Injecting Realistic Burstiness to a Traditional Client-Server Benchmark *

Ningfang Mi[1] Giuliano Casale[2] Ludmila Cherkasova[3] Evgenia Smirni[1]
[1] College of William and Mary, Williamsburg, VA, USA. Email: {ningfang,esmirni}@cs.wm.edu
[2] SAP Research, CEC Belfast, UK. Email: giuliano.casale@sap.com
[3] HP Labs, Palo Alto, CA, USA. Email: lucy.cherkasova@hp.com

ABSTRACT

The design of autonomic systems often relies on representative benchmarks for evaluating system performance and scalability. Despite the fact that experimental observations have established that burstiness is a common workload characteristic that has deleterious effects on user-perceived performance, existing client-server benchmarks do not provide mechanisms for injecting burstiness into the workload. In this paper, we introduce a new methodology for generating workloads that emulate the temporal surge phenomenon in a controllable way, thus provide a mechanism that enables testing and evaluation of client-server system performance under reproducible bursty workloads. This new methodology allows to inject different amounts of burstiness into the arrival stream using the index of dispersion, a single parameter that is as simple to use as a turnable knob.

We exemplify the effectiveness of this new methodology by introducing a new module into the TPC-W, a benchmark that is routinely used for capacity planning of e-commerce systems. This new module injects burstiness into the arrival process of clients in a controllable manner, and hence, enables understanding system performance degradation due to burstiness. Detailed experimentation on a real system shows that this benchmark modification can stress the system under different degrees of burstiness, making a strong case for the usefulness of this modification for capacity planning of autonomic systems.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Reliability, availability, and serviceability; H.3.4 [Systems and Software]: Performance evaluation (efficiency and effectiveness)

General Terms: Experimentation, Performance, Measurement.

Keywords: Client-server benchmarks, burstiness, index of dispersion, performance evaluation of self-managed systems.

*This work is partially supported by NSF grants CNS-0720699 and CCF-0811417, and a gift from HP Labs.

ICAC'09, June 15–19, 2009, Barcelona, Spain.

1. INTRODUCTION

Resource provisioning in contemporary Internet systems that operate using the client-server paradigm requires 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 [23]. Traffic surges are also frequent in other contexts not immediately related to the Slashdot effect, 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*[1] 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. In such environments, autonomic resource provisioning becomes key to business effectiveness.

Benchmarking is a critical step for effective capacity planning and resource provisioning, and consequently may guide the design of autonomic systems. 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. The system behavior under high yet steady client demand may actually be very different than under bursty conditions [18, 17] and mainstream capacity planning models grossly underestimate the negative effects of burstiness. Our thesis is that, because of its tremendous performance implications, burstiness must be accounted in capacity planning and must be incorporated into benchmarking of client-server systems.

In this paper, we propose a robust methodology to inject burstiness into a traditional client-server benchmark via a simple "turnable knob". Extensive research has been carried out in recent years on mechanisms to neutralize the impact of burstiness on web architectures. However, little research has been carried out on workload benchmarks that emulate the traffic surge phenomenon and that are also easily reproducible, scalable, and representative of real workloads. In this paper, we provide a new extension to traditional client-server benchmarks, which enables testing and evaluation of client-server system performance under reproducible and controllable bursty workloads, validation of efficiency of the corresponding management/provisioning solution, and comparison across different management solutions for client-server systems in a reproducible way.

[1]We use the terms traffic burstiness and traffic surge, interchangeably.

In order to derive a credible benchmark that can emulate burstiness, we observe that standard benchmarks, such as TPC-W that is routinely used to evaluate multi-tier architecture, lack the ability to produce burstiness because user arrivals are defined by a Poisson process, i.e., they are always assumed to be independent of their past activity and independent of each other. Here we inject burstiness using simple two-state Markov-modulated processes [8, 19] 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. Markov-modulated processes can capture very well the time-varying characteristics of a workload and describe fluctuations at different time scales, 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. We provide to the user a single parameter, called the *index of dispersion* of the traffic, that controls completely the degree of burstiness in the system. The index of dispersion is used 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, it 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 performance comparisons across different architectures.

Using a real experimental testbed, we show experimentally that this methodology is able 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 code of TPC-W and related scripts at `http://www.cs.wm.edu/~ningfang/tpcw_codes/`.

The remainder of the paper is organized as follows. Section 2 presents motivation for this work. Section 3 defines the new benchmarking methodology starting from an evaluation of the standard TPC-W limitations. Detailed experimentation on a real testbed is presented in Section 4 where we demonstrate that our modified TPC-W benchmark is extremely effective in stressing system performance at different levels. A review of existing research efforts in benchmarking is given in Section 5. Finally, Section 6 draws conclusions.

2. MOTIVATION

Burstiness in the arrival streams and/or service processes is often found in client-server systems. In this section, we argue that burstiness may impact in an unexpected way the performance of different resource allocation mechanisms designed for autonomic system management, and hence testing and evaluating these mechanisms under reproducible and controllable bursty workloads is critical for autonomic system design.

Session-based admission control (*SBAC*) [9] is proposed as a mechanism to prevent e-commerce web sites from overload caused by flash crowds and unexpected bursts in arrival traffic. Typically, access to a web service occurs in the form of a *session* consisting of many individual requests. Placing an order through the web site involves further requests relating to selecting a product, providing shipping information, arranging payment agreement and finally receiving a confirmation. So, for a customer trying to place an order, or a retailer trying to make a sale, the real measure of a web server performance is its ability to process the entire sequence of requests needed to complete a transaction.

The intuition behind the SBAC mechanism is that it accepts a new session only when the system has enough capacity to process all future requests related to the session, i.e., the system can guarantee the successful session completion. If the system is functioning near its capacity, a new session will be rejected (or redirected to another server if one is available).

In [9], the authors proposed an SBAC implementation based on web server CPU utilizations. The server utilization is measured during predefined time intervals (e.g., each second). Using this measured utilization (for the last interval) and some data characterizing server utilization in the recent past, the system computes an "observed" utilization. If the observed utilization is above a specified threshold, then for the next time interval (i.e., the next second), the admission controller rejects all new sessions and will only serve the requests from already admitted sessions. Once the observed utilization drops below the given threshold, the admission controller changes its policy for the next time interval and begins admitting and processing new sessions again. The client requests enter the system via the front (web) server and are stored in the server *queue*. The limit on the queue size further controls the number of simultaneous requests processed by the server. If requests from sessions that are already accepted arrive when the server queue is full, then they are aborted. Since the *useful throughput* of the system is measured as a number of completed sessions, the aborted requests of already accepted sessions are highly undesirable because they compromise the server's ability to process all requests needed to complete a transaction and result in wasted system resources.

We have implemented the SBAC mechanism in a simulation model of a client-server system that is built according to the TPC-W specifications. The SBAC mechanism uses a server utilization threshold equal to 80% for new sessions and we explore its functionality for various fixed queue sizes under non-bursty (traditional, as defined by the TPC-W specifications) versus bursty traffic conditions. Details on how to create bursty arrivals are provided in Section 3.2. In all experiments, the session length is fixed to 5, and we set different limits on the server queue size as 250, 512, and 800.

The *aborted* session ratio, that is the percentage of aborted sessions of the already accepted sessions, is a very important measure of the effectiveness of SBAC. Table 1 illustrates the aborted ratio of requests in SBAC under non-bursty and bursty traffic conditions. While the SBAC mechanism reliably guarantees the completion of already accepted sessions under non-bursty traffic (practically no aborted sessions), the situation is very different for SBAC performance under bursty traffic: there is a high percentage of aborted sessions.

Queue size	aborted ratio (%)	
	Non-bursty	Bursty
250	0.04	11.37
512	0.00	6.28
800	0.00	2.50

Table 1: Illustrating aborted session ratio when the utilization threshold is 80%.

The considered SBAC implementation [9] based on CPU utilization may be sufficient for autonomic system management under non-bursty traffic, but it clearly exhibits its deficiency under bursty traffic conditions. It is apparent, that the admission control mechanism has to take traffic bursti-

ness into account and adapt the system configuration and threshold accordingly in order to effectively deal with changing traffic conditions.

Often, the admission control mechanism is used not only to prevent the overload conditions but to support the SLAs guarantees, i.e., to support given response time guarantees for accepted sessions. The limit on the server queue size is actively used in achieving this goal. Figure 1 shows 90-, 95-, and 98-percentiles of the session response times under non-bursty and bursty traffic arrival models. For non-bursty workloads, it is clear that setting the queue size to 250 is sufficient to guarantee 98-percentiles in user times less than 1.2 seconds. But if the workload is bursty, the response time percentiles become very sensitive to the queue size and may result in significant SLA violations. Therefore, calibrating the SBAC parameters using a non-bursty arrival assumption leads to severe SLA violations when traffic becomes bursty.

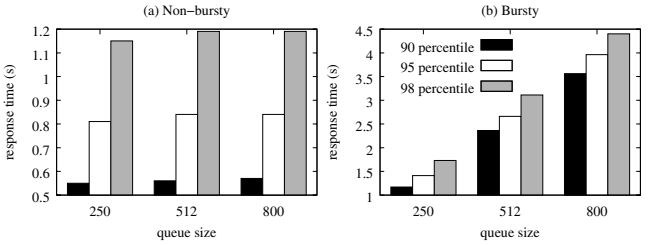

Figure 1: The percentiles of end-to-end response times when the utilization threshold is 80%.

We conclude that supporting SLA guarantees for bursty arrivals is a challenging task that involves understanding and tuning of system parameters under bursty traffic scenarios. In the remainder of the paper, by enhancing the traditional TPC-W benchmark with a turnable knob for burstiness generation, we analyze system and application performance under non-bursty versus bursty traffic in more detail. Section 4 shows that bursty traffic impacts system behavior and the observed metrics in unexpected way. For example, burstiness significantly changes the system utilization profile (it explains why SBAC performs so differently under the bursty traffic conditions). We reveal how burstiness increases the number of simultaneously active clients in the system, and how significantly this impacts the response time distribution. This further justifies that the evaluation of new mechanisms for self-managed systems under reproducible and controllable bursty workloads is a critical and necessary step in autonomic system design.

3. METHODOLOGY

In this section, we describe our approach to inject burstiness into the TPC-W benchmark. Section 3.1 discusses limitations of the standard benchmark with respect to the generation of traffic surges. A new model for think time generation is presented in Section 3.2 and its parameterization approach is introduced in Section 3.3. Section 3.4 defines the modified TPC-W benchmark.

3.1 Limitations of Standard TPC-W

TPC-W is a widely used e-commerce benchmark that simulates the operation of an online bookstore [11]. This multi-tier application uses a three-tier architecture paradigm, which consists of a web server, an application server, and a back-end database. The 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 send 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 $E[Z]$.

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 [11]. 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 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 time scales: 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 intensity 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[2]. As remarked in Section 5, this is a common problem to many request generation techniques based on the user-equivalents approach [6].

In order to address all 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) [19], which have the ability of providing variability at different levels as well as temporal locality effects. Here, we depart from the traditional approach to model increased load in the systems by simply increasing the fixed number of jobs (connections) in the system. Instead, burstiness can occur now in a system with few or many connections by simply handling the duration of user think time. In particular, we propose a new module that creates a set of identical MAPs which are replicated over the different client machines and here *shared* for generation of think times by all clients running on that particular client machine. We exemplify the fluctuation of loads in client-server systems via this new module in our experiments, see Figures 9 and 12. We further stress that this new module can be added to any benchmark with a closed loop structure for clients that use think time.

[2]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.

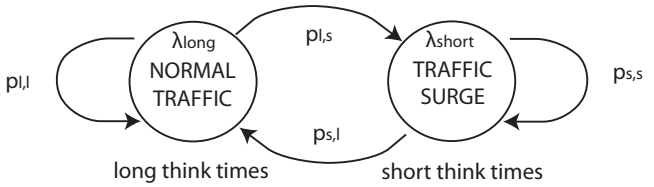

Figure 2: Model of traffic surges based on regulation of think times

3.2 Modeling Traffic Surges

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 to periods of normal traffic. In the "short" state, think times are generated with mean rate λ_{short}, similarly they have mean rate $\lambda_{long} < \lambda_{short}$ in the "long" state. We explain in Section 3.4 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, ..., Z_n, ..., Z_{n_t} from a MAP parameterized by the tuple (λ_{long}, λ_{short}, $p_{l,s}$, $p_{s,l}$, n_t):

```
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 2 summarizes the traffic surge model described above. Note from the pseudo code that the problem of variability of different time scales 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 to state i. This creates fluctuations within the traffic surge. It is also compatible with the observations in Section 3.1 against the exponential think times because the probability of arrival inside the traffic surge is larger than outside it, thanks to the state change mechanism that alters the rate of arrival from λ_{long} to λ_{short}.

3.3 A Turnable Burstiness Knob and Its Realistic Values

We propose to use the *index of dispersion* as a regulator of the intensity of traffic surges. The index of dispersion I is a measure of burstiness in networking engineering [12] and recently has been also introduced for evaluating multi-tier architectures [17].

Consider a sequence of values (e.g., think times, inter-arrival times, service times) with variability quantified by the squared-coefficient of variation (SCV), where the difference in magnitude between consecutive values is summarized by the lag-k autocorrelations[3] ρ_k. Assuming that SCV and ρ_k do not change over time, then the index of dispersion is the quantity $I = SCV\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right)$. For finite length sequences, this value can be estimated accurately, without resorting to an infinite summation, using the methods outlined in [12].

The index of dispersion I has the fundamental property that it grows proportionally with both variability and correlations, thus can be immediately used to identify burstiness in a trace. We point to the experiments in Section 4 and in particular to Figure 4 for a graphical outlook of how increasing values of I in the think times produce increasingly larger traffic surges.

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. For example, we have examined the 1998 FIFA World Cup website trace available at [4] over a period of ten days, finding that dramatic traffic surges connected to sport events can reach values of I slightly larger than 6300,[4] see Figure 3. We remark that although the 1998 FIFA World Cup trace is old, Web workload many characteristics including burstiness persist in recent years [24]. 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 "dramatic burstiness".

3.4 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[5]. 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.*

[3]We recall that the lag-k autocorrelation coefficient is a normalized measure of correlation between random variables X_t and X_{t-k}, with position in the trace differing by k lags. For a trace with mean μ and variance σ^2, $\rho_k = E[(X_t - \mu)(X_{t-k} - \mu)]/\sigma^2, k \geq 1$.

[4]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 [12], Eq. (6).

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

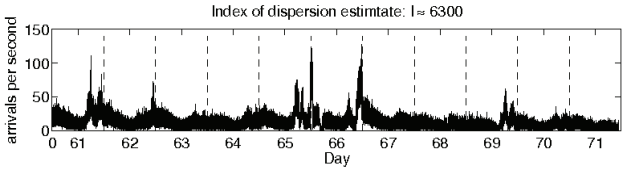

Figure 3: Burstiness of arrivals to server 0 in the 1998 FIFA World Cup trace over ten consecutive days

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 remaining 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 [25, 7].

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:

$$\lambda_{short}^{-1} = (\textstyle\sum_i E[D_i])/f, \quad (1)$$

$$\lambda_{long}^{-1} = f \max(N(\textstyle\sum_i E[D_i]), E[Z]]). \quad (2)$$

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. Eq. (1) 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 λ_{short}^{-1} all clients have submitted requests and the architecture has been yet unable to cope with the traffic surge. Conversely, (2) 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 $\lambda_{long}^{-1} \geq fE[Z]$ is imposed to ensure that the mean think time can be $E[Z]$, which would not be possible if both $\lambda_{short}^{-1} > \lambda_{long}^{-1} > E[Z]$ since $f > 1$ and in MAPs the moments $E[Z], E[Z^2], \ldots$ are

$$E[Z^k] = k! \left(\frac{p_{l,s}}{p_{l,s} + p_{s,l}} \lambda_{short}^{-k} + \frac{p_{s,l}}{p_{l,s} + p_{s,l}} \lambda_{long}^{-k} \right) \quad (3)$$

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

$$p_{l,s} = p_{s,l} \left(\frac{\lambda_{long}^{-1} - E[Z]}{E[Z] - \lambda_{short}^{-1}} \right), \quad (4)$$

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 do 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[6]. The index of dispersion of the MAP can be evaluated at each iteration as [7] [**?**, 19]:

$$I = 1 + \frac{2\, p_{s,l} p_{l,s} (\lambda_{short} - \lambda_{long})^2}{(p_{s,l} + p_{l,s})(\lambda_{short} p_{s,l} + \lambda_{long} p_{l,s})^2}, \quad (5)$$

while the lag-1 autocorrelation coefficient is computed as

$$\rho_1 = \frac{1}{2}(1 - p_{l,s} - p_{s,l}) \left(1 - \frac{E[Z]^2}{E[Z^2] - E[Z]^2} \right), \quad (6)$$

where $E[Z^2]$ is obtained from (3) 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. EXPERIMENTS

In our experimental environment, we have four Pentium D machines running Linux Redhat 9.0 with 4 GB memory each. Two Pentium D machines are used to emulate client activities. If there are N maximum number of clients in the client-server system, then each machine simulates $N/2$ clients. One Pentium D machine is used as the front server with Apache/Tomcat 5.5 installed. The fourth Pentium D machine is used as the back-end database server, which uses a MySQL 5.0 database of 10,000 items in inventory.

TPC-W defines 14 different transactions and three transaction mixes, namely the browsing mix, the shopping mix, and the ordering mix. Capacity planning with TPC-W is typically done as follows: performance measures (e.g., client end-to-end response times and system throughput) are obtained for different maximum number of client connections. Within each experiment, the maximum number of client connections is constant.

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 $E[Z] = 7$ seconds, which is the default value for the

[6]The threshold 0.4 has been chosen since it is the closest round value to the maximum autocorrelation that can be obtained by a two-state MAP.

[7]Note that Eq. (5) slightly differs in the denominator from other expressions of I, such as those reported in [12], because here we consider a MAP that is a generalization of an MMPP process.

TPC-W benchmark. We use a 2-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 do 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 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 4 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 second. In the standard TPC-W experiment, there is no burstiness in the number of arriving clients, which remains stable around 150, see Figure 4(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 Figure 4(b) and Figure 4(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.

4.1 Average Performance

Figure 5 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 Figure 5, see all solid curves). As shown in Figure 5 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, see more details in [17]. 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 does not deteriorate 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 6 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.

4.2 Transient Performance

Here, we examine the performance metrics including the transient CPU utilizations of the front and the database servers, the empirical frequencies of CPU utilizations, and the transient number of active clients in the system as given by the summation of queue lengths at the front server and at the back-end database. The maximum number of client connections in the system is fixed to $N = 1000$.[9]

The Shopping Mix. We first present CPU utilizations of the front and the database servers across time for the shopping mix. We remark that the results for the ordering mix have qualitatively the same trends. In this workload mix, there is no burstiness in either the front or the database service processes. Therefore, if burstiness in CPU utilizations exists, then this must be a direct result of surges. As shown in Figure 7(a), when there are no traffic surges, the utilization at the front server remains stable around 70% while for the database server the utilization levels vary from 10% to 80%, due to high variability in its service times. When surges are generated, the phenomenon of stable utilizations at the front server disappears. Instead, we observe very bursty CPU utilizations at the front server, where the server remains fully utilized (i.e., 100%) for some periods, but then it sharply drops to only 20% during other periods, see Figure 7(b). Meanwhile, the range for the utilizations at the database server is further enlarged up to even 100%. As the intensity of traffic surges increases, the trend for the front server being either overloaded or lightly loaded becomes more evident, see Figure 7(c).

Figure 8 illustrates the empirical frequencies (i.e., empirical PDF) of CPU utilizations at both the front server (see the first row in the figure) and the back-end database (see the second row in the figure). If the arrival process to the system is not bursty, then there is a large mass around 60%-80% in the distribution of utilizations at the front server, which is consistent with the transient results shown in Figure 7(a). For the two cases with burstiness in the arrival process, the distributions are bimodal, an effect that is further accentuated as burstiness increases, see Figure 8(c) and Figure 8(f).

To better understand how traffic surges are generated by using two-state MAPs in user think times, we present the number of active clients (i.e., summation of queue lengths at the front and the database servers) across time for the shopping mix, see Figure 9. This performance metric directly indicates how many active clients are in the system waiting for service. First, as shown in Figure 9(a), we cannot observe any burstiness in the overall queue length, despite the fact that the shopping mix workload is highly variable.

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

[9]For other numbers of maximum client connections, the performance trends are qualitatively similar.

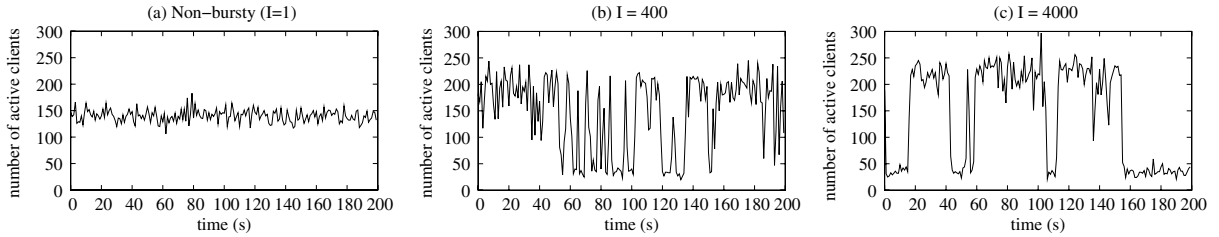

Figure 4: Arriving clients to the system (front server) for the shopping mix with (a) non-bursty (standard TPC-W), (b) $I = 400$, and (c) $I = 4000$ in user think times, where the maximum number of client connections is set to $N = 1000$.

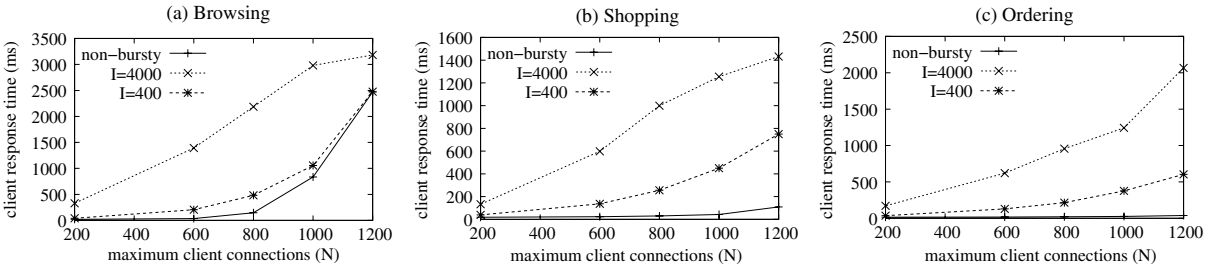

Figure 5: Average latencies as a function of the number of maximum client connections N for (a) browsing mix, (b) shopping mix, and (c) ordering mix with non-bursty and bursty of $I = 4000$ and 400 in the user think times.

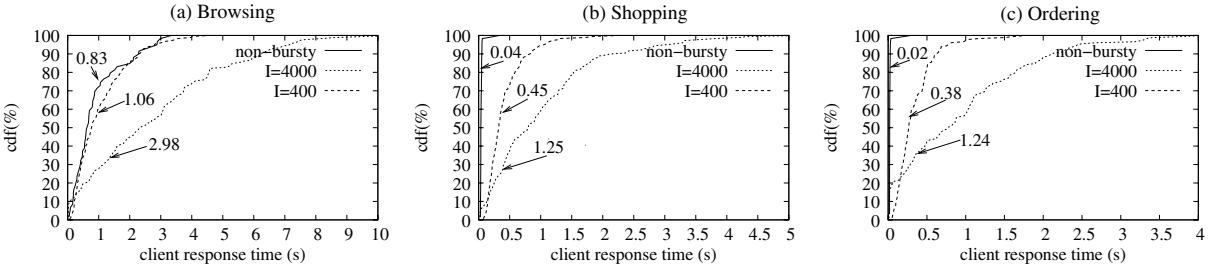

Figure 6: CDFs of latencies for (a) browsing mix, (b) shopping mix, and (c) ordering mix with non-bursty and bursty of $I = 4000$ and 400 in user think times, where $N = 1000$ and the corresponding average latencies are also marked.

When the two-state MAP with $I = 400$ and $I = 4000$ is adopted for user think times, the number of active clients in the system fluctuates dramatically. When $I = 4000$, the system is congested with more than 700 clients for some periods, while it sharply drops to as low as 10 clients during other periods. This exactly matches the burstiness in the CPU utilizations of the front and the database servers.

The Browsing Mix. We now turn to investigate the browsing mix. The distinct difference of this browsing versus shopping or ordering is that there is burstiness in the flows which originates in the database service process. We direct the reader to [17] for detailed discussion on this phenomenon.

In the browsing mix, even if no additional burstiness is injected into the system (i.e., think times are exponential), there does exist burstiness in the CPU utilizations of the front and the database servers, see Figure 10(a). If there is burstiness in think times as well, the burstiness in CPU utilizations becomes more prominent.

We depict the empirical PDF of the CPU utilizations for the browsing mix in Figure 11 for $N = 1000$. Different from the shopping mix, the database utilizations have a bimodal distribution with two peaks around 8% and 100%, this is due to the database correlated service process, see Figure 11(d). For the front server, although most of CPU utilizations are still gathered around 60%-80%, the probabilities of having the front server fully utilized (100%) and fully idle (0%) are as high as 0.16 and 0.06, respectively. CPU utilizations at the front server and the back-end database become extreme, i.e., either very high or very low.

Figure 12 illustrates the number of active clients in the system across time for the browsing mix in the network with $N = 1000$. The observation of the transient number of active clients is consistent with the transient CPU utilizations: under the non-bursty case, the curve of the number of active clients is no longer flat but contains a lot of spikes caused by the burstiness in the database service process; while the additional burstiness in the arrival process continuously increases the spikes in the number of active clients, making the system performance erratic and extremely variable.

5. RELATED WORK

The workload of web sites has been extensively studied and characterized in many research and industrial papers [2, 3, 5, 6, 10]. A number of studies of different sites identified that Internet and web traffic is bursty across several time scales and showed the importance of multiscale analysis of web requests [10, 1, 16, 21, 14]. In [16, 14], 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 time scales; they noted that the frequency of intervals with high or low utilization increased at a finer time scales, and this can impact SLA's guarantees for a significant portion of requests. We are making similar observations in our work. While at a coarser time scale the system CPU utilization may be lower, but at a finer time granularity there are many time periods with high and low CPU

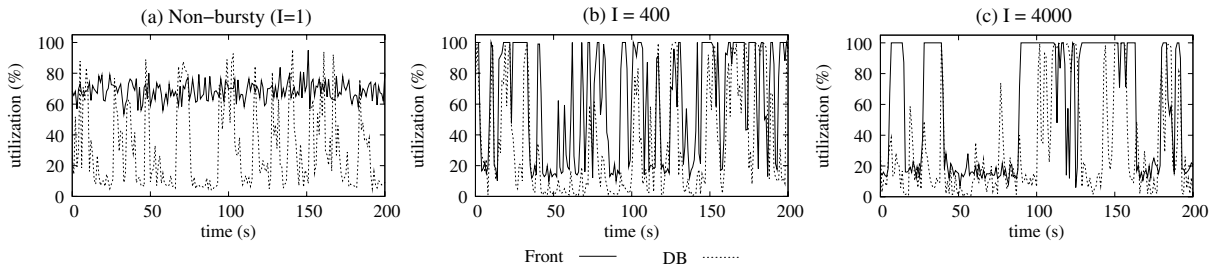

Figure 7: Shopping mix: transient utilizations at the front server and the database server for (a) non-bursty, (b) $I = 400$, and (c) $I = 4000$ in user think times, where $N = 1000$.

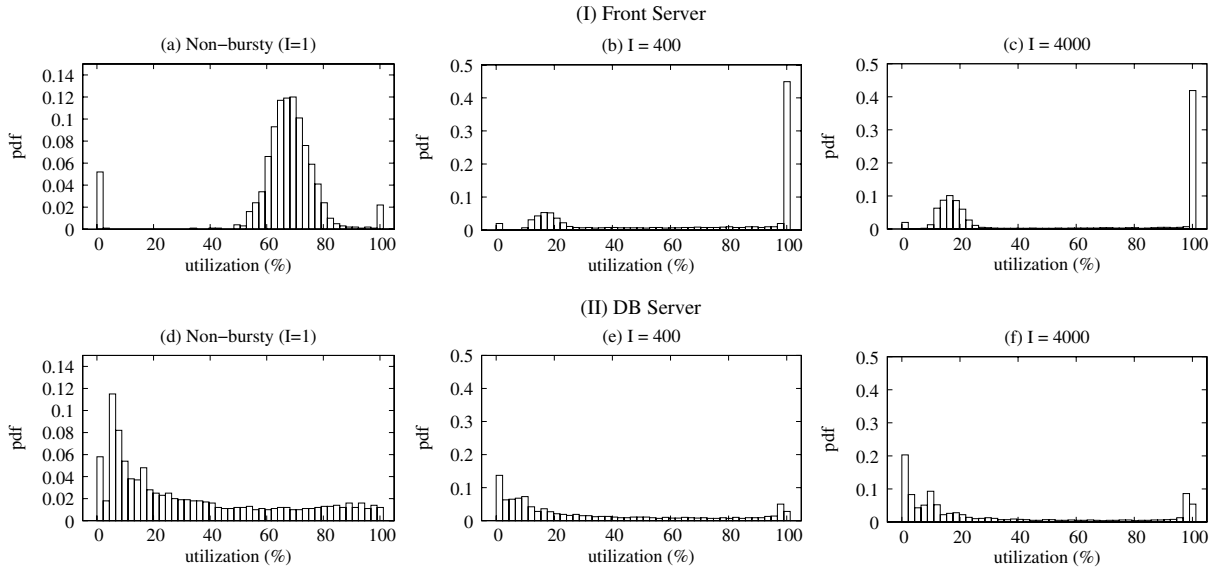

Figure 8: Shopping mix: PDFs of utilizations at (I) the front server and (II) the database server for non-bursty, $I = 400$, and $I = 4000$ in user think times, where $N = 1000$.

utilization periods, and this burstiness significantly impacts the transaction response time.

Several studies have shown that the arrival of requests in a web-based system is self-similar [10, 16]. Self-similar workloads exhibit significant request correlations or bursts over multiple timescales [1]. If a system is not able to support bursts at some timescale, significant queuing delays may occur [22]. Burstiness in TPC-W has been previously observed in the flows of a multi-tier system [18, 17]. The source of burstiness can be located in the front server [18] or in the back-end database [17] and is an effect of the hardware/software configuration of the system. The experiments in [18] and [17] are done on different platforms and show that burstiness may originate in different system components. We stress that experiments in both [18] and [17] are done using the standard TPC-W benchmark, which assumes that there is *no burstiness* in the inter-arrival times of requests into the front server. The main focus of [18, 17] is on the development of effective queueing theory models that capture burstiness.

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 [6, 13, 20, 15].

SURGE [6] is a workload generator for testing Web servers. An offline trace generation engine is used to create a trace of HTTP requests: the tool controls distributions for file size, response size, file popularity, temporal locality, and think time. Users can specify the values of parameters for the distributions or use default values as observed from a large number of traditional Web-based systems. The behavior of a typical Web user can be characterized by durations of activity followed by durations of inactivity [6]. Studies of web and e-commerce workloads have shown that both the active duration, defined as ON time, and inactive duration, defined variously as OFF time or think time, may follow heavy-tailed distributions [6, 10]. While user emulation allows easily convey many user behavioral aspects, this approach is difficult to apply for controlling or enforcing the aggregate traffic characteristics, especially the network impact on the individual user arrival process.

The two commonly used request generation techniques are user-equivalents [6] and aggregate workload generation [13]. The request generation engine submits requests in the trace to the Web server by employing user-equivalents. User-equivalents are software processes that alternate in a loop between issuing a request in the trace (ON state), waiting for the response, and lying idle for a period determined by the think time distribution (OFF state). The limitation of such an approach is that it does not permit control over aggregate request arrival process in the system, and it is not possible to specify the time instances at which requests have to arrive to the system under study. Request generation engines that employ aggregate workload generation address this limitation.

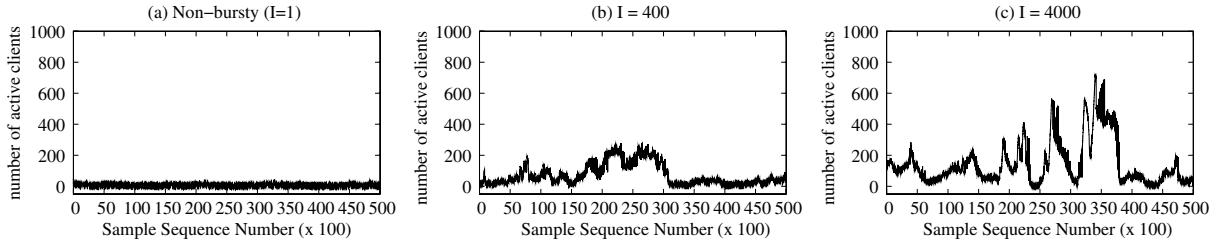

Figure 9: Shopping mix: transient number of active clients in the system, i.e., summation of queue length at the front and the database servers, for (a) non-bursty, (b) $I = 400$, and (c) $I = 4000$ in user think times, where $N = 1000$.

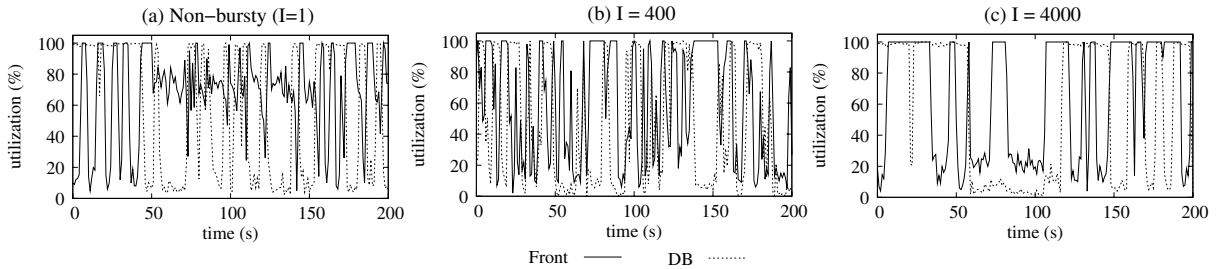

Figure 10: Browsing mix: transient utilizations at the front server and the database server for (a) non-bursty, (b) $I = 400$, and (c) $I = 4000$ in user think times, where $N = 1000$.

The GEIST tool [13] attempts to match the aggregate workload characteristics and models attributes of the request arrival process at the system level. Aggregate workload generation is useful for studies that require explicit control over the characteristics such as the distribution and correlation of inter-arrival times between successive requests to the system. However, the request generator is designed in such a way that it is possible to issue a request in a session before the response to the previous request has been received. This approach may violate inter-request dependencies specific to session-based traffic.

The Httperf [20] tool provides a flexible facility for generating various http workload for measuring web server performance. In Httperf, request generators initiate http calls in two different ways: i) under the first approach, http connections are generated deterministically, at a fixed rate, ii) in the second approach, the request generator creates sessions (sequences of requests) deterministically and at a fixed rate. The authors discuss in [20], a set of difficulties in generating uniformly distributed, average request rates. They notice that generating bursty traffic could severely affect the observed server behavior, and therefore, they make a special effort to avoid generating bursty request arrivals. The authors stress that while measuring web servers under bursty traffic may be an interesting and useful idea, there should be a special, controlled way of enforcing burstiness. However, no such mechanism is proposed by the authors.

The SWAT workload generator [15] is built on top of Httperf. SWAT can accurately emulate such workload attributes as a request mix, session length distribution, and think time distribution. There are two modes for SWAT operation. One mode is for session-based workload generation and it is based on the user emulation approach, with a drawback that it can not enforce certain aggregate traffic (and request arrival) characteristics.. The second mode – the request generation mode – allows better control on the aggregate workload generation at a price of loosing the session-based workload nature.

In our work, we implement a hybrid approach which supports session-based workload and in addition enables a fine control over the aggregate request arrival process in the system.

6. CONCLUSIONS

The evaluation of new mechanisms for self-managed systems under reproducible and controllable bursty workloads is a critical and necessary step in autonomic system design. In this paper, we provide a robust methodology to inject burstiness into the traditional client-server benchmark that can be of great practical use for assessing the effectiveness of mechanisms that counteract burstiness.

To the best of our knowledge, the work presented in this paper is the first concentrated effort to provide a robust methodology to explicitly introduce burstiness in a client-server benchmark. We exemplify the methodology in the well established TPC-W benchmark. Our methodology injects burstiness into the arrival process of the server in a controllable way using simple parameterization. Traffic burstiness is introduced using the index of dispersion, a single parameter. This simple parameterization allows the user to easily 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. Extensive experimentation in a real testbed demonstrates the effectiveness and robustness of the proposed methodology and further demonstrates the importance of evaluating the system under bursty conditions as its performance decidedly worsens as burstiness increases. The proposed extensions to TPC-W benchmark are available to download at `http://www.cs.wm.edu/~ningfang/tpcw_codes/`.

7. REFERENCES

[1] V. Almeida, M. Arlitt, J. Rolia. Analyzing a web-based system's performance measures at multiple time scales. *ACM Perf. Eval. Rev.*, 30(2), pp. 3-9, Sep. 2002.

[2] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing Reference Locality in the WWW. In IEEE Conference on Parallel and Distributed Information Systems, Miami Beach, Florida, pp. 92-103, Dec. 1996.

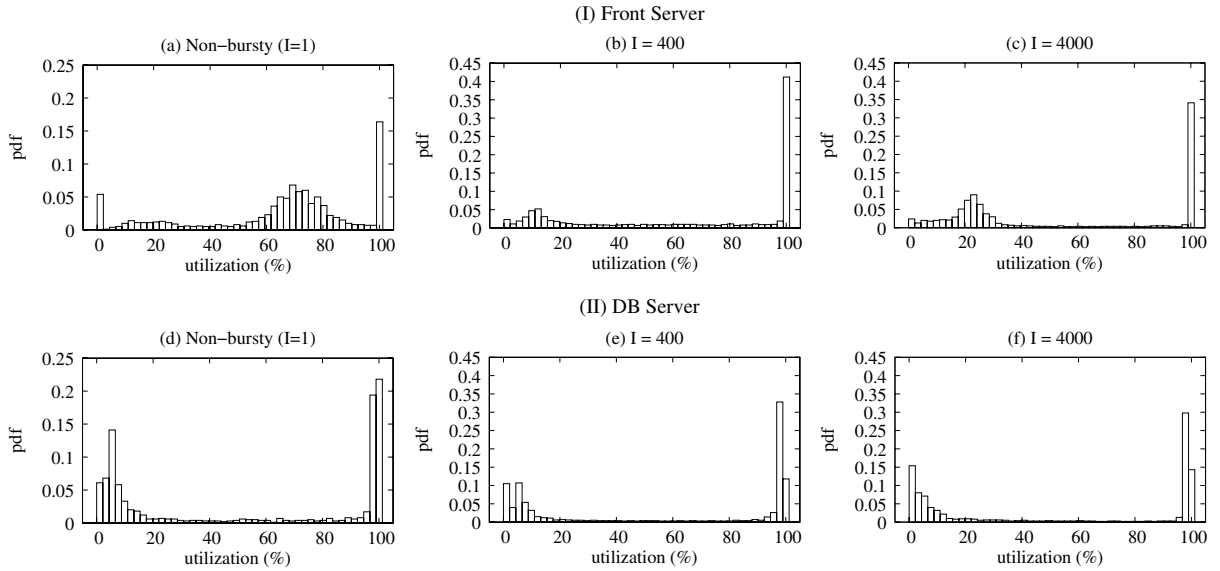

Figure 11: Browsing mix: PDFs of utilizations at (I) the front server and (II) the database server for non-bursty, $I = 400$, and $I = 4000$ in user think times, where $N = 1000$.

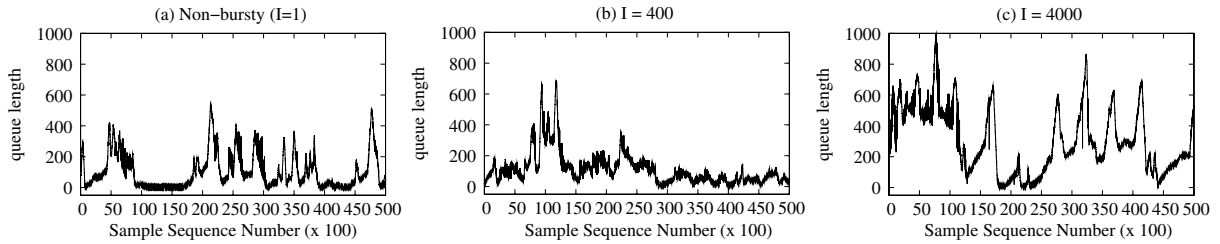

Figure 12: Browsing mix: transient number of active clients in the system, i.e., summation of queue length at the front and the database servers, for (a) non-bursty, (b) $I = 400$, and (c) $I = 4000$ in user think times, where $N = 1000$.

[3] M. Arlitt, R. Friedrich, and T. Jin. Workload Characterization of a Web Proxy in a Cable Environment. *ACM Perf. Eval. Rev.*, 27 (2), pp. 25-36, Aug. 1999.

[4] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-1999-35R1, HP Labs Technical Report, 1999.

[5] M. Arlitt and C. Williamson. Web Server Workload Characterization: the Search for Invariants. *In Proc. 1996 ACM Sigmetrics Conf. Measurement & Modeling of Computer Systems*, Philadelphia, PA, pp. 126-137, May 1996.

[6] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *ACM Perf. Eval. Rev.*, 26 (1), pp. 151-160, 1998.

[7] G. Casale, P. Cremonesi, and R. Turrin. Robust workload estimation in queueing network performance models. In *Proc. of Euromicro PDP*, pp. 183-187, 2008.

[8] G. Casale, E. Zhang, and E. Smirni. KPC-toolbox: Simple yet effective trace fitting using markovian arrival processes. In *Proc. of QEST*, pp. 83-92, 2008.

[9] L. Cherkasova, P. Phaal. Session Based Admission Control: a Mechanism for Peak Load Management of Commercial Web Sites. IEEE J. Transactions on Computers, (TOC), 51 (6), pp. 669-685, June 2002.

[10] M. Crovella and A. Bestravos. Self-Similarity in Word Wide Web Traffic: evidence and possible causes. IEEE/ACM Transactions on Networking, 5 (6), pp. 835-846, 1997.

[11] D. Garcia, J. Garcia. TPC-W E-commerce benchmark evaluation. *IEEE Computer*, pp. 42-48, Feb. 2003.

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

[13] K. Kant, V. Tewary, and R. Iyer. An Internet Traffic Generator for Server Architecture Evaluation. *In Proc. Workshop Computer Architecture Evaluation Using Commercial Workloads*, Jan. 2001.

[14] D. Krishnamurthy and J. Rolia. Predicting the QoS of an Electronic Commerce Server: Those Mean Percentiles. ACM Sigmetrics Performance Evaluation Review, 26 (3), pp. 16-22, December 1998.

[15] D. Krishnamurthy, J. Rolia, and S. Majumdar. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. IEEE Transactions on Software Engineering, 32 (11), pp. 868-882, Nov. 2006.

[16] D. Menasce, V. Almeida, R. Reidi, F. Pelegrinelli, R. Fonesca, and W. Meira Jr.. In Search of Invariants in E-Business Workloads. *In Proc. ACM Conf. Electronic Commerce*, pp. 56-65, Oct. 2000.

[17] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *ACM/IFIP/USENIX 9th Int'l Middleware Conf.*, Leuven, Belgium, pp. 265-286, 2008.

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

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

[20] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. *In Proc. Workshop Internet Server Performance*, pp. 59-67, June 1998.

[21] V. Paxon and S. Floyd. Wide Area Traffic: The Failure of Poisson Modeling. IEEE/ACM Trans. Networking, 3 (3), pp. 226-244, June 1995.

[22] S. Ranjan, J. Rolia, H. Fu, E. Knightly. QoS-Driven Server Migration for Internet Data Center. *In Proc. Int'l Workshop Quality of Service*, pp. 3-12, May 2002.

[23] Slashdot effect, Wikipedia.

[24] A. Williams and M. Arlitt and C. Williamson and K. Barker. Web Workload Characterization: Ten Years Later. 2, pp. 3-21, Springer US, 2005.

[25] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proc. of 4th ICAC*, pp. 27, June 2007.