How to Parameterize Models with Bursty Workloads *

Giuliano Casale
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
casale@cs.wm.edu

Ningfang Mi
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
ningfang@cs.wm.edu

Ludmila Cherkasova
Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304
lucy.cherkasova@hp.com

Evgenia Smirni
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
esmirni@cs.wm.edu

ABSTRACT

Although recent advances in theory indicate that burstiness in the service time process can be handled effectively by queueing models (e.g., MAP queueing networks [2]), there is a lack of understanding and of practical results on how to perform model parameterization, especially when this parameterization must be derived from limited coarse measurements.

We propose a new parameterization methodology based on the index of dispersion of the service process at a server, which is inferred by observing the number of completions within the concatenated busy periods of that server. The index of dispersion together with other measurements that reflect the "estimated" mean and the 95th percentile of service times are used to derive a MAP process that captures well burstiness of the true service process.

Detailed experimentation on a TPC-W testbed where all measurements are obtained via a commercially available tool, the HP (Mercury) Diagnostics, shows that the proposed technique offers a simple yet powerful solution to the difficult problem of inferring accurate descriptors of the service time process from coarse measurements. Experimental and model prediction results are in excellent agreement and argue strongly for the effectiveness of the proposed methodology under bursty or simply variable workloads.

1. INTRODUCTION

Analytical performance models are fundamental in capacity planning to predict the performance of applications under different workload intensities [8, 11]. For example, Web applications running on multi-tier architectures are evaluated using closed queueing networks where cycling requests represent active user sessions [12]. Closed models are better abstractions of these systems than open models because the closed aspect accounts for the limit on the maximum number of concurrent sessions. A simple parameterization of closed models in terms of mean service demands of the incoming requests is sufficient to predict server utilizations and mean end-to-end response times using Mean Value Analysis (MVA) [10]. MVA models provide intuition on the root causes of performance degradation, however, they may be unable to predict performance *accurately* if the workloads are bursty or highly-variable. Burstiness and high-variability can critically degrade performance to an extent that cannot be captured using mean service times only [9, 2]. Therefore, describing burstiness in performance models of Web systems and finding measurement and parameterization techniques that remain simple and practical are important open challenges.

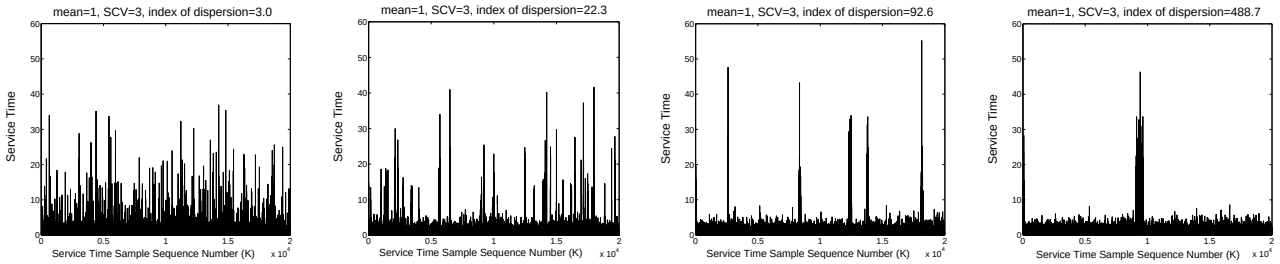
In most cases, measurement data provides the "response time" process, i.e., the service times plus queueing times in a server. Traditional models instead require as input the mean service times only. As response times at low utilization levels imply very little queueing, their mean values are often used as an approximation of mean service times. Yet, it is challenging to extract more accurate descriptors of the service times from such coarse measurements.

Consider a typical situation: we have obtained a trace from a Web server using a non-invasive measurement granularity of a few seconds and we wish to describe service time variability using the collected data. If the measurement granularity is coarse, then the real distribution of service times is different from the one observed in the trace. Under these conditions, only mean service times can be computed reliably, while other descriptors such as service variability or correlations are hidden behind the "measurement granularity wall" and without such indexes it is impossible to characterize effectively burstiness and variability in performance models.

In this paper, we present a new approach to integrate workload burstiness in performance models, which relies on server busy periods (they are immediately obtained from server utilization measurements across time) and measurements of request completions within the busy periods. All measurements are collected with coarse granularity. After giving quantitative examples of the importance of integrating burstiness in performance models, we analyze a real three-tier architecture subject to TPC-W workloads with different burstiness profiles. We show that service burstiness can be inferred effectively from these traces using the *index of dispersion* [6] for counts of completed requests, a measure of burstiness frequently used in the analysis of time series and network traffic. The index of dispersion jointly captures service variability and burstiness in a single number. Furthermore, the index of dispersion can be inferred reliably also if the length of the trace is short. Using the index of dispersion, we show that the accuracy of the model prediction can be increased by up to 30% compared to standard queueing models parameterized only with mean service demands. Exploiting basic properties of bursty processes, we are also able to include in the analysis the 95th percentile of service times, which is widely used in computer performance engineering to quantify the peak-to-mean ratio of service demands. Therefore, our performance models are specified by only three parameters for each server: mean, index of dispersion, and 95th percentile of service demands, making a strong case of being practical, easy, yet surprisingly accurate. To the best of our knowledge, this paper makes a first strong case in the use of a new practical modeling paradigm for capacity planning that encompasses workload burstiness.

The rest of the paper is organized as follows. In Section 2, we introduce burstiness and discuss the measurement of the index of dispersion. In Section 3, we discuss the architecture and workloads used in experiments. The methodology to integrate burstiness in performance models is presented in Section 4 together with experimental results. Finally, Section 5 draws conclusions.

*This work is partially supported by NSF grants CNS-0720699 and CCF-0811417, and a gift from HPLabs. An extended version of this paper titled "Burstiness in Multi-Tier Applications: Symptoms, Causes, and New Models " will appear in the proceedings of Middleware'08.



(a) (b) (c) (d)

Figure 1: Four workload traces drawn from an identical hyper-exponential distribution (mean $\mu^{-1} = 1$, $SCV = 3$), but with different burstiness profiles.

2. BURSTINESS IN PERFORMANCE MODELS: DO WE REALLY NEED IT?

In this section, we show some examples of the importance of burstiness in performance models. In order to gain intuition on the fundamental features of burstiness, let us first consider Figure 1. Each figure represents a sample of $20,000$ service times generated from the same hyperexponential distribution with mean $\mu^{-1} = 1$ and squared coefficient-of-variation $SCV = 3$. The only difference is that we have shaped correlations to impose to each trace a unique burstiness profile. In this way, in Figure 1(b)-(d), the large service times progressively aggregate in bursts, while in Figure 1(a) they appear in random points of the trace. In particular, Figure 1(d) shows the extreme behavior where all large requests are condensed into a single large burst. In the rest of the paper, we use the term "burstiness" to indicate traces that are not just "variable" as the samples in Figure 1(a), but that also present aggregation in "bursty periods" as in Figure 1(b)-(d). What is the performance implication on systems of the different burstiness profiles in Figure 1(a)-(d)? Assuming that the request inter-arrival times to the server follow an exponential distribution with mean $\lambda^{-1} = 2$, a simulation analysis of the $M/G/1$ queue (50% utilization) with correlated service provides the response times shown in Table 1. Irrespectively of the identical properties of the service time distribution, burstiness clearly has paramount importance for queueing prediction, both in terms of mean and tail of response times. For instance, the mean response time for the trace in Figure 1(d) is approximately 40 times slower than with the service times in Figure 1(a) and the 95th percentile of the response times is even 80 times longer. In general, the performance degradation is monotonically increasing with the observed burstiness; therefore it is important to distinguish the different behaviors in Figure 1(a)-(d) with a quantitative index. The index of dispersion discussed in the next section is instrumental to capture the difference in the burstiness profiles.

	Response Time [ms]		*Index of Dispersion*
Workload	mean	95th percentile	I
Figure 1(a)	3.02	14.42	3.0
Figure 1(b)	11.00	83.35	22.3
Figure 1(c)	26.69	252.18	92.6
Figure 1(d)	120.49	1132.40	488.7

Table 1: Response time of the $M/G/1$ queue for the service times traces shown in Figure 1. The server utilization is 50%.

2.1 Characterization of Burstiness

We use the *index of dispersion* I for counts to characterize the burstiness of service times [6]. This is a standard burstiness index used in networking [6], which we here apply to the characterization of workload burstiness in multi-tier applications. To the best of our knowledge, the index of dispersion has not been previously applied to multi-tier application modeling. The index of dispersion has instead a broad applicability and wide popularity in stochastic analysis and engineering.

The index of dispersion of a service process is a measure defined on the squared-coefficient of variation SCV and on the lag-k autocorrelations ρ_k, $k \geq 1$, of the service times as follows:

$$I = SCV \left(1 + 2 \sum_{k=1}^{\infty} \rho_k\right). \quad (1)$$

The joint presence of SCV and autocorrelations in I is sufficient to discriminate traces like those in Figure 1(a)-(d): e.g., for the trace in Figure 1(a) the correlations are statistically negligible, since the probability of a service time being small or large is statistically unrelated to its position in the trace. However, for the trace in Figure 1(d) consecutive samples tend to assume similar values, therefore the sum of autocorrelation in (1) is much greater in Figure 1(d). The last column of Table 1 reports the values of I for the four example traces, the values strongly indicate that I is able to discriminate between the different burstiness levels in Figure 1(a)-(d) and is related directly to the response time results, which is largely expected since autocorrelations severely impact queueing performance.

Since in the exponential case $I = 1$, the index of dispersion may also be interpreted qualitatively as the ratio of the observed service burstiness with respect to a Poisson process; therefore, values of I of the order of hundreds or more indicate a clear departure from the exponentiality assumptions and, unless the real SCV is anomalously high, they are indicators of burstiness. Although the mathematical definition of I in (1) is simple, this formulation is not practical for estimation because of the infinite summation involved and the sensitivity to noise. In addition, it is often hard or impossible to obtain good estimates of the autocorrelation if the measurement window granularity is coarse. In the next subsection, we describe a simple alternative way for estimating I that is also able to overcome the limitations imposed by the measurement granularity wall. We also remark that, if the measurements show long-range dependence, then the index of dispersion value can grow arbitrarily large and other measurements should be used to quantify temporal dependence [7]. In our experiments with TPC-W, we have found that the index of dispersion was very effective to characterize workload burstiness and the value of I was always converging to a finite value. Note also that long-range dependence can be approximated effectively by the class of Markovian processes considered in the remainder of the paper [1].

2.2 Measuring the Index of Dispersion

The index of dispersion enjoys a simple measurement approach which makes it very practical for estimation. Instead of (1), we have an alternative definition of the index of dispersion for a service process as follows. Let N_t be the number of requests completed in a time window of t seconds, where the t seconds are counted *ignoring* the server's idle time (that is, by conditioning on the period where the system is busy, N_t is a property of the service process which is independent of queueing or arrival characteristics). The

index of dispersion I is the limit [6]:

$$I = \lim_{t \to +\infty} Var(N_t)/E[N_t], \quad (2)$$

where $Var(N_t)$ is the variance of the number of completed requests and $E[N_t]$ is the mean service rate during busy periods. Since the value of I depends on the number of completed requests in an asymptotically large observation period, an approximation of this index can be always computed also if the measurements are obtained with coarse granularity. For example, suppose that the sampling resolution is $T = 60$ seconds, and assume to approximate $t \to +\infty$ as $t \approx 2$ hours, then N_t is computed by summing the number of completed requests in 120 consecutive samples. Throughout the paper, we have used the pseudo-code in Figure 2 to estimate I directly from (2). The pseudo-code is a straightforward evaluation of $Var(N_t)/E[N_t]$ for different values of t, which finally estimates I as the limit (2). Intuitively, the algorithm in Figure 2, calculates I of the service process by observing the completions of jobs in concatenated busy period samples. Because of this concatenation, queueing is masked out, and the index of dispersion of job completions serves as a good approximation of the index of dispersion of the service process.

Input
T, the sampling resolution (e.g., 60 seconds)
K, total number of samples, assume $K > 100$
U_k, utilization in the kth period, $1 \leq k \leq K$
n_k, number of completed requests in the kth period, $1 \leq k \leq K$
tol, convergence tolerance (e.g., 0.20)
Estimation of the Index of Dispersion I
1. get the busy time in the kth period $B_k := U_k \cdot T$, $1 \leq k \leq K$
2. initialize $t = T$ and $Y(0) = 0$
3. do
   a. for each $A_k = (B_k, B_{k+1}, \ldots, B_{k+j})$, $\sum_{i=0}^{j} B_{k+i} \approx t$,
      aa. compute $N_t^k = \sum_{i=0}^{j} n_{k+i}$.
   b. if the set of values N_t^k has less than 100 elements.
      bb. The trace is too short. Stop and collect new measures.
   c. $Y(t) = Var(N_t^k)/E[N_t^k]$
   d. increase t by T
   until $|1 - (Y(t)/Y(t-T))| \leq tol$, i.e., the values of $Y(t)$ converge
5. return the last computed value of $Y(t)$ as estimate of I.

Figure 2: Estimation of I from utilization samples.

2.3 Model Parameterization

In this section, we use the measurement of burstiness for the parameterization of a two-phase Markovian Arrival Process (MAP(2)), a class of Markov-modulated process. A MAP(2) can be seen as a Markov chain that jumps between two states and the active state determines the current rate of service. The advantage of MAP(2)s compared to other models is that we can easily fit traces with variability and/or burstiness and the resulting queueing models are computationally tractable [2]. In particular, a MAP(2) is uniquely specified by four descriptors: mean, SCV, skewness, and lag-1 autocorrelation coefficient ρ_1 of the service times; we point to [3] for fitting formulas.

Here, we impose the four values of the MAP(2) descriptors as follows. After estimating the mean service time and the index of dispersion I, we also estimate the 95th percentile of the service times. We distinguish two cases: if the trace has high dispersion (e.g., $I >> 100$), then during a sample interval T all requests are highly correlated, which implies that the 95th percentile of the measured busy times (the B_k values in Figure 2) is approximately equal to the real 95th percentile of the service times[1]. Conversely, if I is small (e.g., $I < 100$), then the assumption of using the same 95th percentile observed in the measured busy times can be inaccurate. Nevertheless, we observe that we can still take this simplification, because under low-burstiness conditions the queueing performance is dominated by the mean and the SCV of the distribution and we do not expect a biased estimate of the 95th percentile to affect accuracy significantly. In practice, we have found this empirical rule to be highly satisfactory for system modeling. Therefore, in all cases we estimate the 95th percentile of the service times equal to the measured 95th percentile of all measured busy period B_k values defined in Figure 2.

Given the mean, the index of dispersion, and the 95th percentile of service times, we search the best combination of the MAP(2) moments and lag-1 autocorrelation that matches these parameters, see [4] for analytical expressions of these descriptors. In particular, we first try to match the index of dispersion and select a subset of MAP(2)s that have less than $\pm 20\%$ relative error on the index of dispersion estimate. Then, we select a MAP(2) from this subset such that the MAP has the 95th percentile of service times closest to the measured value. The search is not computationally challenging given the reduced state space of a MAP(2), usually a couple of minutes are sufficient to obtain the best MAP(2) for each server.

3. EXPERIMENTAL ENVIRONMENT

In our experiments, we use a testbed of a multi-tier e-commerce site that is built according to the TPC-W specifications. TPC-W is a widely used e-commerce benchmark that simulates the operation of an online bookstore [5]. Typically, this multi-tier application uses a three-tier architecture paradigm, which consists of a web server, an application server, and a back-end database. The unit of activity at the client-side corresponds to a download of a web page composed of a HTML file and several embedded objects (images). We opt to put both the web server and the application server on the same machine, henceforth called the front server. We define a *client transaction* as a combination of *all* the processing activities at the server side to deliver an entire web page requested by a client, i.e., generate the main HTML file as well as retrieve embedded objects and perform related database queries.

The number of concurrent user sessions (i.e., customers or emulated browsers (EBs)) is kept constant throughout the experiment. There are 14 different transactions defined by TPC-W. In general, these transactions can be roughly classified of "Browsing" or "Ordering" type. TPC-W defines three standard transaction mixes based on the weight of each type (i.e., browsing or ordering) in the particular transaction mix: (1) the *browsing mix* with 95% browsing and 5% ordering; (2) the *shopping mix* with 80% browsing and 20% ordering; and (3) the *ordering mix* with 50% browsing and 50% ordering. Here, we use a commercially available tool, the HP (Mercury) Diagnostics, to measure the number of completed requests n_k in the kth period. We also use the `sar` command to obtain the utilizations of two servers across time with granularity of one second.

3.1 Bottleneck Switch in TPC-W

For each transaction mix, we run a set of experiments with different numbers of EBs ranging from 25 to 150. Each experiment runs for 3 hours, where the first 5 minutes and the last 5 minutes are considered as warm-up and cool-down periods and thus omitted in the analysis. User think times are exponentially distributed with mean $Z = 0.5$ seconds. Figure 3 presents the overall system throughput, the mean system utilization at the front server and the mean system utilization at the database server as a function of EBs.

[1]This is true because *all* very large requests are likely to appear together in a few bursty periods and this significantly reduces the bias in the estimation of the tail of the service time distribution.

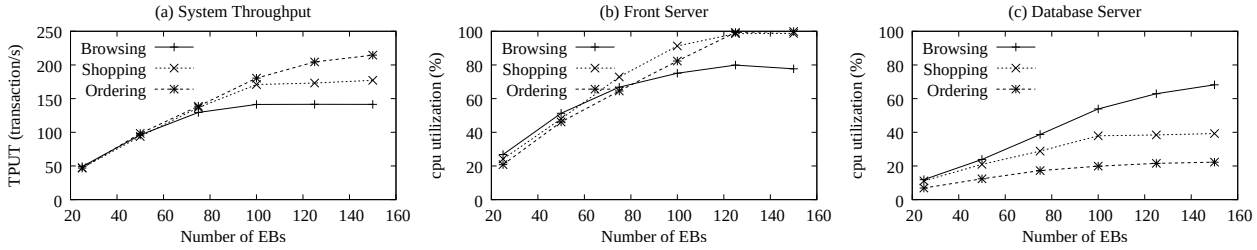


Figure 3: Illustrating a) system overall throughput, b) average CPU utilization of the front server, and c) average CPU utilization of the database server for three TPC-W transaction mixes. The mean think time Z is set to 0.5 seconds.

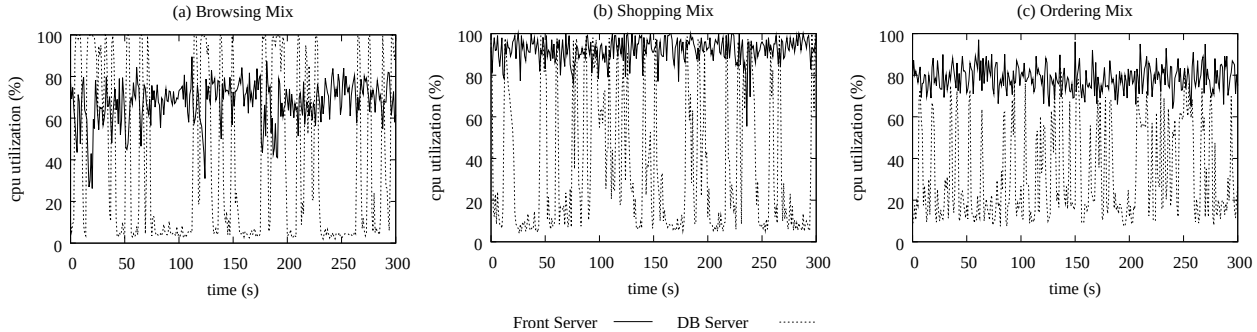


Figure 4: The CPU utilization of the front server and the database server across time with 1 sec granularity for (a) the browsing mix, (b) the shopping mix and (c) the ordering mix under 100 EBs.

Figure 3(a) shows that the system becomes overloaded when the number of EBs reaches 75, 100, and 150 under the browsing mix, the shopping mix and the ordering mix, respectively. The system throughput then remains asymptotically flat with higher EBs. This is due to the "closed loop" aspect of the system, i.e., the fixed number of EBs (customers), that is effectively an upper bound on the number of jobs that circulate in the system at all times.

The results from Figures 3(b) and 3(c) show that under the shopping and ordering mixes, the front server is a bottleneck, where the CPU utilizations are almost 100% at the front tier but only 20-40% at the database tier. For the browsing mix, we see that the CPU utilization of the front server increases very slowly as the number of EBs increases beyond 75, which is consistent with the very slow growth of throughput. Meanwhile, for the browsing mix, the CPU utilization of the database server increases quickly as the number of EBs increases. When the number of EBs is beyond 100, it becomes not obvious which server is responsible for the bottleneck: the average CPU utilizations of two servers are about the same, differing by 10%. In presence of burstiness in the service times, this may suggest that a phenomenon of *bottleneck switching* occurs between the front and the database servers *across time* [9]. That is, a server may become the bottleneck in some periods when processing consecutively large requests, while being lightly loaded during the other periods. In general, additional investigation to determine the existence of bottleneck switching is required when the utilizations are so close or the workloads are known to be highly-variable.

To better understand bottleneck switches, we present CPU utilizations of the front and database servers across time for the browsing mix, as well as the shopping and the ordering mixes, with 100 EBs in Figure 4. A bottleneck switch occurs when the database server utilization becomes larger than the front server utilization, as clearly visible in Figure 4(a). As shown in Figures 4(b) and 4(c), the phenomenon of bottleneck switching cannot be easily observed for the shopping and the ordering mixes, although these two workloads have also high variability. In contrast, there is a continuous *and* obvious switching of bottlenecks between the front and database servers over time for the browsing mix in Figure 4(a). This bottleneck switch is a characteristic effect of burstiness in the service times. This unstable behavior is extremely hard to model. In the next section, we present a solution that takes into account the index of dispersion and enables more accurate modeling of system with bursty workloads.

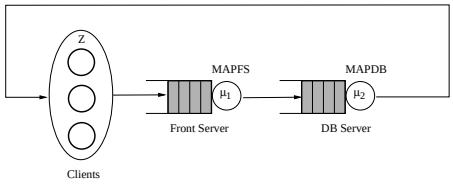


Figure 5: The model for TPC-W.

4. MODEL

We model the multi-tier architecture using a simple queueing network composed by two queues and a delay center, see Figure 5. The two queues model the front server and the database server, respectively. The delay center is instead representative of the average user think time Z between receiving a Web page and the following page download request. The two queues have the first-come first-serve scheduling discipline and are placed in series since the processing at the front server completes almost immediately after database replies are received. In the proposed closed model, the arrival process to the Web server is modeled by the delay which correctly abstracts the behavior of the TPC-W benchmark. Since user think times in TPC-W are exponentially distributed, we equivalently impose an exponentially-distributed service time to the delay server in Figure 5.

The fundamental step of the proposed modeling methodology is the definition of the service time processes at the two servers. As discussed in Section 2, given the mean, the index of dispersion, and the 95th percentile of service times, we obtain the best MAP(2) for each server with a search over the set of feasible MAP(2) parameters. We have noticed that for a small MAP(2), a complete search can be completed in a couple of minutes using MATLAB, therefore the fitting of the MAP(2) service process is not subject to high computational costs. Indeed, if larger MAPs are consid-

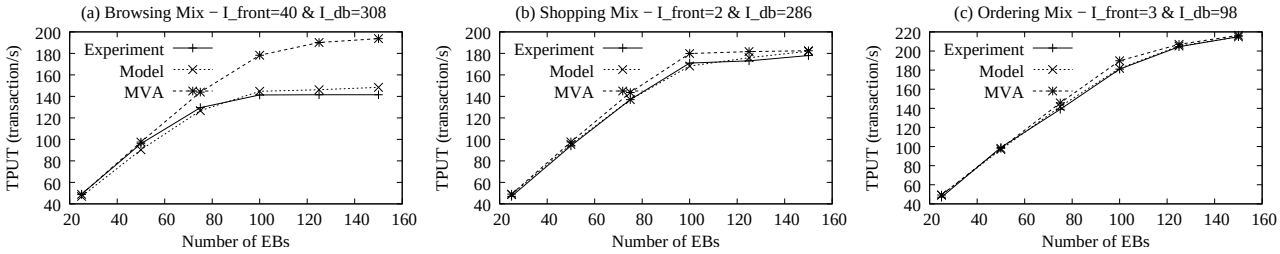


Figure 6: Modeling results for three transaction mixes as a function of the number of EBs.

ered to fit the service processes, more efficient fitting techniques should be used, but resorting to large MAPs introduces the difficult issue of deriving higher-order properties of the inter-arrival times from measurement, e.g., higher-order moments of the correlations [4]. This moments cannot be easily estimated if the correlations between service time samples are significantly altered by the measurement window granularity.

Finally, we parameterize the model in Figure 5 as a MAP queueing network [2] with the service processes that are fitted by two MAP(2)s, denoted as MAPFS and MAPDB, respectively. We then solve the model with MATLAB using an exact numerical evaluation of the underlying Markov chain[2]. In particular, the Markov chain solution gives the probabilities of the different queueing network states, which we aggregate to compute mean utilization and throughput. Other performance indexes such as mean queue-lengths and mean response times can be also computed easily.

Validation: Figure 6 compares the analytic results with the experimental measurements of the real system for (a) the browsing mix, i.e., a bursty workload with significant bottleneck switches, (b) the shopping mix having burstiness at the database server, but negligible bottleneck switching, and (c) the ordering mix with negligible burstiness and bottleneck switches. The values of the index of dispersion for the front and the database service processes are also shown in the figure. Because throughput values immediately provide also the mean response times, here we only present the accuracy of throughput estimates. The response time can be derived from throughput using Little's Law.

Burstiness and bottleneck switches are crucial for the accuracy of forecasting system performance. Figure 6(a) gives evidence that the classic capacity planning models, such as MVA, do not work well for bursty workloads that impose bottleneck switches, as their prediction accuracy dramatically decreases as the system load increases. In contrast, our analytic model based on the index of dispersion achieves substantial gains in the prediction accuracy, as the index of dispersion enables the model to effectively capture *both* burstiness and variability. The results of the proposed analytic model match closely the experimental results for the browsing mix.

The shopping mix presents an interesting case: the MVA model performs well in presence of burstiness because this is a special case where it is not influential in terms of system performance. In fact, regardless of the variation of the workload at the database server, the front server remains the major source of congestion for the system and therefore the model accuracy is high irrespectively of burstiness modeling. This stresses the fact that our modeling methodology provides the largest improvements when the system exhibits complex bottleneck switching phenomena that cannot be modeled, even as approximations, with MVA models. We also remark that if the front server would have been less loaded relatively to the database server, then the behavior of the shopping mix would have been probably similar to the browsing mix.

In the ordering mix, the feature of workload burstiness is almost negligible and the phenomenon of bottleneck switching between the front and the database servers cannot be easily observed, see Section 3.1. For this case, MVA yields prediction errors up to 5%. Yet, as shown in Figure 6(b) and 6(c), our analytic model further improves MVA's prediction accuracy. This happens because of the index of dispersion that is able to capture detailed information on the service characteristics that is not available to the MVA model.

All results shown in Figure 6 validate the analytic model based on index of dispersion: its performance results are in excellent agreement with the experimental values in the systems *with* and *without* workload burstiness.

[2]MATLAB scripts for the solution of MAP queueing network models are available at `http://www.cs.wm.edu/MAPQN/`

5. CONCLUSIONS

In this work, we have presented a solution to the difficult problem of model parametrization by inferring service characteristics from coarse measurements in real systems. After giving quantitative examples of the importance of integrating burstiness in performance models and pointing out its role relatively to the bottleneck switch phenomenon, we show that the coarse measurements can still be used to parameterize queueing models that effectively capture service burstiness and variability. The parameterized queueing model can thus be used to closely predict performance in systems even in the very difficult case where there is persistent bottleneck switch among the various servers.

Detailed experimentation on a multi-tiered system using the TPC-W benchmark validates that the proposed technique offers a robust solution to predicting performance of systems subject to burstiness and bottleneck switches.

6. REFERENCES

[1] A. T. Andersen and B. F. Nielsen. A Markovian approach for modeling packet traffic with long-range dependence. *IEEE JSAC*, 16(5):719–732, 1998.

[2] G. Casale, N. Mi, and E.Smirni. Bound analysis of closed queueing networks with workload burstiness. In *Proc. of ACM SIGMETRICS 2008*, 13-24, 2008.

[3] G. Casale, E. Zhang, and E. Smirni. Interarrival times characterization and fitting for markovian traffic analysis. Number WM-CS-2008-02. Available at http://www.wm.edu/computerscience/techreport/2008/WM-CS-2008-02.pdf.

[4] G. Casale, E. Z. Zhang, and E. Smirni. KPC-Toolbox: Simple yet effective trace fitting using markovian arrival processes. In *to appear in Proc. of the Fifth Conference on Quantitative Evaluation of Systems (QEST)*, Sept. 2008.

[5] D. Garcia and J. Garcia. TPC-W E-commerce benchmark evaluation. *IEEE Computer*, pages 42–48, Feb. 2003.

[6] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE JSAC*, 19(2):203–211, 1991.

[7] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perf. Eval.*, 46(2-3):77–100, 2001.

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

[9] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance impacts of autocorrelated flows in multi-tiered systems. *Perf. Eval.*, 64(9-12):1082-1101, 2007.

[10] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *JACM*, 27(2):312–322, 1980.

[11] M. S. Squillante, D. D. Yao, and L. Zhang. Web traffic modeling and web server performance analysis. *SIGMETRICS Perf. Eval. Rev.*, 27(3):24–27, 1999.

[12] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. of ACM SIGMETRICS 2006*, 291–302, Banff, Canada, June 2005.

Discussion of "How to Parameterize Models with Bursty Workloads"

Presenter: Ningfang Mi (College of William and Mary)
Discussant: Adam Wierman (California Institute of Technology)

This note aims to provide an overview of the discussion at the *HotMetrics 2008 Workshop* surrounding "How to parameterize models with bursty workloads" by Guiliano Casale, Ningfang Mi, Ludmila Cherkasova, and Evgenia Smirni. Ningfang Mis presentation of the work sparked a varied and stimulating dialog among the attendees which I will try to do justice to. But, before we arrive at the discussion, let us briefly overview the goals and contributions of the paper.

1. A BRIEF OVERVIEW

Analytic tools are fundamental to capacity planning in many web applications, e.g., multi-tiered web severs. However, the analytic models at the heart of these tools are often parameterized using only very basic workload statistics–typically just information about the mean values of service demands, inter-arrival times, etc. But, real web workloads are inherently bursty and highly variable, and this causes severe degradation in system performance. In order to capture this behavior, it is necessary to parameterize analytic models with some measure of burstiness, and this is the goal of the paper we are discussing. Parameterizing models with bursty workloads is not an easy task however. Measurements of variability and burstiness are often hidden behind what the authors term the "measurement granularity wall." What this means is that the granularity with which measurements are taken impacts the accuracy of meaurements of burstiness.

The authors suggest that using the "index of dispersion" to parameterize the burstiness of models is an effective technique that is robust to measurement granularity. Further, the index of dispersion jointly captures the variability and burstiness. The authors use the index of dispersion to parameterize a two phase Markovian Arrival Process (MAP(2)), which is then used in the evaluation. A particularly nice feature of this paper is that the authors evaluate their proposed approach by modeling the service demands in the TPCW workload generator, a widely used e-commerce benchmark. So, they illustrate their technique in a very realistic scenario, and the result is that the accuracy of model predictions can be improved by up to 30% compared to parameterizations that ignore burstiness.

2. DISCUSSION OVERVIEW

The audience at HotMetrics clearly recognized the importance of modeling burstiness in web applications. In fact, many members in the audience pointed out that there have been a wide range of models for burstiness in web arrival processes that have emerged from the Sigmetrics community in recent years. With many of those authors present, the discussion sparked by the paper included a number of interesting themes.

Why is "index of dispersion" the right measure?

Clearly the index of dispersion is not the only measure of burstiness that could be used in model parameterizations. For example, it would also be possible to use the autocorrelations and the square coefficient of variation.

The authors explained that the tradeoff between index of dispersion, autocorrelation, and other measures is the granularity of measurements that are required. In particular, most of these measures are highly sensitive to errors resulting from using large sampling intervals, and the index of dispersion is fairly robust to such errors.

Balancing model accuracy and complexity?

Another issue that was brought up was the inherent tradeoff between the accuracy and complexity of the model. In particular, is the benefit in accuracy from using burstiness enough to justify the added complexity when compared with Mean Value Analysis (MVA). In particular, MVA provides significantly more intuition than MAPs. In my opinion, the authors present convincing evidence that burstiness should not be ignored. However, the same issues arise when one considers if a MAP(2) is enough, or whether larger MAPs should be considered. This issue is especially interesting because parameterizing larger MAPs is a much more computationally demanding tasks, and one for which new techniques are needed.

Burstiness in other venues

It was pointed out in the discussion that the context of the paper was a bit limited since the evaluation focused only on the service process of a closed queueing network. However, it was also agreed that the technique will apply more broadly. In particular, it will be interesting to study the applicability of the approach to open queueing models with feedback, which more accurately model the arrivals/departures of user sessions in web applications. Further, it will be interesting to apply the parametrization technique outside of computer applications—the importance of burstiness has been recognized in areas as wide ranging as earthquakes, literature, and weather.

Could burstiness be beneficial?

Burstiness is often viewed as a problem, for example, it worsens the performance in the examples described in the paper.

However, it seems that it is possible to take advantage of it in many cases. I believe one example of this is the increasingly important task of reducing energy usage. In this setting, burstiness may actually increase the effectiveness of speed scaling schemes such as Dynamic Voltage Scaling (DVS). It is interesting to consider what other examples where burstiness is beneficial might be.

Acknowledgement. We would like to thank the session scribe, Abhishek Sharma, of University of South California, for taking detailed notes during the presentation and discussion of this work.