J Internet Serv Appl (2010) 1: 117–134
DOI 10.1007/s13174-010-0012-9

SPECIAL ISSUE: MIDDLEWARE

Sizing multi-tier systems with temporal dependence: benchmarks and analytic models

Ningfang Mi · Giuliano Casale · Ludmila Cherkasova · Evgenia Smirni

Received: 7 March 2010 / Accepted: 8 August 2010 / Published online: 21 September 2010

Abstract Temporal dependence, as a synonym for burstiness, is often found in workloads (i.e., arrival flows and/or service times) in enterprise systems that use the multi-tier paradigm. Despite the fact that burstiness has deleterious effects on performance, existing modeling and benchmarking techniques do not provide an effective capacity planning for multi-tier systems with temporal dependence. In this paper, we first present strong evidence that existing models cannot capture bursty conditions and accurately predict performance. Therefore, we propose a simple and effective sizing methodology to integrate workload burstiness into models and benchmarking tools used in system sizing. This modeling methodology is based on the index of dispersion which jointly captures variability and burstiness of the *service process* in a single number. We report experimentation on a real testbed that validates the accuracy of our modeling technique by showing that experimental and model prediction results are in excellent agreement under both bursty and non-bursty workloads. To further support the capacity planning process under burstiness, we propose an enhanced benchmarking technique that can emulate workload burstiness in systems. We find that most existing benchmarks, like the standard TPC-W benchmark, are designed to assess system performance *only* under non-bursty conditions. In this work, we rectify this deficiency by introducing a new module into existing benchmarks, which allows to inject burstiness into the *arrival stream* in a controllable and reproducible manner by using the index of dispersion as a single turnable knob. This approach enables a better understanding of system performance degradation due to burstiness and makes a strong case for the usefulness of the proposed benchmark enhancement for capacity planning of enterprise systems.

Keywords Enterprise system · Capacity planning · Temporal dependence · Burstiness · Performance benchmarking

This work was partially supported by NSF grants CNS-0720699 and CCF-0811417, a gift from HP Labs, and the Imperial College JRF fellowship.

N. Mi (✉)
Northeastern University, Boston, MA, USA
e-mail: ningfang@ece.neu.edu

G. Casale
Imperial College London, London, UK
e-mail: g.casale@imperial.ac.uk

L. Cherkasova
HP Labs, Palo Alto, CA, USA
e-mail: lucy.cherkasova@hp.com

E. Smirni
College of William and Mary, Williamsburg, VA, USA
e-mail: esmirni@cs.wm.edu

1 Introduction

Capacity planning and resource provisioning for web systems that operate using the client–server paradigm require to take into account emerging Internet phenomena such as the "slashdot effect" where a web page linked by a popular blog or media site suddenly experiences a huge increase of the number of hits [41] with consequent uneven peaks in utilization measurements caused by burstiness. These unexpected surges of traffic are known as *flash crowds* [20]. Traffic surges are also frequent in other contexts, such as in auction sites (e.g., eBay) where users compete to buy an object that is going to be soon assigned to the customer with

the best offer, but also in e-business sites as a result of special offers and marketing campaigns. Burstiness or temporal surges in the incoming requests in an e-commerce server generally turns out to be catastrophic for performance, leading to dramatic server overloading, uncontrolled increase of response times and, in the worst case, service unavailability. 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 by the superposition of several events including database locks, variability in service time of software operations, memory contention, and scheduling characteristics. The above events interact in a complex way with the hardware and software systems involved and with the incoming requests, often resulting in short congestion periods where the entire architecture 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 an extended period of time. 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. For effective capacity planning under bursty workload conditions, capturing this time-varying *bottleneck switch* in multi-tier systems and its performance implications becomes highly critical.

In this paper, we discuss techniques for effective capacity planning under bursty workload conditions that review and extend recent work in the area [13, 30, 31]. After illustrating that existing models of multi-tier architectures can be unacceptably inaccurate if the processed workloads exhibit burstiness, we describe how to integrate workload burstiness in performance models and discuss a validation on an architecture subject to TPC-W workloads with different burstiness profiles. The methodology is based on the *index of dispersion* metric [19], which is a classic indicator for summarizing burstiness in a time series. Using the index of dispersion together with other two parameters, i.e., mean and 95th percentile of service demands, 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.

To further support the capacity planning process under burstiness, we propose a contribution in benchmarking techniques that can emulate the behavior of workload burstiness in systems. Benchmarking is a critical step for effective capacity planning and resource provisioning. An effective benchmark should evaluate the system responsiveness under a wide range of client demands from low to high, but most existing benchmarks are designed to assess the system responsiveness under a *steady* client demand.

We propose to inject burstiness in systems using a simple two-state Markov-modulated processes [34] to regulate the arrival rate of requests to the system. These processes are variations of the popular ON/OFF traffic models used in networking and can be easily shaped to create correlated inter-arrival times. In particular, Markov-modulated processes capture very well the time-varying characteristics of a workload and describe fluctuations at different timescales, e.g., both variability between different surges and fluctuations within the same traffic surge. Starting from this basic idea, we define a modified TPC-W benchmark where sequences of surges with different intensities and durations are created. Consistently with the model-based capacity planning methodology we discuss, the user can describe burstiness in experiment based on the index of dispersion that controls the degree of burstiness in the system. The existence of a *single* parameter to tune burstiness greatly simplifies system benchmarking and allows for a flexible evaluation. We use the index of dispersion to modulate dynamically the think times of users between submission of consecutive requests. Since this approach is independent of the specific nature of the requests sent to the system and only changes their inter-arrival times, our approach can be easily generalized to benchmarks other than TPC-W. In addition, the use of a single parameter for burstiness tuning makes it simple to implement and reproduce the same experiment on different systems, thus enabling the autonomic comparison of client–server performance across different architectures. Using a TPC-W testbed, we show experimentally that this methodology enables to stress the architecture at different levels of performance degradation, thus making the point of being a useful tool for performance robustness assessment of real web systems. We have also released the modified TPC-W at http://www.cs.wm.edu/~esmirni/tpcw_codes/.

The remainder of the paper is organized as follows. In Sect. 2, we introduce burstiness using illustrative examples. We first study burstiness in the service process in a multi-tier enterprise application and present a new approach to integrate workload burstiness in performance models in Sect. 3. We then move to define the new benchmarking methodology starting from an analysis of different sources of burstiness and an evaluation of the standard TPC-W limitations in Sect. 4. Detailed experimentation on a real testbed is presented as well in Sects. 3 and 4, where we validate the accuracy of our performance model in comparison with standard mean-value based capacity planning and demonstrate that our modified TPC-W benchmark is extremely effective in stressing system performance under different levels of burstiness. A review of existing research efforts in capacity

planning and resource provisioning is presented in Sect. 5. Finally, Sect. 6 draws conclusions.

2 Burstiness impact and index of dispersion

In this section, we consider some examples to illustrate the importance of burstiness in performance models and the impact of bursty processes on system performance. Here, we generate three workload traces such that inter-arrival times in each trace are generated from a 2-state Markov-Modulated Poison Process (MMPP(2)) with the same mean $\lambda^{-1} = 10$ ms and squared coefficient-of-variation $SCV = 10$, but different burstiness profiles.

Figure 1(a)–(c) presents the number of incoming arrivals during every 10 ms under these three workloads, respectively. Although three traces have the same variability, a burst of requests aggressively aggregates during a short period in Fig. 1(b)–(c), while no temporal surges exist in Fig. 1(a) as requests come in random points of the trace. Particularly, Fig. 1(c) shows the strongest burstiness case with almost all incoming requests clustering within several short congestion periods. Therefore, we use the term "burstiness" to indicate traces that are not just "variable" as the sample in Fig.1(a), but also aggregate in "bursty periods" as in Fig. 1(b)–(c).

In order to disclose the performance impact of the burstiness, we run simulations of a $trace/M/1$ queue such that request arrival times to the server are obtained from the three inter-arrival time traces in Fig. 1 and request service times follow an exponential distribution with mean $\mu^{-1} = 4$ ms and 8 ms. As a result, we evaluate the performance of this $trace/M/1$ queue under two system loads, i.e., utilizations $\rho = 40\%$ and $\rho = 80\%$, respectively. We also remark that workload burstiness rules out independence of service time samples, thus the classic Pollaczek–Khinchin formula for the $G/M/1$ does not apply and the performance is *not* only determined by mean and squared coefficient-of-variation.

Figure 1(d) depicts the mean response times for the inter-arrival times traces with different burstiness profiles, i.e., $I = 10$, 82, and 402, as shown in Fig. 1(a)–(c). Irrespectively of the identical service time distribution, burstiness in workload traces dramatically degrades the system performance and thus clearly has paramount importance for queueing prediction. For example, when the system is under median load (e.g., 40% utilization), the mean response time for the high variable but non-bursty trace in Fig. 1(a) is not high, but as the dispersion of the burstiness increases, the mean response time becomes approximately 7 and 40 times higher for the traces in Fig. 1(b) and (c), respectively. Furthermore, the performance degradation is monotonically increasing as the observed burstiness increases. Therefore, it is critically important to discriminate the behaviors in Fig. 1(a)–(c) with a quantitative index. Overall, the results in Fig. 1 clearly give intuition that we really need burstiness in performance models.

Furthermore, the burstiness in workloads, such as the inter-arrival times in Fig. 1, can be characterized by the *index of dispersion* I [15, 19]. This is a standard burstiness index used in networking [19], which we apply here to the characterization of workload burstiness in multi-tier applications. To the best of our knowledge, the index of dispersion, while being successfully used in modeling of networking applications, has not been previously applied to modeling of enterprise multi-tier applications.

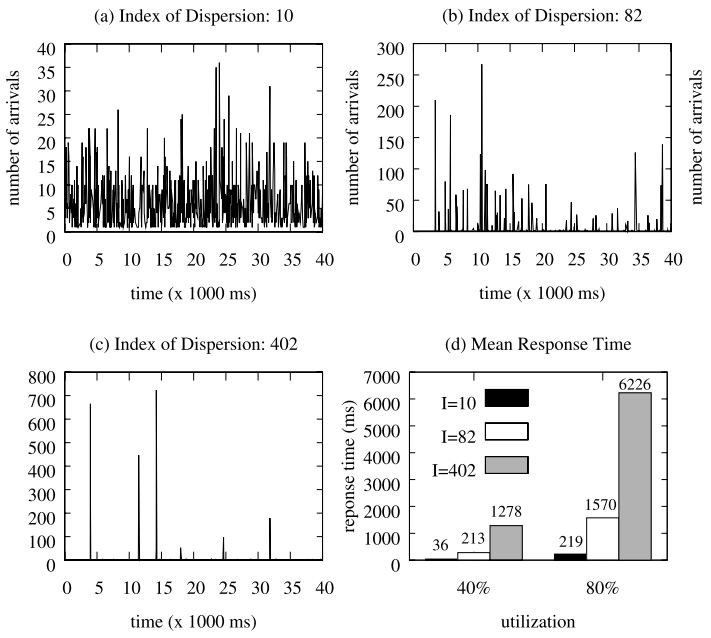

Fig. 1 Three workload traces with MMPP(2) distribution (mean $\lambda^{-1} = 10$ ms, $SCV = 10$), but different burstiness profiles. Plots (**a**)–(**c**) show the number of incoming requests during every 10 ms under the three workload traces and the index of dispersion I is reported on *top of each plot*. Plot (**d**) presents the mean response time of the $trace/M/1$ queue under the system utilization of 40 and 80%

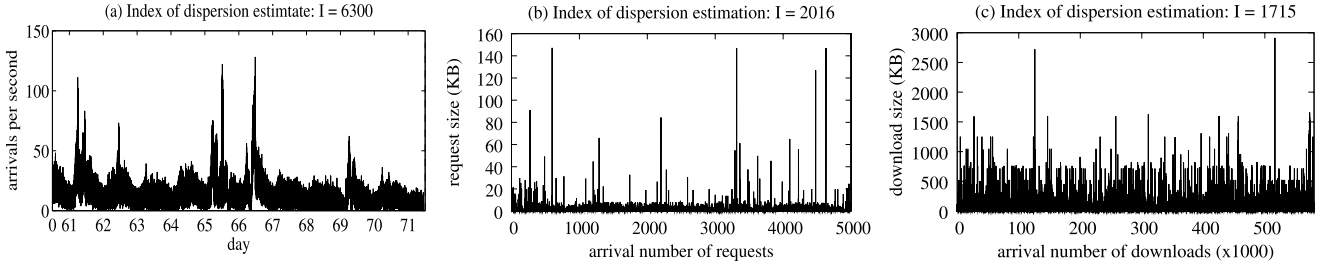

Fig. 2 Burstiness of (**a**) arrivals to server 0 in the 1998 FIFA World Cup trace over ten consecutive days, (**b**) Google HTTP request sizes logged by New York, NY IRCache server over two days, i.e., January 9, 2007 and January 10, 2007, and (**c**) static object download sizes of the HTTP servers at Politecnico di Milano—DEI between September 17, 2006 and September 24, 2006

The characterization of time series with burstiness requires techniques for the statical description of the order in which requests appear in a trace. This topic has been investigated by several works in the literature, a survey of the most popular descriptors can be found in [22]. Here, we focus on the asymptotic index of dispersion I as a metric for characterizing burstiness. Consider a set of n jobs having inter-arrival times $X_1, X_2, \ldots, X_n$, and define: $A_n = X_1 + X_2 + \cdots + X_n$ as the total duration of work imposed on the system by the n jobs. We can define the index of dispersion as the asymptotic limit

$$I = \lim_{n\to+\infty} I_n = \lim_{n\to+\infty} \frac{E[(A_n - E[A_n])^2]}{nE[X_n]^2},$$

where the argument is the index of dispersion for intervals I_n [4]. Noting that $E[A_n] = nE[X_n]$, it is immediate to see that $I_n E[X_n]$ is a relative squared deviation of A_n from expectation, thus I may be seen as a quantifier of the magnitude in fluctuations in an asymptotically large time-series with unit mean.

Other definitions of the index of dispersion of an arrival process are useful for understanding the metric. Call $SCV = \mathrm{Var}(X_n)/E[X_n]^2$ the squared coefficient of variation of the inter-arrival times and denote with ρ_k the lag-k autocorrelations for $k \geq 1$; then the index of dispersion can be written 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 distinguish between traces like those in Fig. 1(a)–(c), as we have reported in the figure title. As the name suggests, the dispersion of the bursty periods increases as the values of I grow because the sum of autocorrelation in (1) is maximal. But, when the correlations become statically negligible, the index of dispersion only captures the characterization of variability with no burstiness presented in workloads. The value of I thus approaches the one of SCV, as shown in Fig. 1(a).

We further give three examples of real world situations where burstiness exists and the index of dispersion well captures the intensity of traffic surges. The first real workload is the 1998 FIFA World Cup website trace available at [5] over a period of ten days, presenting that dramatic traffic surges connected to sport events can reach values of I slightly larger than 6300,[1] see Fig. 2(a). We remark that although the 1998 FIFA World Cup trace is an old web workload, many characteristics including burstiness persist in recent years [46]. We also examined two recently collected web traces: one logged by the New York, NY IRCache server over two days in January, 2007, shows that the number of bytes written to the client by *Google* are not only variable but also bursty, resulting in the estimation of I greater than 2000, see Fig. 2(b); and the other was collected from Politecnico di Milano—DEI between September 17, 2006 and September 24, 2006, showing the large static objects (e.g., gif and jpg image files) aggregated in "bursty periods" with the estimated value of I more than 1715, see Fig. 2(c). In summary, the index of dispersion I can be used as a measure of burstiness in workloads and will be introduced for evaluating multi-tier architectures in Sect. 3 and enhancing benchmarking techniques in Sect. 4.

3 Service process: one source of burstiness

In this section, we first study one source of burstiness—service process—in a multi-tier enterprise application. Then we illustrate that traditional models of multi-tier architectures can be unacceptably inaccurate if the processed workloads exhibit burstiness. We describe how to integrate workload burstiness in performance models by using the index of dispersion and discuss a validation of the proposed technique in a testbed of a multi-tier e-commerce site that is built according to the TPC-W specifications.

[1] Our analysis has focused on the server with label "0" during the period going from day 61 to day 71. The estimation of the index of dispersion I has been done using the theoretical formulas reported in [19], (6).

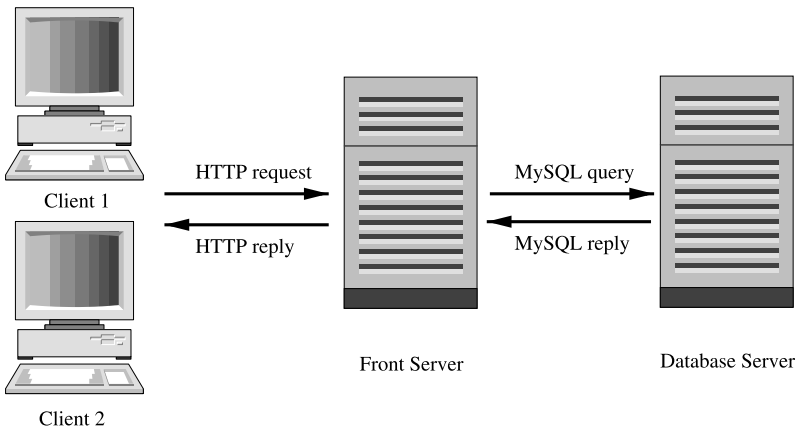

Fig. 3 E-commerce experimental environment

3.1 Burstiness in TPC-W

TPC-W is a widely used e-commerce benchmark that simulates the operation of an online bookstore [18]. Typically, a 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 this web service via a web interface, where the unit of activity at the client-side corresponds to a web page download. 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 web pages. Thus, we opt to put both the web server and the application server on the same machine called the front server.[2] A high-level overview of the experimental setup is illustrated in Fig. 3.

In general, a web page is composed by an HTML file and several embedded objects such as images. 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. Each client sends requests in the system with an average think time that represents the time between receiving a web page and the following page download request.

[2]We use terms "front server" and "application server" interchangeably in this paper.

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 statistically 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. We also have one Pentium D machine serving as the front server, which is installed with Apache/Tomcat 5.5, and one Pentium D machine serving as the back-end database server, which is installed with MySQL 5.0. The database size is determined by the number of items and the number of customers. In our experiments, we use the default database setting, i.e., the one with 10,000 items and 1,440,000 customers in inventory.

There are 14 different transactions defined by TPC-W. In general, these transactions can be roughly classified of "Browsing" or "Ordering" type. Furthermore, TPC-W defines three standard transaction mixes based on the weight of each type in the particular transaction mix:

Transaction mix	Transaction type	
	Browsing	Ordering
Browsing mix	95%	5%
Shopping mix	80%	20%
Ordering mix	50%	50%

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 [47] 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

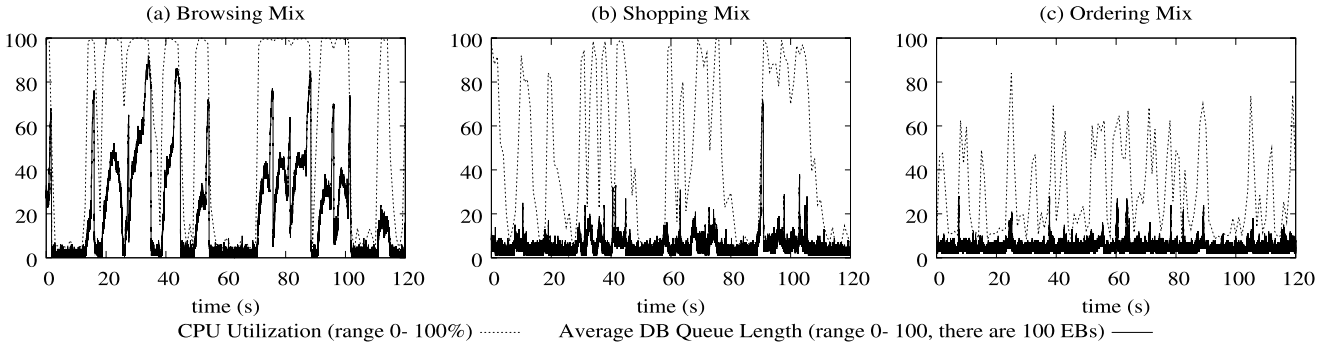

Fig. 4 The CPU utilization of the database server (*dashed lines*) and average queue length at the database server (*solid lines*) across time for (**a**) the browsing mix, (**b**) the shopping mix, and (**c**) the ordering mix. In this figure, the y-axis range of both performance metrics is the same because there are 100 EBs (clients) in the system. The monitoring window is 120 seconds

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.

In TPC-W, 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 conjecture, we measure the queue length across time (see solid lines in Fig. 4) and the CPU utilization across time (see dashed lines in Fig. 4) at the database server under all three transaction mixes, where the transient queue length is recorded at each instance that the database request is issued by the application server and a prepared reply is returned back to the application server. Furthermore, in order to make the figure easy to read, we present the case with 100 EBs such that the queue length is within the range from 0 to 100 and thus the y-axis range for both performance metrics (i.e., queue length and utilization) is the same. First, our conjecture is verified that for the browsing mix 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, see Fig. 4(a). 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 database server being a respective bottleneck). This leads to the alternated bottleneck between the application vs. the database servers. In contrast, for the shopping and the ordering mixes, Figs. 4(b) and (c) in the figure only show high variability in their utilizations but no burstiness in the queue length.

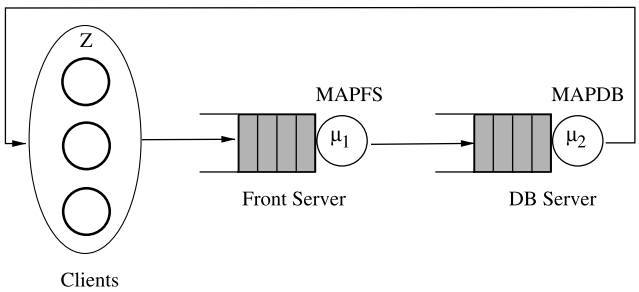

Fig. 5 A closed queueing network for modeling a multi-tier system

3.2 Limitation of traditional performance models

Traditionally, a multi-tier system can be modeled by a closed queueing network, e.g., composed of two queues and a delay center as shown in Fig. 5, and can be solved with inexpensive algorithms, e.g., Mean Value Analysis (MVA) [38]; we refer to these models in the rest of the paper as *MVA models*. In the MVA model shown in Fig. 5, the two queues are representative of the two servers in an enterprise system, i.e., the front server and the database server, respectively. The delay center is instead used to emulate the client activities, such that each server within the delay center models the user think time between receiving a web page and submitting a new page download request.[3] The two queues serve jobs according to a processor-sharing scheduling discipline.

[3]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.

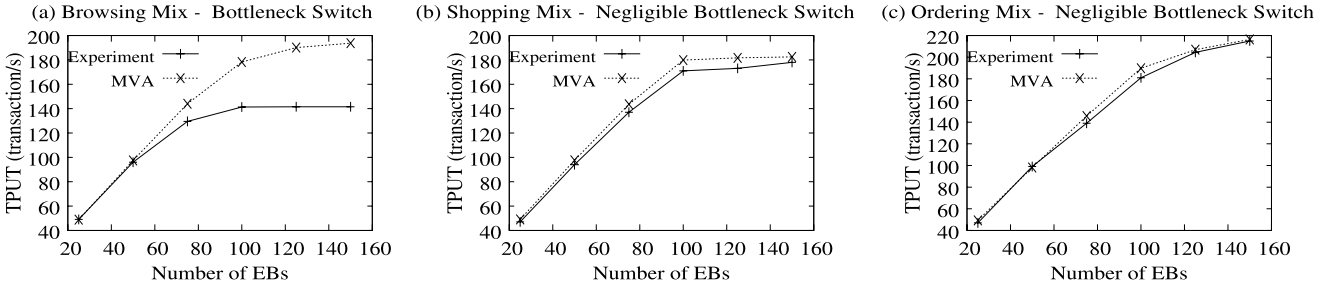

Fig. 6 MVA model predictions versus measured throughput

The proposed MVA model can be immediately parameterized by (1) the mean service time S_{FS} and S_{DB} of the front server and the database server, respectively, (2) the average user think time Z, and (3) the number of emulated browsers (EBs). In TPC-W, a new session is generated in Z seconds (user think time) 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. 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 [48].

Figure 6 presents the results of the MVA model predictions versus the actual measured throughputs (TPUTs) of the system as a function of the number of EBs under the browsing, shopping, and ordering mixes. We observe that the MVA model prediction is quite accurate for the shopping and ordering mixes, see Fig. 6(b) and (c). However, for the browsing mix, the MVA models obtain unacceptable inaccuracy with a large error up to 36% between the predicted and the measured throughputs, see Fig. 6(a). This indicates that MVA models can deal very well with systems *without* burstiness (e.g., the ordering mix) and with systems where burstiness does *not* result in a bottleneck switch (e.g., the shopping mix). However, the fundamental and most challenging case of burstiness reveals the limitation of the MVA modeling technique, see browsing mix in Fig. 6(a). This is consistent with established theoretical results for MVA models, which rule out the possibility of capturing the bottleneck switching phenomenon [8].

3.3 Performance models with burstiness

Although the mathematical definition of the index of dispersion I in (1) is simple, this formulation is not practical for estimation because of the infinite summation involved and its sensitivity to noise. The estimation of the index of dispersion is difficult due to well-known difficulty of estimating autocorrelations reliably [15]. Techniques for estimation of I based on sample measurements are proposed in [19]. Alternatively, one can use the following estimation algorithm, which requires data that is commonly available from system performance measurement tools. 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 known to be equal also to the limit [15]:

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

where $\mathrm{Var}(N_t)$ is the variance of the number of completed requests and $E[N_t]$ is the mean service rate during busy periods. Here I_t represents the index of dispersion for counts, a metric similar to I_n but that describes the variance in time series of counts rather than in intervals. 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 $T = 60$ s, 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 obtain a basic estimate of $\mathrm{Var}(N_t)$ and $E[N_t]$. Based on this approach, the pseudo-code in Fig. 7 can be used to estimate I directly from (2). The pseudo-code is a straightforward evaluation of $\mathrm{Var}(N_t)/E[N_t]$ for different values of t. Intuitively, the algorithm in Fig. 7 calculates I of the service process by observing the completions of jobs in concatenated busy period samples, thus trying to reconstruct the service process time series as if it was measured without the effects of queueing.

In order to integrate the index of dispersion in queueing models, we model service times as a two-phase Markovian Arrival Process (MAP(2)) [34]. The advantage of this approach is that one can then represent service times as MAPs and use the recently proposed class of MAP queueing networks for capacity planning [12, 14]. A MAP(2) is a

Markov process 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, while the other may represent fast service times, and the jumping frequencies between the two states can be chosen to reproduce exactly the burstiness and the distribution of service or arrival times observed in a trace. Typically, given a set of trace moments and the index of dispersion value I, it is straightforward to obtain values of the MAP(2) parameters that uniquely specify the Markov process. We point to (5) and (7) reported later in the paper for equations that related moments and I with MAP(2) parameters and thus can be used directly for MAP(2) fitting.

Input
T, the sampling resolution (e.g., 60 s)
K, total number of samples, assume $K > 100$
U_k, utilization in the kth period, $1 \le k \le K$
n_k, number of completed requests in the kth period, $1 \le k \le K$
tol, convergence tolerance (e.g., 0.20)
Estimation of the index of dispersion I
1. get the busy time in the kth period $B_k := U_k \cdot T$, $1 \le k \le K$
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) $Y(t) = \mathrm{Var}(N_t^k)/E[N_t^k]$
   (d) increase t by T
   until $|1 - (Y(t)/Y(t-T))| \le tol$, i.e., the values of $Y(t)$ converge
4. return the last computed value of $Y(t)$ as estimate of I

Fig. 7 Estimation of I from utilization samples and counts

We can use the 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 ±20% maximal error on I. 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. For the experiments in this section, the 95th percentile is obtained from the 95th percentile of the measured busy times B_k in Fig. 7 scaled by the median number of requests processed in the busy periods, see [30] for details.

We illustrate the accuracy of MAP queueing networks as a capacity planning tool compared to the Mean Value Analysis (MVA) algorithm that is the standard in queueing analysis for IT sizing [22]. Figure 8 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. Figure 8 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. While in the shopping and the ordering mixes, the feature of workload burstiness is almost negligible and thus MVA yields prediction errors up to 10%. Yet, as shown in Fig. 8(b) and (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 cannot be captured by the MVA model. Our experiments provide evidence that the

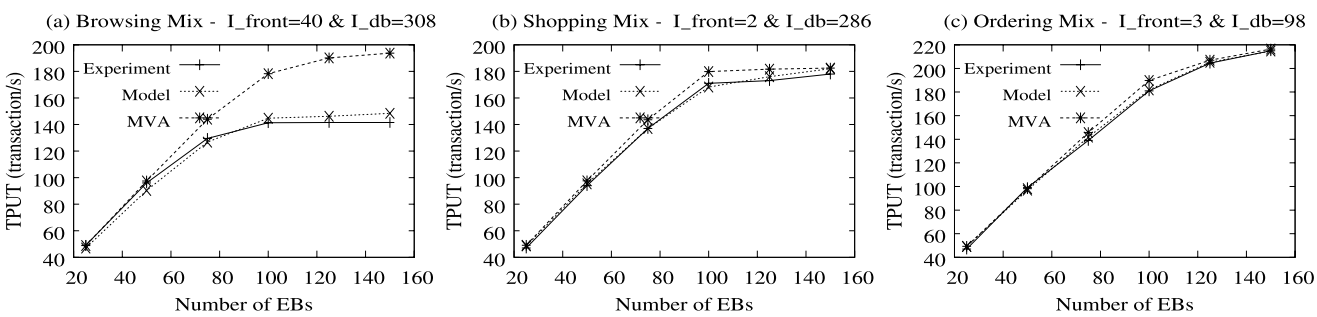

Fig. 8 Modeling results for three transaction mixes as a function of the number of EBs

proposed methodology can work effectively on real-world applications. Indeed, further validation on real workloads is needed in order to further assess the general applicability of the technique.

4 Arrival process: another source of burstiness

Burstiness in arrival streams and/or service processes is often found in client–server systems. Capturing burstiness accurately in performance models for capacity planning [4] becomes extremely important and challenging because this feature is responsible to significant degradation of perceived user performance and system capacity by creating request peak congestion periods in systems.

4.1 Different sources of burstiness

In order to gain intuition about the importance of burstiness in performance models, we use the closed queueing network shown in Fig. 5 to model a multi-tier architecture and then show how burstiness generates traffic surges and thus consistently affects the system performance.

Figure 9 presents the simulation results of the end-to-end client response times, i.e., the summation of the response times at the front server and the back-end database, parameterized according to the TPC-W model presented in Sect. 3. In all simulations, we set the same mean service time (i.e., 5 microseconds) at the front server, as well as the same mean service time (i.e., 3 microseconds) at the database server. The mean user think time is also kept the same in all experiments, i.e., $Z = 7$ seconds. The only difference is that we impose into the model different burstiness profiles: (1) there is no burstiness in neither the two servers, nor the client side, labeled *non-bursty*; (2) burstiness is present only in the front server's service process, labeled *front-server-bursty*; (3) burstiness is present only in the service process at the back-end database, labeled *db-server-bursty*; and (4) burstiness is present only in the arrival process to the front server, labeled *client-bursty*.

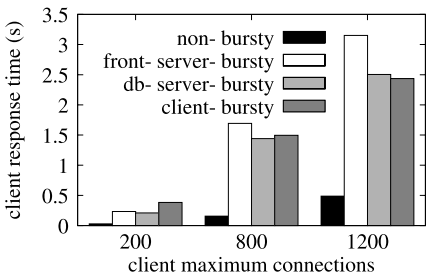

Fig. 9 Illustrating average end-to-end client response time as a function of the number of maximum client connections N

Figure 9 first shows the same results we have presented in Fig. 1 of Sect. 2: when there is burstiness in workloads, the system performance becomes worse compared to the non-bursty case and this performance degradation is consistent over different system loads (i.e., the number of client connections). In addition, we find that burstiness in any system tier (i.e., the service processes at the front and the database servers) or client side (i.e., the arrival process) has a similar negative impact on overall system performance. This motivates us to consider the importance of burstiness in capacity planning no matter if it exists in the arrival process or the service process of one queue.

Observe also that while the maximum number of clients in this closed system is fixed, the number of clients receiving service from the system does fluctuate. Figures 10–13 show the number of clients receiving service (i.e., the first column in the figures) under four different workloads: (1) no burstiness in the system, see Fig. 10, (2) when burstiness is present in the front server's service process, see Fig. 11, (3) when burstiness is present in the database server's service process, see Fig. 12, and (4) when burstiness is present in the arrival process to the front server, see Fig. 13. The transient utilization levels at the front and back-end database servers are also depicted in the figures, see the corresponding second and third columns. Observe that when there is no burstiness in the system, the number of clients is quite low and no traffic congestion exists in the system. As a result, the utilizations at both the front and database servers are highly variable across time only and the best performance is obtained among all four workloads. In contrast, non-negligible burstiness is found under the other three workloads: when there is burstiness in the arrivals to the front server, we observe intensive traffic surges (i.e., bursts in the number of clients), as well as the corresponding burstiness in the front and database utilizations; and when there is burstiness in the front (resp. database) service times, we observe strong burstiness in the front (resp. database) utilizations but negligible burstiness in the database (resp. front) utilizations across times.

Furthermore, due to the propagation of burstiness, the number of clients under workloads *front-server-bursty* and *db-server-bursty* presents the burstiness, see Figs. 11 and 12(a), which however is not as strong as the one under the workload with bursty arrivals, see Fig. 13(a). It follows that both the front and the database servers experience longer saturated periods (i.e., the system utilization almost reaches 100%) under the case of bursty arrivals, see Fig. 13(b) and (c). All these results give the further explanation about the poor performance of bursty workloads in Fig. 9: it is more difficult for the system to recover when there is a huge accumulation of jobs and this is immediately reflected in the user-perceived performance.

[4] In this paper, we focus on performance models for capacity planning to represent the client–server systems in terms of their performance.

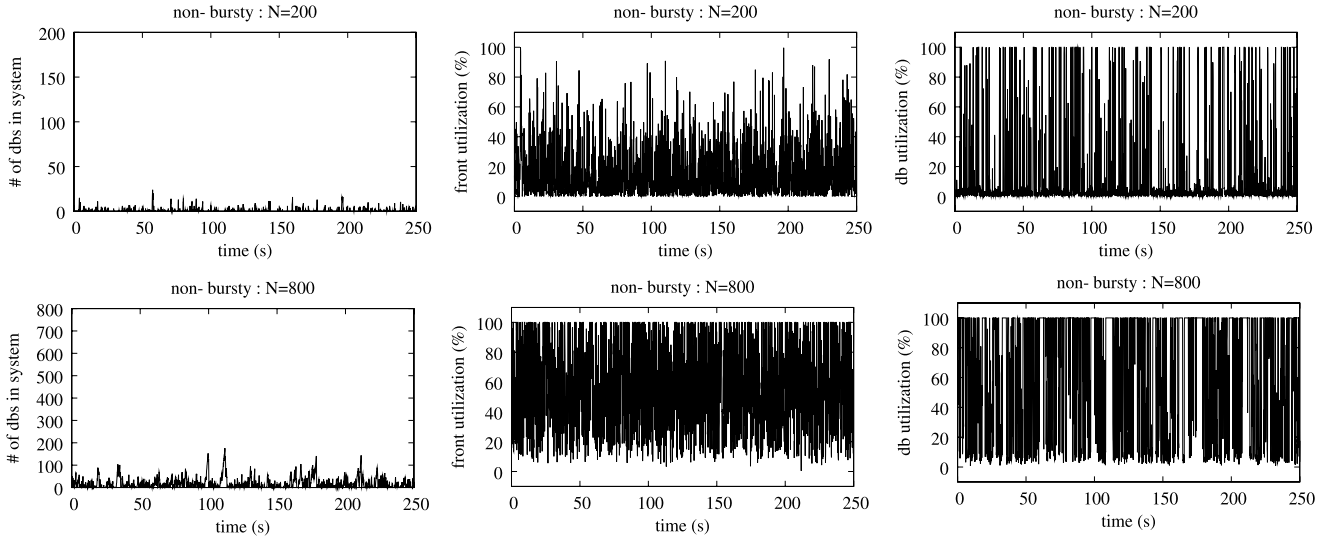

Fig. 10 Illustrating number of requests in the servers, transient utilizations at the front server, and transient utilizations at the back-end database, when there is no burstiness in systems

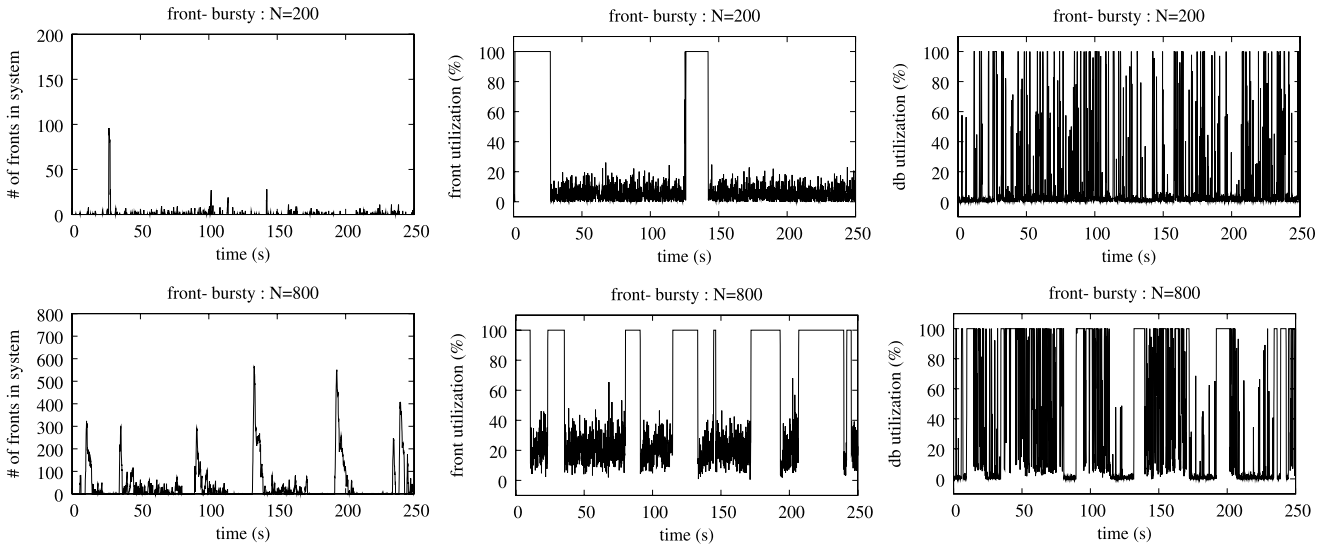

Fig. 11 Illustrating number of requests in the servers, transient utilizations at the front server, and transient utilizations at the back-end database, when burstiness is present in the front server's service process

From the implementation point of view, if one wants to introduce a burstiness "knob" in the benchmark, it is much harder to introduce and control burstiness at the front or the database tiers of the system without significantly changing the TPC-W implementation and possibly even application processing functionality. The most natural, simple, and controllable place of introducing burstiness is at the arrival process, i.e., at the client side. Therefore, in this section, we introduce a new module into TPC-W that injects burstiness into the arrival process in a controllable manner and thus enables detailed performance studies for evaluating system performance degradation due to burstiness. Most importantly, burstiness in the arrivals to the multi-tier system also capture the performance effect of traffic surges.

4.2 Limitations of standard TPC-W

The standard TPC-W benchmark implements a fixed number of emulated browsers in the system that is equal to the maximum number of client connections. Each emulated browser sends requests in the system with an average think time $E[Z]$ that represents the time between receiving a web page and the following page download request. Fluctuations of the number of jobs in the system is regulated by the average user think time $E[Z]$.

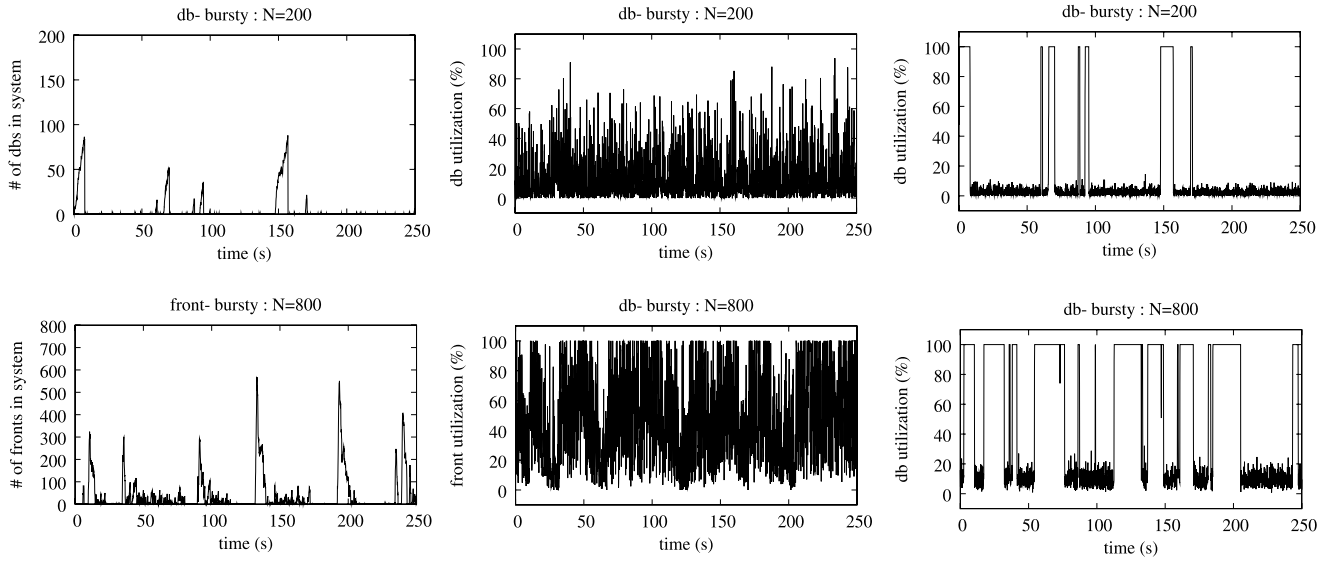

Fig. 12 Illustrating number of requests in the servers, transient utilizations at the front server, and transient utilizations at the back-end database, when burstiness is present in the database server's service process

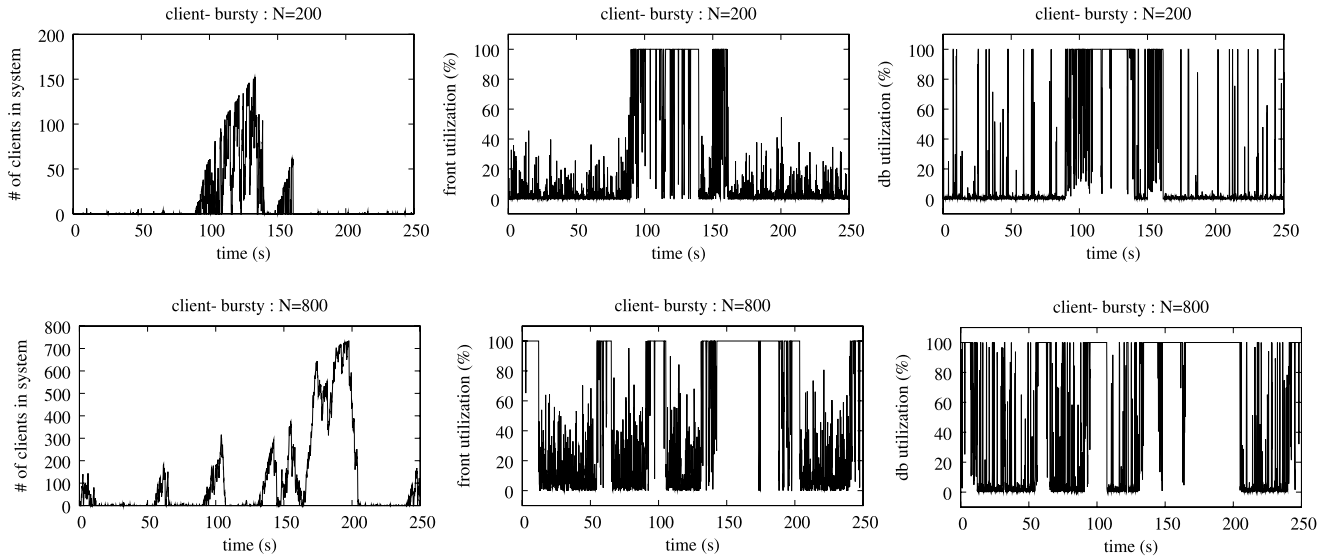

Fig. 13 Illustrating number of requests in the servers, transient utilizations at the front server, and transient utilizations at the back-end database, when burstiness is present in the arrival process to the front server

Here, we propose to inject burstiness into the incoming traffic by modifying the way think times are generated in the client machines. Think times in the standard TPC-W benchmark are drawn randomly from an exponential distribution that is identical for all clients [18]. Because of the memoryless property of the exponential distribution, this is equivalent to imposing that clients operate independently of their past actions. However, exponential think times are incompatible with the notion of burstiness for several reasons:

Temporal locality: intuitively, under conditions of burstiness, arrivals from different customers cannot happen at random instants of time, but they are instead condensed in short periods across time. Therefore, the probability of sending a request inside this period is much larger than outside of it. This behavior is inconsistent with classic distributions considered in performance engineering of web architectures, such as Poisson, hyper-exponential, Zipf, and Pareto, which all miss the ability of describing temporal locality within a process.

Variability of different timescales: Variability within a traffic surge is a relevant characteristic for testing peak performance degradation. Therefore, a benchmarking model for burstiness should not only create surges of variable in-

tensity and duration, but also create fluctuations within a surge. This implies a hierarchy of variability levels that cannot be described by a simple exponential distribution and instead requires a more structured arrival process.

Lack of aggregation: in the standard TPC-W, each thread on the client machines uses a dedicated stream of random numbers, thus think times of different users are always independent. This is representative of normal traffic, but fails in capturing the essential property of traffic surges: users act in an aggregated fashion which is mostly incompatible with independence assumptions.[5] As remarked in Sect. 5, this is a common problem to many request generation techniques based on the user-equivalent approach [10].

In order to address all the above points, we propose to regulate the arrival rate of requests to the system using a class of Markov-modulated processes known as Markovian Arrival Processes (MAPs) [34], which have the ability of providing variability at different levels as well as temporal locality effects. Recent work in [17] proposed two new metrics, i.e., marginal entropy and coefficient of variation, to capture temporal locality of web reference streams; however, the coefficient of variation used as a metric in [17] is not sufficient to measure the correlation component of temporal locality. Thus, we use a MAP parameterized by the index of dispersion to create sequences of surges with different intensities and durations in the following sections.

4.3 A turnable burstiness knob and its realistic values

A MAP can be seen as a simple mathematical model of a time series, such as a sequence of think times, for which we can accurately shape distribution and correlations between successive values. Correlations among consecutive think times are instrumental to capture periods of the time series where think times are consecutively small and thus a surge occurs, as well as to determine the surge duration.

We use a class of MAPs with two states only, one responsible for the generation of "short" think times implying that users produce closely spaced arrivals, possibly resulting in surges, while the other is responsible for the generation of "long" think times associated with periods of normal traffic. In the "short" state, think times are generated with mean rate λ_{short}, similarly they have mean rate $\lambda_{\text{long}} < \lambda_{\text{short}}$ in the "long" state. We explain in Sect. 4.4 how to assign values for λ_{short} and λ_{long} starting from standard TPC-W measurements. In order to create correlation between different events, after the generation of a new think time sample, our model has a probability $p_{s,s}$ that two consecutive think times are short and a different probability $p_{l,l}$ of two consecutive think times being both long. The probability $p_{s,l} = 1 - p_{s,s}$ (resp., $p_{l,s} = 1 - p_{l,l}$) determines the frequency of jump from the short (resp., long) state to the long (resp., short) state. Thus, the values of $p_{s,s}$, $p_{s,l}$, $p_{l,s}$ and $p_{l,l}$ shape the correlations between consecutive think times and are instrumental to determine the duration of the traffic surge; see the next subsection for further details. Henceforth, we focus only on the independent values $p_{l,s}$ and $p_{s,l}$.

In order to gain intuition on the way this model works, we provide the following pseudo-code to generate a sample of n_t think time values $Z_1, Z_2, \ldots, Z_n, \ldots, Z_{n_t}$ from a MAP parameterized by the tuple $(\lambda_{\text{long}}, \lambda_{\text{short}}, p_{l,s}, p_{s,l})$:

```
function: MAP_sample(λ_long, λ_short, p_{l,s}, p_{s,l}, n_t)
/* initialization in normal traffic state */
active_state = "long";
for n = 1, 2, ..., n_t
/* generate sample in current state */
        Z_n = sample from exponential distribution
                with rate λ_active_state;
/* update MAP state */
        r = random number in [0, 1];
        if active_state ="long" and r ≤ p_{l,s}
                active_state = "short";
        else if active_state = "short" and r ≤ p_{s,l}
                active_state = "long";
        end
end
```

Figure 14 summarizes the traffic surge model described above. Note from the pseudo-code that the problem of variability of different timescales is solved effectively in MAPs: if the MAP is in a state i, then samples are generated by an exponential distribution with rate λ_i associated with state i. This creates fluctuations within the traffic surge. It is also compatible with the observations in Sect. 4.2 against the exponential think times because the probability of arrival inside the traffic surge is larger than outside of it, due to the state change mechanism that alters the rate of arrival from λ_{long} to λ_{short}.

We propose to use the *index of dispersion* as a regulator of the intensity of traffic surges. The index of dispersion I has the fundamental property that it grows proportionally with

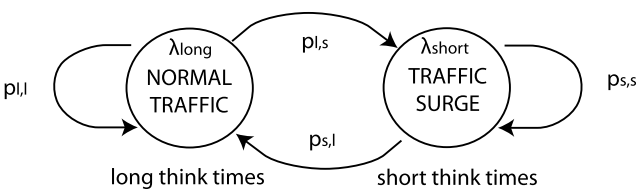

Fig. 14 Model of traffic surges based on regulation of think times

[5] As already observed in the introduction, we do not assume in any point of this paper that users explicitly coordinate their submission of requests. Instead, we impose a loose synchronization which leaves large room for fluctuations within a traffic surge.

both variability and correlations, and thus can be immediately used to identify burstiness in a trace. When there is no burstiness, the value of I is equal to the squared coefficient-of-variation of the distribution, e.g., $I = SCV = 1$ for the exponential distribution, while it grows to values of thousands on bursty processes. We point to the three real traces in Fig. 2 of Sect. 2 for a graphical outlook of how the values of I capture the intensity of burstiness in workloads. For example, 1998 FIFA World Cup website trace [5] presents dramatic traffic surges caused by particular important sport events, which results in the values of I slightly larger than 6300, see Fig. 2(a). Thus, a parameterization of I spanning a range from single to multiple digits can give a good sense of scalability between workloads with "no burstiness" and workloads with "very high burstiness."

4.4 Integrating burstiness in TPC-W

To avoid inter-machine communication and keep the modifications to TPC-W simple, we propose to use a *shared* MAP process to draw think times for all users emulated on the same client machine.[6] This solves immediately the problem of independence between requests of different users and is a paradigm change, because *we no longer model in the TPC-W benchmark the individual think times; instead, we shape directly the behavior of all clients*.

The most complex aspect of this new approach is the parameterization of the MAP process: how should we define the arrival stream in order to stress effectively a system? The fundamental problem is how to determine a parameterization (λ_{long}, λ_{short}, $p_{l,s}$, $p_{s,l}$) that produces a sequence of surges in the incoming traffic that is always capable of stressing the system and highlighting scalability problems. Further, this parameterization must remain representative of a realistic (i.e., probabilistic, non-DDoS-like) scenario. Henceforth, we assume that the user gives to the modified TPC-W benchmark the desired values of the mean think time $E[Z]$ and of the index of dispersion I which specifies the burstiness level. The benchmark automatically generates a parameterization (λ_{long}, λ_{short}, $p_{l,s}$, $p_{s,l}$) capable of stressing the system. We also assume that the standard TPC-W benchmark has been previously run on the architecture and that the mean service demand $E[D_i]$ of each server i has been estimated from utilization measurements, e.g., using linear regression methods [49].

The mean think time $E[Z]$ can be parameterized as in the standard TPC-W benchmark as $Z = 7$ seconds, while the index of dispersion I is the additional parameter that can be used to tune the level of burstiness of the benchmark. Our approach to fully define the properties of MAP think times other than the mean $E[Z]$ starts by the following parameterization equations:

$$\lambda_{\text{short}}^{-1} = \left(\sum_i E[D_i]\right) \Big/ f, \tag{3}$$

$$\lambda_{\text{long}}^{-1} = f \max\left(N\left(\sum_i E[D_i]\right), E[Z]\right). \tag{4}$$

Here, $f \geq 1$ is a free parameter, N is the maximum number of client connections considered in the benchmarking experiment, $\sum_i E[D_i]$ is the minimum time taken by a request to complete at all servers, and $N(\sum_i E[D_i])$ provides an upper bound to the time required by the system to respond to all requests. Equation (3) states that, in order to create surges, the think times should be smaller than the time required by the system to respond to requests. Thus, assuming that all N clients are simultaneously waiting to submit a new request, one may reasonably expect that after a few multiples of $\lambda_{\text{short}}^{-1}$ all clients have submitted requests and the architecture has been yet unable to cope with the traffic surge. Conversely, (4) defines think times that on average give to the system enough time to cope with any request, i.e., the normal traffic regime. Note that the condition $\lambda_{\text{long}}^{-1} \geq f E[Z]$ is imposed to ensure that the mean think time can be $E[Z]$, which would not be possible if both $\lambda_{\text{short}}^{-1} > \lambda_{\text{long}}^{-1} > E[Z]$ since $f > 1$ and in MAPs the moments $E[Z], E[Z^2], \ldots$ are

$$E[Z^k] = k!\left(\frac{p_{l,s}}{p_{l,s} + p_{s,l}}\lambda_{\text{short}}^{-k} + \frac{p_{s,l}}{p_{l,s} + p_{s,l}}\lambda_{\text{long}}^{-k}\right). \tag{5}$$

The above formula for $k = 1$ implies that $E[Z]$ has a value in-between of $\lambda_{\text{short}}^{-1}$ and $\lambda_{\text{long}}^{-1}$, which is not compatible with $\lambda_{\text{short}}^{-1} \geq \lambda_{\text{long}}^{-1} \geq f E[Z]$. According to the last formula, the MAP parameterization can always impose the user-defined $E[Z]$ if

$$p_{l,s} = p_{s,l}\left(\frac{\lambda_{\text{long}}^{-1} - E[Z]}{E[Z] - \lambda_{\text{short}}^{-1}}\right), \tag{6}$$

and we use this condition in the modified TPC-W benchmark to impose the mean think time.

In order to fix the values of $p_{s,l}$ and f in the above equations, we first carry a simple search on the space ($0 \leq p_{s,l} \leq 1$, $f \geq 1$) where at each iteration we check the value of the index of dispersion I and lag-1 autocorrelation coefficient ρ_1 from the current values of $p_{s,l}$ and f. We stop searching when we find a MAP with an I that is within 1% of the target user-specified index of dispersion and the lag-1 autocorrelation is at least $\rho_1 \geq 0.4$ in order to have consistent probability of formation of surges within short time periods. Here, the threshold 0.4 has been chosen since it is the closest round value to the maximum autocorrelation that can

[6] Often, TPC-W setup involves multiple client machines to generate enough user requests to load the benchmarked system.

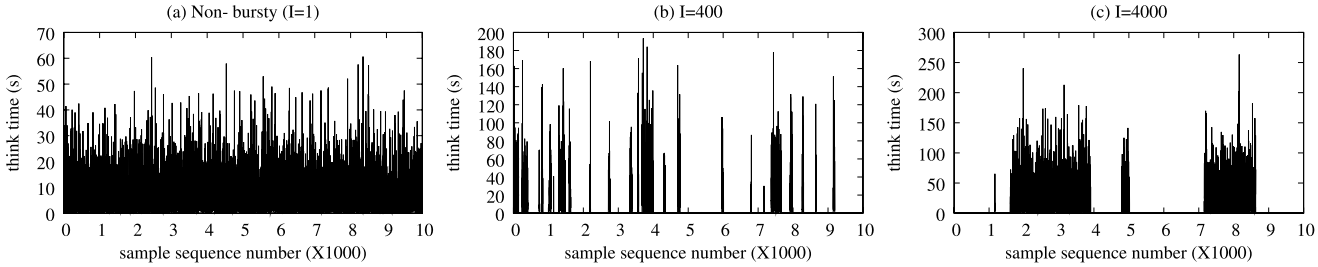

Fig. 15 User think times for the shopping mix with (**a**) non-bursty (standard TPC-W), (**b**) $I = 400$, and (**c**) $I = 4000$

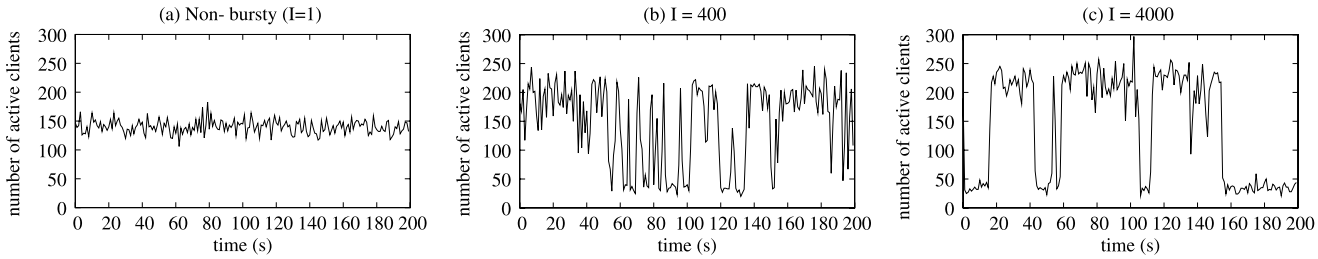

Fig. 16 Arriving clients to the system (front server) for the shopping mix with (**a**) non-bursty (standard TPC-W), (**b**) $I = 400$, and (**c**) $I = 4000$ in user think times, where the maximum number of client connections is set to $N = 1000$

be obtained by a two-state MAP. The index of dispersion of the MAP can be evaluated at each iteration as [7] [11, 34]

$$I = 1 + \frac{2\, p_{s,l}\, p_{l,s} (\lambda_{\text{short}} - \lambda_{\text{long}})^2}{(p_{s,l} + p_{l,s})(\lambda_{\text{short}}\, p_{s,l} + \lambda_{\text{long}}\, p_{l,s})^2}, \tag{7}$$

while the lag-1 autocorrelation coefficient is computed as

$$\rho_1 = \frac{1}{2}(1 - p_{l,s} - p_{s,l})\left(1 - \frac{E[Z]^2}{E[Z^2] - E[Z]^2}\right), \tag{8}$$

where $E[Z^2]$ is obtained from (5) for $k = 2$. We remark that if no MAP exists with at least $\rho_1 \geq 0.4$, then the benchmark should search for the MAP with largest ρ_1 in order to facilitate the formation of surges which persists over several units of time.

4.5 Experiments

In order to demonstrate our modified TPC-W benchmark, we conduct detailed experimentation in the TPC-W testbed under three standard transaction mixes. For each transaction mix, we run a set of experiments with different number of maximum client connections (fixed within each experiment) ranging from 200 to 1200. As a result, we evaluate the new methodology under various system loads with utilization levels at the front and the database servers within the range of 12–98% and 6–74%, respectively. In all experiments, the average user think time is set to $E[Z] = 7$ sec, which is the default value for the TPC-W benchmark. We use a two-state MAP to generate the user think times as described in the previous section. Our experiments are done with two different MAPs that result in index of dispersion equal to $I = 400$ (mild burstiness) and $I = 4000$ (severe burstiness).

For comparison, we also perform experiments with the standard configuration, i.e., think times are exponentially distributed with mean $E[Z] = 7$ seconds and squared coefficient-of-variation $SCV = 1$. All experiments are run for 3 hours each, where the first 5 minutes and the last 5 minutes are considered as warm-up and cool-down periods and thus omitted in the measurements.

Figure 15 illustrates the user think times under the shopping mix, which are generated by the standard TPC-W and our extended TPC-W with $I = 400$ and $I = 4000$ in MAPs. Clearly, the user think times in the standard TPC-W benchmark are exponentially distributed, see Fig. 15(a), while mild and strong burstiness is presented in user think times under the two MAPs. Consequently, the two MAPs (with $I = 400$ and $I = 4000$) for user think times inject the burstiness into the arrival process. Figure 16 demonstrates the arrival processes to the system under the shopping mix,[8] where we depict the number of arriving clients to the system (i.e., the front server) in monitoring windows of 1 sec-

[7] Note that (7) slightly differs in the denominator from other expressions of I, such as those reported in [19], because here we consider a MAP that is a generalization of an MMPP process.

[8] The results for the browsing and the ordering mixes are qualitatively the same and are not presented here due to lack of space.

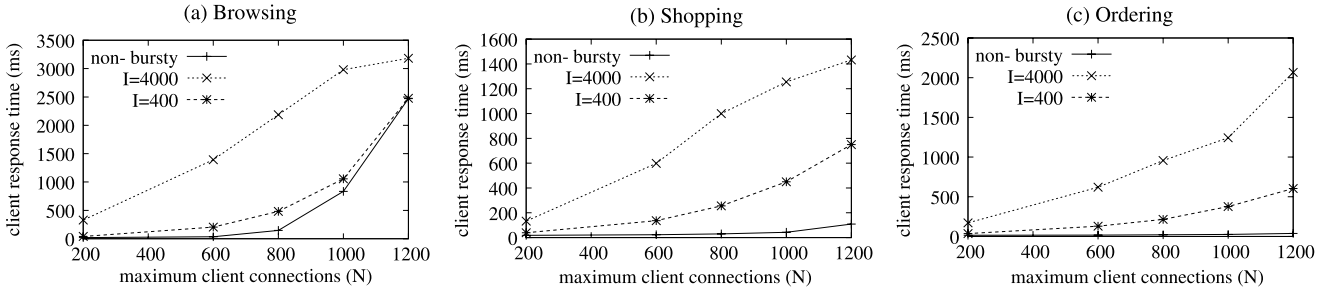

Fig. 17 Average latencies as a function of the number of maximum client connections N for (**a**) browsing mix, (**b**) shopping mix, and (**c**) ordering mix with non-bursty and bursty of $I = 4000$ and 400 in the user think times

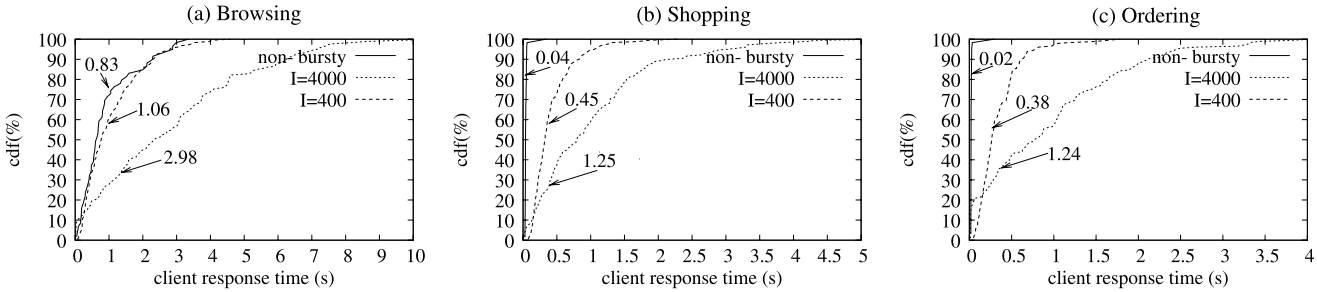

Fig. 18 CDFs of latencies for (**a**) browsing mix, (**b**) shopping mix, and (**c**) ordering mix with non-bursty and bursty of $I = 4000$ and 400 in user think times, where $N = 1000$ and the corresponding average latencies are also marked

ond. In the standard TPC-W experiment, there is no burstiness in the number of arriving clients, which remains stable around 150, see Fig. 16(a). When we adopt two-state MAPs in think times, surges are generated in the arrivals as shown by periods of continuous peak arrival rates, see Fig. 16(b) and (c). We stress that all three arrival processes have the same mean. As the index of dispersion increases from $I = 400$ to $I = 4000$, there are sharp surges in the number of active clients, consistently with our purpose to "create" bursty conditions.

Figure 17 presents the *average latency* for a client transaction, which is the interval from the moment when the client sends an HTTP request to the moment when an entire HTTP web page (including embedded objects) is retrieved. We first direct the reader's attention to the system performance under the standard TPC-W experiment (i.e., exponential think times, labeled *non-bursty* in Fig. 17, see all solid curves). As shown in Fig. 17 across all workloads, average latencies increase as the maximum number of client connections increases. Especially for the browsing mix, the latency becomes two orders of magnitude larger when N is increased from 200 to 1200. This is due to the presence of burstiness in the service times at the database server, which dramatically degrades the overall system performance. For the shopping and the ordering mixes, there is no burstiness in neither the front nor the database service processes, although these two workload mixes are highly variable. Consequently, a large number of clients do not deteriorate their performance as severely as in the browsing mix.

When burstiness is injected into the arrival flows, the overall system performance becomes significantly worse for all three transaction mixes. For instance, for the shopping and the ordering mixes, when the index of dispersion in the two-state MAP for user think times is $I = 4000$ and the maximum number of client connections is beyond 600, the average latency is increased by at least 13 times and 35 times, respectively, compared to the non-bursty case. As the index of dispersion decreases, e.g., $I = 400$, the degradation caused by burstiness on the overall system performance becomes weaker yet visible as latencies remain at least 6 times slower. For the browsing mix, the newly injected burstiness in arrivals further deteriorates average latencies. Yet, as the maximum number of client connections reaches 1200, the system performance under $I = 400$ is similar to the non-bursty case. This happens because the system is already overloaded, regardless of burstiness.

In addition to average latency values, we also evaluate the distribution of latencies. Figure 18 shows the cumulative distribution function (CDF) of the latency of the three transaction mixes when $N = 1000$. The corresponding average latencies are also marked in the figure. With bursty arrivals, the mass of clients experience significantly worse performance and much longer tails in the latency distributions. This essentially argues that QoS guarantees cannot be given for significant percentiles of the workload and further

highlights the pressing need to evaluate client–server systems under bursty conditions.

5 Related work

Capacity planning of multi-tier systems is a critical part of the architecture design process and requires reliable quantitative methods, see [27] for an introduction. Queueing models are popular for predicting system performance and answering what-if capacity planning questions [27, 43–45]. Single-tier queueing models focus on capturing the performance of the most-congested resource only (i.e., bottleneck tier): [45] describes the application tier of an e-commerce system as an $M/GI/1/PS$ queue; [37] abstracts the application tier of an 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 [43, 44] 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 [40] 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 [26], 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 [25], which fits the real data using the mixture of distributions.

However, the observations in [29] show that autocorrelation in multi-tier systems flows, which is ignored by standard capacity planning models, must be accounted for accurate performance evaluation. Indeed, [9] 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 the 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 in this paper that can capture the effects of burstiness has been first introduced in [12, 14] together with a bounding technique for approximate model solution. In [12, 14] the authors provide the theoretical methods for capacity planning under bursty workloads. But, a practical issue often encountered is that the model parameterization must be derived from limited coarse measurements. Thus, to address this issue, 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 95th percentile of service times. The index of dispersion has been frequently adopted in the networking literature for describing traffic burstiness [19, 42]; 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 index of dispersion [42]. Further results concerning the characterization of index of dispersion in MAPs can be found in [3].

To analyze performance of systems, one needs a good understanding of fundamental features and properties of web workloads. The workload of websites has been extensively studied and characterized in many research and industrial papers [1, 6, 7, 10, 16]. A number of studies of different sites identified that Internet and web traffic is bursty across several timescales and showed the importance of multiscale analysis of web requests [2, 16, 23, 28, 35]. In [23, 28], the authors consider the relationship between response time percentiles and CPU utilization for a web-based shopping system. The authors noted that for bursty workloads it is important to consider different timescales; they noted that the frequency of intervals with high or low utilization increased at a finer timescales, and this can impact SLA's guarantees for a significant portion of requests.

Several studies have shown that the arrival of requests in a web-based system is self-similar [16, 28]. Self-similar workloads exhibit significant request correlations or bursts over multiple timescales [2]. A system's ability to handle such bursts is determined by its features and system resources such as the system capacity, scheduling disciplines, maximum allowable queue lengths, etc. If a system is not able to support bursts at some timescale, significant queueing delays may occur [36]. When choosing an e-commerce site's hardware and software configuration, one needs to access whether considered configurations could handle a desired load level while providing acceptable performance. Considerable effort has been focused on synthetic workload generators for traditional web-based systems [10, 21, 33]. SURGE [10] is a workload generator for testing web servers. The GEIST tool [21] attempts to match the aggregate workload characteristics and models attributes of the request arrival process at the system level. The Httperf [33] tool provides a flexible facility for generating various http workloads for measuring web server performance.

Workload models [24, 32, 39] have been recently studied to generate synthetic traces which can represent real networking traffic with the characteristics of long range dependence (LRD) and/or burstiness. For example, the multifractal wavelet model (MWM) has been developed for characterizing and synthesizing positive LRD data [39]. Later, Li [24] used the MWM to model the LRD job arrivals in Grids and Minh and Wolters [32] modified the MWM to model both LRD and burstiness in the job arrival process.

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 real system. After giving quantitative examples of the importance of integrating burstiness in performance models pointing out its role relatively to the bottleneck switch 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.

We have also developed a new methodology to explicitly introduce burstiness in a client–server benchmark. We exemplify this methodology in the well established TPC-W benchmark. Our methodology injects burstiness into the arrival process of the server in a controllable way using the index of dispersion. This simple parameterization allows the user to introduce traffic surges of different intensity into the system, thus allowing for accurate benchmarking as well as evaluation of the system under various what-if scenarios. Looking to the future, we will investigate the robustness of our methodology and focus on early detection of traffic surges and on proactive solutions ranging from load balancing to work shedding.

References

1. Almeida V, Bestavros A, Crovella M, de Oliveira A (1996) Characterizing reference locality in the WWW. In: IEEE conference on parallel and distributed information systems, Dec 1996
2. Almeida V, Arlitt M, Rolia J (2002) Analyzing a web-based system's performance measures at multiple timescales. ACM Perform Eval Rev 30(2):3–9
3. Andersen AT, Nielsen BF (1998) A Markovian approach for modeling packet traffic with long-range dependence. IEEE J Sel Areas Commun 16(5):719–732
4. Andersen AT, Nielsen BF (2002) On the use of second-order descriptors to predict queueing behavior of MAPs. Nav Res Logist 49(4):391–409
5. Arlitt M, Jin T (1999) Workload characterization of the 1998 World Cup website. Technical Report HPL-1999-35R1
6. Arlitt M, Williamson C (1996) Web server workload characterization: the search for invariants. In: Proc of ACM SIGMETRICS, pp 126–137
7. Arlitt M, Friedrich R, Jin T (1999) Workload characterization of a web proxy in a cable environment. ACM Perform Eval Rev 27(2):25–36
8. Balbo G, Serazzi G (1996) Asymptotic analysis of multiclass closed queueing networks: common bottlenecks. Perform Eval 26(1):51–72
9. Banga G, Druschel P (1999) Measuring the capacity of a web server under realistic loads. WWW 2(1–2):69–83
10. Barford P, Crovella M (1998) Generating representative web workloads for network and server performance evaluation. ACM Perform Eval Rev 26(1):151–160
11. Casale G, Zhang E, Smirni E (2007) Characterization of moments and autocorrelation in MAPs. ACM Perform Eval Rev 35(1):27–29. Special issue on MAMA workshop
12. Casale G, Mi N, Smirni E (2008) Bound analysis of closed queueing networks with workload burstiness. In: Proc of SIGMETRICS, pp 13–24
13. Casale G, Mi N, Cherkasova L, Smirni E (2010) Dealing with burstiness in multi-tier applications: new models and their parameterization (under submission)
14. Casale G, Mi N, Smirni E (2010) Model-driven system capacity planning under workload burstiness. IEEE Trans Comput 59(1):66–80
15. Cox DR, Lewis PAW (1966) The statistical analysis of series of events. Methuen, London
16. Crovella M, Bestravos A (1996) Self-similarity in Word Wide Web traffic: evidence and possible causes. In: Proc of SIGMETRICS
17. Fonseca R, Almeida V, Crovella M, Abrahao B (2003) On the intrinsic locality properties of web reference streams. Proc IEEE INFOCOM
18. Garcia D, Garcia J (2003) TPC-W e-commerce benchmark evaluation. IEEE Comput 36:42–48
19. Gusella R (1991) Characterizing the variability of arrival processes with indexes of dispersion. IEEE J Sel Areas Commun 19(2):203–211
20. Jung J, Krishnamurthy B, Rabinovich M (2002) Flash crowds and denial of service attacks: characterization and implications for CDNs and websites. In: Proc of WWW, pp 293–304
21. Kant K, Tewary V, Iyer R (2001) An internet traffic generator for server architecture evaluation. In: Proc of workshop computer architecture evaluation using commercial workloads
22. Kobayashi H, Mark BL (2009) System modeling and analysis: foundations of system performance evaluation
23. Krishnamurthy D, Rolia J (1998) Predicting the QoS of an electronic commerce server: those mean percentiles. ACM Sigmetrics Perform Eva Rev 26(3):16–22
24. Li H (2010) Realistic workload modeling and its performance impacts in large-scale eScience grids. IEEE Trans Parallel Distrib Syst 21(4):1045–9219
25. Liu Z, Niclausse N, Jalpa-Villanueva C (2001) Traffic model and performance evaluation of web servers. Perform Eval 46(2–3)
26. Liu Z, Wynter L, Xia CH, Zhang F (2006) Parameter inference of queueing models for it systems using end-to-end measurements. Perform Eval 63(1):36–60
27. Menascé DA, Almeida VAF, Dowdy WL (1994) Capacity planning and performance modeling: from mainframes to client–server systems
28. Menascé DA, Almeida VAF, Reidi R, Pelegrinelli. R. Fonesca F, Meira W Jr. (2000) In search of invariants in e-business workloads. In: Proc of ACM conf electronic commerce, pp 56–65
29. Mi N, Zhang Q, Riska A, Smirni E, Riedel E (2007) Performance impacts of autocorrelated flows in multi-tiered systems. Perform Eval 64(9–12):1082–1101

30. Mi N, Casale G, Cherkasova L, Smirni E (2008) Burstiness in multi-tier applications: symptoms, causes, and new models. In: Proc of Middleware
31. Mi N, Casale G, Cherkasova L, Smirni E (2009) Injecting realistic burstiness into a traditional client–server benchmark. In: Proc of ICAC
32. Minh TN, Wolters L (2009) Modeling job arrival process with long range dependence and burstiness characteristics. In: Proc of int'l symp on cluster computing and the grid, pp 324–330
33. Mosberger D, Jin T (1998) httperf: a tool for measuring web server performance. In: Proc of workshop internet server performance
34. Neuts MF (1989) Structured stochastic matrices of $M/G/1$ type and their applications. Dekker, New York
35. Paxon V, Floyd S (1995) Wide area traffic: the failure of poisson modeling. IEEE/ACM Trans Netw 3(3):226–244
36. Ranjan S, Rolia J, Fu H, Knightly E (2002) QoS-driven server migration for internet data center. In: Proc of IWQoS, pp 3–12
37. Ranjan S, Rolia J, Fu H, Knightly F (2002) Qos-driven server migration for Internet data centers. In: Proc of IWQoS
38. Reiser M, Lavenberg S (1980) Mean-value analysis of closed multichain queueing networks. J ACM 27(2):312–322
39. Riedi RH, Crouse MS, Ribeiro VJ, Baraniuk RG (1999) A multifractal wavelet model with application to network traffic. IEEE Trans Inf Theory 45(4):992–1018
40. Rolia J, Vetland V (1998) Correlating resource demand information with arm data for application services. In: Proc of WOSP, pp 219–230
41. Slashdot effect, Wikipedia, Oct 13, 2008. http://en.wikipedia.org/wiki/Slashdot_effect
42. Sriram K, Whitt W (1986) Characterizing superposition arrival processes in packet multiplexers for voice and data. IEEE J Sel Areas Commun 4(6):833–846
43. Urgaonkar B, Pacifici G, Shenoy P, Spreitzer M, Tantawi A (2005) An analytical model for multi-tier internet services and its applications. In: Proc of ACM SIGMETRICS, pp 291–302
44. Urgaonkar B, Shenoy P, Chandra A, Goyal P (2005) Dynamic provisioning of multi-tier internet applications. In: Proc of ICAC
45. Villela D, Pradhan P, Rubenstein D (2002) Provisioning servers in the application tier for e-commerce systems. ACM Trans Internet Technol 7(1):7
46. Williams A, Arlitt M, Williamson C, Barker K (2005) Web workload characterization: ten years later. Springer, New York
47. www.mercury.com/us/products/diagnostics. HP (Mercury) diagnostics
48. Zhang Q, Cherkasova L, Mathews G, Greene W, Smirni E (2007) R-capriccio: a capacity planning and anomaly detection tool for enterprise services with live workloads. In: Proc of Middleware, pp 244–265
49. Zhang Q, Cherkasova L, Smirni E (2007) A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In: Proc of ICAC