Availability and Performance Oriented Availability Modeling of Webserver Cache Hierarchies

Geof Pawlicki, Berkeley
Archana Sathaye, San José State University, San Jose

Key Words: Performance Availability Modeling Stochastic Reward Networks CARP Webserver Reverse Proxy

SUMMARY & CONCLUSIONS

Large scale Webserver configurations are widely implemented by Internet portals, and hence have evolved to rely upon numerous performance and high availability system mechanisms. The principle objective of this paper is to provide a method to compare key hierarchical features of webserver configurations with respect to availability. Towards this goal, we model their structure and behavioral characteristics using Stochastic Reward Nets. Several performance and availability measures are also introduced to compare two configurations: (1) a single cluster with multiple servers each featuring a percentage of it's workload cached in main memory and (2) a two cluster site, a minimal star architecture providing benefits of geographic hierarchy. These are differentiated by the varying rates of cache service afforded by in-memory and on-disk caching, and the varying rates of cache service imposed by network latency in a distributed hierarchy. Both rely behaviorally upon the Cache Array Routing Protocol (CARP) to provide parent-child hierachical caching, distributed parallel processing, hash based load balancing and dynamic repair and failover to peer systems. Principally, our results show that using a local cache hierarchy of both in-memory and disk caching significantly offsets the disadvantage of increased network latency occasioned by a geographically centralized site. Specifically, a geographic hierarchy alone is shown to be a much less efficient means of accommodating a heavy "read" workload.

1. *INTRODUCTION*

The performance and availability of webservers - particularly clusters of web proxy servers - is of central importance to the mass acceptance of web-based commerce. A recent study conducted by Carnegie-Mellon University shows that in cases when a user is unable to connect to a site there is a 50% probability that they will never return. Modeling offers one method for the owners of portals and other large websites to predict and provide for such service anomalies. Much of the work to date by Menasce[7] and others has concentrated primarily on performance issues. While of critical importance, these do not reflect availability and the effects of fault tolerance mechanisms. Similarly, pure availability measures do not characterize the queuing behavior and resulting performance degradation. In this paper we combine these perspectives. Measures are provided for availability, performance, and their combination to investigate performance within defined thresholds of acceptable performance. We provide comparative results between parameterized versions of two common webserver configurations. Specifically, we model (1) a single cluster with multiple servers each featuring a percentage of it's workload cached in main memory and (2) a two cluster site, a minimal star architecture, providing the benefit of geographic hierarchy. All clusters are assumed to implement the Cache Array Routing Protocol (CARP) [12,13] in which the namespace of the files served is divided by means of a hash function amongst the local disks of the available proxies.

Such modeling has traditionally been conducted in terms of product form queuing networks, which are generally insufficient to describe systems demonstrating concurrent behavior. While Markov chains are useful for small-scale problems, they are effectively limited by the prohibitive complexity arising from the enumeration of reachable states in larger configurations. Petri Nets[2] provide a higher level formalism, accessible through a graphically intuitive interface, to model logic and time constrained transitions between places representing decision and queuing resources. The state space generated by the reachability graph of a Petri Net can be mapped into a Markov chain model, which in turn is solved by numerical methods[8]. The addition of priorities either explicitly between enabled transitions or by the marking of one place inhibiting transition to another - results in Generalized Stochastic Petri Nets

(GSPN) with computational power equivalent to a Turing machine. In particular, we model the configurations using Stochastic Reward Nets, which allow extensive marking dependenices and reward rate specifications for computing complex measures[2]. Beyond exponentially distributed service and interarrival times, we utilize a Zipf distribution [16] to model to calculate the utilization of files cached in-memory. The models are analyzed using the Stochastic Petri Net Package (SPNP) [10]. Our results principally show the advantage of a local in-memory cache hierarchy to supplement disk caching in a reverse proxy server when compared to geographically distributed mirror sites to offset the network latency.

This paper is organized as follows. Section 2 presents a description of key features of the configurations modeled, along with a discussion of the general modeling techniques utilized relative to the types of measures sought. Section 3 presents the results obtained from analyzing the models. Section 4 summarizes the paper and presents conclusions and future research directions.

2. *MODELING WEBSERVER CLUSTER CONFIGURATIONS*

The generic internet client-server configuration shown in Figure 1 utilizes Proxy servers on both client and server sides. Proxies are common at client sites, e.g. between a router and a corporate intranet. They originated as simple firewalls, characteristically preventing unauthorized access to particular subnetworks by particular users or traffic types. Because of this unique sentry position, Proxy servers were later utilized to cache data obtained from prior HTTP reads.

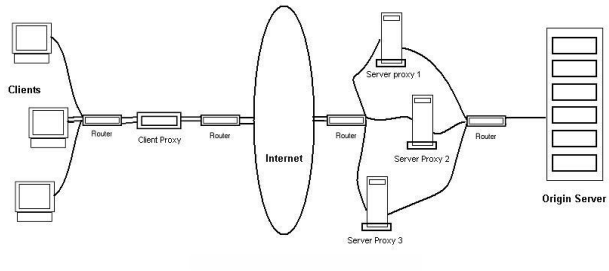

Figure 1: Internet with Client and Server side Proxies

By contrast dynamically generated content - such as server-parsed HTML, CGI scripts, custom-server API applications and specialized servers - never run on proxies. In this paper we refer to Proxy servers simply as servers, and reserve the term Origin Server for the other functions. If the proxy has the requested file, it responds itself in stead of the Origin server it represents, otherwise it forwards the request upstream, toward the origin of the requested file. Whereas a 30-60% cache hit rate can be expected on the client side, the rate for the Proxy server cluster can be 99%+ [6]. This paper focuses on server side cluster configurations, also known as reverse proxy servers. They will be referred to simply as webservers.

2.1 *Stochastic Reward nets (SRN)*

We model these complex clusters using Stochastic Reward nets, which are extensions of Generalized Stochastic Petri Nets (GSPNS). A GSPN is a special case of Stochastic Petri Nets. Formally, a Stochastic Petri net is defined as a 11-tuple[2]:

$$SRN = (P, T, DI, DO, DH, PRIO, EN, F, PS, POL, MO)$$

where: $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,

$T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,

$\forall p_i \in P, \forall t_i \in T, DI_{i,j} : \mathrm{N}^m \rightarrow \mathrm{N}$ is the (possibly) marking dependent multiplicity of the input arc from place p_i to transition t_j; if the multiplicity is zero, the input arc is absent.

$\forall p_i \in P, \forall t_i \in T, DO_{i,j} : \mathrm{N}^m \rightarrow \mathrm{N}$ is the (possibly) marking dependent multiplicity of the output arc from transition t_i to place p_j; if the multiplicity is zero, the output arc is absent.

$\forall p_i \in P, \forall t_i \in T, DH_{i,j} : \mathrm{N}^m \rightarrow \mathrm{N}$ is the (possibly) marking dependent multiplicity of the inhibitor arc from place p_i to transition t_j; if the multiplicity is zero, the inhibitor arc is absent.

$\forall t_i \in T, PRIO_i \in \mathrm{N}$ is the priority of the transition t_i.

$\forall t_i \in T, EN_i : N^m \rightarrow \{0,1\}$ is the marking dependent enabling of the transition t_i.; if no enabling function is defined for t_i, then $EN_i = 1$.

$\forall t_i \in T, F_i : \mathrm{N}^m \rightarrow F$ is the (possibly) marking dependent firing time distribution of the transition t_i (*F* is the family of distributions having a non-negative image).

Let $S \in 2^T, \forall s_j \in T_i, \forall s \in S, PS_{i,j} : \mathrm{N}^m \rightarrow \{0,1\}$ is the marking dependent firing time distribution of the ransition t_j given that the transitions in $s \in S$ are enabled and are scheduled to fire at the same time.

$\forall s \in S, \forall t_i \in T, POL_j : \mathrm{N}^m \rightarrow \{PRI, PRD, PRS\}$ is the marking dependent interruption policy for transition t_j given that the interruption was caused by the simultaneous firing of the set of transitions in s (a less general definition is usually adequate, where the policy is not dependent on the cause of the interruption, or even on the marking).

$M0 \in \mathrm{N}^M$ is the initial marking.

A Stochastic Petri Net is a Stochastic Reward Net (SRN) if $\forall t_j \in T$ either F_j is exponentially distributed with rate of firing $m(t_j) : \mathrm{N}^M \rightarrow \Re^+$ or F_j is deterministic distributed with value 0.

$\forall s \in S, \forall t_i \in T, POL_{s,i} : \mathrm{N}^m \rightarrow \{PRD, PRS\}$ is the marking dependent interruption policy for transition t_j .[2]

As mentioned previously, these readily admit to graphic representation using the conventions shown in Figure 2.

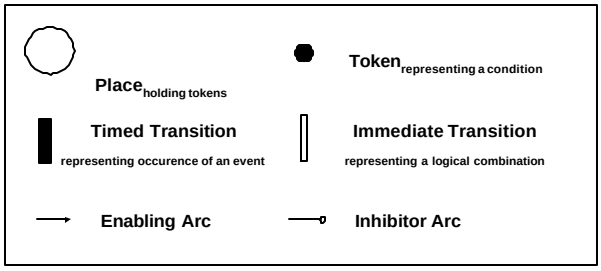

Figure 2: Graphical Notation for Petri Nets

2.2 *Stochastic Reward Nets (SRN)*

Our basic model is given in Figure 3. Jobs arrive from the upper left via the always enabled transition T_{into_entry} and are subsequently dispatched with equal probability to either the service places of either the origin or the mirror site. A job may also be rejected at the cluster due to overflow of the service place. Similarly, rates for transitions out of the service places are determined by reward functions associated with the markings of places representing server status.

The specific marking dependent rates and probabilities associated with each of the transitions are given in Table 1. Each transition name is given only once, reflecting the fact that the mirror site is identical to the origin, excepting as noted for the enabling function of the recache transition. In this case, a mirror recaches from the Parent cache, whereas the Parent must go to the Origin servers at a slower rate.

Transition	Rate or Probability
T_{into_entry}	Lamba = jobs/second
$T_{arrival}$	Immediate transition, enabled if ($\#P_{service}$ < 50 x #servers) and $\#P_{service}$
$T_{overflow}$	Immediate transition, enabled ($\#P_{service}$ < 50 x #servers)
$T_{Lostjob}$	($\#P_{down}$ x disk_service_rate)

$T_{from\_memory\_service}$
 	
Let :
 	
tp[0] =
traffic percentage[0];
 	
p5k[0] = percentage 5k traffic at cluster 0;
 	
nwm[0] = network weighting multiplier[0];
	if (virtual_mirrors)
	{
	rate = (tp[0] * nwm [0]))
	* ((p5k [0] * mem_rate_5k) +
	(p10k [0] * mem_rate_10k) +
	(p38k [0] * mem_rate_38k))
	* percent_memory_cache_hit[0]
	* num_servers_up;
	}
	else
	{
	for each i < mirror_count;
	{
	rate += (tp [i] / nwp [i]))
	* ((p5k [0] * mem_rate_5k) +
	(p10k [0] * mem_rate_10k) +
	(p38k [0] * mem_rate_38k))
	* percent_memory_cache_hit[i]
	* num_servers_up;
	}
	}

$T_{from\_recache}$	DEFAULT_RATE_RECACHE
$T_{failure}$	$\#P_{up}$ x lambda_fault
T_{into_repair}	HEARTBEAT = 30 seconds
T_{reboot}	mu_reboot = 1/hr.= 1.0 / (5 * 60)
$T_{replace}$	mu_replace = 1/hr = 1.0 / (60 * 60)

$T_{from\_disk\_service}$
 	
Let :
 	
tp[0] = traffic percentage[0];
 	
DP = Default_percentage
p5k[0] = percentage 5k traffic at cluster 0;
 	
nwm[0] = network weighting multiplier[0];
	tp[0] *
	(((DP_5K - p5k [0]) > 0 ?
	(DP _5K - p5k [0]) *
	rate_5k : 0) +
	(((DP_10K – p10k [0]) > 0 ?
	(DP _10K – p10k [0]) *
	rate_10k : 0) +
	(((DP_38K – p38k [0]) > 0 ?
	(DP _38K – p38k [0]) *
	rate_38k : 0) +
	(DP_350K +
	(p5k [0] < DP _5K ?
	p5k [0] : DP _5K) +
	(p10k [0] < DP _10K ?
	p10k [0] : DP _10K) +
	(p38k [0] < DP _38K ?
	p38k [0] : DP _38K))
	* rate_350k)

	* (PERCENTAGE_CACHE_HIT) * num_servers_up / (1.0 + (nwm[i] - 1) * 0.5));

Table 1: Transitions and their enabling functions

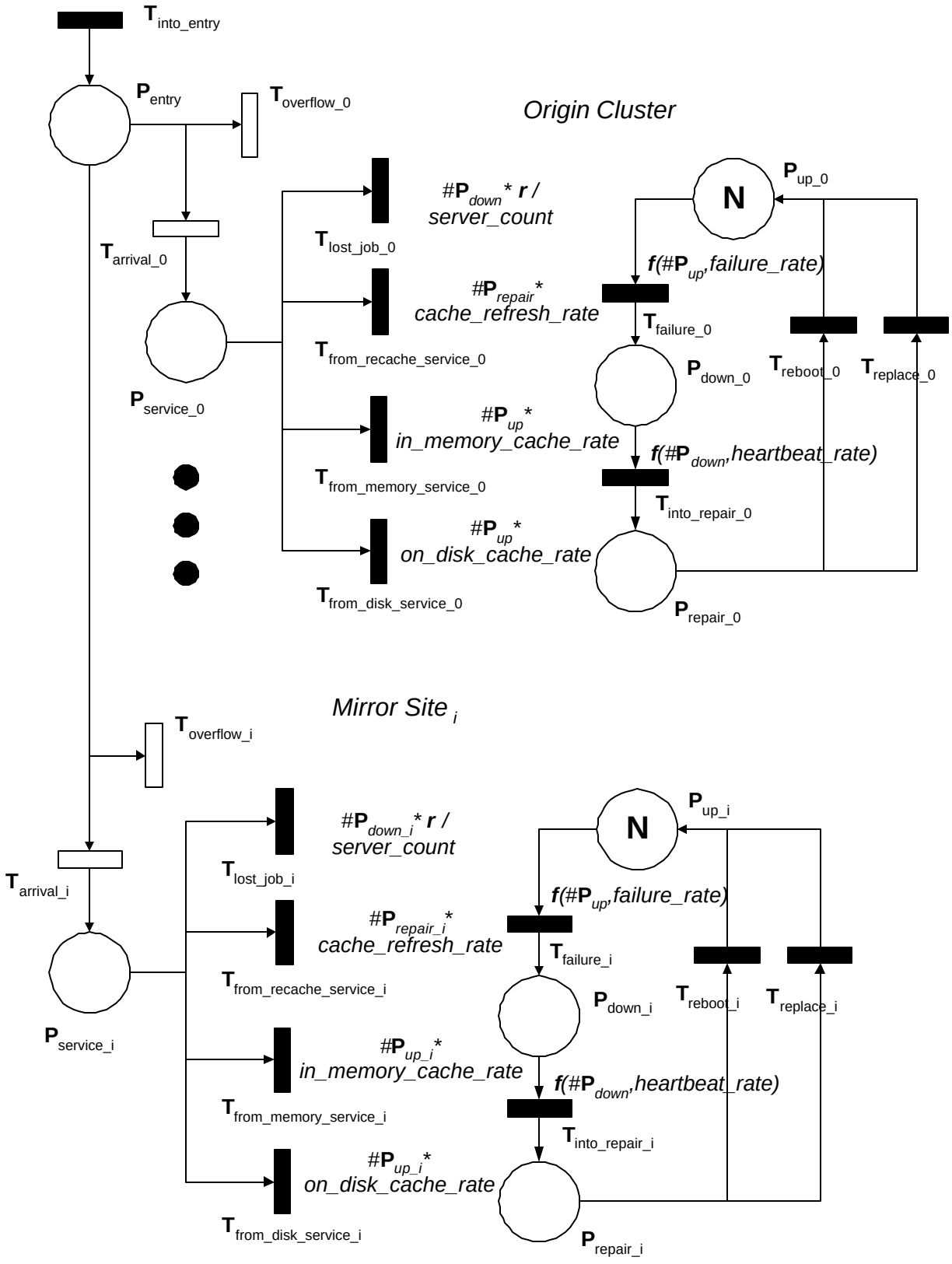

Figure 3. Distributed CARP Website

The features of central importance to the investigation of the effects of cache hierarchy on availability are the disk and memory cache service rates. The disk service rate is determined by a default work load characterized by a percentage of jobs at each of 4 representative sizes – 5, 10, 38 and 350 kbytes- as given in [7] multiplied by an associated per class average rate. From these percentages, the percentage of jobs in service by the in-memory cache is subtracted. The associated in-memory rate is derivative of the disk cache rate less the network latency. Because the memory cache is configured to only apply to the smaller files, there is effectively disk bandwidth available to service a greater number of larger jobs at their correspondingly slower per job rate.

3. ANALYTIC RESULTS

Since the principle objective of this paper is to investigate the effect of the cache hierarchy on the availability of the cluster, other conventional availability measures such as system availability and are the effects of repair [11,17] are de-emphasized. In all there appear 5 series, with datapoints at intervals on the horizontal axis representing memory cache sizes of 0, 1, 2, 4, and 8 Megabytes. The effect of these changes in size is shown on the expected memory throughput rates shown in Figure 4. The maximum rates achieved represent a range of from 44% to 100 % of the cacheable files. For a 1MB cache, the former corresponds to 64 individual file experiencing hit rates ranging from 4.6% to 0.28 % of the workload. This relatively uniform, heavy-tailed scaling, compared to a distribution surrounding a highly popular homepage, can be thought of as representing the service of many small homepages, or the slicing of popular graphics.

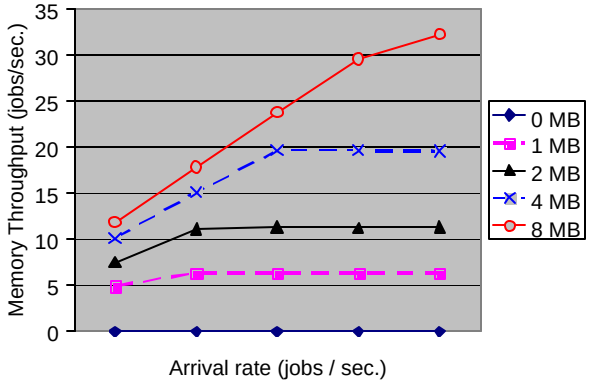

Figure 4: Memory cache throughput – single cluster

The increased throughput translates into the admission of more jobs to service shown in Figure 5.

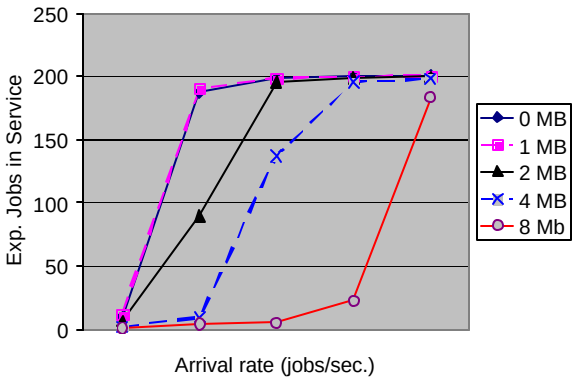

Figure 5 Total number of jobs in service – mirrored site

Clearly the work load is relatively high for the available buffer size since the maximum number of job enqueued is reached for all configurations of memory cache size and mirrored distribution. The net gain in throughput corresponding to memory cache increase is shown in Figures 6 and 7. The elimination of the network weight – that portion of a single server configuration's workload attributable to the mirrored workload of the second configuration, represented by a simple factor of 2 multiplier - produces only a fractional gain in the overall throughput of the mirrored configuration, regardless of the in-memory cache size.

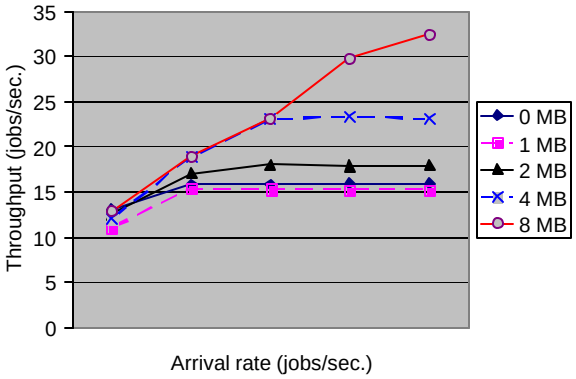

Figure 6: Total Throughput – single cluster

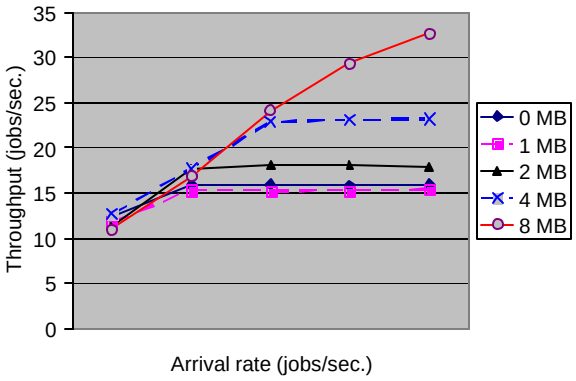

Figure 7: Total Throughput – d-istributed cluster

Fig. 8 shows for the mirrored configuration that the number of jobs admitted is reciprocally related to the number of jobs lost due to buffer overflow.

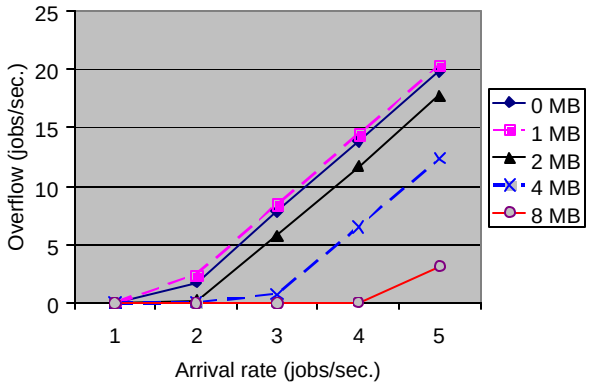

Figure 8: Lost jobs due to buffer over flow

We also compute measures for job loss due to due to excessive delay. Figures 9 show results when parameterized to allow 50 concurrent jobs per server with an arbitrary maximum delay of 10 seconds. The probability of delay is derived from the degenerate case of the general formula for probability of a specific delay with parallel queues and truncation[3]. For each marking

$$reward = \sum_{i=0}^{n} \frac{(mt)^i e^{-mt}}{i!}$$

of the queue, where n is the number of jobs, t is delay threshold and m is the computed expected service rate. The results once again shows that even with tightly controlled buffer counts, there is the possibility of significant job loss in all cases unless there is a suitably scaled acceleration due to in-memory caching.

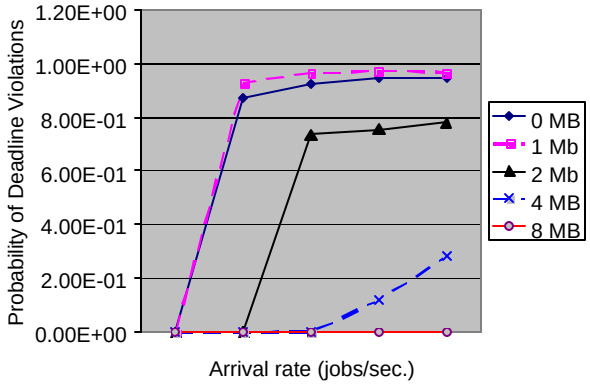

Figure 9: Job Loss due to deadline violation

4. CONCLUSIONS

The present model assumes numerous fundamental limitation to the scale of the configurations modeled – chiefly in terms of the number of jobs in service (tokens allowed per place), but also the number of servers available (a run time limitation) and the allowed number of buffers per server (limited by the computation of a large factorial). It can nonetheless be seen that a workload characterized by HTTP read access benefits more significantly – and less expensively - from an in-memory cache of small, frequently used files than from an additional mirrored cluster to compensate for network latency in servicing remote requests. We are exploring techniques to extend the model to reduce the complexity of the state space explosion by using modular decomposition and Fluid Stochastic Petri Nets[4]. Either will allow an increased number of active concurrent clients. Particularly, we expect that by allowing the replacement of integral marking with fractional values at sink and source places surrounding timed transitions, FSPNs could directly extend the current Discrete Event simulation to a more numerically tractable Continuous event simulation, with significantly reduced run times.

REFERENCES

[1] P. Albiz and C. Liu, *DNS and BIND*, 3rd ed., 1998, O'Reilly & Associates, Inc., Sebastopol, CA.

[2] G. Ciardo, *Analysis of large stochastic Petri Net models: Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University*, 1989.

[3] D. Gross and C. M. Harris, *Fundamentals of Queuing Theory*, (Wiley Series in Probability and Mathematical Statistics), 3rd ed., 1997, John Wiley & Sons.

[4] G. Horton, V.G. Kulkarni, D.M. Nicol and K.S. Trivedi, "Fluid Stochastic Petri nets: Theory, Applications and solution techniques," 1993, source unknown.

[5] O.C. Ibe, K.S. Trivedi and A. Sathaye, "Stochastic Petri net modeling of VAXcluster system availability," *International Conference of Fault Tolerant Systems and Diagnostics*, Varna, Bulgaria, June 20-22 1990. c. Digital Equipment Corporation; Andover MA.

[6] A. Luotonen, *Web Proxy Servers*, 1998, Netscape Communications Corporation, Prentice Hall.

[7] D.A. Menasce and V.A. F. Almeida, *Capacity Planning for Web Performance : Metrics, Models, and Methods*, June 1998, Prentice Hall.

[8] T. Murata, "Petri Nets: Properties, Analysis and Applications," April 1989, *Proceedings of the IEEE*, Vol. 77, no. 4.

[9] Robin Sahner, Kishor Trivedi and Antonio Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example Based Approach using the SHARPE software package*, 1996, Kluwer Academic Publishers, New York, NY.
[10] Kishor Trivedi, *SPNP Version 6.1*, Software, March 2000, Duke University;
Kishor Trivedi, *SPNP User's Manual Version 6.0*, September 1999, Duke University.
[11] Kishor S. Trivedi, Archana S. Sathaye, Oliver C. Ibe and Richard C. Howe, "Should I add a processor?", 1990 *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, IEEE Computer Society Press.
[12] "Cache Array Routing Protocol (CARP) v1.0 Specifications," 1998 Microsoft. [An Internet Draft] http://www.linofee.org/~elkner/da/papers/CarpSpec.htm
[13] "Cache Array Routing Protocol and MS Proxy Server version 2.0," White Paper, http://www.microsoft.com/technet/Proxy/technote/prxcarp.asp, 1997 Microsoft Corporation.
[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", Network Working Group, Request for Comments #2068, Standards Track, January 1997.
[15] L. Fan, P.Cao, J.Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol", 1998, http://www.cs.wisc.edu/~cao/papers/summary-cache/.
[16] "Zipf Curves and Website Popularity," Jakob Nielsen's Alertbox, April 15,1997, http://www.zdnet.com/devhead/alertbox/zipf.html
[17] G. Pawlicki, A. Sathaye, "Performance and Availability Modeling of Webserver Configurations", (with appendixes), *Proceedings*, Vol. 3, The 7th World Multiconference on Systemics, Cybernetics, and Informatics, http://www.pawlicki.net/_content/SCI2003a.pdf.

BIOGRAPHIES

Geof Pawlicki, M.S.C.S, B.M.
2135 Oregon
Berkeley, CA, 94705 USA
Internet (email) geof@pawlicki.net

Geof Pawlicki is a software engineer currently working as a database and website consultant. He holds an MSCS from San Jose State University in 2001 and a Bachelor of Music Composition from the College of Fine Arts at the University Of Illinois Urbana-Champaign. His prior professional engagements have included programming of network management software for real-time high availability optical network elements with Ciena Corp. and BlueLeaf Networks.

Archana Sathaye, Ph.D.
Department of Math and Computer Science
San Jose State University
San Jose CA 95192 USA
Internet (email) sathaye@mathcs.sjsu.edu

Archana Sathaye is on the faculty of the Computer Science department at San Jose State University. Prior to this, she was a consultant to Digital Equipment Corporation where she developed performance models for computer systems under the TPC C workload. Dr. Sathaye also worked as a Principal Engineer with Digital Equipment Corporation, where her primary responsibility was to develop availability, reliability, and performability models for VAXclusters, multiprocessors, and storage systems. She also taught Electrical Engineering at the University of Pittsburgh. Dr. Sathaye has an M.Sc in Mathematics from the University of Bombay, an M.S in Applied Mathematics from Virginia Tech, and her Ph.D. in Electrical & Computer Engineering from Carnegie Mellon University. Her research interests are Petri nets, Fault-tolerance and performance modeling of computer, manufacturing, network, and database systems, Database mining, control of discrete event systems.