Performance and Reliability Analysis of Web Server Software Architectures*

Swapna S. Gokhale, Paul J. Vandal and Jijun Lu
Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269, USA
Email: {ssg,pjvandal,jijun.lu}@engr.uconn.edu

Abstract

Our increasing reliance on the information and services provided by modern Web servers mandates that these services be offered with superior performance and reliability. The architecture of a Web server has a profound impact on its performance and reliability. One of the dimensions used to characterize the architecture of a Web server is the processing model employed in the server, which describes the type of process or threading model used to support a Web server operation. The main options for a processing model are process-based, thread-based or a hybrid of the process-based and the thread-based models. These options have unique advantages and disadvantages in terms of their performance and reliability tradeoffs. In this paper we propose an analysis methodology based on the Stochastic Reward Net (SRN) modeling paradigm to quantify the performance and the reliability tradeoffs in the process-based and the thread-based Web server software architectures. We demonstrate the capability of the methodology to facilitate systematic, quantitative tradeoffs using several examples.

1 Introduction and motivation

The services offered over the World Wide Web (WWW) have rapidly permeated our lives, primarily due to the comfort, convenience and low costs associated with their use. Although in the past, flexibility, ease of use and reduced costs have been the dominant factors in encouraging the use of these services, their growing prevalence in business and critical domains indicates that these services must be offered with superior performance and reliability to retain the current users and attract new ones. For example, online banking, stock trading, and bill payment services must be fast, efficient and reliable to be widely accepted [1, 20, 29].

An important component of any WWW service is a Web server. A Web server's performance and reliability can be significantly affected by its software architecture [19]. One of the dimensions characterizing the architecture of a Web server is the processing model employed in the Web server. The processing model describes the type of process or threading model used to support a Web server operation. The main options for a processing model are process-based, thread-based or a hybrid of the process-based and the thread-based models. These options have unique advantages and disadvantages in terms of their performance and reliability tradeoffs. A systematic, quantitative evaluation of these tradeoffs within the context of the application domain is necessary prior to deciding the architecture and the configuration parameters of the architecture that should be employed for a given service. In this paper we propose an analysis methodology based on the Stochastic Reward Net (SRN) modeling paradigm to conduct performance and reliability tradeoffs in the process-based and the thread-based Web server software architectures. We demonstrate the capability of the methodology to enable systematic, quantitative performance and reliability tradeoffs with several examples.

The balance of this paper is organized as follows: Section 2 describes modern Web server software architectures, along with a discussion of their advantages and disadvantages. Section 3 provides an overview of the Stochastic Reward Net (SRN) modeling paradigm. Section 4 presents the SRN-based methodology for the performance and the reliability analysis of Web server software architectures. Section 5 illustrates the potential of the methodology to enable performance and reliability tradeoffs using several examples. Section 6 summarizes the related research and places our work in the context of prevalent efforts. Section 7 offers concluding remarks and directions for future research.

2 Web server software architectures

A Web server's performance and reliability can be significantly affected by its software architecture [19]. The two dimensions characterizing the Web server software architecture include the processing model and the pool size be-

*This research is supported in part by a Large Grant from Univ. of Connecticut Research Foundation.

12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)
0-7695-2724-8/06 $20.00 © 2006 IEEE

havior. The processing model determines the type of the process or the threading model used for concurrent Web server operations. The pool size behavior indicates whether and how the number of threads/processes vary as a function of time and workload.

In a process-based architecture, the server consists of multiple single-threaded processes, each of which handles one request at a time. In a thread-based architecture, the server consists of a single multi-threaded process; each thread handles one request at a time. The hybrid model consists of multiple multi-threaded processes, with each thread of any process handling one request at a time. An example of a thread-based Web server is Microsoft IIS server [18], a process-based server is Apache HTTP server 1.3 [17], and a hybrid server is Apache HTTP server 2.0 [17].

The two options for the pool size behavior include, static and dynamic. If the pool size is static, then the Web server creates a limited number of threads/processes a priori to service incoming requests. If all these threads/processes are busy, then an incoming request is queued. In the case of dynamic pool size, the number of processes/threads vary with load. The size of the pool increases with load allowing more requests to be serviced simultaneously, while the pool size is reduced if the load is low. An example of dynamic pool size is Apache HTTP server [17]. For most modern Web server implementations, such as the Apache HTTP server [17] and the Microsoft IIS server [18], the pool size can be made static through simple configurations. In this paper we assume that the pool size is static, leaving the consideration of dynamic pool size for the future.

A primary advantage of the process-based architecture is its stability. The crash or failure of any process generally does not affect the others. The major drawback of the process-based architecture is related to performance: creating and killing processes overloads the Web server, mainly because of address-space management operations. Moreover, high-volume Web sites require many processes, which leads to non-negligible memory requirements and increased context-switching overhead [19] and this may also cause performance degradation. A thread-based architecture is not as stable as a process-based one. A single malfunctioning thread can bring the entire Web server down because all the threads share the same address space. However, the memory requirements of a thread-based architecture are much smaller than the memory requirements of a process-based architecture. Spawning threads of the same process is much more efficient than forking processes because the new threads do not need an additional address space. An additional advantage of the thread-based architecture is that the various threads can easily share data structures such as caches [19]. The hybrid architecture combines the advantages of both methods and reduces their disadvantages. If a thread crashes, it can bring down the process in which it runs, but all the other threads continue to process their requests. Less memory is required in this approach than to run the same number of requests in the process-based architecture [19]. To summarize, qualitatively, the process-based architecture is expected to be more reliable, albeit at the expense of reduced performance, whereas, the thread-based architecture offers better performance but lower reliability. In the subsequent sections we describe a methodology to quantify these tradeoffs.

3 Overview of SRNs

A SRN is a directed graph, which contains two types of nodes: *places* and *transitions*. A directed arc connecting a place (transition) to a transition (place) is called an *input (output) arc*. Arcs have a positive integer number called *multiplicity* associated with them. Places can contain *tokens* that move from one place to another through transitions. A transition is enabled when each of the places connected to it by its input arc has at least the number of tokens equal to the multiplicity of those arcs. When an enabled transition fires, a number of tokens equal to the input arc multiplicity is removed from each of the corresponding input places and a number of tokens equal to the output arc multiplicity is deposited in each of the corresponding output places. A SRN may also include an *inhibitor arc* which can also have a multiplicity associated with it. An inhibitor arc inhibits the transition it is connected to if the place it is connected to at its other end has a number of tokens equal to at least its multiplicity [26]. The state of a SRN with P places is represented by a vector $(m_1, m_2, \cdots, m_p)$ called the *marking* of the SRN, where m_i is the number of tokens in place i. A SRN marking with at least one immediate transition enabled is called a *vanishing marking*, while a marking with no immediate transitions enabled is called a *tangible marking*. A reward rate may be associated with each tangible marking of a SRN. The tangible markings of a SRN and their rates of transition from one marking to another are in fact equivalent to the corresponding states and transitions between the states of an underlying continuous time Markov chain (CTMC) [23]. Hence a SRN can be mapped into an equivalent Markov reward model (MRM) [27]. Software tools such as SPNP [2] can automate the translation of a SRN to its equivalent MRM and solve it. The SRN models thus allow the concise specification of various reward functions. To extend the power of specification, a SRN may also include the specification of *enabling* (or *guard*) *functions* for each transition. The transition is enabled only if the enabling function returns 1.

Stochastic Reward Nets (SRNs) substantially extend the modeling power of Generalized Stochastic Petri Nets (GSPNs) [25], which are an extension of Petri nets [24]. A SRN is a modeling technique that is concise in its spec-

ification and closer to a designer's intuition about what a model should look like. SRNs have been extensively used for performance, reliability and performability analysis of a variety of systems [9, 10, 8, 31, 22].

4 Analysis methodology

In this section we describe the SRN models for performance and reliability analysis of the process- and thread-based Web server software architectures. We also present the relevant performance and reliability metrics and describe how these metrics can be obtained by an appropriate assignment of the reward rates at the net level.

4.1 Process-based architecture

Figure 1 shows the SRN model for the process-based architecture. Place P_S represents the process pool, with the number of active processes serving client requests equal to the number of tokens. The maximum number of tokens in place P_S is s, where s is the size of the process pool. Place P_Q represents the buffer used to hold the incoming client requests if no process in the process pool is available. The maximum number of tokens in place P_Q is q, where q is the size of the buffer. Transition T_A represents the arrival of client requests. If the arrival rate is λ, then transition T_A is expected to fire at rate λ. Transition T_S represents the service of client requests, while transition T_F represents the failure of a request while it is being serviced by the server. Let μ and γ denote the service and the failure rates of a single client request. The firing rates of transitions T_S and T_F depend on the number of active processes serving requests. These firing rates are in Table 1. The maximum rates of transitions T_S and T_F are $s\mu$ and $s\gamma$ respectively. The inhibitor arc from place P_Q to transition T_A prevents its firing when there are q tokens in place P_Q, indicating that the buffer is full and there is no room to hold additional client requests. The inhibitor arc from place P_S to transition $T_{i,1}$ prevents the firing of transition $T_{i,1}$ when there are s tokens in place P_S, indicating that all the processes are busy and the incoming request must be queued.

Table 1. Rate functions (Process- and thread-based architectures)

Transition	Rate function
T_S	$\#(P_S)\mu \ if \ (\#(P_S) < s)$ $s\mu$ otherwise
T_F	$\#(P_S)\gamma \ if \ (\#(P_S) < s)$ $s\gamma$ otherwise

To understand the dynamic evolution of the net, we describe the trajectory of an incoming client request as modeled by the SRN shown in Figure 1. Transition T_A fires if there are less than q tokens in place P_Q and the firing of T_A deposits a token in place P_Q. If the number of tokens in place P_S is less than s, then the immediate transition $T_{i,1}$ fires and the service of the request begins immediately. If the number of tokens in place P_S is s, then the client request is queued. If the request is serviced to completion without failure, then transition T_S fires. On the other hand, if a failure occurs during service then transition T_F fires. Thus, there is a race between transitions T_S and T_F, and the probability of firing these transitions will depend on the actual values of T_S and T_F.

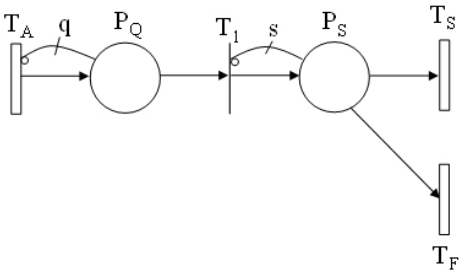

Figure 1. SRN model of process-based architecture

The performance metric of interest is the throughput, which is the rate at which the requests are processed successfully, without failure, by the server. The reliability metrics of interest are the rate at which requests that are accepted by the server are rejected due to failure and the rate at which the incoming requests are rejected. The expected or the average throughput is given by the effective firing rate of transition T_S. The rejection rate of the accepted requests is given by the firing rate of transition T_F. Since the incoming requests are rejected when the buffer is full, the rate at which incoming requests are rejected (not accepted) is given by the probability that the buffer is full times the incoming rate. The reward rates used to obtain the performance and reliability metrics for the process-based architecture are summarized in Table 2.

Table 2. Reward rates (Process-based architecture)

Metric	Reward rate
Throughput	$rate(''T_S'')$
Rejection rate (accepted)	$rate(''T_F'')$
Rejection rate (incoming)	$\lambda \times ((\#P_Q == q)?1 : 0)$

4.2 Thread-based architecture

Figure 2 shows the SRN model of the thread-based architecture. A majority of the places and transitions in this model are the same as in the SRN model of the process-based architecture shown in Figure 1. The following additional places and transitions are added to account for the failure of the server caused by thread failure. Place P_R is added to represent the failure of the server. Transitions $T_{p,1}$ and $T_{p,2}$ are added to purge the outstanding client requests in the queue and the requests currently being serviced when a thread fails. Transition T_R represents the restarting of the server after failure. The firing rates of transitions T_S and T_F are the same as in the case of the process-based architecture and are summarized in Table 1.

The dynamic evolution of the net in this case is similar to the evolution of the net representing the process-based architecture until a failure, which is represented by the firing of transition T_F occurs. Once a thread fails, however, the whole server fails and all the queued and in service client requests are lost. This is modeled as follows. The firing of transition T_F deposits a token in place P_R. The presence of a token in place P_R enables transition $T_{p,1}$, provided there are one or more tokens in place P_Q. Similarly, transition $T_{p,2}$ is enabled by the presence of a token in place P_R and one or more tokens in place P_S. Transitions $T_{p,1}$ and $T_{p,2}$ fire and purge all the tokens in places P_Q and P_S. After all the tokens in places P_S and P_Q are purged, the timed transition T_R fires which represents server restart. ν, the restart rate is the firing rate of transition T_R. The server does not accept incoming client requests when it is in the failed state. The guard functions for transitions $T_{p,1}$, $T_{p,2}$ and T_R in the thread-based architecture are in Table 3.

Table 3. Guard functions (thread-based architecture)

Transition	Guard
$T_{p,1}$	$((\#P_S > 0)\&\&(\#P_R == 1))?1 : 0$
$T_{p,2}$	$((\#P_Q > 0)\&\&(\#P_R == 1))?1 : 0$
T_A	$(\#P_R == 1))?0 : 1$

The performance and reliability metrics of interest in the case of the thread-based architecture are the same as in the case of the process-based architecture. These metrics can be obtained as follows. The throughput is the firing rate of transition T_S. To compute the rejection rate of incoming requests, we note that the incoming requests are rejected when the server is in the failed state, which is in addition to the request rejection that occurs when the queue is full as in the case of the process-based architecture. The rate at which accepted requests are lost due to server failure is given by

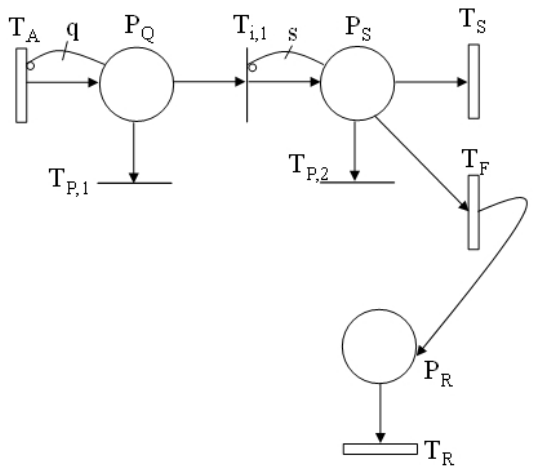

Figure 2. SRN model of thread-based architecture

the average number of requests that are present in the server (queued plus in service) at the time of failure times the failure rate of the server. The effective failure rate of the server is the firing rate of transition T_F, while the average number of requests present in the server is given by the sum of the average number of tokens in places P_Q and P_S. The reward rates to obtain the performance and reliability metrics for the thread-based architecture are summarized in Table 4.

5 Illustrations

In this section we demonstrate the potential of the analysis methodology presented in Section 4 to enable performance and reliability tradeoffs using several examples.

The pool size s and queue size q for both the process and the thread-based architectures are set to 5 and 10 respectively. Typically, in a given server the service rates of the process-based and thread-based architectures are specific to its implementation. Since we are interested in the performance and reliability tradeoffs of these two architectures, we set the service rate of the thread-based architecture to 10/sec., and vary the service rate of the process-based architecture relative to the service rate of the thread-based architecture. We set the service rate of the process-based architecture from 50% to 100% of the service rate of the thread-based architecture in steps of 10%. The arrival rate for the two architectures is varied from 5/sec. to 30/sec. in steps of 5/sec. The failure rate of each request is set to 0.5/sec. This value of the failure rate was chosen so that for the values of the service rate considered the probability that a request may fail is very low. The server reset rate, which is relevant only in the case of the thread-based architecture

0-7695-2724-8/06 $20.00

Table 4. Reward rates (thread-based architecture)

Metric	Reward rate
Throughput	$rate(''T''_S)$
Rejection rate (incoming)	$\lambda \times (((\#P_Q == q)\|\|(\#P_R == 1))?1 : 0)$
Rejection rate (accepted)	$rate(''T''_F) \times (mark(''P''_Q) + mark(''P''_S))$

was set to 100/sec. The reset rate was chosen such that the reset operation is on an average ten times faster than the processing of a service request.

Table 5. Parameters of process and thread-based architectures

Parameter	Thread	Process
Pool size (s)	5	5
Queue size (q)	10	10
Arrival rate (λ)	5-30/sec.	5-30/sec.
Service rate (μ)	10/sec.	5, 6, 7, 8 9, 10/sec.
Failure rate (γ)	0.5/sec.	0.5/sec.
Reset rate (ν)	100/sec.	–

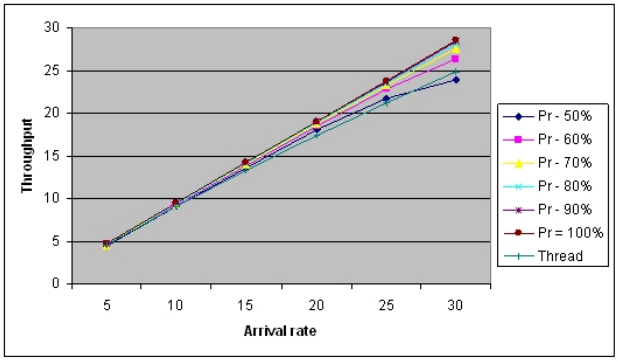

Figure 3. Throughput as a function of request arrival rate

Figure 3 shows the throughput as a function of the request arrival rate, for the thread-based architecture and different service rates of the process-based architecture. Figure 3 indicates that when the arrival rate is low, the throughputs of both the process-based and the thread-based architectures are very similar, even when the service rate of the process-based architecture is half of that of the thread-based architecture. As the arrival rate increases, the process-based architecture provides a slightly higher throughput than the thread-based architecture for all the service rates. This is because the thread-based architecture rejects accepted requests at a higher rate, due to the server failure caused by thread failure and this can be seen in Figure 5. When the arrival rate exceeds 25/sec. the throughput of the process-based architecture with service rate of half of that of the thread-based architecture is slightly lower than the thread-based architecture. Based on the standard results in queuing theory [32], it can be seen that when the arrival rate exceeds 25/sec. and the service rate is 5/sec., the traffic intensity presented to the Web server exceeds 1.0. This results in a sharp increase in the rejection rate of the incoming requests, which leads to lower throughput. This sharp increase in the rejection rate of the incoming requests for the process-based architecture can be observed in Figure 4. Thus, if the service rate of the process-based architecture compared to the thread-based architecture is lower than a certain threshold, the process-based architecture may be simply incapable of handling all the presented load.

The rejection rate of the incoming requests shown in Figure 4 indicates that for service rates of 5/sec. and 6/sec. of the process-based architecture, the rejection rate increases dramatically once the arrival rate exceeds a certain value. As discussed above, this occurs when the traffic intensity presented to the server exceeds 1.0. The rejection rate of the thread-based architecture, on the other hand increases linearly and at a slower place as a function of the arrival rate. This is because in the thread-based architecture, the dominant factor that causes the rejection of incoming requests is the down time of the server caused by thread failure rather than the overloading of the server. In fact, the maximum traffic intensity presented to the thread-based architecture, computed for even the highest value of the arrival rate is only 60%. As a result, even for low arrival rates, the rejection rate of the thread-based architecture is not negligible unlike the process-based architecture which has a near zero rejection rate for smaller arrival rates.

Figure 5 shows the rejection rate of the accepted requests as a function of the request arrival rate for the process-based and the thread-based architectures. The figure indicates that for a single service rate of the process-based architecture, the rejection rate of the accepted requests increases almost linearly as a function of the arrival rate. This is intuitive, since as more requests are processed by the server, the chance of failure increases. The figure also shows that for a given arrival rate, the rejection rate drops as the service rate of the process-based architecture increases. This is because the probability that a request will fail during processing is

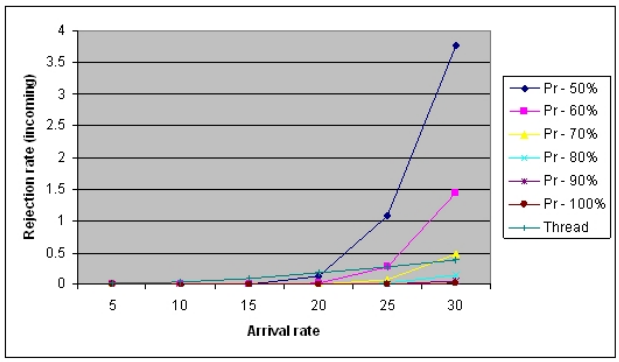

Figure 4. Rejection rate (incoming) as a function of request arrival rate

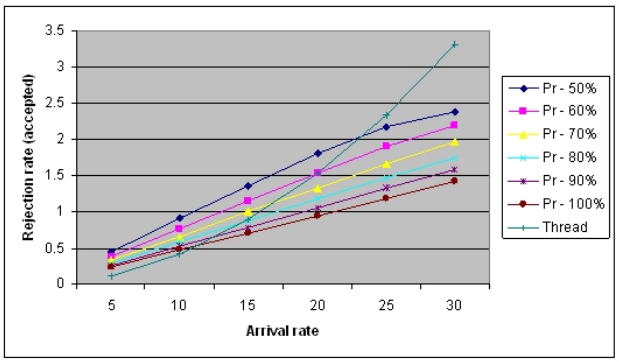

Figure 5. Rejection rate (accepted) as a function of request arrival rate

determined by the time taken to process the request. The higher the service rate, the smaller is the processing time and the lower is the failure probability leading to a lower rejection rate of accepted requests. The rejection rate for the thread-based architecture also increases with the arrival rate, however, the increase is more rapid as compared to the process-based architecture. In the thread-based architecture, an accepted request is rejected not only due to a failure that occurs during its processing but also because of the failure of another thread that brings down the server. As the arrival rate increases, the average number of requests present in the server at the time of failure increases, which contributes to the increased rejection rate. The figure also shows that the rejection rate plot of the thread-based architecture intersects the rejection rate plot of the process-based architecture for a given service rate of the process-based architecture. For arrival rates which are lower than the arrival rate at the intersection point of the two plots, the rejection rate of the thread-based architecture is lower than that of the process-based architecture, whereas, for arrival rates above the arrival rate at the intersection point the rejection rate of the thread-based architecture is higher. Thus, the higher arrival rates which may not be sustainable by the process-based architecture may be possibly handled by the thread-based architecture, albeit with an increase in the rejection rate of the accepted requests.

The above results illustrate the tradeoffs between the process and the thread-based architectures. The specific architectural choice along with the parameters will depend on the requirements of the application domain. For applications with high throughput requirements, but which can tolerate the loss of service requests (incoming and accepted), the thread-based architecture may be an attractive choice. On the other hand, for applications which are intolerant to request loss, but can tolerate lower throughput, the process-based architecture may be better. Further, a given implementation of the process-based architecture may simply be unable to sustain the expected load, in which case, it may be necessary to re-implement the process-based architecture if high reliability is desired.

6 Related research

Performance modeling and analysis of Web servers has been an active area of research. Existing approaches can be broadly classified into two categories: (i) analytic/simulation-based, (ii) measurement-based. Some techniques use a combination of analysis and measurements. We summarize the research in these two categories briefly.

Slothouber [30] proposes to model a Web server as an open queuing network. Heidemann *et al.* [6] present analytical models for the interaction of HTTP with several transport layers. Van der Mei *et al.* [33] present an end-to-end queuing model for the performance of Web servers, encompassing the impact of client workload characteristics, server hardware/software configuration, communication protocols and interconnect topologies. Kamra *et al.* [13] present a control-theoretic approach that both prevents overload and enforces absolute response times. Liu *et al.* [16] provide a model of a three-tiered (Web server, application server and database server) Web services architecture, where each of the three tiers of the Web service architecture is modeled by a multi-station queuing center. Wells *et al.* [34] present a general framework for modeling distributed computing environments for performance analysis by means of Timed Hierarchical Colored Petri Nets. The proposed framework was used to build and analyze a Colored Petri Net model of a Web server. Analysis of the performance of the Web server model reveals how the Web server will respond to changes in the arrival rate of requests, and alternative configurations of the Web server model are examined. Gaeta *et*

al. [3] present an approximate generalized stochastic Petri net (GSPN) model for the analysis of the workload offered to replica servers in a DNS based redirection architecture. Gvozdanovic *et al.* [4] propose a Petri net model that addresses traffic generation patterns of Internet-based real-time block-transfer applications. Scarpa *et al.* [28] propose a methodological approach for the performance analysis of Web-based searching applications on the Internet. They describe how Petri net models can be developed to derive performance indices which can help the designers to improve the efficiency of distributed applications.

Kant *et al.* [14] describe a queuing network model for a multiprocessor system running a static Web workload. The model is based on detailed measurements from a baseline system and a few of its variants. Hu *et al.* [7] measure and analyze the behavior of the Apache Web server driven by the SPECweb96 and the WebStone benchmark. Techniques to improve the performance are also proposed. Hardwick *et al.* [5] use Indy, a new performance modeling framework, to create a performance analysis tool for database-backed Web sites. This tool is validated using the predicted and observed performance of a sample e-commerce site. Kohavi and Parekh [15] offer several recommendations for supplementary analyses which have been found to be very useful in practice. Iyengar *et al.* [11] present several techniques that can be used at popular sites to improve performance and availability based on two case studies.

There are also some efforts which consider availability/dependability analysis of a Web server. Kaaniche *et al.* [12] present and illustrate a hierarchical modeling framework for the availability/dependability evaluation of an Internet-based travel agency. Merzbacher *et al.* [21] first present the results of a series of long-term experiments that measured availability of select Web sites and services. Based on these measurements, they propose a new metric for availability that goes beyond the traditional sole measure of uptime.

There is not much research on performance and reliability analysis of a Web server based on its software architecture. Menascé [19] provides a classification of Web server software architectures and present an approach based on queuing networks to study the pool size behavior. Their research, however, does not distinguish between the different characteristics of the two processing models nor does it explore the inherent tradeoffs which is the goal of the present paper.

7 Conclusions and future research

The software architecture of a Web server has a significant influence on its performance and reliability. In this paper we propose an analysis methodology based on the Stochastic Reward Net (SRN) modeling paradigm to quantify the performance and reliability tradeoffs in the process-based and thread-based Web server software architectures. We illustrate the value of the methodology with several examples.

Our future research includes extending the methodology to: consider: (i) hybrid architecture, (ii) dynamic pool size behavior, and (iii) load-dependent failures.

References

[1] Y. Bakos. The emerging role of electronic marketplaces on the Internet. *Communications of the ACM*, 41(8):35–42, 1998.

[2] G. Ciardo, J. K. Muppala, and K. S. Trivedi. SPNP: Stochastic Petri Net Package. In *Proceedings of 3rd International Workshop on Petri Nets and Performance Models*, pages 142–150, 1989.

[3] R. Gaeta, M. Gribaudo, D. Manini, and M. Sereno. A GSPN model for the analysis of DNS-based redirection in distributed Web systems. In *Proceedings of 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pages 39–48, 2004.

[4] D. Gvozdanovic, D. Simic, U. Vizek, M. Matijasevic, K. P. Valavanis, and D. Huljenic. Petri net based modeling of application layer traffic characteristics. In *EUROCON'01*, pages 424–427, 2001.

[5] J. C. Hardwick, E., Papaefstathiou, and D. Guimbellot. Modeling the performance of e-commerce sites. In *Proceedings of the 27th International Conference of the Computer Measurement Group*, pages 3–12, 2001.

[6] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking*, 5(5):616–630, 1997.

[7] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the Apache Web server. In *IEEE International Performance, Computing and Communications Conference (IPCCC'99)*, pages 261–267, 1999.

[8] O. Ibe, A. Sathaye, R. Howe, and K. S. Trivedi. Stochastic Petri net modeling of VAXCluster availability. In *Proc. of Third International Workshop on Petri Nets and Performance Models*, pages 142–151, 1989.

[9] O. Ibe and K. S. Trivedi. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.

0-7695-2724-8/06 $20.00 © 2006 IEEE

[10] O. Ibe and K. S. Trivedi. Stochastic Petri net analysis of finite-population queueing systems. *Queueing Systems: Theory and Applications*, 8(2):111–128, 1991.

[11] A. Iyengar, J. Challenger, D. Dias, and P. Dantzig. High-performance Web site design techniques. *IEEE Internet Computing*, 4(2):17–26, March 2000.

[12] M. Kaaniche, K. Kanoun, and M. Martinello. A user-perceived availability evaluation of a Web based travel agency. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03)*, pages 709–718, 2003.

[13] A. Kamra, V. Misra, and E. Nahum. Controlling the performance of 3-tiered Web sites: modeling, design and implementation. In *SIGMETRICS 2004/PERFORMANCE 2004*, pages 414–415, 2004.

[14] K. Kant and C. R. M. Sundaram. A server performance model for static Web workloads. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'00)*, pages 201–206, 2000.

[15] R. Kohavi and R. Parekh. Ten supplementary analyses to improve e-commerce Web sites. In *Proceedings of the Fifth WEBKDD workshop (WEBKDD'03)*, pages 29–36, 2003.

[16] X. Liu, J. Heo, and L. Sha. Modeling 3-tiered Web applications. In *13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'05)*, pages 307–310, 2005.

[17] Apache Software Foundation. Apache HTTP server project. `http://httpd.apache.org/`.

[18] Microsoft Corporation. Internet information services (iis). `http://www.microsoft.com/WindowsServer2003/iis/default.mspx`.

[19] D. Menascé. Web server software architecture. *IEEE Internet Computing*, 7(6):78–81, 2003.

[20] D. A. Menascé and V. A. F. Almeida. *Capacity planning for Web services: metrics, models and methods*. Prentice Hall, Upper Saddle River, NJ, 2002.

[21] M. Merzbacher and D. Patterson. Measuring end-user availability on the Web: Practical experience. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN'02)*, pages 473–477, 2002.

[22] J. Muppala, G. Ciardo, and K. S. Trivedi. Stochastic reward nets for reliability prediction. *Communications in Reliability, Maintainability and Serviceability: An International Journal Published by SAE Internationa*, 1(2):9–20, July 1994.

[23] J. R. Norris. *Markov Chains*. University of Cambridge, 1998.

[24] J. L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[25] A. Puliafito, M. Telek, and K. S. Trivedi. The evolution of stochastic Petri nets. In *Proceedings of World Congress on Systems Simulation*, pages 3–15, 1997.

[26] S. Ramani, K. S. Trivedi, and B. Dasarathy. Performance analysis of the CORBA event service using stochastic reward nets. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 238–247, 2000.

[27] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems : An Example-Based Approach Using the SHARPE Software Package*. Springer, 1995.

[28] M. Scarpa, A. Puliafito, M. Villari, and A. Zaia. A modeling technique for the performance analysis of Web searching applications. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1339–1356, November 2004.

[29] S. S. Y. Shim, V. S. Pendyala, M. Sundaram, and J. Z. Gao. Business-to-business e-commerce frameworks. *Computer*, 33(10):40–47, 2000.

[30] L. Slothouber. A model of Web server performance. In *Proceedings of the Fifth International World Wide Web Conference*, 1996.

[31] H. Sun, X. Zang, and K. S. Trivedi. A stochastic reward net model for performance analysis of prioritized DQDB MAN. *Computer Communications*, 22(9):858–870, July 2000.

[32] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley and Sons, 2001.

[33] R. D. van der Mei, R. Hariharan, and P. Reeser. Web server performance modeling. *Telecommunication Systems*, 16(3-4):361–378, 2001.

[34] L. Wells, S. Christensen, L. M. Kristensen, and K. H. Mortensen. Simulation based performance analysis of Web servers. In *Proceedings of 9th International Workshop on Petri Nets and Performance Models*, pages 59–68, 2001.