DECISION SCIENCES
A JOURNAL OF THE DECISION SCIENCES INSTITUTE

Decision Sciences
Volume 45 Number 3
June 2014

Capacity Planning and Allocation for Web-Based Applications

Srimathy Mohan†
Department of Supply Chain Management, W.P. Carey School of Business, Arizona State University, PO Box 874706, Tempe, AZ 25857-4706, e-mail: Srimathy@asu.edu

Ferdous M. Alam
Supply Chain Business Intelligence, Nestle USA Inc, 800 North Brand Boulevard (5A24), Glendale, CA 91203, e-mail: Ferdous.Alam@us.nestle.com

John W. Fowler, Mohan Gopalakrishnan, and Antonios Printezis
Department of Supply Chain Management, W.P. Carey School of Business, Arizona State University, PO Box 874706, Tempe, AZ 25857-4706, e-mail: John.Fowler@asu.edu, Mohan@asu.edu, Printezis@asu.edu

ABSTRACT

Motivated by the technology division of a financial services firm, we study the problem of capacity planning and allocation for Web-based applications. The steady growth in Web traffic has affected the quality of service (QoS) as measured by response time (RT), for numerous e-businesses. In addition, the lack of understanding of system interactions and availability of proper planning tools has impeded effective capacity management. Managers typically make decisions to add server capacity on an ad hoc basis when systems reach critical response levels. Very often this turns out to be too late and results in extremely long response times and the system crashes. We present an analytical model to understand system interactions with the goal of making better server capacity decisions based on the results. The model studies the relationships and important interactions between the various components of a Web-based application using a continuous time Markov chain embedded in a queuing network as the basic framework. We use several structured aggregation schemes to appropriately represent a complex system, and demonstrate how the model can be used to quickly predict system performance, which facilitates effective capacity allocation decision making. Using simulation as a benchmark, we show that our model produces results within 5% accuracy at a fraction of the time of simulation, even at high traffic intensities. This knowledge helps managers quickly analyze the performance of the system and better plan server capacity to maintain desirable levels of QoS. We also demonstrate how to utilize a combination of dedicated and shared resources to achieve QoS using fewer servers. [Submitted: October 21, 2011. Revised: February 26, 2013. Accepted: March 13, 2013.]

†Corresponding author.

Subject Areas: Capacity Planning and Allocation, Markov Decision Process, Quality of Service, Queuing Theory, Services, and Web-Based Applications.

INTRODUCTION

In this article, we address the capacity planning and allocation decision that a manager of a Web-based application system has to face on a daily basis. Our study was motivated by the capacity and performance management problem in the technology division of a Fortune 100 financial services firm. A typical Web-based application system consists of Web servers, application servers, various external servers (e.g., databases and mainframe computers), and Web-application logic that controls the flow and sequence of customer requests through all the elements of the system. The performance of the system varies significantly based on the architecture, application logic, and capacity. The application server is central to any Web-based application as it serves as the traffic controller and controls the flow of information between the end-user and across the several tiers in the network.

An important decision that a system manager faces is determining and maintaining the balance between incoming demand for services and application server configuration. Managers have to maintain sufficient server capacity to guarantee a desired quality of service (QoS). QoS is measured by response time (RT) or the time it takes to respond to user requests. As demand increases, QoS deteriorates if additional capacity is not added. Due to the high performance expectations associated with online services, it is important to have the ability to anticipate performance degradation and quickly react to changing conditions in order to ensure QoS. In the real-world application that we studied, managers added capacity on an ad hoc basis without clearly understanding the resulting effect on RT. Also, due to the lack of a clear understanding of the relationship between RT and capacity levels, it was difficult to anticipate the rate of deterioration in QoS as demand increased. As a result, the systems would reach near crash situations very rapidly and capacity additions at that point yielded the desired results very slowly.

In most computers systems, the relationship between server capacity and RT is nonlinear due to underlying system interactions and the effects get magnified under high traffic conditions. Capacity planning and allocation requires a thorough understanding of system interactions and performance in order to meet and maintain satisfactory customer service, and to minimize the operating and capital costs. It is typically feasible to measure the performance of an existing system for a given configuration and traffic load. However, it is quite challenging to predict the performance of new systems in planning stages or even existing systems under highly varying loads. Due to a lack of good analytical approaches to capturing the capacity footprint on the application servers, managers relied on knowledge from past crash situations and tend to overestimate server capacity, thus increasing operating costs. Hence, managers constantly are faced with the conflicting objectives of meeting customer expectations in terms of QoS and keeping IT costs reasonable and under control (Almeida & Menascé, 2002b).

This problem is not unique to the financial services firm that we studied, but is prevalent in most Web-based applications. The constantly increasing use of the internet for personal and business activities will only aggravate this problem for managers of Web-based applications. For example, in the United States, online shopping between November 1 and December 31 in 2012 increased by 14% compared to the same time in 2010 and resulted in 42.3 billion dollars in sales (Lipsman, 2013). This directly translates to capacity management issues for managers handling the e-commerce applications for these businesses. Another application that is bound to increase the need for effective server capacity planning and allocation is electronic health records (EHRs). The American Recovery and Reinvestment Act of 2009 (ARRA) and the health care bill passed in 2010 in the United States have provided incentives and mandates for the "meaningful use" of health care information technology (HIT) in improving the quality and cost effectiveness of care. This includes the implementation of EHRs and computerized physician order entry systems as critical building blocks for hospitals to avail themselves of incentives and avoid penalties. This is increasing the growth of electronic traffic in the Web-based HITs, which will impact storage and server capacities in a significant manner. Hence, it is critical for managers to understand system behavior and plan and allocate server capacity based on this understanding.

Application servers are the central element of any Web-based system. Each application server contains a limited number of channels that can be used by incoming requests. While customer requests are processed in multiple stages and routed through the application server, the application server channel serving an arriving request is typically "locked" and is not available for other arriving requests. When all channels are being used, the application server is completely busy and cannot respond to other incoming requests. For example, when a customer accesses online banking services to check an account balance, the request is first routed to an application server through a Web server. The application server collects the username and password information and sends the request to an external security server to authenticate the user. During authentication, the application server channel that handles the request is "locked" and cannot be used for other arriving requests. When authentication approval is received, the application server sends the account balance request to an external database server. When account balance information is received, it is sent back to the user and the application server channel is released. In this example, the first stage processing is completed at the application server when it collects information from the user. The second stage consists of two services completed by the security server and the database server. While the second part of the service is being completed at one or more external servers, the application server channel that handles that request remains locked and cannot be used to serve another customer request. We refer to this as "resource locking." Resource locking is a very important phenomenon that significantly complicates the interactions among servers and has to be considered explicitly during performance analysis and capacity planning.

An increase in electronic traffic leads to higher traffic intensity at the application server as well as the external servers. Higher traffic intensity at the external servers results in application server channels remaining locked longer and eventually leads to servers crashing. As a result, service levels start deteriorating and

potential revenue could be impacted until the problem is fixed. To address this, Bucholtz and Wright (2001) discuss "hot servers" that could provision capacity on the fly. These servers contain a selection of critical applications that are loaded and can be brought online within a very short time. However, Web administrators and managers still need a model or an approach that can predict system performance under various loads so as to help find the "right timing" to bring hot servers online. This research develops an analytical model of a Web-based application system to measure RTs and to develop insights for performance and capacity management. The objective is to develop a model that can predict performance quickly enough that it can be used as a process monitoring tool. The results from the model at frequent intervals can help identify increasing trends in RTs and signal the need to bring additional capacity online. The concept is similar to using a quality control chart to monitor processes and take action when the process tends toward being out of control. Specifically, we answer the following questions:

- How many application servers are necessary to maintain a predetermined service quality given a certain traffic load?
- With an increase in demand, when should additional application servers be installed in order to maintain the same level of service quality?
- What is the impact of resource locking on QoS and required server capacity? How would capacity requirements change if we ignored resource locking during the planning stages?
- Given competing applications, where should hot servers be utilized to maximize QoS?

Answers to these questions will provide managers the information required to make informed decisions about server capacity additions and allocations. We use a continuous time Markov chain (CTMC) embedded in a queuing network to model and analyze the system. The CTMC model captures many of the real-world characteristics while maintaining prudence, resulting in computational efficiency. While a simulation model can be used to answer the same questions, these models take a long time to reach steady state and cannot be used as an aid for close-to-real-time adaptive capacity management. In this study, we make the following contributions.

- We have developed an analytical model of the two-tier architecture of a Web-based application system that includes resource locking directly. The model aids in performance measurement as well as capacity planning and allocation.
- Experimentation shows that our model produces results comparable to simulation at a fraction of the time. Experimentation also helped identify that the application server is the bottleneck tier when compared to the external servers.
- Our analyses indicate that ignoring resource locking leads to heavy overestimation of server requirements. This leads to unused capacity and tied up capital in real-world applications.

- Server capacity and QoS can be effectively managed using quantitative tools that produce quick, reliable results and flexible servers that can be shared across applications.

The rest of the article is organized as follows. First, we present an overview of the related literature and identify the gap in the literature addressed by this research. We next provide a detailed description of our modeling approach. The following section presents results, validates our model, and discusses the managerial implications. The final section concludes with directions for future research.

LITERATURE REVIEW

There are two classes of studies on performance modeling and capacity planning for Web services. The first models the Web server performance at a high level and is useful for identifying performance trade-offs and making higher level server sizing, additions, and allocations. The second class of models captures the low-level details of the Hypertext Transfer Protocol (HTTP) and Transmission Control Protocol/Internet Protocol (TCP/IP) protocols and software components. The second class of studies generally tends to include system interactions to a greater extent than the first class of models. In this section, we summarize some of the relevant research in both classes of models and identify the gaps in literature addressed by our research.

The first class of models (Menascé, 2002; Cao, Anderson, Nyberg, & Kihl, 2003; Liu, Heo, Sha, & Zhu, 2006) describes Web-based applications as a single-tier architecture. The Web/application server is modeled as an M/G/1 queue and all of the downstream processing is combined into the service time of this queue. Almeida and Menascé (2002a) present a general methodology for capacity planning for Web services. The authors highlight the fact that capacity planning techniques rely very heavily on accurate performance prediction. Cao et al. (2003) use an M/G/1 queuing model with processor sharing service discipline to model a Web server. They derive closed-form expressions for average RT, throughput time, and blocking probability, and test their model against an experimental setup. Liu et al. (2006) also use an M/G/1 processor sharing queue to predict performance, and use an online adaptive feedback loop that enforces admission control to ensure QoS.

These single-tier architecture models do not entirely capture the effects of downstream congestion on Web/application server capacity because resource locking is not addressed in this stream of research. However, the use of a general distribution to approximate the cumulative service time and processor sharing service discipline are important contributions. The second class of models that we summarize later, attempts to address the concept of resource locking directly and to address performance modeling and capacity planning at a lower level.

Resource locking is similar to blocking, but is more restrictive. Blocking of an upstream resource occurs when the downstream queue is full. However, a resource can be locked even if the downstream queue is empty. Resource locking occurs when the upstream resource is waiting for a response from a downstream resource. In the case of multiple upstream servers, they all become blocked simultaneously. Onvural (1990), Perros (1994), and Balsamo, Persone, and Onruval (2001)

present works that study queuing networks with blocking and have proposed exact solution techniques for very simple and special cases, and several approximate solution techniques for cyclic queuing networks. Approximate analysis for queuing networks with simultaneous resource possession (entities can hold two resources simultaneously) has been studied by Jacobson and Lazowska (1982) and Freund and Bexfield (1983). In both studies, the second or downstream resource is used only by one class of customers. The models cannot be directly extended to include secondary subsystems that also receive external customers from other sources.

Layered queuing networks (LQN) and stochastic rendezvous networks (SRVN) have been used to model software architecture systems with multiple layers of servers (Woodside, 1989; Woodside, Neilson, Petriu, & Majumdar, 1995; Neilson, Woodside, Petriu, & Majumdar, 1995; Rolia & Sevcik, 1995). Rolia and Sevcik (1995) have used LQN and have developed the method of layers (MOL) to estimate the performance of distributed applications. Omari, Franks, Woodside, and Pan (2005) have developed a solution procedure for LQN with replicated subsystems to model large client–server systems with several identical subsystems. Omari, Franks, Woodside, and Pan (2006) extended this methodology to consider parallel subsystems in the network. However, none of these consider resource locking.

Reeser and Hariharan (2000, 2002) present an analytical model for Web server performance evaluation. They do consider resource locking, but the downstream resources or external servers receive entities only from the upstream resource (application server). Our model allows the downstream resources to process arrivals from the application servers as well as other sources in the network. Urgaonkar, Pacifici, Shenoy, Spreitzer, and Tantawi (2005) have presented a model for multilayer internet services. However, the authors acknowledge not modeling two critical issues that affect performance, namely, multiresource capture at a layer and resources held simultaneously at multiple layers. The latter is the focus of our study that we define as "resource locking." Ramesh and Perros (2000a, 2000b) have presented a model for distributed software systems that considers client–server communication. They consider a mix of synchronous (client gets locked) and asynchronous (client does not get locked) messages when a client communicates with a downstream server. The authors present a method to estimate RTs when there are no asynchronous messages. Then they extend their method to accommodate for the asynchronous messages using a service reduction technique. Comparison with results from equivalent simulation models shows that the method produces good results (average deviation up to 10%) for traffic intensity up to 70%. The authors indicate that the accuracy of their method would decrease considerably at higher traffic intensities. Also, the model presented can only calculate aggregate RTs for synchronous and asynchronous messages. Moreover, the authors consider servers of unit capacity and no external entities, and communication between the servers can occur only if they are located on adjacent layers. Reeser and Hariharan (2000, 2002), 2005, Urgaonkar et al. (2005), and Ramesh and Perros (2000a, 2000b) consider systems with Poisson arrivals, exponential service times, and first-come, first-serve (FCFS) service discipline.

Mohan, Printezis, and Alam (2009) have proposed a Markovian model for a very simple Web-based application. They consider a system with one application server, one external server, and a single service step for arriving entities. Their

model allows locking entities from the application server (referred to as type-1 entities) and nonlocking entities from other external sources (referred to as type-2 entities) to be processed at the external server. They explain why they used a direct modeling approach as opposed to standard queuing models. If the application server was approximated as a G/G/c queuing system, the total service time of entities would be the sum of the service time in the application server, the waiting time in the external server queues, and the service time in the external servers. However, estimating the mean and variance of the waiting time in the external queues is complicated due to resource locking. They also show that because of resource locking the waiting time of type-1 and type-2 entities in the external queues are not identical. Real-world Web-based application systems are significantly more complex, with several application and external servers, and service consisting of several steps and therefore, the Mohan et al. (2009) model cannot be used to evaluate the performance of such systems as is.

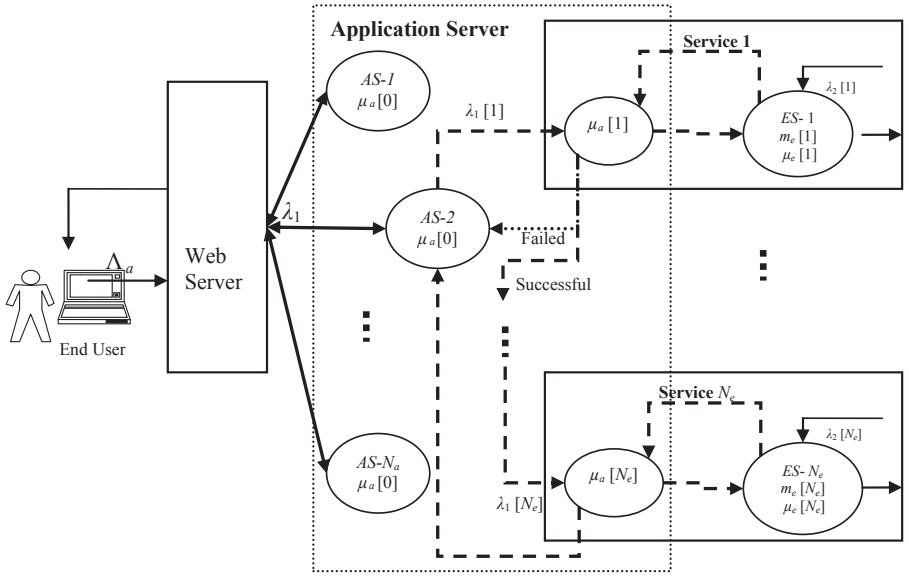
In this study, we use the basic model proposed by Mohan et al. (2009) as a building block for modeling a more complex and realistic Web-based application system that can estimate RT accurately and quickly for a wide range of traffic intensities and system configurations. Our model is similar to the first class of models in that we address a higher level performance prediction and capacity planning and allocation problem. However, we address the important system interactions such as resource locking, similar to the second class of models. Thus, our research bridges the gap between the two streams of research in Web-server capacity planning and management.

MODELING APPROACH

Web-based applications collect user-specific information from multiple sources and display it back to the end-user. When the end-user requests a particular set of information, the Web server routes the request to one of the many application servers in the system. The application server then translates the user request into transactions and determines the number and sequence of service steps necessary to process the request. The routing of each request is dependent on the type of request and also on the processing result of the previous service step. Each service step is performed by a corresponding external server. When an external server completes its processing the request is routed back to the application server. The application server then sends it to the next external server for further processing or back to the user if service is complete. Figure 1 describes the flow of entities through a typical Web-based application system.

The system contains N_a application servers and N_e external servers. Each external server performs a unique service that consists of two parts. The first part is completed at the application server, and the second part at one or more appropriate external servers. The two parts of the service are represented together as a service block in Figure 1. During the second part of the service, the application server channel remains locked. The application server channel that handles an entity remains locked throughout the entity's stay in the system. When all service steps are completed, the locked application server channel is released and the entity leaves the application server. The external servers receive requests from different sources. We define arrivals from the application server to an external server as

Figure 1: Web-based application system and entity flow through the system.



type-1 entities and arrivals from other sources as type-2 entities. The dashed line in Figure 1 shows the path of a type-1 entity. Entities from every application server follow similar paths. If service for a type-1 entity fails at an external server, the entity returns to the application server and further processing may be halted. Next, we describe the notation used to describe our model.

Notation

Subscripts and indices:

a = application server (subscript)

e = external server (subscript)

s = service (index)

Parameters:

$N_{a(e)}$ = *number of application servers (external servers, i.e., unique services)*

Λ_a = *arrival rate of type-1 entities to the system*

λ_1 = *arrival rate of type-1 entities per application server* $= \Lambda_a / N_a$

$SR[s]$ = *success rate of type-1 entities in external server s*

$\lambda_1[s]$ = *arrival rate of type-1 entities to external server* $s = SR[s-1] \times \lambda_1[s-1]$

$\mu_a[0]$ = *service rate of the application server for the initial preprocessing of type-1 entities*

$\mu_a[s]$ = *service rate of the application server for the first part of service s*

m_a = *number of parallel channels in each application server*

$\lambda_2[s]$ = *arrival rate of type-2 entities to external server s*

$\mu_e[s]$ = *service rate of external server s*

$m_e[s]$ = *number of parallel channels in external server s*

$\rho_e[s]$ = *traffic intensity of external server s*

ρ_a = *traffic intensity of each application server*

Performance measures:

$Wq_1[s]$ = *average time in queue for type-1 entities in external server s*

$W_1[s]$ = *average time in system for type-1 entities in external server s*

Wq = *average waiting of time of customer requests in application server queue*

W = *average RT of customer requests*

Model Assumptions

The objective of this article is to develop a model that captures the important characteristics of the underlying real-world application, while providing an efficient methodology to quickly analyze system performance and make capacity decisions. In order to achieve this, we make the following assumptions.

Assumption 1: Arrivals of type-1 entities to the system are from a Poisson process. Arrivals of type-1 entities to each application server are also from a Poisson process.

Most of the existing research on modeling Web-based applications uses Poisson arrivals to the system (Menascé, 2002; Reeser & Hariharan, 2002; Ramesh & Perros, 2000a; Cao et al., 2003; Liu et al., 2006; Mohan et al., 2009). This provides a good approximation while keeping the model simple. Many Web sites that use multiple application servers to support their architecture utilize a load balancer to distribute the load evenly across the servers (Menasce, 2002). Hence, in steady state, we define the arrival rate of type-1 entities to each application server as λ_1 by dividing the total arrival rate (Λ_a) by the number of application servers (N_a). We can then solve this as a system with a single application server. However, type-1 entities from the additional $N_a - 1$ application servers affect the total load on the external servers. To account for this additional load, we combine the type-1 entities from the remaining $N_a - 1$ application servers with the type-2

entities arriving at the external server. So, the modified arrival rate of type-2 entities to the external server is $\lambda_2[s] = \lambda_2[s] + (N_a - 1) \times \lambda_1[s]$.

Assumption 2: Service times at the application and external servers are exponential and service discipline is FCFS.

The second class of research we described earlier assumes exponential service times and FCFS service discipline to keep the analytical models tractable when addressing complicated system interactions such as resource locking. The results from these models show a good representation of real-world performance metrics. A recent research study (Deslauriers, Ecuyer, Pichitlamken, Ingolfsson, & Avramidis, 2007) on telephone call centers with call blending uses an M/M/n queue with FCFS service discipline to predict the performance of the call center. Extensive testing and comparison with real-world data from a call center showed that the difference between the models and exact simulation was less than 1% for important performance measures.

Model of a Simple Web-Based Application System

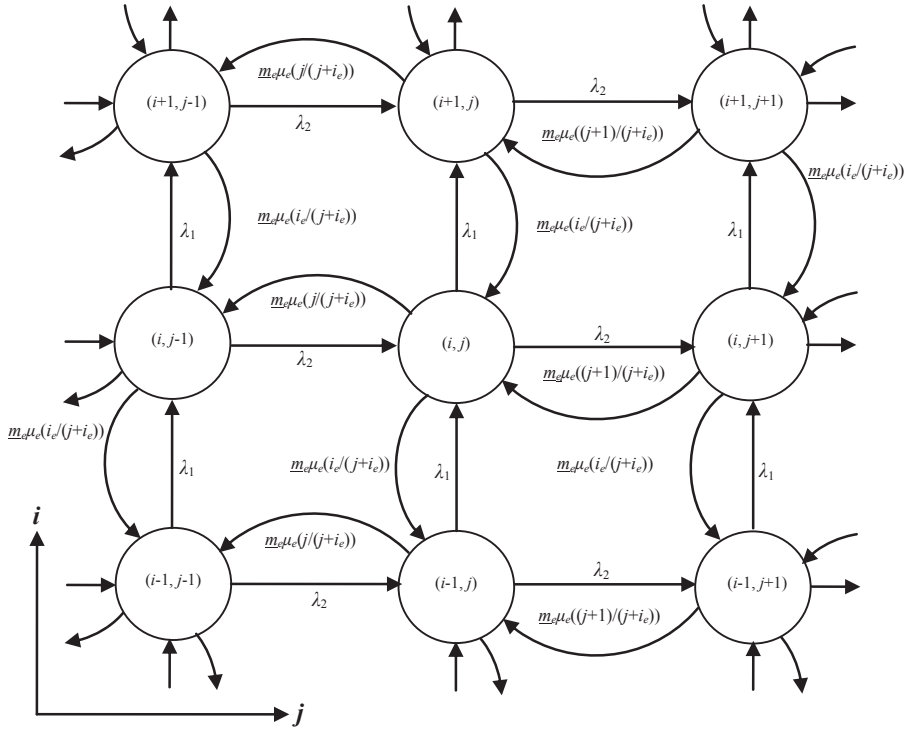
Mohan et al. (2009) present a direct modeling approach to represent a system similar to a single service block in Figure 1. The system has one application server and one external server. The external server serves both type-1 entities and type-2 entities at the same rate. Because we only have one external server in the basic model, the parameters related to the external server do not have the index *s*. The authors first consider a system without service in the application server; in other words, when the application server channel is available, the entities are sent to the external server immediately. Thus, this model describes a service block in Figure 1. The state of the system is defined as (i, j), where i and j are the number of type-1 and type-2 entities in the system. The authors use a two-dimensional CTMC assuming a uniform splitting rule to determine the transition out of a state when a service is completed in the external server. Figure 2 shows a portion of the transition diagram.

The general expression for the steady-state probability of the system being in state (i, j) is

$$p_{i,j} = \frac{1}{\lambda_2 + \lambda_1 + \underline{\mu_e}} \left[\lambda_2 p_{i,j-1} + \lambda_1 p_{i-1,j} + \frac{i+1}{i+j+1} \underline{\mu_e} p_{i+1,j} + \frac{j+1}{i+j+1} \underline{\mu_e} p_{i,j+1} \right], \tag{1}$$

where $\underline{\mu_e} = \text{Min}(m_e, i_e + j)\mu_e$ is the total service rate of the external server and $i_e = \text{Min}(m_a, i)$ is the number of type-1 entities in the external server at state (i, j). The steady-state probabilities of the truncated state space are then estimated numerically by limiting the number of type-1 and type-2 entities in the system. We use modified versions of the two-dimensional CTMC and Equation (1) to represent a service block in Figure 1.

Figure 2: Transition diagram for a system with application server, one external server, and no service in the application server.



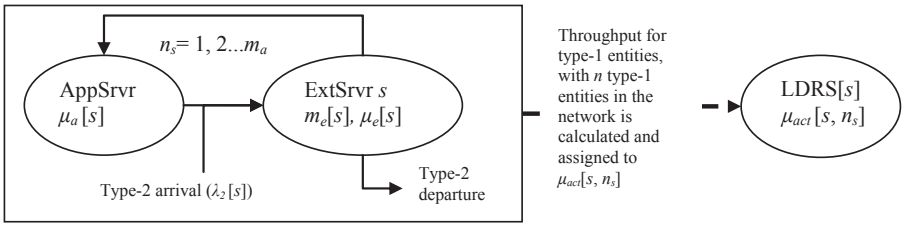
Extension of Model to Include Multiple External Servers

Case 1: At most one visit to each unique external server

In a typical Web-based application, there are different kinds of external servers such as authentication servers, servers to hold cached information, mainframe computers, and servers to customize marketing information. Each type of server performs a unique service. The application server sends the customer request to more than one type of external server, depending on the service requested. Thus, we can represent a type-1 entity's path through the system as a movement through a sequence of service blocks. We first model the case where the entity visits each service block at most once. There are no repetitive visits (i.e., reentrant flows) to service blocks.

We first approximate each service block with a single load-dependent service-rate server (LDRS) by considering it to be a closed network for type-1 entities. Type-2 entities enter and exit external server *s* directly. Figure 3 shows a representative service block. We use the two-dimensional CTMC described earlier to calculate the throughput for type-1 entities, conditional on the number of type-1

Figure 3: Service block approximated by an load-dependent service-rate server (LDRS).



entities in the network. We assign this throughput as the conditional service rate for the LDRS, given that there are $n_s$type-1 entities.

Even though our model includes service within the application server, it is sufficient to use the two-dimensional CTMC because we condition on the number of type-1 entities in the network. We define the state of the system as (i_e, j), where i_e is the number of type-1 entities in the external server and j is the number of type-2 entities in the system. The number of type-1 entities in the application server is then $n_s - i_e$. In state (i_e, j) the total service rates of the application server and external server are, respectively, $\underline{\mu_a} = (n_s - i_e)\mu_a[s]$ and $\underline{\mu_e} = \text{Min}(i_e + j, m_e[s])\mu_e[s]$. The transition diagram of the system is similar to that shown in Figure 2, with $\lambda_1 = \underline{\mu_a}$ and $\lambda_2 = \lambda_2[s]$. The expression for the steady-state probabilities are

$$p_{i_e,j} = \frac{1}{\lambda_2[s] + \underline{\mu_e} + \underline{\mu_a}} \left[\lambda_2[s] p_{i_e,j-1} + \underline{\mu_a} p_{i_e-1,j} + \frac{i_e + 1}{i_e + j + 1} \underline{\mu_e} p_{i_e+1,j} + \frac{j+1}{i_e + j + 1} \underline{\mu_e} p_{i_e,j+1} \right]. \tag{2}$$

We solve for the steady-state probabilities numerically, limiting the number of type-2 entities in the system to $n_e[s]$.

The conditional service rate, given n_s type-1 entities in block s, is then calculated as $\mu_{\text{act}}[s, n_s] = \sum_{i_e=0}^{n_s} p_{i_e,j}(n_s - i_e)\mu_a[s]$. The conditional service rate is calculated for each value of n_s between 0 and m_a (the number of type-1 entities in a service block). This gives us *LDRS*[s] with service rates $\mu_{\text{act}}[s, n_s]$, which approximates the service block s. Because the service rate of *LDRS*[s] cannot be more than the capacity of external server s after serving type-2 entities (i.e., $\mu_{\text{act}}[s, n_s] \leq (\mu_e[s] \times m_e[s] - \lambda_2[s])$), we stop calculating the service rates when we find an $n_s = n$ for which $\mu_{\text{act}}[s, n_s]$ is close to the maximum service rate. For $n_s > n$ we assign this maximum service rate to $\mu_{\text{act}}[s, n_s]$. The procedure is repeated for all service blocks corresponding to each of the N_e services.

Case 2: Repeated visits to external servers

We have so far assumed that an arriving entity requires each service at most once and thus, it needs to visit each external server at most once. In reality however, many entity types require visiting the same server more than once during the required processing. We now allow service requests that require more than one visit to the same external service. We introduce the following additional notation to characterize systems with re-entrant flow (i.e., with repetitive services).

$$N_{st} = \textit{the number of service steps (if there is no service repetition, } N_{st} = N_e\textit{)}$$

$$ROUTE[st, s] = \textit{entity service sequence} = \begin{cases} 1 & \text{if step } st \text{ requires service } s \\ 0 & \text{otherwise} \end{cases}$$

$$SR[st] = \textit{success rate of type-1 entities at service step } st = \sum_{s=1}^{N_e} ROUTE[st, s] \times SR[s]$$

$$\lambda_1[st] = \textit{arrival rate of type-1 entities to service step } st = \lambda_1[st-1] \times SR[st-1]$$

$$VR[s] = \textit{the number of repetitions of service } s = \sum_{st=1}^{N_{st}} ROUTE[st, s]$$

$$SR[s] = \textit{cumulative success rate of type-1 entities in service } s = SR[s]^{VR[s]}$$

$$\lambda_1[s] = \textit{cumulative arrival rate of type-1 entities to service } s = \sum_{st=1}^{N_{st}} ROUTE[st, s] \times \lambda_1[st]$$

We modify the service rates for all LDRS to accommodate service repetitions. We divide the service rates of $LDRS[s]$ by $VR[s]$ to estimate the aggregated service rate for $VR[s]$ visits. The service rates for $LDRS[0]$ are set to $\mu_{\text{act}}[0, n_0] = n_0\mu_a[0]$, where $1/\mu_a[0]$ is the initial processing time of entities in the application server. We then modify these LDRS to replace the service blocks and simplify the representation of the system as shown in Figure 4. Figure 5 provides a detailed description of the LDRS algorithm.

Sequential Aggregation Method (SAM)

Next, we use an SAM to combine all the LDRS nodes sequentially and obtain a single LDRS node that represents the entire network. We use the following additional notation to describe the SAM.

$$ALDRS[s] = \textit{aggregated LDRS node, after aggregating the first } s \textit{ LDRSs}$$

$$\mu_{\text{agg}}[s, n] = \textit{service rate of the aggregated node in step } s \textit{ with } n \textit{ entities in the node}$$

$$ASR[s] = \textit{success rate of entities in the aggregated node in step } s$$

Figure 4: Representation of the system with service blocks replaced by load-dependent service-rate server (LDRS).

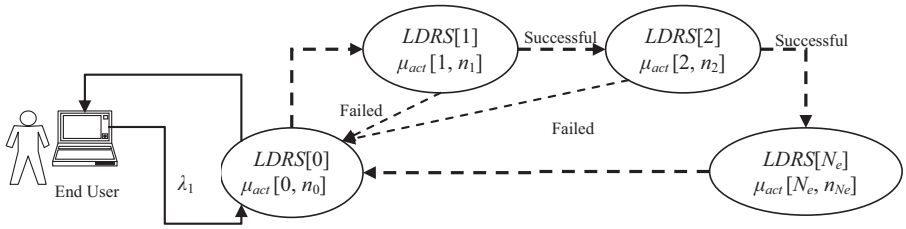


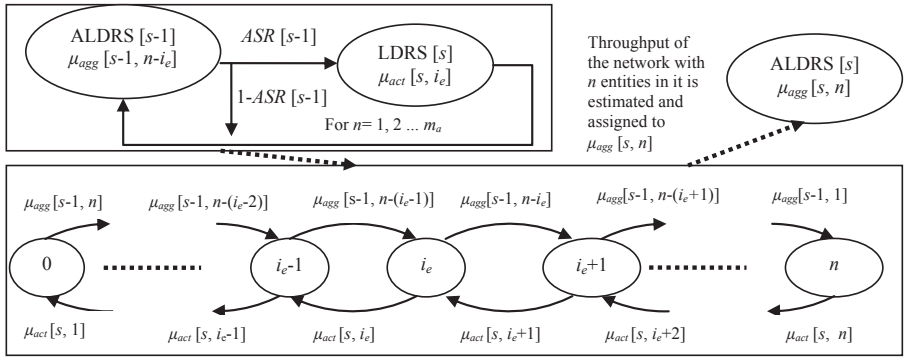
Figure 5: Algorithm load-dependent service-rate server (LDRS).

1. ***Initialization***
   Read $N_a, \Lambda_a, m_a, \mu_a[0], N_e, N_{st}, \mu_a[s], \mu_e[s], m_e[s], \lambda_2[s], n_e[s], SR[s], ROUTE[st, s]$
   Calculate λ_1 and modify $\lambda_2[s]$. Calculate $VR[s]$, $SR[st]$, $\lambda_1[st]$, and modify $SR[s]$, $\lambda_1[s]$
   Set service rates for $LDRS[0]$: $\mu_{act}[0, n_0] = n_0 \mu_a[0]$; for $n_0 = 0$ to m_a, *accuracy* = accuracy level

2. ***Derive all the Load Dependent service Rate Servers***
   For $s = 1$ to N_e, set $\mu_{act}[s,0] = 0$
   For $n_s = 1$ to m_a
   i. If $\mu_{act}[s, n_s - 1] * VR[s] \geq 0.999 * (\mu_e[s] * m_e[s] - \lambda_2[s])$
   set $\mu_{act}[s, n_s] = (\mu_e[s] * m_e[s] - \lambda_2[s])$, Go To step v.
   ii. Initialize $p_{i_e, j} = 1 / N_{tot}$ for $i_e = 0$ to n_s and $j = 0$ to $n_e[s]$
   where, $N_{tot} = (n_s + 1) * (n_e[s] + 1)$ = total number of states.
   Calculate $\mu_{act}[s, n_s] = \sum_{i_e=0}^{n_s} p_{i_e, j} \mu_a[s](n_s - i_e)$; set $LastValue = \mu_{act}[s, n_s]$.
   iii. Update $p_{i_e, j}$ for $i_e = 0$ to n_s and $j = 0$ to n_e s using Equation (2).
   iv. Calculate $\mu_{act}[s, n_s] = \sum_{i_e=0}^{n_s} p_{i_e, j} \mu_a[s](n_s - i_e)$.
   Calculate $Shift = 100 * Abs(LastValue - \mu_{act}[s, n_s]) / LastValue$.
   If $Shift > accuracy$, then go to step iii.
   v. Stop iteration; Calculate $\mu_{act}[s, n_s] = \mu_{act}[s, n_s] / VR[s]$; evaluate next n_s.

In each step of the SAM, we aggregate one LDRS node to the previously aggregated node. We start with *ALDRS*[0]with service rate $\mu_{\text{agg}}[0, n] = n\mu_a[0]$ for $n = 1$ to m_a. We then aggregate *LDRS*[1] with *ALDRS*[0] to derive a new aggregated node *ALDRS*[1] with estimated service rates, $\mu_{\text{agg}}[1, n]$. The process continues until we aggregate all LDRS into a final aggregated node *ALDRS*[N_e] with service rate $\mu_{\text{agg}}[N_e, n]$. Aggregation in step s is performed using the one-dimensional CTMC illustrated in Figure 6.

Figure 6: Step s of sequential aggregation—Estimating service rates of aggregated node.



Consider a closed queuing network that includes the nodes $ALDRS[s-1]$ and $LDRS[s]$. The state of the system is defined as i_e, the number of entities in node $LDRS[s]$, and the expressions for steady-state probabilities are shown in Equations (3) and (4):

$$p_1 = \frac{1}{\mu_{\text{act}}[s,1]}\mu_{\text{agg}}[s-1,n]p_0, \tag{3}$$

$$p_{i_e} = \frac{1}{\mu_{\text{act}}[s,i_e]} \left[\begin{array}{c} (ASR[s-1] \times \mu_{\text{agg}}[s-1, n-i_e+1] + \mu_{\text{act}}[s,i_e-1])p_{i_e-1} \\ +(ASR[s-1] \times \mu_{\text{agg}}[s-1, n-i_e+2])p_{i_e-2} \end{array} \right]$$
$$\text{for } i_e = 2,3,\ldots,n. \tag{4}$$

The value of p_0 is calculated from the normalizing equation $\sum_{i_e=0}^{n} p_{i_e} = 1$. We determine the steady-state probabilities numerically, and calculate the throughput of the network as $\mu_{\text{agg}}[s,n] = \sum_{i_e=0}^{n} p_{i_e} \times \mu_{\text{act}}[s,i_e]$. This is the service rate of the new aggregated node $ALDRS[s]$. We calculate the service rates for $n = 1, 2, \ldots m_a$ and the success rate of entities in the aggregated node as $ASR[s] = ASR[s-1] \times SR[s]$.

By repeating the above procedure N_e times we combine all services into a single $ALDRS[N_e]$ with service rate $\mu_{\text{agg}}[N_e, n]$. We then use the CTMC to model node $ALDRS[N_e]$ with arrival rate λ_1. The state of the system is defined as i, the number of entities in the system and the probability of the system being in state i is defined as p_i. We then solve for the steady-state probabilities and calculate the average performance measures.

Variance Estimation

Initial experimentation indicated that this method produced results very close to the ones obtained using simulation for traffic intensities of up to 70%. For higher traffic intensities, the deviation from simulation results increased significantly. The main reason was an overestimation of the variance of the total service time of type-1 entities in the final aggregated node. When the variance of the service time is overestimated, the average waiting time in queue and the average total time in system are also overestimated. Therefore, in addition to the service rates, we also need to estimate the variance of the service time for the aggregated node at each step. The procedure that we use to estimate the variance in each aggregated node is described later. This method significantly improves the estimation of variance, but we need to note that it does not entirely eliminate the overestimation. The following additional parameters, all related to type-1 entities, are used in the variance estimation procedure:

$ST_a[s]$ = *service time for part of service s in application server; for exponential service time* $E[ST_a[s]] = 1/\mu_a[s]$; $\text{Var}[ST_a[s]] = (1/\mu_a[s])^2$

$ST_e[s]$ = *service time for part of service s in external server; for exponential service time* $E[ST_e[s]] = 1/\mu_e[s]$; $\text{Var}[ST_e[s]] = (1/\mu_e[s])^2$

$RT[s]$ = *total residence time of type-1 entities in service s;* $E[RT[s]] = 1/\mu_a[s] + W_1[s]$

$ST[s]$ = *total service time of type-1 entities up to service s;* $ST[s] = \sum_{i=0}^{s} RT[i]$

$VWq_1[s]$ = *variance of waiting time in queue of type-1 entities in external server s*

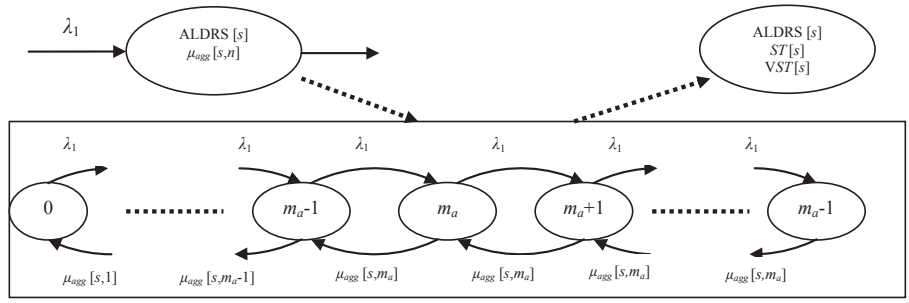
At step s of the SAM, we solve the queuing system with the node $ALDRS[s]$ and arrival rate λ_1 using the one-dimensional CTMC above. The number of entities in the system, i, defines the state of the system. There are $\text{Min}(i, m_a)$ entities in service and the remaining entities wait in queue. We derive the expressions for the steady-state probabilities using Equations (5) and (6) and solve them numerically limiting the number of type-1 entities in the application system to n_a.

$$p_1 = \frac{1}{\mu_{\text{agg}}[s, 1]} \lambda_1 p_0, \tag{5}$$

$$p_i = \frac{1}{\mu_{\text{agg}}[s, \text{Min}(i, m_a)]} \left[(\lambda_1 + \mu_{\text{agg}}[s, \text{Min}(i-1, m_a]) p_{i-1} + \lambda_1 p_{i-2} \right] \quad \text{for } i = 2, 3 \ldots n_a. \tag{6}$$

The value of p_0 is calculated using the normalizing equation $\sum_{i=0}^{n_a} p_i = 1$ and the average total time in service is calculated as $E[ST[s]] = \sum_{i=0}^{n_a} \text{Min}(i, m_a) \times p_i/\lambda_1$.

Figure 7: Step s of sequential aggregation method—Variance estimation.



The total time is the sum of type-1 entities' service time in $ALDRS[s-1]$ and residence time in $LDRS[s]$. Therefore, the average residence time of entities in $LDRS[s]$ is estimated as $E[RT[s]] = (E[ST[s]] - E[ST[s-1]])/VR[s]$. The residence time $RT[s]$ is the sum of the service time inside the application server, waiting time in external server queue, and the service time in the external server. Assuming that the components of $ST[s]$ are independent, the mean and variance of the total service time can be calculated using Equations (7) and (8).

$$E[ST[s]] = E[ST[s-1]] + E[ST_a[s]] + Wq_1[s] + E[ST_e[s]], \quad (7)$$

$$\mathrm{Var}[ST[s]] = \mathrm{Var}[ST[s-1]] + \mathrm{Var}[ST_a[s]] + VWq_1[s] + \mathrm{Var}[ST_e[s]]. \quad (8)$$

The only unknown component in Equation (8) is $VWq_1[s]$. From Gross and Harris (1998), for an M/M/$m_e[s]$ queue with arrival rate $(\lambda_1[s] + \lambda_2[s])$ and service rate $\mu_e[s]$, the waiting time in queue distribution is $Wq_1(t) = 1 - \frac{r^c p_0}{c!(1-\rho)} e^{-(m_e[s]\mu_e[s] - (\lambda_1[s]+\lambda_2[s]))t}$. If we assume that the relationship between the mean and the variance of the waiting time of type-1 entities in the external server queue is similar to the relationship between the mean and the variance of an M/M/c queue, it follows that the variance of the waiting time, $VWq_1[s]$, can be expressed as

$$VWq_1[s] = \frac{2Wq_1[s]}{m_e[s]\mu_e[s] - (\lambda_1[s] + \lambda_2[s])} - (Wq_1[s])^2. \quad (9)$$

Repeating this procedure N_e times we obtain the final aggregated node $ALDRS[N_e]$ with mean $E[ST[N_e]]$ and variance of total service time $\mathrm{Var}[ST[N_e]]$. Figure 7 presents the variance estimation at step s of the SAM. Finally, with the estimated mean and variance of service time, the final aggregated node is modeled as an M/G/m_a queuing system with arrival rate λ_1, expected service time $E[ST[N_e]]$, and variance of service time $\mathrm{Var}[ST[N_e]]$.

Next, we need to evaluate the average waiting time in queue, WQ, and the average time in system, W, for type-1 entities. If WQ_E and WQ_D are, respectively, the waiting time of entities in queue of an M/M/m_a and an M/D/m_a queue with the

Figure 8: Algorithm sequential aggregation method (SAM).

1. ***Initialize*** $ALDRS[0]$ with service rate $\mu_{agg}[0,n] = n\mu_a[0]$ for $n = 1$ to m_a
2. ***Aggregate LDRS nodes***
   For $s = 1$ to N_e, solve the closed network with $ALDRS[s-1]$ and $LDRS[s]$
   (a) Estimate the service rates for the aggregated node. For $n = 1$ to m_a
      i. Calculate p_{i_e} for $i_e = 1$ to n using Equations (3), (4) and the normalizing equation
      ii. Calculate $\mu_{agg}[s,n] = \sum_{i_e=0}^{n} p_{i_e} * \mu_{act}[s,n]$ and $ASR[s] = ASR[s-1] * (SR[s])$
   (b) Estimate the variance of the service time in the aggregated node
      Solve the aggregated node $ALDRS[s]$ with arrival rate λ_1
      i. Calculate p_i for $i = 0$ to n_a using Equations (5), (6) and the normalizing equation $\sum_{i=0}^{n_a} p_i = 1$
      ii. Calculate $E[ST[s]] = \sum_{i=0}^{n_a} Min(i, m_a) * p_i / \lambda_1$
      iii. Calculate $W_{q_1}[s]$, $Var[ST[s]]$, and $VW_{q_1}[s]$ using Equations (7), (8), and (9)
3. ***Solve final aggregated node; determine average system performance measures using Equations (10)–(12)***

same service rate as the M/G/m_a queue, then Boxma, Cohen, and Huffels (1979) show that

$$WQ_D = 0.5WQ_E\left[1 + \frac{(1-\rho)(m_a-1)(\sqrt{4+5m_a}-2)}{16\rho m_a}\right], \tag{10}$$

$$WQ = WQ_D(1-COV) + WQ_E(COV), \tag{11}$$

$$W = WQ + E\,[ST\,[N_e]]\,, \tag{12}$$

where, $\rho = \lambda_1/(m_a\mu)$ is the traffic intensity with $\mu = 1/E\,[ST[N_e]]$, and $COV = \mathrm{Var}\,[ST[N_e]]\,/\,(E\,[ST[N_e]])^2$ is the squared coefficient of variation (COV).

While, from Gross and Harris (1998), WQ_E can be evaluated as $WQ_E = \frac{r^{m_a}}{m_a!(m_a\mu_{\mathrm{act}})(1-\rho)^2}\,p_0$,

with $p_0 = \left(\sum_{n=0}^{m_a-1}\frac{r^n}{n!} + \sum_{n=c}^{\infty}\frac{r^n}{m_a^{n-m_a}m_a!}\right)^{-1}$, $\mu_{\mathrm{act}} = 1/ST[N_e]$, and $r = \lambda_1/\mu_{\mathrm{act}}$.

Figure 8 presents a step-by-step description of the SAM.

RESULTS AND DISCUSSION

We tested the model extensively and compared the results to the results from equivalent simulation models. The use of simulation as a benchmark for evaluating the accuracy of complex queuing models is standard practice. Simulation models can portray the behavior of a real-world system without any modifications or assumptions, while even the most complex analytical models make some assumptions in order to obtain closed-form or approximate solutions. In these cases, a comparison

with simulation helps us to understand the loss of fidelity in the analytical solutions. If the resulting gap from simulation is acceptable, the analytical model tends to be more useful because it can produce results much faster than a simulation model.

We tested our model for a variety of configurations with and without service repetition. The baseline data that we have used for the experimentation were derived from the real-world application that motivated our study. We collected data related to one specific application related to account summary at the financial services firm we were studying. The application-specific data were the types and number of services required for different types of requests arriving at the application, frequently used service routings, types and number of external servers, and service times at the application and external servers. We collected actual data for a period of 3 months and determined the distributions for service times. In cases where the actual service distributions were not Poisson, we assumed them to be Poisson with the same mean, because our analytical models handle only Poisson distributions for arrival and service distributions. The service rates for the application and external servers that we have used in our experimentation (100 per second and 200 per second for the application servers and 100 per second for external servers) are the approximate rates at which the two types of application servers dedicated to the account summary application and the external servers were functioning at during our data collection phase.

The main parameter that influences RTs is the traffic intensity at the application and external servers. Traffic intensity intrinsically captures the capacity of the servers and the demand. Hence, we used a simple experimental design to vary the traffic intensity at the application and external servers. Our data reflected the change in traffic patterns during typical daily operations and we used this as a guide for varying traffic intensities at the servers in our experimentation. We also used the information obtained from our contacts at the firm. Spikes in demand do happen at times, and in order to study the effect of these increases, we allowed the traffic to increase beyond levels observed normally during the data collection phase. Specifically, during the 3-month data collection phase, we observed 10 instances when the system completely crashed due to extremely high traffic intensity. Thirteen percent of the observations occurred when traffic intensity was between 85% and 90%. These were the occasions that significantly affected RTs.

We also present stability analysis results and discuss the computational requirements of our algorithm in comparison to simulation. We define the deviation in RT of our model from the simulation as follows:

$$\%\text{Deviation} = \frac{\text{Model RT - Simulation RT}}{\text{Simulation RT}} \times 100\%.$$

After establishing the validity of our model, we discuss the use of the model to develop managerial insights and to aid capacity planning and management.

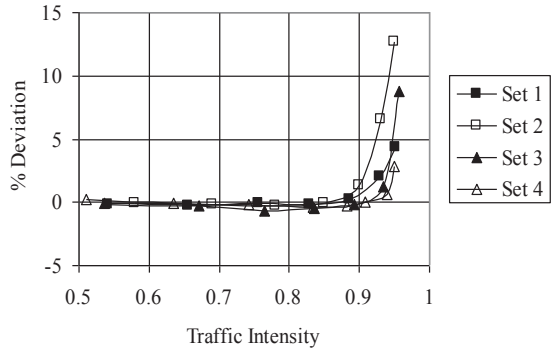
Evaluation of Model for Systems without Service Repetition

We first compared the performance of our model with simulation for applications without service repetition. A customer request does not visit an external server more than once during its processing. We considered four scenarios of different

Table 1: Parameters for evaluation of model for cases with no repetition of services.

						Parameter for Services				
Set	N_a	Λ_a	m_a	$\mu_a[0]$	s	$\mu_a[s]$	$\lambda_2[s]$	$\mu_e[s]$	$m_e[s]$	$\rho_e[s]$
1	1	38, 46, 53, 58, 62, 65, 66.5	5	100	1–3	100	200	100	5	44–53%
2	5	200, 235, 265, 285, 300, 310, 315	5	100	1–3	100	500	100	10	70–82%
3	5	250, 300, 330, 350, 365, 375, 380	10	100	1–6	200	500	100	10	75–88%
4	5	150, 180, 205, 225, 235, 240, 245, 247	10	100	1–10	200	600	100	10	75–85%

Figure 9: Deviation in response time between model and simulation—No repetitive services.



sizes. Problem sets 1 and 2 require three services, and sets 3 and 4 require six and ten services, respectively. In all cases, we adjust the arrival rate of type-1 entities such that the application server traffic intensity varies from 50% to 95%. Table 1 summarizes the various parameters used for the four problem sets. Figure 9 shows the deviation in total RT between our model and simulation.

The results indicate that the deviation of our model from the simulation stays below 5% for traffic intensities up to 95%. The deviation in set 2 is higher than in set 1 due to an increase in the number of application servers. Problem sets 3 and 4, with a larger number of services, have lower deviations than problem sets 1 and 2. As the number of services increases, the squared COV of the aggregated total service time becomes smaller, and as a result, the overestimation of variance (during the sequential aggregation) will have a smaller impact on the overestimation of the total RT. Even with a large number of services, the overestimation of variance and hence the deviation of RT becomes magnified at high traffic intensities. Experience from real-world instances indicates that the nonlinear increase in RT occurs at traffic

Table 2: Parameters for evaluation of model for systems with repetitive services.

			Parameter for Services					
Set	Λ_a	st	s	$\mu_a[s]$	$\lambda_2[s]$	$\mu_e[s]$	$m_e[s]$	$\rho_e[s]$
5	60, 70, 80, 87, 92, 96, 98	1, 10	1	100	500	50	20	62–70%
		2, 8, 12	2	100	500	50	20	68–79%
		3, 9, 13	3	100	500	50	20	68–79%
		4, 7, 14	4	100	500	50	20	68–79%
		5, 11	5	100	500	50	20	62–70%
		6, 15	6	100	500	50	20	62–70%
6	60, 70, 78, 85, 88, 90, 91	1, 9	1	200	300	50	12	70–80%
		2, 7, 13	2	200	250	50	12	72–87%
		3, 8	3	200	350	50	12	78–89%
		4, 10, 14	4	200	300	50	12	80–96%
		5, 12	5	200	300	50	12	70–80%
		6, 11, 15	6	200	250	50	12	72–87%

intensities below 95%. Therefore, the decision to intervene and add capacity or reroute requests has to happen at traffic intensities below 95% where our model consistently produces accurate results. Also, utilizing our model as a quality control tool to monitor for increasing trends in traffic helps managers to adaptively add capacity and prevent the system from reaching extremely high traffic intensities and encountering crashes.

Evaluation of Model for Complex Systems with Service Repetitions

We use problem sets 5 and 6 to evaluate the performance of our model for situations with service repetitions. These sets use the following common parameters: $N_a = 5$, $N_{st} = 15$, $N_e = 6$, $\mu_a[0] = 50$, $m_a = 10$. Both problem sets have six unique external services, but the type-1 entities have 15 service steps that have to be completed. The other system characteristics and parameters are shown in Table 2. For example, in problem set 5, the entities have to visit external server 1 for the first and tenth steps of their service, external server 2 for the second, eighth, and tenth steps, and so on. Figure 10 presents the deviation of our model from simulation. These results indicate that the deviation of our algorithm remains less than 5% for traffic intensities up to 90%.

Stability Analysis

We next evaluate the robustness of our model for different configurations and traffic intensities. We choose set 7 with six service steps and six services (i.e., no service repetition) and set 8 with 15 service steps and six services (i.e., with service repetition). The following parameters are common to both sets: $N_a = 5$, $m_a = 10$, $\mu_a[0] = 50$. The load from type-1 entities is adjusted so that the traffic intensity of application servers varies from 50% to 95%. For each set, we use three

Figure 10: Deviation in response time between model and simulation—Repetitive services.

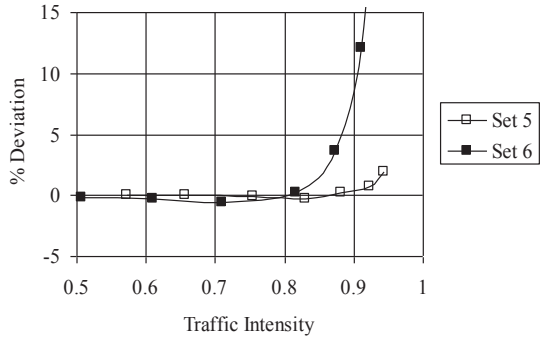


Table 3: Loads and parameters used for stability analysis.

			Parameters for Services				
Set		Λ_a	$\mu_a[s]$	$\lambda_2[s]$	$\mu_e[s]$	$m_e[s]$	$\rho_e[s]$
7	LEL	130, 160, 185, 200, 212, 220, 222	100	150	50	10	56–74%
	MEL	120, 150, 165, 175, 183, 188, 190	100	250	50	10	74–88%
	HEL	100, 108, 116, 120, 123, 125	100	350	50	10	90–95%
8	LEL	60, 70, 80, 87, 93, 98, 100	100	300	50	20	48–66%
	MEL	60, 70, 80, 87, 92, 96, 98	100	500	50	20	68–80%
	HEL	55, 63, 70, 75, 78, 81, 82	100	700	50	20	87–95%

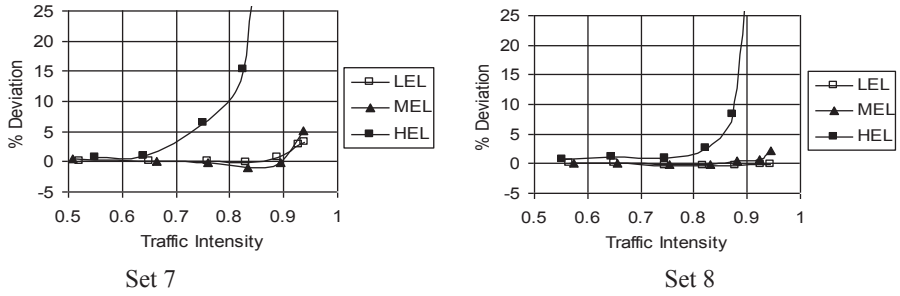
subsets of problems based on the percentage of load on the external servers from type-2 entities. These are low external load (LEL; 30% of external server capacity from type-2 entities), medium external load (MEL; 50%), and high external load (HEL; 70%). Table 3 presents the configurations and the resulting range of traffic intensities on the external servers.

Figure 11 presents the deviation of our model from simulation.

The results indicate that when the load on the external servers is low to medium (LEL and MEL) our model produces very good results even at high application server traffic intensity. The deviation stays below 5% for application server traffic intensities up to 95%. However, when external server load is high (HEL) the deviation increases rapidly at around 80% to 85%.

We have experimented with a wide range of configurations and traffic intensity levels and we have generally found that our model produces good results

Figure 11: Deviation of our model from simulation.



(within 5% error) when external server traffic intensity does not exceed 90% and the application server traffic intensity is at, or below 95%. Also, the most significant parameters that influence the deviation are the traffic intensity of the application and external servers. Our model is not significantly affected by the capacity of the application servers, the number of external servers, the capacity of the external servers, or the number of services service steps and service repetition.

Computational Effort

The accuracy of the results produced by our algorithm depends on the values of n_a and $n_e[s]$ that determine the size of the state space when solving for the steady-state probabilities. The computation time depends heavily on the choice of $n_e[s]$ that is used for the calculation of the service rates for the $LDRS[s]$. If we choose a small value for $n_e[s]$ when solving for the steady-state probabilities of a truncated state space, we lose a large percentage of $\lambda_2[s]$ (arrivals to the external server) and as a result we overestimate the service rates of the $LDRS[s]$. Therefore, to accurately estimate service rates for the $LDRS[s]$ we use values for $n_e[s]$ such that the percentage of lost arrivals, $\lambda_2[s]$, is less than 0.1%. When external server traffic intensity is high, we need to choose a large value for $n_e[s]$, which increases the computation time.

In order to estimate the computation time required by the simulation model, we run the simulation long enough to produce stable results; specifically the simulation is terminated only when the half width of the 90% confidence interval for the average time in system is approximately 2% of the average. We consider four problem sets from Table 3 (Set 7-LEL, Set 7-MEL, Set 8-LEL, and Set 8-MEL) and express the computation time required by our algorithm as a percentage of the time required by simulation. For problem sets with high HEL, the traffic intensity at the external server goes over 100% and the external system becomes unstable, which also affects the application server. We observed results to this effect in Figure 11 as well. For the HEL scenarios, we obtained results using our algorithm. Because we did not obtain stable results using simulation, we have not included the HEL scenarios in the comparison. The results are presented in Figure 12.

Figure 12: Computation time comparison between our algorithm and simulation.

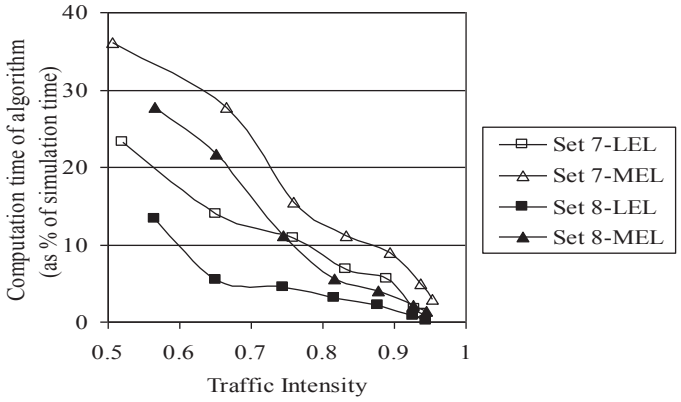


Figure 12 shows that the computation time of our algorithm is consistently lower than simulation throughout the whole range of traffic intensities for all problem sets. At high traffic intensities, our algorithm presents a much faster alternative. In fact, as traffic increases, simulation takes considerably longer to reach steady state due to the increase in variability of RTs. The responses of simulated congested queues are known to be difficult to estimate accurately, because both the mean and the variance of the steady-state output typically become unbounded as the traffic intensity gets closer to saturation. As a result, simulation models have to be run longer to maintain accuracy in estimating waiting times and queue lengths. Hence, the rate of increase in computation time for simulation is much higher than for our algorithm. Specifically, at 90% traffic intensity, our algorithm would require less than 10% of the time required by simulation for all the four problem sets considered here. Even at traffic intensities as low as 50%, our algorithm requires only about 40% of the time taken by simulation. This would be essential in situations where managers want to use performance prediction and monitoring as a process control mechanism to effectively add servers to augment capacity in real time. Thus, our analytical model and solution algorithm provide efficient and reliable alternatives to simulation for performance measurement and capacity planning.

Effect of Resource Locking

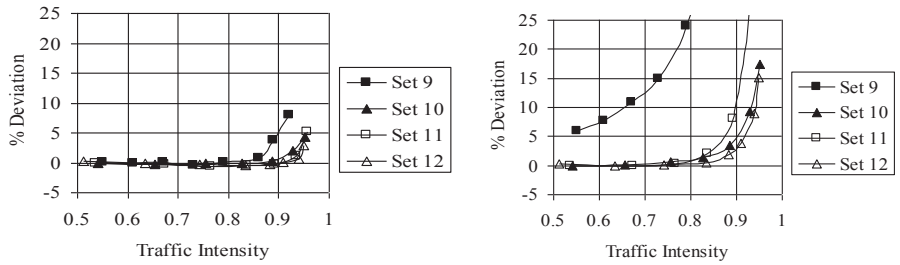
One of the research gaps that we address is the concept of resource locking directly in a higher level analytical model of Web-based applications to aid performance measurement and capacity planning. This section looks at the impact of ignoring resource locking on performance assessment and capacity planning. Although real-world systems utilize resource locking, most existing models ignore it, and this section tries to understand the impact of using such models for capacity planning for the real-world system with resource locking.

We compare the performance of two similar systems, one with and one without resource locking. We ignore resource locking by analyzing each external

Table 4: Parameters for problems—Impact of ignoring resource locking.

Set	N_a	N_e	m_a	m_e	λ_1	$\lambda_2[s]$	$\rho_e[s]$
9	1	2	1	5	9–15	400	82–83%
10	1	3	5	5	38–6.5	200	44–53 %
11	5	6	10	10	250–380	500	75–88 %
12	5	10	10	10	150–247	600	75–85%

Figure 13: Comparison of proposed model and no-locking approximation (NLA) model.



(a) Deviation of our model from simulation

(b) Deviation of NLA model from simulation

server as an M/M/m_e queue and estimate the mean and variance of the total time in the external server system. We then estimate the mean and variance of the first part of the service to determine the total RT for that service. Finally, we add the service times for all services to estimate the mean and variance of the time taken to complete all services. We can now model the application server as an M/G/m_a queue to evaluate the average waiting and average RT within the application server. The queue discipline is first-in, first-out in both models. However, when we ignore resource locking, entities that return to the application server after processing do not have a dedicated channel and hence, would have to wait with all other requests. We refer to this model as the no-locking approximation (NLA) model and compare the performance estimated by this model with our proposed model. We use RT as the comparison criterion and calculate the deviation of both the NLA model and our model from the simulation model. We measure RTs as the duration between the start of service at the application server and the completion of all external services and return to customer. We consider four different problem sets (sets 9, 10, 11, and 12) with different system configurations. The common parameters used for the problem sets are $\mu_a[0] = 100$, $\mu_a[s] = 100$, and $\mu_e[s] = 100$. Table 4 presents other parameters and the varying arrival rates and traffic intensities. Figures 13(a) and (b) present the deviation of our model and the NLA model from simulation.

The results in Figure 13 indicate that for the NLA model the RT is much higher than our model. The results seem to be counterintuitive at first sight, but do reflect the underlying situations realistically. When the model considers resource locking, each arriving customer request gets a dedicated application server channel,

which is a true reflection of the real system. When the entity returns from an external server after processing, the application server is ready to process it with the dedicated server channel. Hence, the RT is a function of only the delays and processing times at the various external servers. On the other hand, when we choose to ignore resource locking, there are no dedicated application server channels. So, when the entities come back to the application server after processing at the external servers, they do not get a higher priority and are queued with several other arriving entities. Now, the RT is a function of processing times and delays at external servers, as well as delays and waits at the application server. At higher traffic intensities, the problem gets exaggerated more and RTs are very large.

In fact, Figure 13 shows that the error of the NLA model is more than double the error of our model at higher traffic intensities. This shows that it is very critical to consider the effect of resource locking at higher traffic intensities. Choosing to ignore the effects of resource locking in capacity planning will lead to maintaining a lot more servers than necessary to maintain the desired QoS. This would eventually result in a lot of unused capacity and tied up capital.

MANAGERIAL IMPLICATIONS

Managers of Web applications strive to maintain high levels of customer service and satisfaction while minimizing operating expenses through efficient management of resources. They generally try to keep utilization and system congestion within a certain range (often at a predetermined RT) in order to minimize the probability of server failure. Hence it is critical to understand the trade-offs between operating cost, efficient capacity utilization, RT, and reliability, and use this knowledge effectively to deliver high-quality and reliable service at a reasonable cost.

Determining Number of Servers to Maintain QoS

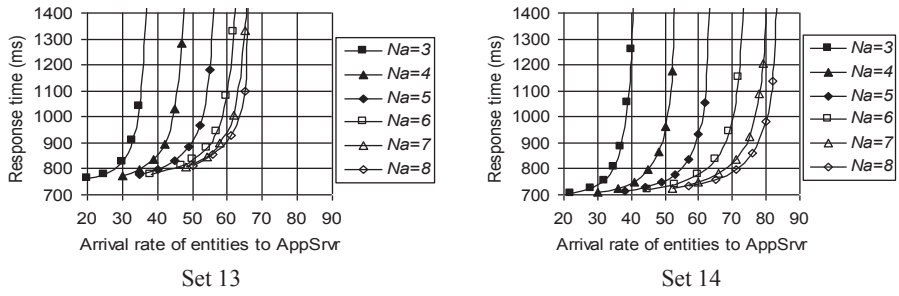
An issue constantly faced by managers is determining the number of application servers needed to maintain a predetermined service quality, given a certain traffic load. In order to understand this issue, we consider a system with six services and 15 service steps ($N_e = 6, N_{st} = 15$) that approximates the complexity of a typical, real-world, Web-based application. Problem sets 13 and 14 (shown in Table 5) are considered. Problem set 14 has 10% higher service rates for its external servers, than set 13. The average RT of the system while varying load and number of application servers, N_a, is presented in Figure 14.

The nonlinear relationship between traffic load and RT is clearly illustrated in Figure 14. It is important to note that resource locking dramatically increases this effect and ignoring it would lead to system failures earlier than expected. Increasing traffic to the application servers also increases the load on the external servers. As a result, requests from the external servers are further delayed, which in turn keeps application server resources locked longer, until finally the system fails. This feedback loop between application and external servers compounds the problem and is the main cause for the highly nonlinear RTs as traffic intensity increases. Due to the shape of the RT function, current network delay alone is not a good performance indicator. Our model can be used to predict the system load for

Table 5: Parameters for problems in set 13 and set 14.

		Route and Parameters for Services						
						$\mu_e[s]$		
$\mu_a[0]$	m_a	st	s	$\mu_a[s]$	$\lambda_e[s]$	Set 13	Set 14	$m_e[s]$
20	10	1, 6, 10, 13	1	100	450	30	33	25
		2, 4, 8	2	100	450	30	33	25
		3, 5, 9, 12	3	100	540	30	33	30
		7, 11	4	100	375	25	27.5	25
		14	5	100	120	20	22	10
		15	6	100	120	20	22	10

Figure 14: Response time of entities with increasing load for different values of N_a.



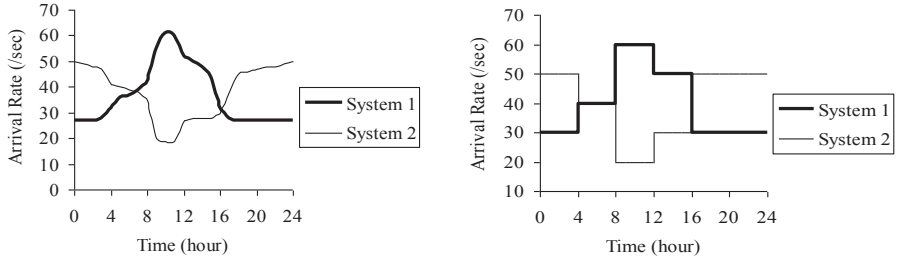
which the nonlinear increase in RT will occur. This could give advance warning about imminent server failures, if no action is taken. With an a priori analysis like this, the system manager has the ability to react and/or plan in advance, and connect additional application servers to the system or reroute the requests, in order to avoid system overloading and failure.

To improve reliability, it is important to assess the traffic intensity the system will be able to handle with the existing capacity. It is also important to estimate the capacity requirement for a certain arrival rate. The model presented can be used for both. For example, for set 14, the results presented in Figure 14 indicate that if the target RT is below 1000 ms and the peak load is 60 arrivals per second, five application servers are required. Figure 14 can also be used to determine the load the system can handle with a given number of application servers.

Determining When to Add Capacity to Maintain QoS

Our model can also be used to estimate when intervention points (e.g., adding servers to increase capacity) are necessary and provide enough time to react. For example, if the target RT is below 1000 ms, then Figure 14 indicates that for set 13, we can serve a load of approximately 33 arrivals per second with three application

Figure 15: Loads for two systems at different hours of a day.



(a) Actual load (b) Target load for capacity planning

Table 6: Service sequence for systems 1 and 2.

ServiceService Step	1	2	3	4	5	6
System 1	1, 10	2, 8, 12	3, 9, 13	4, 7, 14	5, 11	6, 15
System 2	1, 9	2, 7, 13	3, 8	4, 10, 14	5, 12	6, 11, 15

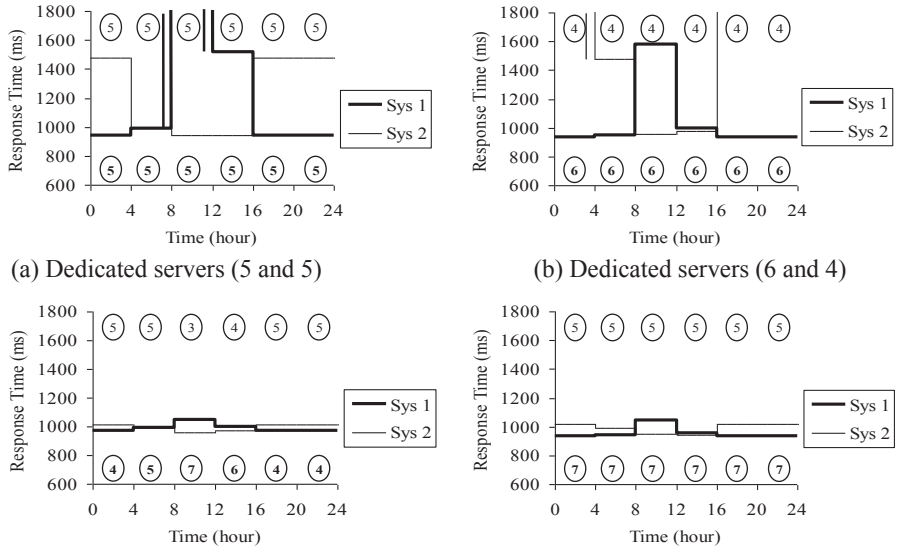
servers. With increasing loads, additional application servers need to be added at arrival rates of approximately 45, 53, 60, 63, 65, etc. Note that the interval between intervention points decreases as traffic increases. This is again due to the sharp nonlinear relationship between the RT and load due to resource locking.

Adaptive Capacity Management Using Hot Servers

One of the uses of our model is to aid managers in adaptive capacity planning and management. This is especially important in companies with multiple applications. Each application has its own customers, service requirements for customers, and load conditions. If these companies have the option of maintaining "hot servers," they can allocate a small number of dedicated application servers for each application and a pool of shared hot servers for the entire system. The hot servers can be brought online depending on the demand on the individual application. We conjecture that the total server requirements will be smaller with this resource allocation system and illustrate this using the example discussed below.

Let us consider a service company, which has two applications with varying load conditions as shown in Figure 15(a). For example, system 1 could be demand for U.S. customers and system 2, demand from overseas customers. The services that customersrequire and their sequences are presented in Table 6. The application server capacity is $m_a = 10$ and the parameters for the services are as follows: $\mu_a[0] = 50$, $\mu_a[s] = 50$, $\mu_e[s] = 25$, $m_e[s] = 40$. Both applications have average daily demand of approximately 40 arrivals per second. However their load peaks occur at different times. Let us assume that, to ensure good QoS, the target is to keep the average RT below 1200 ms for both applications. We want to determine the number of application servers required to maintain the reliability of the system

Figure 16: Adaptive capacity management with hot servers.



at minimum cost. To estimate the capacity needed at different periods of day, we divide a day into six 4-hour periods with approximate load in each period as shown in Figure 15(b).

If we have only 10 application servers and we maintain dedicated servers for each application, we can allocate the servers two ways: (i) allocate five servers to each application, as both have the same average load over the whole day, or (ii) allocate six servers to application 1 and four to application 2, as application 1 has a higher peak load than application 2. Figures 16(a) and (b) present the results for these two cases. The lower and upper edges of the plot area in Figure 16 show the number of servers used, at different periods of time, for applications 1 and 2, respectively. It is clear from the figures that the RT is over 1800 ms for application 1 during some time blocks using the equal allocation, and for application 2 using the unequal allocation. On the other hand, if we allocate four and three dedicated servers for applications 1 and 2, respectively, and three hot servers that can be brought into service as needed, we can meet the target RT with 10 application servers. Figure 16(c) presents the results for this case. If we have to meet the target using only dedicated servers, we can see that we need a total of 12 servers from the results in Figure 16(d).

This example clearly demonstrates the advantage of maintaining flexible resources. By adopting this scheme for capacity management, a firm can clearly achieve the desired QoS with a smaller number of servers. However, the loads on the various applications are not always known exactly in advance. Therefore, in order to achieve the advantage of using hot servers, firms need the capability to

evaluate the performance of their applications and react quickly. Our modeling and solution methodology addresses this need and can provide very reliable results relatively quickly.

CONCLUSIONS

In this article, we have presented a Markovian model for performance evaluation and adaptive capacity planning of Web-based computer application systems. Specifically, our model addresses the concept of resource locking directly. We use RT as the performance metric and measure the deviation of our model from equivalent simulation models. Our model produces excellent results and the deviation from simulation stays below 5% when external server traffic intensity does not exceed 90% and the application server traffic intensity is at or below 95%. These conditions reflect real-world Web application scenarios very well and hence the value and applicability of our model. Another advantage of our model is that it produces results much faster than simulation, especially at high traffic intensity levels. This is particularly important when attempting to quickly assign hot servers to applications that are reaching crash thresholds. We showed how our model can be used to determine how many servers are needed as well as when and where they should be added to maintain target RTs across all applications in a firm.

While the model is quite versatile and useful in answering several managerial questions, there are a few limitations. The proposed model deals with only one class of type-1 entities. In Web-based computer application systems, there can be multiple classes of type-1 entities and the routing can depend on entity classes as well as on the current state. Extending our model to accommodate multiple classes of type-1 entities and state-dependent routing is an appealing extension for further study. Another extension is considering unequal capacities for the various application servers. Finally, arriving entities might be assigned to specific application servers based on certain rules such as least busy server or round robin. While we are currently comparing the proposed adaptive capacity management method to the static fixed capacity, it would be interesting to explore other capacity management options and compare the performance of those methods to the proposed method. We leave these as directions for future research.

REFERENCES

Almeida, V., & Menascé, D. (2002a). Capacity planning: An essential tool for managing Web services. *IEEE IT Professional*, *2*(2), 33–38.

Almeida, V., & Menascé, D. (2002b). *Capacity planning for Web services: Metrics, models and methods*. Upper Saddle River, NJ: Prentice Hall.

Balsamo, S., Persone, V. D. N., & Onvural, R. (2001). *Analysis of queueing networks with blocking*. Boston, MA: Kluwer Academic.

Boxma, O. J., Cohen, J. W., & Huffels, N. (1979). Approximations of mean waiting time in an M/G/s queueing system. *Operations Research*, *27*(6), 1115–1122.

Bucholtz, C., & Wright, R. (2001). 20 tools you need to beat the economy, *CRN Magazine*, accessed January 29, 2014, available at http://www.crn.com/news/channel-programs/18823815/20-tools-you-need-to-beat-the-economy.htm

Cao, J., Andersson, M., Nyberg, C., & Kihl, M. (2003). Web server performance modeling using an M/G/1/K*PS queue. *Proceedings of the 10th International Conference on Telecommunications*, Papeete, Tahiti: IEEE, 1501–1506.

Deslauriers, A., L'Ecuyer, P., Pichitlamken, J., Ingolfsson, A., & Avramidis, A. (2007). Markov chain models of a telephone call center with call blending. *Computers & Operations Research*, *34*(6), 1616–1645.

Freund, D. J., & Bexfield, J. N. (1983). A new aggregation approximation procedure for solving closed queueing networks with simultaneous resource possession. *Journal of the ACM*, *25*(1), 214–223.

Gross, D., & Harris, C. M. (1998). *Fundamentals of queueing theory*. New York, NY: Wiley.

Jacobson, P. A., & Lazowaska, E. D. (1982). Analyzing queueing networks with simultaneous resource possession. *Communications of the ACM*, *25*(2), 142–151.

Lipsman, A. (2013). 2012 U.S. online holiday spending grows 14 percent vs. year ago to $42.3 billion, accessed January 29, 2014, comScore, Inc., available at http://www.comscore.com/Insights/Press_Releases/2013/1/2012_U.S._Online_Holiday_Spending_Grows_14_Percent_vs_Year_Ago_to_42.3_Billion

Liu, X., Heo, J., Sha, L., & Zhu, X. (2006). Adaptive control of multi-tiered Web applications using queueing predictor. *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium*, Vancouver, Canada: IEEE, 106–114.

Menascé, D. (2002). Trade-offs in designing Web clusters. *IEEE Internet computing*, *6*(5), 76–80.

Mohan, S., Printezis, A., & Alam, M. F. (2009). A framework for modeling Web-based applications with resource locking. *International Journal of Operational Research*, *6*(3), 289–303.

Neilson, J. E., Woodside, C. M., Petriu, D. C., & Majumdar, S. (1995). Software bottlenecking in client-server systems and rendezvous networks. *IEEE Transactions on Software Engineering*, *21*(9), 776–782.

Omari, T., Franks, G., Woodside, M., & Pan, A. (2005). Solving layered queueing networks of large client server systems with symmetric replication. *Proceedings of the 5th International Workshop on Software and Performance*, Palma, Illes Balears, Spain: ACM, 159–166.

Omari, T., Franks, G., Woodside, M., & Pan, A. (2006). Efficient performance models for layered server systems with replicated servers and parallel behaviour. *Journal of Systems and Software*, *80*(4), 510–527.

Onvural, R. O. (1990). Survey of closed queueing networks with blocking. *ACM Computing Surveys*, *22*(2), 83–121.

Perros, H. G. (1994). *Queueing networks with blocking*. New York, NY: Oxford University Press.

Ramesh, S., & Perros, H. G. (2000a). A two-level queueing network model with blocking and non-blocking messages. *Annals of Operations Research*, *93*(1), 357–372.

Ramesh, S., & Perros, H. G. (2000b). A multi-layer client-server queueing network model with synchronous and asynchronous messages. *IEEE Transactions on Software Engineering*, *26*(11), 1086–1100.

Reeser, P., & Hariharan, R. (2000). Analytic model of Web servers in distributed environments. *Proceedings of the 2nd International Workshop on Software and Performance*. Ottawa, Ontario, Canada: ACM, 158–167.

Reeser, P., & Hariharan, R. (2002). An analytic model of Web servers in distributed computing environments. *Telecommunication Systems*, *21*(2), 283–299.

Rolia, J. A., & Sevcik, K. C. (1995). The method of layers. *IEEE Transactions on Software Engineering*, *21*(8), 689–700.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., & Tantawi, A. (2005). An analytical model for multi-tier internet services and its applications. *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada: ACM, 291–302.

Woodside, C. M. (1989). Throughput calculation of basic stochastic rendezvous networks. *Performance Evaluation*, *9*(2), 143–160.

Woodside, C. M., Neilson, J. E., Petriu, D. C., & Majumdar, S. (1995). The stochastic rendezvous network model for performance of synchronous multitasking distributed software. *IEEE Transactions on Computers*, *44*(1), 20–34.

Srimathy Mohan is an associate professor in the Supply Chain Management department in the W.P. Carey School of Business at Arizona State University. Her current research interests include patient flow, material flow, and information flow modeling and optimization. Her research has spanned healthcare, manufacturing, and financial services domains. Her research has been published in *Management Science, Journal of Supply Chain Management, European Journal of Operational Research, International Journal of Production Research,* and *International Journal of Production Economics*. Her teaching interests include business analytics and operations management.

Ferdous Alam has been working in the Supply Chain Performance Analytics & Optimization group at Nestle USA, Inc., as a supply chain operations research analyst, where he is engaged in modeling for master production schedule optimization, network optimization, transportation forecasting, and other supply chain–related problems. Prior to joining Nestle, he worked as an operations research analyst at Aviation Logistics Center (ALC), United States Coast Guard (USCG) at Elizabeth City, NC, where he was engaged in demand analysis and forecasting, modeling

and analysis for capacity planning, and resource allocation to support efficient inventory control and supply chain management for aviation spare parts at ALC, USCG. He has a PhD in industrial engineering with major in operations research from Arizona State University, Tempe, AZ. He is very interested in studying and applying operations research techniques in solving real life problems.

John Fowler is the Motorola Professor and Chair of the Supply Chain Management department at Arizona State University (ASU). He is also a professor of industrial engineering and was previously the program chair for IE at ASU. His research interests include discrete event simulation, deterministic scheduling, and multicriteria decision making. He has published over 100 journal articles and over 100 conference papers. He was the Program Chair for the 2008 Industrial Engineering Research Conference, the 2008 Winter Simulation Conference (WSC), and Co-Program Chair for the 2012 INFORMS National meeting. He is currently serving as editor-in-chief for a new Institute of Industrial Engineers journal focused on health care delivery systems entitled *IIE Transactions on Healthcare Systems Engineering*. He is also an editor of the *Journal of Simulation* and an associate editor for *IEEE Transactions on Semiconductor Manufacturing*. He is a Fellow of the Institute of Industrial Engineers (IIE) and currently serves as the IIE Vice President for Continuing Education, is a former INFORMS Vice President, and is an SCS representative on the Winter Simulation Conference Board of Directors.

Mohan Gopalakrishnan is an associate professor in the Supply Chain Management department in the W.P. Carey School of Business at Arizona State University. His current research interests include health care logistics, humanitarian logistics, and global logistics. Specifically, he is interested in studying the structure and impact of the fully integrated supply chain organization in healthcare. His research also covers information flow congestion, implications and capacity management and location, inventory positioning, and flow issues in humanitarian logistics. He primarily teaches logistics and operations management courses across the different undergraduate and graduate platforms and also manages the industry relationships through capstone course in applied projects.

Antonios Printezis is Supply Chain Management clinical faculty at W. P. Carey School of Business at Arizona State University (ASU). He received his doctorate from Case Western Reserve University in operations research and holds a master's degree in chemical engineering. Prior to ASU, he taught courses in the department of Business Statistics, Operations Management, and Technology & Innovation at the Weatherhead School of Management at Case Western Reserve University and at Cleveland State University in the Department of Operations Management and Business Statistics. Prior to joining the academic community he held a quality control engineer position for PepsiCo. His current projects and research interests focus on sustainability and supply chain management. He has been teaching and developing courses for W. P. Carey and the School of Sustainability at ASU on topics including global supply operations, business and sustainability, and operations management