Burstiness in Multi-Tier Applications: Symptoms, Causes, and New Models *

Ningfang Mi[1], Giuliano Casale[1], Ludmila Cherkasova[2], and Evgenia Smirni[1]

[1] College of William and Mary, Williamsburg, VA 23187, USA
{ningfang,casale,esmirni}@cs.wm.edu
[2] Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA
lucy.cherkasova@hp.com

Abstract. Workload flows in enterprise systems that use the multi-tier paradigm are often characterized as bursty, i.e., exhibit a form of temporal dependence. Burstiness often results in dramatic degradation of the perceived user performance, which is extremely difficult to capture with existing capacity planning models. The main reason behind this deficiency of traditional capacity planning models is that the user perceived performance is the result of the complex interaction of a very complex workload with a very complex system. In this paper, we propose a simple and effective methodology for detecting burstiness symptoms in multi-tier systems rather than identifying the low-level *exact* cause of burstiness as traditional models would require. We provide an effective way to incorporate this information into a surprisingly simple and effective modeling methodology. This new modeling methodology is 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 Markov-modulated process that captures well burstiness and variability of the true service process, despite inevitable inaccuracies that result from inexact and limited measurements. Detailed experimentation on a TPC-W testbed where all measurements are obtained by HP (Mercury) Diagnostics, a commercially available tool, 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 of a given system. Experimental and model prediction results are in excellent agreement and argue strongly for the effectiveness of the proposed methodology under both bursty and non-bursty workloads.
Keywords: capacity planning, multi-tier systems, transactions, sessions, bursty workload, bottleneck switch, index of dispersion.

1 Introduction

The performance of a multi-tier system is determined by the interactions between the incoming requests and the different hardware architectures and software systems that serve them. In order to model these interactions for capacity planning, a detailed characterization of the workloads and of the application is needed, but such a "customized" analysis and modeling may be very time consuming, error-prone, and inefficient in practice. An alternative approach is to rely on live system measurements and to assume that the performance of each software or hardware resource is completely characterized

* This work is partially supported by NSF grants CNS-0720699 and CCF-0811417, and a gift from HPLabs. A short version of this paper titled "How to Parameterize Models with Bursty Workloads" appeared in the HotMetrics 2008 Workshop (non-copyrighted) [5].

by its *mean* service time, a quantity that is easy to obtain with simple measurement procedures. The mean service times of different classes of transaction requests together with the transaction mix can be used as inputs to the widely-used Mean Value Analysis (MVA) models [13, 26, 30] to predict the overall system performance under various load conditions. The popularity of MVA-based models is due to their simplicity and their ability to capture complex systems and workloads in a straightforward manner. In this paper, we present strong evidence that MVA models of multi-tier architectures can be unacceptably inaccurate if the processed workloads exhibit *burstiness*, i.e., short uneven spikes of peak congestion during the lifetime of the system. Motivated by this problem, we define here a new methodology for effective capacity planning under bursty workload conditions.

Internet flash-crowds are familiar examples of bursty traffic and are characterized by periods of continuous peak arrival rate that significantly deviate from the average traffic intensity. Similarly, a footprint of burstiness in system workloads is the presence of short uneven peaks in utilization measurements, which indicate that the server periodically faces congestion. In multi-tier systems, congestion may arise from the super-position of several events including database locks, variability in service time of software operations, memory contention, and/or characteristics of the scheduling algorithms. The above events interact in a complex way with the underlying hardware/software systems and with the incoming requests, often resulting in short congestion periods where the entire system is significantly slowed down. For example, even for multi-tier systems where the database server is highly-efficient, a locking condition on a database table may slow down the service of multiple requests that try to access the same data and make the database the bottleneck server for a time period. During that period of time, the database performance dominates the performance of the overall system, while most of the time another resource, e.g., the application server, may be the primary cause of delays in the system. Thus, the performance of the multi-tier system can vary in time depending on which is the current bottleneck resource and can be significantly conditioned by *dependencies* between servers that cannot by captured by MVA models. However, to the best of our knowledge, no simple methodology exists that captures in a simple way this time-varying *bottleneck switch* in multi-tier systems and its performance implications.

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 traces using the *index of dispersion* for counts of completed requests, a measure of burstiness frequently used in the analysis of time series and network traffic [8, 11]. 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 [4]. 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 [21].

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 perfor-

mance engineering to quantify the peak-to-mean ratio of service demands. Therefore, our performance models are specified by three parameters only for each server: the mean, the index of dispersion, and the 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. We stress that the prediction models we propose do not require explicit identification of the cause(s) of the observed burstiness. Instead, they use a powerful but simple abstraction that captures the effects of burstiness in complex multi-tiered environments.

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 proposed model. In Section 3, we discuss the multi-tier architecture and the TPC-W workloads used in experiments and show that existing queueing models can not work if bottleneck switch exists in the system. The proposed modeling paradigm that integrates burstiness in performance models is presented in Section 4. Section 4 also shows the experimental results that validate the accuracy of the new methodology in comparison with standard mean-value based capacity planning. Finally, Section 6 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 show that burstiness can consistently affect the performance of a system and gain intuition about its fundamental features, we use a simple example. Let us consider the four workloads shown in Figure 1.

Each plot 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 impose to each trace a unique burstiness profile. 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 case where all large requests are compressed into a single large burst. Thus, we use the term "burstiness" to indicate traces that are not just "variable" as the sample in Figure 1(a), but that also aggregate 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 $\lambda^{-1} = 2$ and 1.25, a simulation analysis of the $M/Trace/1$ queue[3] at 50% and 80% utilization, respectively, provides the response times, i.e., the service time plus waiting/queueing times in a server, 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 response time mean and tail. For instance, at 50% utilization the mean response time for the trace in Figure 1(d) is approximately 40 times slower than the service times in Figure 1(a) and the 95th percentile of the response times is nearly 80 times longer. In general, the performance degradation is monotonically increasing with burstiness; therefore it is important to distinguish the behaviors in Figure 1(a)-(d) via a quantitative index.

[3] We remark that workload burstiness rules out independence of service time samples, thus the classic Pollaczek-Khinchin formula for the $M/G/1$ queue does not apply if the service time distribution is bursty.

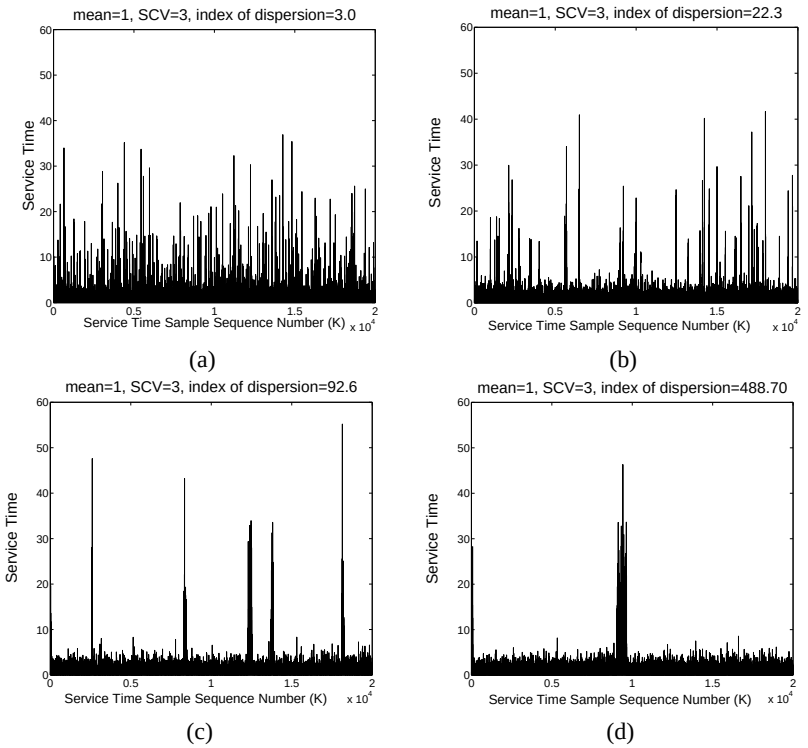

Fig. 1. Four workload traces with identical hyper-exponential distribution (mean $\mu^{-1} = 1$, $SCV = 3$), but different burstiness profiles. Given the identical variability, trace (d) represents the case of maximum burstiness where all large service times appear consecutively in a large burst. The 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, the dispersion of the bursty periods increases up to the limit case in Figure (d) as I grows.

	Response Time (util=0.5)		*Response Time (util*=0.8)		*Index of Dispersion*
Workload	mean	95th percentile	mean	95th percentile	I
Fig. 1(a)	3.02	14.42	8.70	33.26	3.0
Fig. 1(b)	11.00	83.35	43.35	211.76	22.3
Fig. 1(c)	26.69	252.18	72.31	485.42	92.6
Fig. 1(d)	120.49	1132.40	150.32	1346.53	488.7

Table 1. Response time of the $M/Trace/1$ queue relatively to the service times traces shown in Figure 1. The server is evaluated for utilizations $\rho = 0.5$ and $\rho = 0.8$.

Overall the results in Table 1 give intuition that we really need burstiness in performance models. The index of dispersion introduced in the next section is instrumental to capture the difference in the burstiness profiles and provides a simple way to generalize queueing models to effectively capture the performance of bursty workloads and the effects of bottleneck switch.

2.1 Characterization of Burstiness: the Index of Dispersion

We use the *index of dispersion* I for counts to characterize the burstiness of service times [8, 11]. This is a standard burstiness index used in networking [11], which we apply here to the characterization of workload burstiness in multi-tier applications.

The index of dispersion has a broad applicability and wide popularity in stochastic analysis and engineering [8]. From a mathematical perspective, 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[4] ρ_k, $k \geq 1$, of the service times as follows:

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

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 (1) is maximal 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 reflect the different burstiness levels in Figure 1(a)-(d) which directly affect the performance results.

Note that $I = 1$ if service times are exponential, thus 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, I can be used as a good indicator 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 its sensitivity to noise. In the next subsection, we describe a simple alternative way of estimating I.

2.2 Measuring the Index of Dispersion

Instead of (1), we provide 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). If we regard N_t as a random variable, that is, if we perform several experiments by varying the time window placement in the trace and obtain different values of N_t, then the index of dispersion I is the limit [8]:

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

[4] Autocorrelation is used as a statistical measure of the relationship between a random variable and itself [4]. In a time series of random variables $\{X_n\}$, where $n = 0, \ldots, \infty$, ρ_k expresses the value of the autocorrelation coefficient as follows: $\rho_k = \frac{E[(X_t - \mu^{-1})(X_{t+k} - \mu^{-1})]}{\sigma^2}$, where μ^{-1} is the mean, σ^2 is the common variance of $\{X_n\}$, and k denotes the time separation between the occurrences X_t and X_{t+k}.

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 also computed if the measurements are obtained with coarse granularity. For example, suppose that the sampling resolution is $T = 60s$, 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. Repeating the evaluation for different positions of the time window of length t, we compute $Var(N_t)$ and $E[N_t]$. Here, we use the pseudo-code in Figure 2 to estimate I directly from (2). The pseudo-code is a straight-forward evaluation of $Var(N_t)/E[N_t]$ for different values of t. 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., $60s$)
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. stop and collect new measures because the trace is too short;
   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.

Fig. 2. Estimation of I from utilization samples.

3 Burstiness in Multi-Tier Applications: Symptoms and Causes

Today, a multi-tier architecture has become the 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. This allows to conduct experiments under different settings in a controlled environment, which then allows to evaluate the proposed modeling methodology that is based on the index of dispersion.

3.1 Experimental Environment

TPC-W is a widely used e-commerce benchmark that simulates the operation of an online bookstore [10]. Typically, this multi-tier application uses a three-tier architecture paradigm, which consists of a web server, an application server, and a back-end

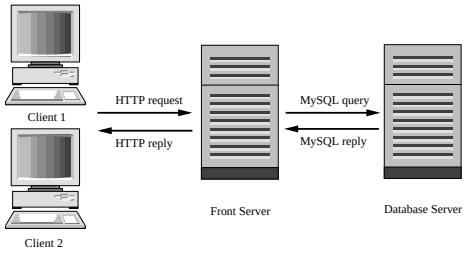

Fig. 3. E-commerce experimental environment.

database. A client communicates with this web service via a web interface, where the unit of activity at the client-side corresponds to a webpage download. In general, a web page is composed by an HTML file and several embedded objects such as images. In a production environment, it is common that the web and the application servers 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[5]. A high-level overview of the experimental set-up is illustrated in Figure 3 and specifics of the software/hardware used are given in Table 2.

	Processor	RAM	OS
Clients (Emulated-Browsers)	Pentium D, 2-way x 3.2 GHz	4 GB	Linux Redhat 9.0
Front Server - Apache/Tomcat 5.5	Pentium D, 1-way x 3.2 GHz	4 GB	Linux Redhat 9.0
Database Server - MySQL5.0	Pentium D, 2-way x 3.2 GHz	4 GB	Linux Redhat 9.0

Table 2. Hardware/software components of the TPC-W testbed.

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. Accurate CPU consumption estimates are required for building an effective application provisioning model but there is no practical way to effectively measure the service times for *all* page objects. To address this problem, we define a *client transaction* as a combination of *all* processing activities that 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. According to the TPC-W specification, the number of concurrent sessions (i.e., customers) or emulated browsers (EBs) is kept constant throughout the experiment. For each EB, the TPC-W benchmark defines the user session length, the user think time, and the queries that are generated by the session. In our experimental environment, two Pentium D machines are used to simulate the EBs. If there are m EBs in the system, then each machine emulates $m/2$ EBs. One Pentium D machine is used as the back-end database server, which is installed with MySQL 5.0 having a database of 10,000 items in inventory.

There are 14 different transactions defined by TPC-W. In general, these transactions can be roughly classified of “Browsing” or “Ordering” type, as shown in Table 3. Furthermore, TPC-W defines three standard transaction mixes based on the weight of

[5] We use terms “front server” and “application server” interchangeably in this paper.

Browsing Type	**Ordering** Type
Home	Shopping Cart
New Products	Customer Registration
Best Sellers	Buy Request
Product detail	Buy Confirm
Search Request	Order Inquiry
Execute Search	Order Display
	Admin Request
	Admin Confirm

Table 3. The 14 transactions defined in TPC-W.

each type (i.e., browsing or ordering) in the particular transaction mix:

- the *browsing mix* with 95% browsing and 5% ordering;
- the *shopping mix* with 80% browsing and 20% ordering;
- the *ordering mix* with 50% browsing and 50% ordering.

One way to capture the navigation pattern within a session is through the *Customer Behavior Model Graph (CBMG)* [16], which describes patterns of user behavior, i.e., how users navigate through the site, and where arcs connecting states (transactions) reflect the probability of the next transaction type. TPC-W is parameterized by the set of probabilities that drive user behavior from one state to another at the user session level. During a session, each EB cycles through a process of sending a transaction request, receiving the response web page, and selecting the next transaction request. Typically, a user session starts with a Home transaction request.

The TPC-W implementation is based on the J2EE standard – a Java platform which is used for web application development and designed to meet the computing needs of large enterprises. For transaction monitoring, we use the HP (Mercury) Diagnostics [29] tool which offers a monitoring solution for J2EE applications. The Diagnostics tool collects performance and diagnostic data from applications without the need for application source code modification or recompilation. It uses bytecode instrumentation, which enables a tool to record processed transactions and their database calls over time as well as to measure their execution time (both transactions and their database calls). We use the Diagnostics tool to measure the number of completed requests n_k in the kth period having a granularity of 5 seconds. We also use the `sar` command to obtain the utilizations of two servers across time with one second granularity.

3.2 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 $Z = 0.5s$. Figure 4 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 4(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. Beyond these EB values, the system throughput remains asymptotically flat. 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.

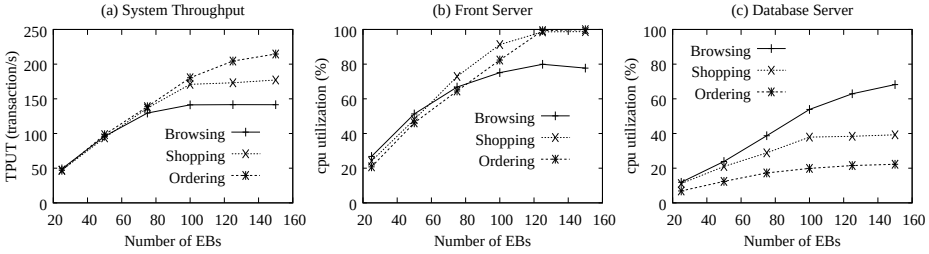

Fig. 4. Illustrating a) system overall throughput, b) average CPU utilization of the front server, and c) average CPU utilization of the database server for three TPC-W transaction mixes. The mean think time Z is set to 0.5 seconds.

The results from Figures 4(b) and 4(c) show that under the shopping and the 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 is not obvious which server is responsible for the bottleneck: the average CPU utilizations of two servers are about the same, differing by a statistically insignificant margin. In presence of burstiness in the service times, this may suggest that the phenomenon of *bottleneck switch* occurs between the front and the database servers *across time*. This phenomenon is not specific to the testbed described in the current work. In an earlier paper [31], a similar situation was observed for a different TPC-W testbed. That is, a server may become the bottleneck while processing consecutively large requests, but be lightly loaded during other periods. In general, additional investigation to determine the existence of bottleneck switch is required when the average utilizations are relatively close or when the workloads are known to be highly variable.

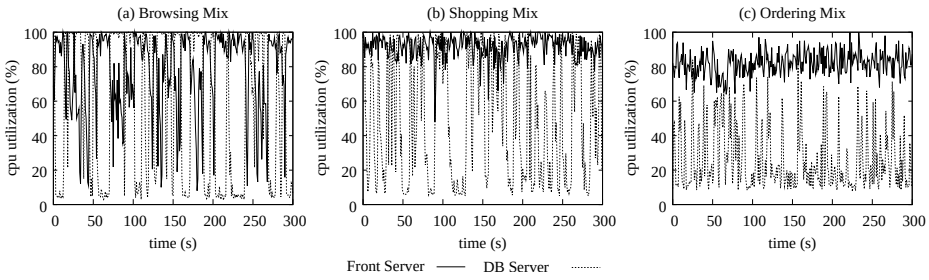

Fig. 5. The CPU utilization of the front server and the database server across time with 1 second granularity for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix under 100 EBs. The monitoring window is 300 seconds.

To confirm our conjecture about the existence of bottleneck switch in the browsing mix experiment, we present CPU utilizations of the front and the database servers across time for the browsing mix, as well as for the shopping and the ordering mixes with 100 EBs, see Figure 5. A bottleneck switch occurs when the database server utilization

becomes significantly higher than the front server utilization, as clearly visible in Figure 5(a) under the browsing mix workload. As shown in Figures 5(b) and 5(c), there is no bottleneck switch for the shopping and the ordering mixes, although these two workloads are also highly variable.

The bottleneck switch is a characteristic effect of burstiness in the service times. This unstable behavior is extremely hard to model. Later, in Section 4.3, we show that the browsing mix exhibits a significantly higher index of dispersion for both the front and database server compared to the shopping and ordering mixes.

3.3 The Analysis of Bottleneck Switch

Now, we focus on the burstiness in a multi-tier application to further analyze the symptoms and possible causes of the bottleneck switch. Indeed, for a typical request-reply transaction, the application server may issue multiple database calls while preparing the reply of a web page. This cascading effect of various tasks breaks down the overall transaction service time into several parts, including the transaction processing time at the application server as well as all related query processing times at the database server. Therefore, the application characteristics and the high variability in database server may cause burstiness in the overall transaction service times.

To verify the above congecture, we record the queue length at the database server at each instance that the database request is issued by the application server and a prepared reply is returned back to the application server. Figure 6 presents the queue length across time at the database server (see solid lines in the figure) as well as the CPU utilizations of the database server (see dashed lines in the figure) for all three transaction mixes.

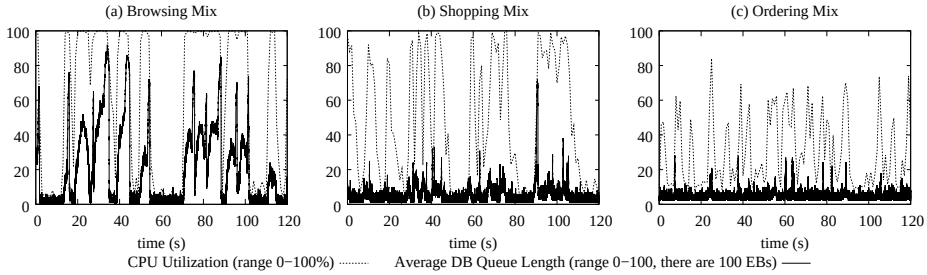

Fig. 6. The CPU utilization of the database server (dashed lines) and average queue length at the database server (solid lines) across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix. In this figure, the y-axis range of both performance metrics is the same because there are 100 EBs (clients) in the system. The monitoring window is 120 seconds.

Here, in order to make the figure easy to read, we show the case with 100 EBs such that the y-axis range for both performance metrics (i.e., queue length and utilization) is the same. First of all, the results for the browsing mix in Figure 6(a) verify that burstiness does exist in the queue length at the database server, where the queue holds less than 10 jobs for some periods, while sharply increases to as high as 90 jobs during other periods. More importantly, the burstiness in the database queue length exactly matches the burstiness in the CPU utilizations of the database server. Thus, at some periods almost all the transaction processing happens either at the application server (with the application server being a bottleneck) or at the database server (with the

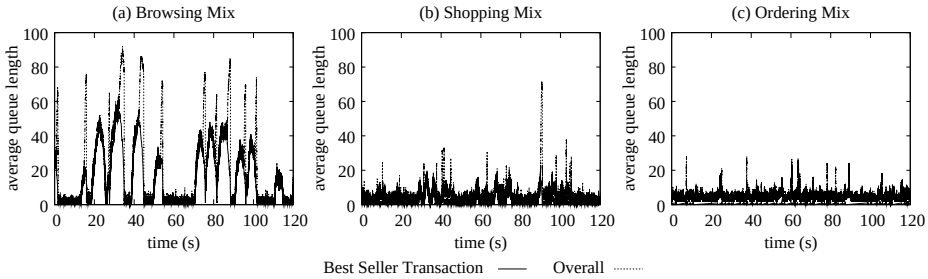

Fig. 7. The overall queue length at the database server (dashed lines) and the number of current requests in system for the *Best Seller* transaction (solid lines) across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix, with 100 EBs and mean think time equal to $0.5s$. The monitoring window is 120 seconds.

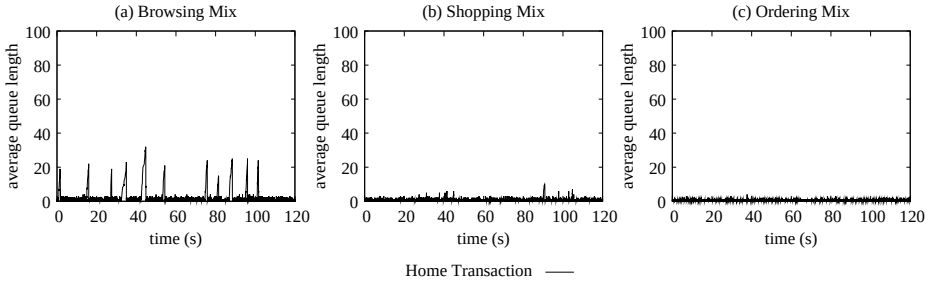

Fig. 8. The number of current requests in system for the *Home* transaction across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix, with 100 EBs and mean think time equal to $0.5s$. The monitoring window is 120 seconds.

database server being a respective bottleneck). This leads to the alternated bottleneck between the application vs the database servers.

In contrast, no burstiness can be observed in the queue length for the shopping and the ordering mixes, although these two workloads have also high variability in their utilizations, see Figures 6(b) and 6(c). These results are consistent with those shown in Figures 5(b) and 5(c), where the application server is the main system bottleneck.

According to the TPC-W specification, different transaction types may have different number of outbound database queries. For example, the *Home* transaction has two database queries in maximum and one in minimum for each transaction request while the *Best Seller* transaction always has two outbound database queries per transaction request. To analyze whether burstiness in the database queue length originates from some particular transaction types, we measure the number of current requests for each transaction type over time. After revisiting all 14 transaction types, we find that the sources of this burstiness are indeed due to specific transaction types. Figures 7 and 8 show the results for two representative transaction types, the *Best Seller* transaction and the *Home* transaction, under three transaction mixes.

In Figure 7, the overall database queue length across time is also plotted as a base line. As shown in Figure 7(a), although in the browsing mix only 11% of requests belongs to the *Best Seller* transaction type, the number of these requests dominates the overall database queue length: the spikes in the overall queue length in the database clearly originate from this particular transaction type. Furthermore, there is burstiness

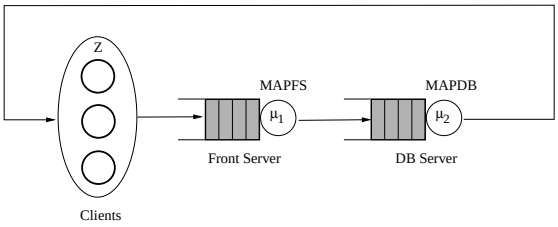

Fig. 9. The closed queueing network for modeling the multi-tier system.

in the number of requests for this transaction type and this burstiness "matches" well the overall queue length in the database server. In addition, for some extremely high spikes, e.g., at timestamp 40 in Figure 7(a), the requests of another popular transaction type, the *Home* transaction, also contribute to burstiness (see Figure 8(a)). These figures indicate that *Best Seller* and *Home* transactions share some resources required for their processing at the database server, and it leads to extreme burstiness during such time periods.

For the shopping and the ordering mixes, there is no visible burstiness in either the queue length at the database server or the number of current requests for each transaction type, as shown in Figure 7(b)-(c) and Figure 8(b)-(c), respectively.

In summary, we showed that

- burstiness in the service times can be a result of a certain workload combination (mix) in the multi-tier applications (e.g., burstiness in the service times may exist under the browsing mix in the TPC-W testbed);
- burstiness in the service times can be caused by a bottleneck switch between the tiers, and can be a result of "hidden" resource contention between the transactions of different types and across different tiers.

Systems with burstiness result in unstable behavior that is extremely hard to express and model. The super-position of several events, such as database locking conditions, variability in service time of software operations, memory contention, and/or characteristics of the scheduling algorithms, may interact in a complex way, resulting in burstiness in the system. The question is whether instead of identifying the low-level *exact* causes of burstiness as traditional models would require, one can provide an effective way to infer this information using live system measurements in order to capture burstiness into new capacity planning models.

3.4 Traditional MVA Performance Models Do not Work

In this section, we use standard performance evaluation methodologies to define an analytical model of the multi-tier architecture presented in Section 3.1. Our goal is to show that existing queueing models can be largely inaccurate in performance prediction if the system is subject to bottleneck switches. We show in Section 4 how performance models can be generalized to correctly account for burstiness and bottleneck switches based on the index of dispersion.

We model the multi-tier architecture studied in our experiments by a closed queueing network composed of two queues and a delay center as shown in Figure 9. Closed queueing networks (see [13] for an introduction) are established as the standard capacity planning models for predicting the performance of distributed architectures using inexpensive algorithms, e.g., Mean Value Analysis (MVA) [22]; we refer to these models in the rest of the paper as *MVA models*.

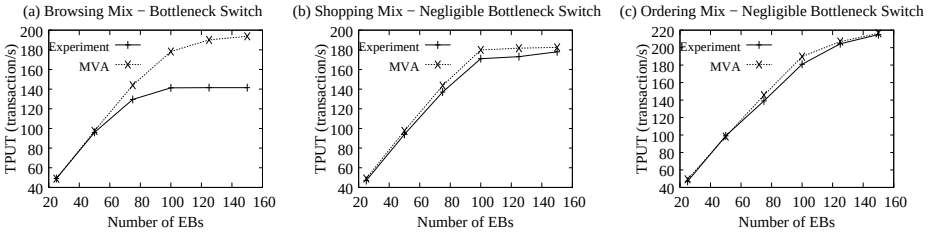

Fig. 10. MVA model predictions versus measured throughput.

In the MVA model shown in Figure 9, the two queues are used to abstract performance of the front server and of the database server, respectively. The delay center is instead representative of the average user think time Z between receiving a Web page and submitting a new page download request[6]. The two queues serve jobs according to a processor-sharing scheduling discipline. In the real application, the servlet code is a mix of instructions at the front server and the database server: without an expensive analysis of the source code, it is truly difficult to characterize the switch of the execution from the front server to the database server and back, we thus make a simplification by assuming that requests first execute at the front server without any interruption and then the residual service time is processed at the database server[7]. Consequently, with this simplification, the two queues in Figure 9 are connected in series.

The proposed MVA model can be immediately parameterized by the following values:

- the mean service time S_{FS} of the front server;
- the mean service time S_{DB} of the database server;
- the average user think time Z;
- the number of emulated browsers (EBs).

Note that the arrival process at the multi-tier system, which is in the real system the arrival of new TPC-W sessions, is fully reproduced by the Z parameter. In fact, a new TPC-W session is generated in Z seconds after completion of a previously-running user session: thus, the feedback-loop aspect of TPC-W is fully captured by the closed nature of the queueing network and the user think time Z completes the model of the TPC-W arrival process.

The values of S_{FS} and S_{DB} can be determined with linear regression methods from the CPU utilization samples measured across time at the two servers [30]. Instead, Z and the number of EBs are imposed to set a specific scenario. For example, in Figure 10, we evaluate an increase of the number of EBs under the fixed think time $Z = 0.5s$; other choices of the delay are possible, see Section 4.2 for a discussion. Indeed, increasing the EB number is a typical way in capacity planning to explore the impact of increasingly larger traffic intensities on system performance. Figure 10 shows the results of the MVA

[6] The main difference between a queue and a delay server is that the mean response time at the latter is *independent* of the number of requests present.

[7] In the following sections, we consider the burstiness associated to the execution of these requests at the front server and at the database server. Our abstraction ignores the order of execution of portions of the servlet code and has no impact on the burstiness estimates because the requests complete *faster* than the monitoring window of the measurement tool. Thus, for an external observer, it would be impossible to distinguish between samples collected from the real system and those of the abstracted system where the code first executes only at the front server and then completes at the database server.

model predictions versus the actual measured throughputs (TPUTs) of the system as a function of the number of EBs.

The three plots in the figure illustrate the accuracy of the MVA model under the browsing, shopping, and ordering mixes. The results show that the MVA model prediction is quite accurate for the shopping and ordering mixes, while there exists a large error up to 36% between the predicted and the measured throughputs for the browsing mix, see Figure 10(a). This indicates that MVA models can deal very well with systems *without* burstiness (e.g., the ordering mix in Figure 10(c)) and with systems where burstiness does *not* result in a bottleneck switch (e.g., the shopping mix in Figure 10(b)). However, the fundamental and most challenging case of burstiness that causes bottleneck switches reveals the limitation of the MVA modeling technique, see Figure 10(a). This is consistent with established theoretical results for MVA models, which rule out the possibility of capturing the bottleneck switching phenomenon [2].

4 Integrating Burstiness in Performance Models

Here, we use a measure of burstiness for the parameterization of the performance model presented in Figure 9. In Section 4.1, we first present the methodology for integrating the burstiness in queueing models and then discuss the impact of measurement granularity in Section 4.2. The experimental results that validate the proposed model are given in Section 4.3.

4.1 Integrating I in Performance Models

In order to integrate the index of dispersion in queueing models, we model service times as a two-phase Markovian Arrival Process (MAP(2)) [19, 23, 6]. A MAP(2) is a Markov chain that jumps between two states and the active state determines the current rate of service. For example, one state may be associated with slow service times, the other may represent fast service times. While processing the sequence of jobs, the MAP(2) jumps between these two states according to predefined frequencies. Simultaneously, the service rate offered to the jobs changes according to the current state. The variation of service rates of the MAP(2) is sufficient to reproduce the burstiness observed in the measured trace. The challenge is to assign the service rates of the two states and the jumping frequencies such that the service times received by the jobs served by the MAP(2) in the queueing model have the same burstiness properties of the service times in the measured trace. Fortunately, MAP(2) service rates and jumping frequencies can be fitted with closed-form formulas given the mean, SCV, skewness, and lag-1 autocorrelation coefficient ρ_1 of the measured service times [9, 7].

We use these closed-form formulas to define the MAP(2) as follows. After estimating the mean service time and the index of dispersion I of the trace, we also estimate the 95th percentile of the service times as we describe at the end of this subsection. Given the mean, the index of dispersion I, and the 95th percentile of service times, we generate a set of MAP(2)s that have $\pm 20\%$ maximal error on I, see [12, 1] for computational formulas of I in MAP(2)s. Among this set of MAP(2)s, we choose the one with its 95th percentile closest to the trace. Overall, the computational cost of fitting the MAP(2)s is negligible both in time and space requirements. For instance, the fitting

of the MAP(2)s has been performed in MATLAB in less than five minutes[8] for the experiments in this paper.

We conclude by explaining how to estimate the 95th percentile of the service times from the measured trace. We compute the 95th percentile of the measured busy times B_k in Figure 2 and scale it by the median number of requests processed in the busy periods. If the trace has high dispersion (e.g., $I >> 100$), this estimate is very accurate because the n_k jobs that are served in the kth busy period receive a similar service time S_k and the busy time is therefore $B_k \approx n_k S_k$. This approximation consists in assuming that n_k is always constant and equal to its median value $med(n_k)$. Under this hypothesis the 95th percentile of B_k is simply $med(n_k)$ times the 95th percentile of S_k. Conversely, if the trace has low dispersion (e.g., $I < 100$), the estimation is inaccurate. Nevertheless, we observe that we can still use this simplification, because under low-burstiness conditions the queueing performance is dominated by the mean and the SCV of the distribution, and therefore a biased estimate of the 95th percentile does not have any appreciable effect on accuracy. In practice, we have found this estimation approach to be highly satisfactory for system modeling as shown by the experimental results reported in the next sections.

4.2 Impact of Measurement Granularity and Monitoring Windows

Starting from the MAP-based model defined in the previous section, we validate the accuracy of the new analytic model using the same experimental setup as in Section 3.4. We denote by Z_{qn} the think time used in the capacity planning queueing network model that represents the system presented in Section 3.4. For validation, we always compare the predictions of this model with a real experiment where the TPC-W has think time Z_{qn}. The notation Z_{estim} denotes the TPC-W think time used in experiments to generate the traces from which we estimate I and the MAP(2)s. In general, Z_{estim} can differ from Z_{qn}, e.g., if we want to explore the sensitivity of the system to different think times we may consider models with different Z_{qn}, but the MAP(2)s are parameterized from the same experimental trace obtained for a certain $Z_{estim} \neq Z_{qn}$. A robust modeling methodology could predict well the performance of the system also for $Z_{qn} \neq Z_{estim}$ and we are seeking for a robust characterization of the service processes which is insensitive to the value Z_{estim} that describes a characteristic of the arrival process to the multi-tier system, rather than a property of the servers.

In all validations, we set $Z_{qn} = 0.5s$ and evaluate throughput and an increase of the number of EBs. The default think time value for the TPC-W benchmark is $7s$, but setting $Z_{qn} = 7s$ we would need to set the number of EBs as high as 1200 to reach heavy-load. Unfortunately, no existing numerical approach can solve the model for exact solutions when the system has such a large number of EBs. Since in this work we are interested in validating models with respect to their exact accuracy, we have explored exact solutions in Section 3.4 by reducing the user think time to $Z_{qn} = 0.5s$, such that the system becomes overloaded when the number of EBs is around $100 - 150$. Models with larger number of EBs should be evaluated with approximations, e.g., with the class of performance bounds presented in [6]. In the rest of paper, we only consider

[8] Occasionally, and only for certain combinations of I and 95th percentile, there may exist more than one MAP(2) with identical mean, I, and 95th percentile. We have not found this case during the experiments in this paper, but in general we recommend to choose the MAP(2) with largest lag-1 autocorrelation since this results in a slightly more aggressive burstiness profile that provides conservative capacity planning estimates.

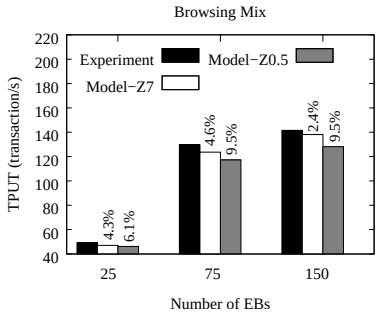

Fig. 11. Comparing the results for the model which fits MAPs with different $Z_{estim} = 0.5s$ and $Z_{estim} = 7s$. On each bar, the relative error with respect to the experimental data is also reported.

queueing network models with $Z_{qn} = 0.5s$. By building the underlying Markov chain and solving the system of linear equations, we solve the new analytic model and get the analytic results, see [6] for a description of the Markov chain underlying a MAP queueing network.

Here, we first present validation results on the browsing mix for different values of the measurement granularity Z_{estim}. Since measurements should not interfere with normal server operations, we have set the monitoring window resolution of the Diagnostics tool to a standard $W = 5s$, which means that hundreds of requests may be served between the collection of two consecutive utilization samples. For instance, when the user think time in TPC-W is set to $Z_{estim} = 0.5s$ and the number of EBs is 50, there are on average 465 requests completed in a monitoring window of $W = 5s$. A reduction of the frequency of sampling makes it difficult to collect a large number of samples (e.g., tens of thousands), and this significantly reduces the statistical robustness of the index of dispersion estimates[9]. Conversely, we have found that decreasing the mean throughput of the system by an increase of Z_{estim} can have beneficial effects on the quality of the index of dispersion estimation without having to modify the monitoring window resolution.

Figure 11 compares the analytic results with the experimental measurements of the real system for the browsing mix. A summary of the think time values used in the two models is given in Table 4. In all models, we set the mean user think time to $Z_{qn} = 0.5s$ and vary the system loads with different EBs. To evaluate the effect of the measurement granularity on the analytic model, we have estimated 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 $Z_{estim} = 0.5s$, and $Z_{estim} = 7s$, respectively. As Z_{estim} increases, we are getting monitoring data of finer granularity, because in the same monitoring window W a smaller number of requests is completed. This makes the estimation of the variance of N_t in the algorithm in Figure 2 more accurate as the finer granularity reveals better the nature of the service times. This is intuitive, e.g., in the extreme case where Z_{estim} is so large that only a single request is

[9] Robustness depends on the relative frequency of service time peaks, e.g., if congestion events due to bursty arrivals as in Figure 1(d) are not frequent, then a large volume of experimental data may be needed to distinguish such events from outliers and correctly identify the bursty behavior.

	Queueing Network	*MAP*(2) *Estimation*
Model-Z0.5	$Z_{qn} = 0.5s$	$Z_{estim} = 0.5s$
Model-Z7	$Z_{qn} = 0.5s$	$Z_{estim} = 7s$

Table 4. Think time values considered in the accuracy validation experiments.

completed during a single monitoring window W, then our measurement corresponds to a direct measure of the request service time and the estimation becomes optimal[10].

In Figure 11, the corresponding relative prediction error, which is the ratio of the absolute difference between the analytic result over the measured result, is shown on each bar. The figure shows that precision increases non-negligibly when a finer granularity of monitoring data is used. As the system becomes heavily loaded, the model with finer granularity (i.e., Z_{estim} as high as $7s$) dramatically reduces the relative prediction error to 2.4%.

4.3 Validation of Prediction Accuracy on Different Transaction Mixes

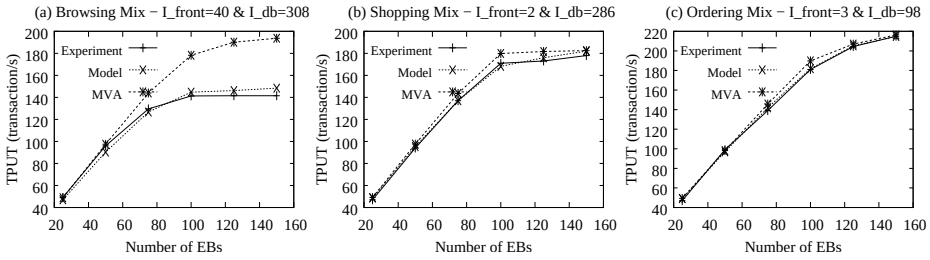

Fig. 12. Modeling results for three transaction mixes as a function of the number of EBs.

Figure 12 compares the analytical results with the experimental measurements of the real system for the three transaction mixes. 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_{qn} is set to $Z_{qn} = 0.5s$; the MAP(2)s are obtained from experimental data collected with $Z_{estim} = 7s$.

Figure 12 gives evidence that the new analytic model based on the index of dispersion achieves gains in the prediction accuracy with respect to the MVA model on *all* workload mixes, showing that it is reliable also when the workloads are not bursty. In the browsing mix, the index of dispersion enables the queueing model to effectively capture *both* burstiness and bottleneck switch. The results of the proposed analytic model match closely the experimental results for the browsing mix, while remaining robust in all other cases.

The shopping mix presents an interesting case: as already observed in Section 3.4, the MVA model performs well on the shopping mix despite the existing burstiness

[10] Indeed, a large increase of Z_{estim} to this level would be unrealistic because it would hide possible slowdowns in service times that become evident only when several requests are served simultaneously, e.g., increased memory access times in algorithms due to an increase in size of shared data structures. For this reason, it is always advisable to increase Z_{estim} such that there are some tens of requests completed in a time window W during the experiment.

because, regardless of the variation of the workload at the database server, the front server remains the major source of congestion for the system and the model behaves similarly to a MVA model (i.e., there is no bottleneck switch).

In the ordering mix, the feature of workload burstiness is almost negligible and the phenomenon of bottleneck switch between the front and the database servers cannot be easily observed, see Section 3.2. For this case, MVA yields prediction errors up to 5%. Yet, as shown in Figure 12(b) and 12(c), our analytic model further improves MVA's prediction accuracy. This happens because the index of dispersion I is able to capture detailed properties of the service time process, which can not be captured by the MVA model.

All results shown in Figure 12 validate the analytic model based on the index of dispersion: its performance results are in excellent agreement with the experimental values in the system, and it remains robust in systems *with* and *without* the feature of workload burstiness and bottleneck switch.

5 Related Work

Capacity planning of multi-tier systems is a critical part of the architecture design process and requires reliable quantitative methods, see [17] for an introduction. Queueing models are popular for predicting system performance and answering what-if capacity planning questions [17, 28, 27, 26]. Single-tier queueing models focus on capturing the performance of the most-congested resource only (i.e., bottleneck tier): [28] describes the application tier of an e-commerce system as a M/GI/1/PS queue; [20] abstracts the application tier of a N-node cluster as a multi-server G/G/N queue.

Mean Value Analysis (MVA) queueing models that capture all the multi-tier architecture performance have been validated in [27, 26] using synthetic workloads running on real systems. The parameterization of these MVA models requires only the mean service demand placed by requests at the different resources. In [24] the authors use multiple linear regression techniques for estimating from utilization measurements the mean service demands of applications in a single-threaded software server. In [15], Liu et al. calibrate queueing model parameters using inference techniques based on end-to-end response time measurements. A traffic model for Web traffic has been proposed in [14], which fits real data using mixtures of distributions.

However, the observations in [18] show that autocorrelation in multi-tier systems flows, which is ignored by standard capacity planning models, must be accounted for accurate performance prediction of multi-tiered systems. Indeed, [3] presents that burstiness in the World Wide Web and its related applications peaks the load of the Web server beyond its capacity, which results in significant degradation of the actual server performance. In this paper we have proposed for the first time robust solutions for capacity planning under workload burstiness. The class of MAP queueing networks considered here has been first introduced in [6] together with a bounding technique for approximate model solution. In this paper, we have proposed a parameterization of MAP queueing networks using for the service process of each server its mean service time, the index of dispersion, and the 95-th percentile of service times. The index of dispersion has been frequently adopted in the networking literature for describing traffic burstiness [25, 11]; in particular, it is known that the performance of the G/M/1/FCFS queue in heavy-traffic is completely determined by its mean service time and the index of dispersion [25]. Further results concerning the characterization of index of dispersion in MAPs can be found in [1].

6 Conclusions

Today's IT and Services departments are faced with the difficult task of ensuring that enterprise business-critical applications are always available and provide adequate performance. Predicting and controlling the issues surrounding system performance is a difficult and overwhelming task for IT administrators. With complexity of enterprise systems increasing over time and customer requirements for QoS growing, effective models for quick and automatic evaluation of required system resources in production systems become a priority item on the service provider's "wish list".

In this work, we have presented a solution to the difficult problem of model parameterization by inferring essential process information from coarse measurements in a real system. After giving quantitative examples of the importance of integrating burstiness in performance models pointing out its role relatively to the bottleneck switching phenomenon, we show that 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 predict performance of systems subject to burstiness and bottleneck switching conditions.

The proposed approach is based on measurements that can be routinely obtained from 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.

References

1. A. T. Andersen and B. F. Nielsen. On the statistical implications of certain random permutations in markovian arrival processes (MAPs) and second-order self-similar processes. *Perf. Eval.*, 41(2-3):67–82, 2000.
2. G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: Common bottlenecks. *Perf. Eval.*, 26(1):51–72, 1996.
3. G. Banga and P. Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.
4. J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, 1994.
5. G. Casale, N. Mi, L. Cherkasova, and E. Smirni. How to parameterize models with bursty workloads. In *Proceedings of First Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics'08)*, 2008.
6. G. Casale, N. Mi, and E.Smirni. Bound analysis of closed queueing networks with workload burstiness. In *Proceedings of ACM SIGMETRICS*, pages 13–24, 2008.
7. 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, 2008.
8. D. Cox and P. Lewis. *The Statistical Analysis of Series of Events*. John Wiley and Sons, New York, 1966.
9. 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.
10. D. Garcia and J. Garcia. TPC-W E-commerce benchmark evaluation. *IEEE Computer*, pages 42–48, Feb. 2003.
11. R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE JSAC*, 19(2):203–211, 1991.

12. A. Heindl. *Traffic-Based Decomposition of General Queueing Networks with Correlated Input Processes*. Ph.D. Thesis, Shaker Verlag, Aachen, 2001.
13. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.
14. Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perform. Eval.*, 46(2-3), 2001.
15. Z. Liu, L. Wynter, C. H. Xia, and F. Zhang. Parameter inference of queueing models for it systems using end-to-end measurements. *Perf. Eval.*, 63(1):36–60, 2006.
16. D. A. Menascé and V. A. F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice-Hall, Inc., 2000.
17. D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy. *Capacity planning and performance modeling: from mainframes to client-server systems*. Prentice-Hall, Inc., 1994.
18. 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.
19. M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.
20. S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *the 10th International Workshop on Quality of Service (IWQoS'02)*, 2002.
21. M. Reiser. Mean-value analysis and convolution method for queue-dependent servers in closed queueing networks. *Perf. Eval.*, 1:7–18, 1981.
22. M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *JACM*, 27(2):312–322, 1980.
23. T. G. Robertazzi. *Computer Networks and Systems*. Springer, 2000.
24. J. Rolia and V. Vetland. Correlating resource demand information with arm data for application services. In *Proceedings of WOSP '98*, pages 219–230. ACM, 1998.
25. K. Sriram and W. Whitt. Characterizing superposition arrival processes in packet multiplexers for voice and data. *IEEE JSAC*, 4(6):833–846, 1986.
26. 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.
27. B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 217–228, 2005.
28. D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Interet Technol.*, 7(1):7, 2007.
29. www.mercury.com/us/products/diagnostics. *Mercury Diagnostics*.
30. Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Proceedings of Middleware*, pages 244–265, Newport Beach, CA, 2007.
31. Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of ICAC'07*, page 27, 2007.