment consisting of Apache, Tomcat and MySQL discussed above for better explanation and understanding of the intuitions behind the analytical model. In other Web services deployments, individual tier's architecture is similar; hence the proposed model can be easily applied to other set-ups with little modification.

### 3.1. Web Server Architecture and Modeling

Modern Web server typically utilizes a multi-threaded (or multi-process) architecture. It is structured as a pool of worker threads to handle HTTP requests.

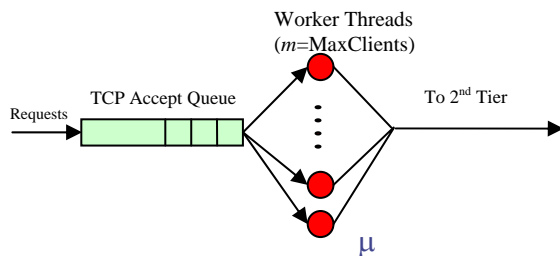
Apache [18] has been the most popular Web server on the Internet. The February 2005 Netcraft Web Server Survey [21] shows that 68.8% of the Web sites on the Internet use Apache, which makes it the dominant Web server deployed.



**Figure 2. Apache multi-threaded architecture**

The flow of servicing a HTTP request in Apache (version 1.3.x and version 2.x) is displayed in Figure 2. Requests enter the TCP accept queue where they wait for a worker thread. A worker thread processes a single request to completion before accepting another new request. After a HTTP request is serviced by Web server, it is forwarded to 2<sup>nd</sup> tier using AJP (Apache JServ Protocol) for further processing.

We use a multi-station queueing center to model the multi-threaded Web server. The corresponding queueing model is shown in Figure 3.



**Figure 3. Queueing model of multi-threaded Web server architecture**

In this model, each worker thread is represented by a station. The total number of stations (i.e. worker threads) is denoted by  $m$ . Requests have mean service time  $D$  in the station, which corresponds to a service rate  $\mu = 1/D$  per station.

### 3.2. Application Server Architecture and Modeling

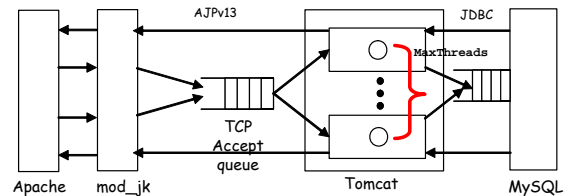
An application server is a component-based product that resides in the middle-tier of the 3-tiered Web services architecture. It provides middleware

services for security and state maintenance, along with data access and persistence support.

Application server typically consists of *Servlet Engine* and *EJB (Enterprise Java Beans) container*. *Servlet Engine* is a place to execute Java Servlets, a basic building block of Web applications. EJB container is the environment that Java beans will execute in, and EJB container will manage transactions, thread pools, and connection pools.

Tomcat [19] is a servlet engine that is used in the official reference implementation for the Java Servlet and Java Server Pages (JSP) technologies [22]. Unlike commercial Web application servers, Tomcat is not an EJB container but only a servlet engine. This means it supports servlets and plain Java objects but not EJBs. Here, we use Tomcat for the 2<sup>nd</sup> tier in our testbed, since it is easy to configure and deploy applications. The functionalities such as transaction management which are usually provided by EJB container are implemented in the application level in our testbed.

Tomcat has a similar structure as Apache. It also maintains a pool of threads. When a request is received, an idle thread is assigned to execute the request. The flow of requests in Tomcat is shown in Figure 4. For the modeling purpose, we can also use the multi-station queueing center to model Tomcat (similar to Figure 3).



**Figure 4. Tomcat multi-threaded architecture**

### 3.3. Database Server Architecture and Modeling

MySQL [20] is the world's most popular open source database with more than 2 million installations. The unique design of separation of the core server from the table handler makes it possible to run MySQL under strict transaction control or with ultrafast transactionless disk accesses, which makes MySQL most appropriate for Web services deployment.

Like Apache and Tomcat, MySQL also has a multi-threaded architecture. However, in terms of thread management, MySQL is different from Apache or Tomcat. It doesn't use thread pool model

but uses thread cache instead, where the latter is a group of threads that can be reused by subsequent SQL queries. Thread cache is different from thread pool in the sense that thread cache does not pre-create threads at system startup. Instead, the threads are managed in a dynamic fashion. When workload is heavy such that the number of required concurrent threads exceeds the cache size, it creates new threads to serve extra requests. After that only the cache size number of threads are reused and maintained alive.

In terms of modeling, we can use load-dependent [3] multi-station queueing center to model thread cache. This means the service rate per station is modeled as changing with the total number of requests being serviced in the system. However, using load-dependent service rate increases both the complexity of service time measurement and the complexity of the solution to the model. So we propose an approximated model which still uses load-independent multi-station queueing center to model MySQL. In the approximated model, the number of stations is the averaged number of all worker threads of MySQL during a run.

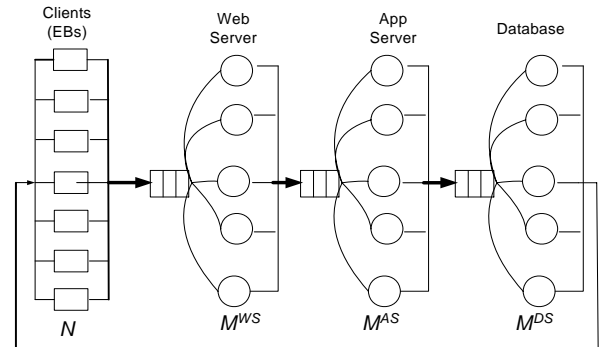
### 3.4. Queueing Network Model of 3-Tiered Architecture

With individual tier's queueing models given above, we are now ready to present the model of the 3-tiered Web services architecture. Figure 5 shows the closed queueing network model (QNM) of the 3-tiered Web services.

The concurrency of multiple clients accessing a Web site is modeled by  $N$  users in a queueing system. Each client sends requests and gets responses from the 3-tiered system. A request is generated by a client through its browser, and is sent to the front-end Web server via HTTP protocol. The request then goes through the 3 tiers until the response is formed and sent back. Each tier is modeled by a multi-station queueing center as discussed in previous subsections, with the number of stations being the server's total number of worker threads. As discussed in Section 3.3, for database server which does not use a thread pool structure, the number of stations corresponds to the time averaged number of worker threads in a run. We use  $M^{WS}$ ,  $M^{AS}$ ,  $M^{DS}$  to denote the number of worker threads at each tier respectively. Similarly, the mean service time of requests at each tier is denoted by  $D^{WS}$ ,  $D^{AS}$  and  $D^{DS}$ .

A client typically waits until the previous request's response is back 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 client is called the "think time". We use  $Z$  to denote think time.



**Figure 5. Queueing network model of 3-Tiered Web services architecture**

Given the parameters  $\{ N, M^{WS}, M^{AS}, M^{DS}, D^{WS}, D^{AS}, D^{DS}, Z \}$ , the proposed closed queueing network model can be solved analytically by assuming the service time and think time are exponentially distributed. In Section 4, we will present the solution technique we used.

## 4. Solution Techniques for Evaluating the Model

Mean-value analysis (MVA) is an efficient algorithm for evaluating closed queueing network models with exact solutions [4, 23]. Traditional MVA deals with single-station queue at each service center. It takes the following set of parameters as inputs:

- $K$  the number of servers (i.e. centers);
- $n$  the total population in the system;
- $Z$  the think time;
- $D_k$  the mean service demand of each request at server  $k$  ( $k = 1, \dots, K$ ).

Using MVA, the following performance measures are determined as outputs:

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

MVA is an iterative algorithm. It is based on the following three recursive equations:

**Response time equation:**

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

**Throughput equation:**

$$X(n) = \frac{n}{\sum_{k=1}^K R_k(n)} \quad (2)$$

**Queue length equation:**

$$Q_k(n) = X \times R_k(n) \quad (3)$$

where  $Q_k(n)$  is the mean queue length at server  $k$  when the total population in the system is  $n$ .

From equations (1)-(3), if we know the performance measures when system population is  $(n-1)$ , we can compute the performance measures when system population is  $n$ . i.e.

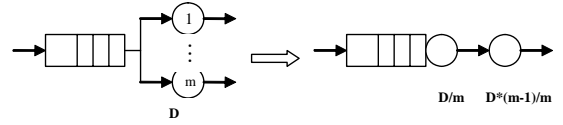
$$\begin{aligned} & \{R_k(n-1), X(n-1), Q_k(n-1)\} \\ & \rightarrow \{R_k(n), X(n), Q_k(n)\}. \end{aligned} \quad (4)$$

So beginning from the initial conditions when the system population is 1, the performance measures of any population can be derived using equations (1)-(3).

The iterative MVA algorithm described above only applies to the single-station queueing center model. However, in our proposed model of 3-tiered Web services, the multiple worker threads structure in each tier is modeled with a multi-station queueing center. In order to use the above mentioned MVA algorithm for solution, here we adopt an approximation method proposed by Seidmann et al. [24].

The idea is to replace a queueing center who has  $m$  stations and service demand  $D$  at each station with two queues in tandem (See Figure 6). The first queue is a single-station queue with service demand  $D/m$  (i.e. with a new center works  $m$  times faster than any station in the original center). The second queue is a pure delay center, with delay  $D(m-1)/m$ . For reasoning behind this approximation method, please see [24].

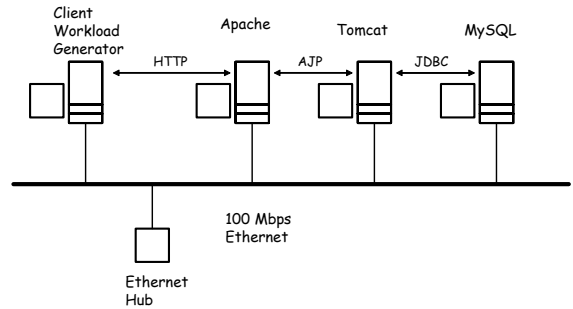
It has been shown in [3] that the error introduced by this approximation is small. By applying this approximation to each tier, our 3-tiered multi-station queueing network model shown in Figure 5 is converted to a new closed queueing model, with 3 single-station queueing centers and 3 additional single-station pure delay centers. After the conversion, the model is readily solvable using the MVA algorithm.



**Figure 6. Approximation method for multi-station queueing center**

## 5. Implementation and Performance Measurements

To validate the correctness and accuracy of our model, we built a testbed as shown in Figure 7. The testbed is composed of four machines. One of them is used as client workload generator and the other three machines are used as Web server, application server and database server respectively. They are connected with 100 Mbps Ethernet connections.



**Figure 7. Testbed infrastructure**

The client machine is equipped with a 2.8 GHZ Intel Pentium IV processor and 512MB RAM. TPC-W client emulator [25] is used as synthetic workload generator on the client. The Web server machine has a 1GHZ Intel Pentium III processor and 256MB RAM, which runs Apache 2.0.7 [18]. The application server machine has a 1.5GHZ Intel Pentium III processor and 256MB RAM, which runs Tomcat 5.0 [19]. The database server machine has a 1GHZ Intel Pentium III processor and 512MB RAM, which runs MySQL 4.1.7 [20].

### 5.1. Client Workload Generation

We use TPC-W [25], an industry standard e-Commerce benchmarking tool as a client request generator. TPC-W from the Transaction Processing Council (TPC) is a transactional Web benchmark specifically designed for evaluating e-commerce

systems. Thus, it implements all functionalities that typical e-commerce Web sites provide, including multiple online browser sessions, dynamic Web page generations, database transactions, authentications and authorizations. TPC-W specifies 14 unique Web interactions, which are different from each other in the sense that they require different amount of server side work. Most interactions require generation of dynamic pages and database queries, range from simple select SQL statements to complicated transactions.

We modified a Java implementation of TPC-W from the PHARM group at the University of Wisconsin [26] to make it compatible with the newest version of Tomcat and MySQL installed in our testbed. It implements all functionalities in TPC-W specification. The database is configured to contain 10,000 items and 288,000 customers.

TPC-W benchmark generates requests by starting a number of emulated browsers (EB). EBs send requests to the Web server and receive HTML pages back as response. Between when an EB received the last byte of a Web page and when it begins to send the first byte of a new request, it waits for certain amount of *Think Time* to mimic user's think time.

## 5.2. Performance Measurements

A performance model aims at representing the behavior of a real system in terms of its performance. The representativeness of a model depends directly on the quality of its input parameters [3]. Measuring details of input parameters such as service time (service demand) becomes complicated and difficult in a multi-tiered environment due to the lack of centralized and transaction-oriented monitoring facilities.

We modified the source code of Apache, Tomcat and MySQL to measure model input and output parameters, these include: request throughput at each tier, request's total response time, request's service time in each tier. For MySQL, which uses the thread cache mechanism and doesn't provide a constant thread pool, in order to get the time averaged number of threads, we also changed MySQL's source code to constantly measure the number of the threads running currently. It is worth noting that measuring service time in each tier is not an easy task, since each request involves several operations on each tier. What is more, during an operation, the server on each tier may block waiting for a reply from its next neighboring tier more than once. To solve this, we come up with a measuring technique to measure service time accurately in each tier. Due to space

limit, we have to omit the detailed discussions in this paper, interested readers are referred to our technical report [27].

## 6. Results and Model Validation

We carried out extensive experiments to validate the model presented in Section 3 based on the measurements obtained from the testbed. The validation method is as follows. First, for each designed test case, we measure or estimate the input parameters of the model as well as the performance measures during the tests. Using the input parameters and the solution techniques presented in section 4, we can predict the Web services performance such as response time and throughput. The validation is done by comparing the value of performance data got from predicted model output and those measured in the testbed. Due to space limit, we present only one set of validation results in this section.

**Table 1. Notations used in measurement data**

$N$	# of emulated browsers (EB) in TPC-W
$Z$	Think time of EB (sec)
$M^{WS}$	# of worker threads in Apache
$M^{AS}$	# of worker threads in Tomcat
$M^{DS}$	Average # of worker threads in MySQL
$D^{WS}$	Mean service time of requests in Apache threads (sec)
$D^{AS}$	Mean service time of requests in Tomcat threads (sec)
$D^{DS}$	Mean service time of requests in MySQL threads (sec)
$D_{norm}^{DS}$	Normalized Mean service time of each transaction in MySQL threads (sec) (See explanation below)
$X^{WS}$	Throughput of Apache (requests/sec)
$X^{AS}$	Throughput of Tomcat (requests/sec)
$X^{DS}$	Throughput of MySQL (requests/sec)
$X$	Throughput of the system (transactions/sec)
$T$	Mean response time of transactions (sec)

In this test case, we change the workload of client requests to the 3-tiered Web services system by varying the number of EBs in TPC-W. In each individual test of this test case group, the number of EBs is fixed and each run lasts 200 seconds after the 60 seconds of warm-up period. We measure different model input and output parameters during each test. Their notations and meanings are summarized in Table 1. In all these tests, the *Think Time* of TPC-W EBs is set to 0.035 sec. We varied the number of EBs from 1 to 200. The test results with calculated model predicted performance data are shown in Table 2.

Note in Table 2, the throughputs at each tier ( $D^{WS}, D^{AS}, D^{DS}$ ) are measured in terms of *requests/second* and service times are averaged service time per request. Each transaction in TPC-W contains one HTTP request. After this HTTP request is parsed at Web server, if it is a servlet request, then it is forwarded to the application server using AJP (Apache JServ Protocol) and waits for Tomcat to reply. However, each servlet may contain one or several SQL requests which are sent to and executed on database server. As a result, each transaction serviced by the 3-tiered system will approximately contribute to one HTTP request at the Web server tier, one servlet request at the application server tier and one or several SQL requests at the database tier. That explains why the measured throughputs in terms of *requests/sec* is different at each tier, with the relationship of  $X^{WS} \simeq X^{AS} < X^{DS}$ . In order to compensate this to get performance measures with respect to *transactions/sec* in the analytical model, we normalize the measured average per request service time at database ( $D^{DS}$ ) to approximate average per transaction service time  $D_{norm}^{DS}$  by multiplying the ratio of  $X^{DS} / X^{AS}$ . This is because one transaction invokes roughly  $X^{DS} / X^{AS}$  SQL requests in database server.

When the input parameters are applied, the MVA algorithm determines the (predicted) performance

data for each individual test with a fixed client population  $N$ . Model predicted response times and throughputs are also shown in Table 2 for validation.

From the experiments' result shown in Table 2, we find that the analytic model does predict the performance of 3-tiered Web services system accurately. Under different workload conditions, the model predicted response time ( $T$ ) and throughput of the system ( $X$ ) are very close to those directly measured in the testbed. The small difference between the model-predicted values and measured values may be due to two reasons: First, in testbed measurements, the service times at each tier are not exponentially distributed as assumed by the analytical model. Second, as stated in Section 4, the approximated MVA algorithm we used for the solution of performance measures is an approximation for the multi-station queueing center. However, we see from the results that even with those approximations, the proposed model still predicts the performance of the 3-tiered Web services system accurately.

## 7. Conclusions and Future Work

In this paper, we presented a queueing network model of the 3-tiered Web services architecture. In this model, the server at each tier is modeled as a multi-station queueing center, which represents the

**Table 2. Model predicted and measurement data collected from testbed**

$N$	1	5	10	50	100	150	200
$M^{WS}$	50	50	50	50	50	50	50
$L^{WS}$	0.5817	4.2548	9.1394	49.0384	99.0240	148.6923	198.9567
$X^{WS}$	11.1391	23.5220	27.0849	28.8355	29.2947	28.7636	27.5298
$D^{WS}$	0.0012	0.0014	0.0016	0.0016	0.0014	0.0015	0.0015
$M^{AS}$	50	50	50	50	50	50	50
$X^{AS}$	11.1186	23.4356	26.9335	28.7792	29.2895	28.6878	27.4815
$D^{AS}$	0.0151	0.0971	0.2798	1.6507	1.6749	1.7048	1.7895
$M^{DS}$ (rounded value)	0.8923 (1)	3.1394 (3)	5.2740 (5)	25.6778 (26)	26.1442 (26)	25.4230 (25)	25.7788 (25)
$X^{DS}$	17.8217	33.6850	38.5728	41.4092	42.1601	41.4999	39.9598
$D^{DS}$	0.0254	0.0471	0.0353	0.0320	0.0298	0.0309	0.0340
$D_{norm}^{DS}$	0.0367	0.0679	0.0506	0.0460	0.0429	0.0448	0.0495
$T$ (measured)	0.0531	0.1664	0.3353	1.7019	3.3948	5.2089	7.3343
$T$ (model predicted)	0.0417	0.1527	0.3191	1.8339	3.3148	5.0793	7.1230
$X$ (model predicted)	13.0463	26.6437	28.2397	26.7543	29.8524	29.3297	27.9407

multi-threaded architecture commonly structured in the modern Web, application and database servers. A testbed is built based on the widely deployed combination of Apache, Tomcat and MySQL, with the client request generator being the industry standard TPC-W benchmark. Validation tests show the proposed model can predict performance measures such as response time and throughput very accurately.

Our future work includes using the proposed model for better QoS provisioning and better design of overload control schemes for 3-tiered Web services.

## 8. References

- [1] census.gov, "QUARTERLY RETAIL E-COMMERCE SALES, 3rd QUARTER 2004," vol. 2004: The Census Bureau of the Department of Commerce, 2004.
- [2] UDDI, "Universal Description, Discovery, and Integration of Business for the Web"
- [3] D. Menasce and V. Almeida, Capacity Planning for Web Services: Metrics, Models, and Methods: Prentice Hall PTR, 2001.
- [4] R. Jain, The Art of Computer Systems Performance Analysis: John Wiley & Sons, 1991.
- [5] M. Welsh and D. Culler, "Adaptive Overload Control for Busy Internet Servers," presented at 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA, 2003.
- [6] H. Chen and P. Mohapatra, "Session-Based Overload Control in QoS-Aware Web Servers," presented at IEEE INFOCOM, 2002.
- [7] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility," presented at USITS, 2003.
- [8] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing Model Based Network Server Performance Control," in Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02): IEEE Computer Society, 2002, pp. 81.
- [9] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," presented at IEEE Real-Time Technology and Applications Symposium, TaiPei, Taiwan, 2001.
- [10] L. Wells, S. Christensen, L. M. Kristensen, and K. H. Mortensen, "Simulation Based Performance Analysis of Web Servers," 9th International Workshop on Petri Nets and Performance Models, 2001.
- [11] J. Dille, R. Friedrich, T. Jin, and J. Rolia, "Web server performance measurement and modeling techniques," Performance Evaluation, vol. 33, pp. 5-26, 1998.
- [12] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, "Web server performance modeling using an M/G/1/K\*PS queue," presented at 10th International Conference on Telecommunications (ICT 2003.), 2003.
- [13] M. Andersson, J. Cao, M. Kihl, and C. Nyberg, "Performance Modeling of an Apache Web Server with Bursty Arrival Traffic," presented at International Conference on Internet Computing (IC), 2003.
- [14] L. Cherkasova and P. Phaal, "Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites," IEEE Trans. Comput., vol. 51, pp. 669--685, 2002.
- [15] A. Kamra, V. Misra, and E. Nahum, "Yaksha: A Controller for Managing the Performance of 3-Tiered Websites," presented at The Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004), Toronto, Canada, 2004.
- [16] R. Nagappan, R. Skoczylas, and R. P. Sriganesh., Developing Java Web Services: Architecting and Developing Secure Web Services Using Java: Wiley, 2003.
- [17] D. K. Barry, Web Services and Service-Oriented Architectures: The Savvy Manager's Guide: Morgan Kaufmann, 2003.
- [18] The Apache Group, "Apache HTTP server project 2.0.7", available at <http://httpd.apache.org/>
- [19] The Apache Jakarta Project, "Tomcat 5.0", available at <http://jakarta.apache.org/tomcat/>.
- [20] MySQL Group, "MySQL," <http://www.mysql.com/>.
- [21] Netcraft, "February 2005 Web Server Survey," available at <http://news.netcraft.com>, 2005.
- [22] Java Comm. Process, "The Java Community Process Program," <http://jcp.org/en/introduction/overview>.
- [23] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, "Quantitative system performance: computer system analysis using queueing network models," Prentice-Hall, Inc., 1984.
- [24] A. Seidmann, P. J. Schweitzer, and S. Shalev-Oren, "Computerized Closed Queueing Network Models of Flexible Manufacturing Systems," Large Scale Systems, vol. 12, pp. 91-107, 1987.
- [25] TPC Council, "TPC-W," <http://www.tpc.org/tpcw>.
- [26] T. Bezenek, H. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, "Characterizing a Java implementation of TPC-W," in 3rd Workshop On Computer Architecture Evaluation Using Commercial Workloads (CAECW), Toulouse, France, January 2000.
- [27] X. Liu, J. Heo, and L. Sha, "Modeling 3-Tiered Web Services," Technical Report, available at <http://www-sal.cs.uiuc.edu/~xueliu/TRModeling.pdf>.