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 model parameterization must be derived from limited coarse measurements as is often encountered in practice. We propose a new modeling 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 and variability of the true service process, despite inevitable inaccuracies that result from inexact measurements. 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 [10]. For example, Web applications running on multi-tier architectures are evaluated using closed queueing networks where cycling requests represent active user sessions [16]; a simple parameterization of these queueing models in terms of mean service demands of incoming requests is then sufficient to predict server utilizations and mean end-to-end response times using Mean Value Analysis (MVA) [14]. However, can we really use queueing models based only on mean service times to predict performance if the workloads are bursty or highly-variable? The answer is often negative: burstiness and high-variability can critically degrade performance to an extent that cannot be captured using mean service times only [11, 2]. Therefore, describing burstiness in performance models of contemporary systems and finding measurement and parameterization techniques that remain simple and practical to use are important open challenges.

In most cases, measurement data provides the "response time" process, i.e., the service times plus waiting/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 process from such coarse measurements.

Consider a typical situation: we have obtained a trace from a live Web server using a non-invasive measurement granularity of a few seconds and we wish to describe service time variability using the collected data. Unfortunately, if the measurement granularity is coarse, then the real distribution of service times is different from the one observed in the trace. For instance, a large request that appears in a sequence of hundreds of small requests may not be immediately visible from the trace if the server is sampled too coarsely. 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 burstiness in the service process can be inferred effectively from these traces using the *index of dispersion* [4, 8] 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 and can also be related to the well-known Hurst parameter used in the analysis of long-range dependence [1]. 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 [13]. 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-

*This work is partially supported by NSF grants ITR-0428330 and CNS-0720699, and a gift from HPLabs.

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 service burstiness using illustrative examples and present the methodology for the measurement of the index of dispersion to parameterize the model. In Section 3, we discuss the multi-tier architecture and the TPC-W workloads used in experiments. The proposed modeling paradigm to integrate burstiness in performance models is presented in Section 4. Section 4 also shows the experimental results that validate the accuracy of the methodology in comparison with standard mean-value based capacity planning. Finally, Section 5 draws conclusions.

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 such a 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 arrival times to the server follow an exponential distribution with mean $\lambda^{-1} = 2$, a simulation analysis of the $M/G/1$ queue[1] (50% utilization) 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 25 times slower than with the service times in Figure 1(a) and the 95th percentile of the response times is even 45 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.

2.1 Characterization of Burstiness

We use the *index of dispersion* I for counts to characterize the burstiness of service times [4, 8]. This is a standard burstiness index used in networking [8], 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 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 statically 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 Eq. (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 related directly to the response time results, which is largely expected since autocorrelations severely impact queueing performance [7].

Note that since in the exponential case $I = 1$, the index of dispersion may 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 Eq.(1) is simple, this formulation is not practical for estimation because of the infinite summation involved and the sensitivity to noise. 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.

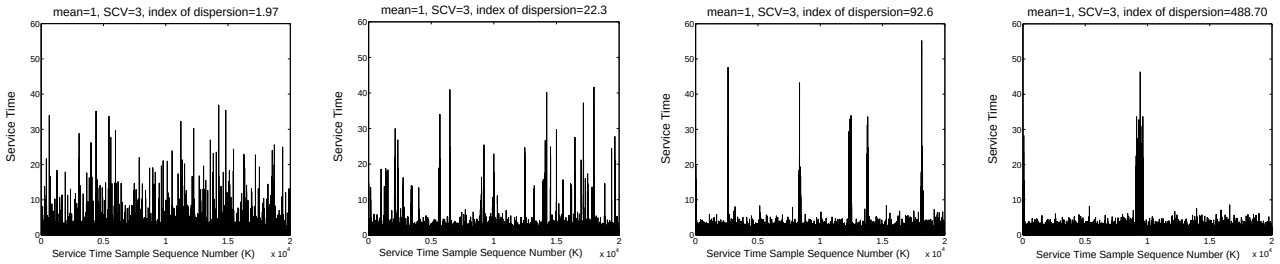
2.2 Measuring the Index of Dispersion

The index of dispersion enjoys a simple measurement approach which makes it very practical for estimation. Instead of Eq.(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). Then as shown in [4], the index of dispersion I is the limit:

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

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 Eq.(2). The pseudo-code is a straight-forward evaluation of $Var(N_t)/E[N_t]$ for different values of t, which finally uses linear regression to estimate 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

[1] We remark that workload burstiness rules out independence of service time samples, thus the classic Pollaczek-Khinchin formula for the $M/G/1$ does not apply and the performance is *not* only determined by mean and squared coefficient-of-variation.



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

Figure 1: Four workload traces with identical hyper-exponential distribution (mean $\mu^{-1} = 1$, $SCV = 3$), but different burstiness profiles. Given the identical variability, trace (d) represents the case of maximum burstiness where all large service times appear consecutively in a large burst. The index of dispersion I, introduced in this paper for the characterization of workloads in multi-tier architectures and reported on top of each figure, is able to capture the significantly different burstiness of the four workloads. As the name suggest, as I grows the dispersion of the bursty periods increases up to the limit case in Figure (d).

serves as a good approximation of the index of dispersion of the service process.

Workload	*Response Time* [ms] mean	95th percentile	*Index of Dispersion* I
Figure 1(a)	3.11	3.23	1.97
Figure 1(b)	9.98	11.98	22.3
Figure 1(c)	51.05	60.91	92.6
Figure 1(d)	75.73	144.47	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%.

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, see [15] for an excellent introduction. A MAP(2) is essentially 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 [12, 2]. In particular, a MAP(2) is uniquely specified by four parameters: mean, SCV, skewness, and lag-1 autocorrelation coefficient ρ_1 of the service times; we point to [5, 3] for fitting formulas.

Here, we impose the four values of the MAP(2) parameters 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[2]. 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, e.g, we are in the same assumptions of the Pollaczek-Khinchin formula for the $M/G/1$ queue, and thus 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, see Figure 2.

Given the mean, the index of dispersion, and the 95th percentile of service times, we search exhaustively the best combination of the MAP(2) moments and lag-1 autocorrelation that matches these parameters, see [9] for MAP(2) expressions of I and percentiles. 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[3]. 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 exhaustive 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

Today, a multi-tier architecture has become an industry standard for implementing scalable client-server enterprise applications. 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 [6]. 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. A client communicates with such a web service via a web interface, where the unit of activity at the client-side corresponds to a download of a web page. In general, a web page is composed of

[2]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.

[3]If $I > 1$, then arbitrarily small error on the index of dispersion can be achieved by proper choice of the MAP(2)'s SCV and ρ_1.

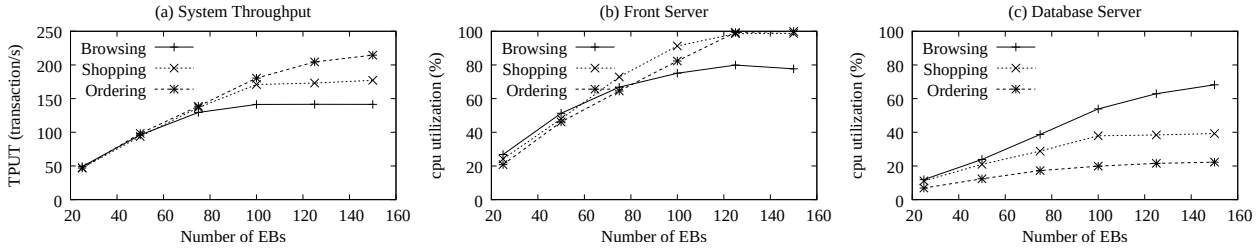


Figure 3: Illustrating a) system overall throughput, b) average CPU utilization of the front server, and c) average CPU utilization of the database server for three TPC-W transaction mixes. The mean think time Z is set to 0.5 seconds.

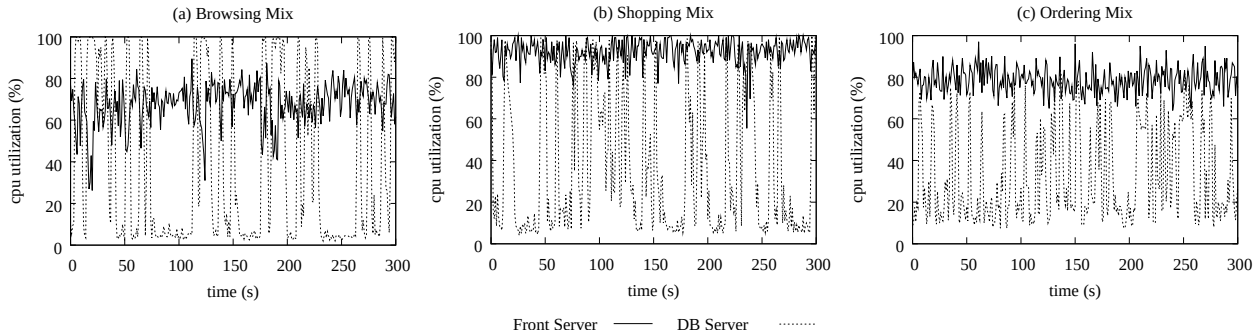


Figure 4: The CPU utilization of the front server and the database server across time with 1 sec granularity for (a) the browsing mix, (b) the shopping mix and (c) the ordering mix under 100 EBs.

an HTML file and several embedded objects such as images. In a production environment, it is common that a web server and application server reside on the same hardware, and shared resources are used by the application and web servers to generate main HTML files as well as to retrieve page embedded objects. We opt to put both the Web server and the application server on the same machine called the front server. Since the HTTP protocol does not provide any means to delimit the beginning or the end of a web page, it is very difficult to accurately measure the aggregate resources consumed due to web page processing at the server side. There is no practical way to effectively measure the service times for *all* page objects, although accurate CPU consumption estimates are required for building an effective application provisioning model. To address this problem, 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.

Typically, a continuous period of time during which a client accesses a Web service is referred to as a *User Session* which consists of a sequence of consecutive individual transaction requests. The number of concurrent 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 in one second granularity.

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. 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. For example, when the front server is already 100% utilized under the shopping and the ordering mixes, the front server for the browsing mix is just around 80%. 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* [11]. That is, a server may become the bottleneck while 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 switching, 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

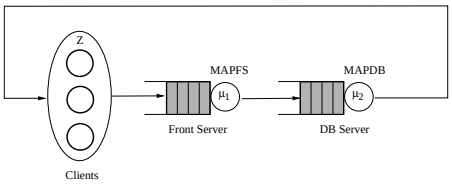


Figure 5: The model for TPC-W.

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 switching is a characteristic effect of burstiness in the service times. This unstable behavior is extremely hard to model. Later, in Section 4.2, we will show that browsing mix exhibits a significantly higher index of dispersion for both the front and database server compared to shopping and ordering mixes. 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.

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[4].

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 by the exhaustive search. 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[5]. 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.

4.1 Discussions on Measurement Granularity

According to TPC-W specification, 7 seconds is the default mean user think time. To evaluate the TPC-W testbed in heavy-load conditions when $Z = 7\ s$, we need to set the number of EBs as high as 1200. To the best of our knowledge, no existing numerical approach can solve the model for exact solutions when the system has such a high population. One optional way is to use the approximation presented in [2]. Here, we obtain the exact solutions by reducing the user think time, e.g., $Z = 0.5\ s$, such that the system becomes overloaded when the number of EBs is around 100 - 150. The minimum and default size of the monitoring window is set to 5 seconds in the Diagnostics tool. As the user think time becomes smaller (i.e., 0.5 s), more requests are completed in a monitoring window. For example, for the browsing mix, when Z is set to 0.5 s and the number of EBs is 50, there are 465 requests completed in every 5 seconds on the average. The information of these jobs are *all* aggregated in one monitoring window. This indicates that smaller Z is equivalent to coarser measurement granularity. Therefore, as Z decreases, more information on variability and burstiness is hidden behind the measurement granularity wall. Unfortunately, if the measurement granularity is coarse, then the values of $Y(t)$ in Figure 2 cannot converge to a stabilized number, see Step **3** in Figure 2. Therefore, a finer measurement granularity is critical for modeling systems with workload burtiness.

There are two approaches to improve the coarse measurement granularity: (1) decreasing the size of the monitoring windows in the tool; and (2) running experiments with the increased user think time Z. The minimum granularity used by Diagnostics for reporting the measured metrics is 5 seconds. Therefore, we alternatively gather the information from the experiments which have a large user think time (e.g., 7 seconds) to estimate I for the front and the database servers, and then obtain the 2-phase MAPFS and MAPDB fitting the service processes at the front server and the backend database server, respectively. Now, for the browsing mix with Z of 7 seconds and 50 EBs, only 17 requests are completed in every 5 second on the average. Note that the real service processes at the front and the database serves are independent on the user think time. This allows us to plug the estimated MAP(2)s into the model shown in Figure 5 to predict the performance metrics for the systems with small user think times, and essentially deal with the measurement granularity problem.

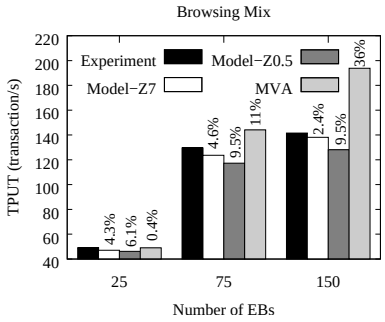


Figure 6: Comparing the results for the model which fits MAPs with different $Z = 0.5$ and 7 seconds.

Figure 6 compares the analytic results with the experimental measurements of the real system for the browsing mix. In all experiments, we set the mean user think time to 0.5 seconds and vary the system loads with different EBs. To evaluate the effect of the measurement granularity on the analytic model, we estimate two sets of MAP(2)s by using the measured traces from the experiments with 50 EBs and two different levels of measurement granularity, i.e., the user think time $Z = 0.5$, and 7 seconds, respectively. As Z increases, we are getting a finer granularity monitoring data. The analytic results from the classic MVA method are plotted as a comparison baseline. Furthermore, the corresponding relative prediction error, which is the ratio of the absolute difference between the analytic result and the measured result over the measured result, is shown on each bar. Figure 6 shows that precision increases non-

[4]Indeed, processor sharing would be a better model of the real system scheduling. This would also avoid database responses to the queue at the front-server. However, only small single-queue processor-sharing models for bursty workloads exist in the literature because of the state-space explosion problem; therefore, we resort to first-come first-served scheduling with two queues in series to avoid overestimation of queueing at the front server.

[5]MATLAB scripts for the solution of MAP queueing network models like the one consider here will be made available at `http://www.cs.wm.edu/MAPQN/`.

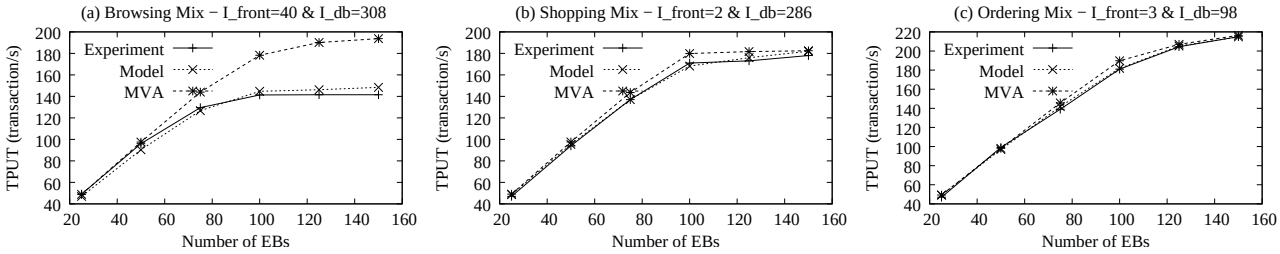


Figure 7: Modeling results for three transaction mixes as a function of the number of EBs.

negligibly when a finer granularity of monitoring data is used. As the system becomes heavily loaded, the model with finer granularity (i.e., Z as high as 7 seconds) dramatically reduces the relative prediction error to 2.4%, while MVA yields a significant error of 36%. In summary, results show that the new model produces superior results.

4.2 Validation

Now, we turn to validate the accuracy of our analytic model for resource usage evaluation through a detailed performance case study in the TPC-W testbed. Figure 7 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 switching, (b) the shopping mix having burstiness at the database server, but negligible bottleneck switching, and (c) the ordering mix with negligible burstiness and bottleneck switching. The values of the index of dispersion for the front and the database service processes are also shown in the figure. Throughout all experiments, the mean user think time Z is set to 0.5 seconds.

Burstiness and bottleneck switching are crucial for the accuracy of forecasting system performance. Figure 7(a) gives evidence that the classic capacity planning models, such as MVA, do not work well for bursty workloads that impose bottleneck switching, 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 7(b) and 7(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 time characteristics that is not available to the MVA model.

All results shown in Figure 7 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* the feature of workload burstiness.

5. CONCLUSIONS

In this work, we have presented a solution to the difficult problem of model parametrization by inferring essential process information from coarse measurements in real systems. After giving quantitative examples of the importance of integrating burstiness in performance models and pointed 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 burstiness and variability of the true process. 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 switching conditions.

The proposed approach is based on measurements that can be routinely obtained from the existing commercial monitoring tools. The resulting parameterized models are practical and robust for a variety of capacity planning and performance modeling tasks in production environments.

6. REFERENCES

[1] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, 1994.

[2] G. Casale, N. Mi, and E.Smirni. Bound analysis of closed queueing networks with workload burstiness. In *Proceedings of ACM SIGMETRICS 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] D. Cox and P. Lewis. *The Statistical Analysis of Series of Events*. John Wiley and Sons, New York, 1966.

[5] H. Ferng and J. Chang. Connection-wise end-to-end performance analysis of queueing networks with MMPP inputs. *Perf. Eval.*, 43(1):39–62, 2001.

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

[7] M. Grossglauser and J. Bolot. On the relevance of long-range dependence in network traffic. In *Proc. of SIGCOMM Conf.*, pages 15–24, 1996.

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

[9] A. Heindl. *Traffic-Based Decomposition of General Queueing Networks with Correlated Input Processes*. Ph.D. Thesis, Shaker Verlag, Aachen, 2001.

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

[11] 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.

[12] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.

[13] M. Reiser. Mean-value analysis and convolution method for queue-dependent servers in closed queueing networks. *Perf. Eval.*, 1:7–18, 1981.

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

[15] T. G. Robertazzi. *Computer Networks and Systems*. Springer, 2000.

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