BURN: Enabling Workload Burstiness in Customized Service Benchmarks

Giuliano Casale*, Amir Kalbasi, Diwakar Krishnamurthy*, and Jerry Rolia*
*Member, IEEE

Abstract—We introduce BURN, a methodology to create customized benchmarks for testing multi-tier applications under time-varying resource usage conditions. Starting from a set of pre-existing test workloads, BURN finds a policy that interleaves their execution to stress the multi-tier application and generate controlled burstiness in resource consumption. This is useful to study, in a controlled way, the robustness of software services to sudden changes in the workload characteristics and in the usage levels of the resources. The problem is tackled by a model-based technique which first generates Markov models to describe resource consumption patterns of each test workload. Then, a policy is generated using an optimization program which sets as constraints a target request mix and user-specified levels of burstiness at the different resources in the system. Burstiness is quantified using a novel metric called *overdemand*, which describes in a natural way the tendency of a workload to keep a resource congested for long periods of time and across multiple requests. A case study based on a three-tier application testbed shows that our method is able to control and predict burstiness for session service demands at a fine-grained scale. Furthermore, experiments demonstrate that for any given request mix our approach can expose latency and throughput degradations not found with non-bursty workloads having the same request mix.

Index Terms—Benchmarking, performance, burstiness, bottleneck migration, overdemand.

1 Introduction

Burstiness refers to workload patterns of an application that cause serial correlations in the service demands placed at various system resources. Recent work has shown that serial correlation occurs in services that rely on multi-tier infrastructures [27], [29], leading to performance degradations that are not visible for workloads with random, uncorrelated, patterns. For example, burstiness can trigger bottleneck migrations where the role of the most congested resource changes suddenly over time [27]. Despite theoretical work done to characterize such transient effects [3], [12], [25], performance prediction under burstiness remains a challenging task. Consequently, there is the need for methodologies and testing tools that may help in a practical way to assess the scalability and robustness of an application under the adverse conditions created by burstiness. Unfortunately, due to the session-oriented nature of application workloads, this is a non-trivial task for multi-tier systems. A synthetic workload for such systems must simultaneously match many different characteristics, such as ensuring a user-specified request mix or keeping a certain resource at a given utilization level, while using only semantically correct request sequences. Furthermore, the creation of controlled burstiness requires a detailed understanding of characteristics of the workload and its resource consumption which are hard to estimate, especially with resource usage monitors in production systems that collect observations only every few seconds.

In this paper, we tackle this problem by introducing the BURstiness eNabling method (BURN), an automated approach to generate customized service benchmarks with controlled burstiness. The problem solved by BURN can be formulated as follows. Consider a multi-tier application with a pre-existing set of G *test suites* $g = 1, \dots G$. A test suite is a script or client that executes service invocation scenarios such as those initiated by real users of the application. Each suite g is seen as as a randomly ordered sequence of sessions chosen from S available *session types* for a system. Each session type is a semantically correct fixed sequence of requests invoking a service of the multi-tier application. Thus a session is an instance of a given session type. Requests are assumed to belong to one of R available request types and are submitted serially within a session. Our goal is to automatically create a benchmark B that submits a sequence of sessions to the system by "drawing" sessions from the G suites, e.g., by alternating execution of the test suites over time. BURN is a technique that controls the sequence of sessions within the benchmark B such that a user-specified request mix $\boldsymbol{\rho} = (\rho_1, \dots, \rho_R)$, $\sum_r \rho_r = 1$, where $\rho_r \in [0,1]$ describes the proportion of type-r requests submitted, is matched while simultaneously causing a user-specified level of burstiness in

- *G. Casale is with Imperial College London, Department of Computing, 180 Queen's Gate, London SW7 2AZ, UK,* g.casale@imperial.ac.uk
- *A. Kalbasi and D. Krishnamurthy are with the University of Calgary, Calgary, AB, Canada,* {akalbasi,dkrishna}@ucalgary.ca
- *J.Rolia is with the Services Research Lab, HP Labs, Palo Alto, CA, USA,* jerry.rolia@hp.com

Part of the work of G. Casale and D. Krishnamurthy was done while at SAP Research CEC Belfast, The Concourse, Queen's Road, Queen's Island, Titanic Quarter, Belfast BT3 9DT, UK. This research was partially funded by the InvestNI/SAP project MORE, by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Imperial College Junior Research Fellowship. We thank Stephen Dawson for helpful feedback during the preparation of this work.

resource consumption, e.g., a given level of fluctuation in the utilization of a database server.

The proposed methodology has three steps: *characterization*, *composition*, and *search*. The characterization step automatically deduces for each suite g a service demand distribution model for a random session generated from that suite. It relies on coarse grained resource usage measurements for sessions in the test suites when executed in isolation. The composition step uses the service demand distribution models of the G test suites to describe the resource consumption pattern of benchmark B. Finally, the search step combines the results of the previous steps within a linear optimization program to search for a benchmark B that satisfies desired request mix and burstiness level. This is achieved by identifying an appropriate session submission policy that governs the sequence of execution of the G test suites.

Usage scenarios for BURN include, but are not limited to, application sizing, admission control, controller tuning, and deployment validation exercises:

- Sizing requires representative burstiness in synthetic test workloads to ensure a system can handle its load with appropriate response times. BURN can help one to validate a deployment for an application by creating limiting stress conditions that are not present in traditional benchmarks and which may reveal poor resource allocation decisions.
- Admission control determines whether an additional customer can be served by a shared application. BURN can help in verifying that an admission control policy is effective, robust to burstiness, and fair for a variety of request mixes entering the admission queue.
- In shared virtualized environments, application resource allocations may be governed dynamically by automated controllers. BURN can be helpful in ensuring that these controllers are tuned appropriately to react effectively to representative bursts in application workloads and any corresponding bottleneck migrations.
- Application deployment validation also benefits from fine control over burstiness, e.g., it can help determine the impact of burstiness on cache misses and virtual memory swapping effects that may not be present with random workloads.
- BURN may also be helpful for identifying anomalous conflicts between requests types arising from their simultaneous execution which are not evident in traditional benchmarks. For instance, we show in our validation experiments that some workload mixes result in significant slowdowns during the start of a bottleneck switch, that are not visible in uncorrelated workloads.

We remark that this paper extends our early work on generating benchmarks with controlled burstiness [7]. A main innovation in this paper, with respect to [7], is that we introduce the *overdemand*, a new metric that enables the search for a session submission policy using linear programming instead of nonlinear programming. The overdemand of a benchmark is its tendency to keep one or more resource busy serving a burst of requests for a continuous period of time. We show that BURN's policy generation approach is scalable and can be used to find a benchmark B in a few minutes for systems with hundreds of session test suites. The overdemand has important advantages over other descriptors. For example, it allows a tester to determine the minimum and maximum burstiness possible for a resource and consequently the limitations of a given benchmark with respect to its ability to generate burstiness. We propose in Section 5.3 a quantitative evaluation of BURN linear optimization approach compared to the nonlinear method for automatic benchmark synthesis we proposed in our earlier work [7]. Additionally, we present in Section 3.3 new experiments showing that the BURN workload modeling approach is justified by measurements observed in real multi-tier applications.

Summarizing, the proposed methodology has the following main advantages: (i) it causes a controlled level of burstiness in service demands using the overdemand metric; (ii) it is automated, combining pre-existing non-bursty, semantically correct sessions for the definition of a benchmark with burstiness based on the solution of an optimization program; (iii) it has wide applicability since it only requires information about mean service demands of the pre-existing sessions. Such demands can be deduced directly via system performance measurements or by using techniques such as linear regression [5], operational analysis [5], or the linear programming based Demand Estimation with Confidence (DEC) technique [36].

The remainder of the paper is organized as follows. Section 2 describes related work. The characterization step is examined in Section 3, the composition and search steps are discussed in Section 4. A case study is offered in Section 5, followed by summary and concluding remarks in Section 6.

2 Related Work

Benchmarking is a well accepted method for evaluating the behavior of software platforms [16]. In [35], benchmark-driven methods are described that support the performance engineering of customized software service instances. This paper focuses on the creation of customized benchmarks with specific properties that are used to evaluate the performance of such services.

Dujmovic describes a benchmark design theory that models benchmarks using an algebraic space and minimizes the number of benchmark tests needed to provide maximum information [14]. Dujmovic's work informally describes the concept of interpreting the results of a mix of different test suites to better predict the behavior of a customized system, but no formal method is given to

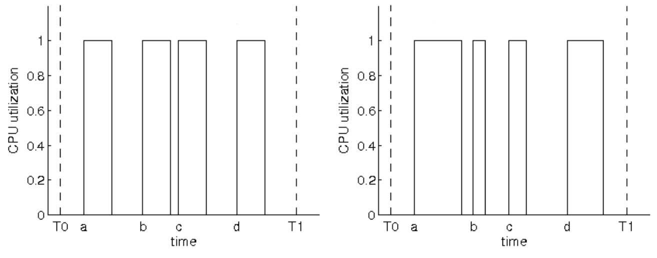
TABLE 1
Summary of Main Notation

B	benchmark generated by BURN
R	number of request types
S	number of session types
G	number of test suites
M	number of resources
$\boldsymbol{\rho} = (\rho_r)$	request mix
$\boldsymbol{\sigma} = (\sigma_s)$	session mix
$\boldsymbol{\gamma} = (\gamma_g)$	test suite mix
$U^{(j)}$	resource utilization in the jth sample period
$D_{req,r}$	service demand of type-r request on a given resource
$D_{sess,s}$	service demand of type-s session on a given resource
$D_{tst,g}$	service demand of test suite g on a given resource
D	service demand of a random session generated by B
$(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i)$	service demand model of test suite g at resource i
$(\mathbf{D}_0^i, \mathbf{D}_1^i)$	model of benchmark effects on resource i
$\mathbf{P}$	session submission policy of B
ϕ_i	benchmark overdemand at resource i
$ID(i)$	(asymptotic) index of dispersion for B at resource i
T	service time

compute the mix. Krishnaswamy and Scherson [23] also model benchmarks as an algebraic space but also do not consider the problem of finding a mix.

The approach presented in this paper is motivated by the previous work of Krishnamurthy *et al.* on synthetic workload generation for session-based systems [22]. The work has developed the Session-Based Web Application Tester (SWAT) tool. The tool includes a method that exploits an algebraic space to automatically select a subset of pre-existing semantically correct user session types from a session-based system and computes a mix of session types $\boldsymbol{\sigma}_j = (\sigma_{j,1}, \dots, \sigma_{j,s}, \dots, \sigma_{j,S})$ to achieve specific workload characteristics, where $\sigma_{j,s} \in [0,1]$ is the percentage of type-s sessions used in the submitted workload. (A summary of the notation used throughout this paper is given in Table 1.) For example, SWAT can reuse the existing sessions according to $\boldsymbol{\sigma}$ to simultaneously match a user specified request mix $\boldsymbol{\rho}$ and a particular session length distribution. It can also prepare a corresponding synthetic workload to be submitted to the system. BURN exploits and extends these concepts. SWAT can be used to find a mix of session types $\boldsymbol{\sigma}$ that matches a request mix $\boldsymbol{\rho}$ and other session properties. Then BURN decides the order in which sessions are executed to match a desired level of burstiness for service demands at system resources.

Burstiness in service demands has recently been recognized as an important feature of multi-tier systems that can be responsible for major performance degradations [29]. Service demand burstiness differs substantially from the well-understood burstiness in the arrival of requests to a system. Arrival burstiness has been systematically examined in networking [24] and there are many benchmarking tools that can shape correlations between arrivals [4], [21], [22], [28]. In contrast, service demand burstiness can be seen as the result of serially correlated service demands placed by consecutive requests on a software system [27], [29], rather than a feature of the inter-arrival times between requests. For instance, the use of caches at the disk drive, memory, database, and application layers inevitably involves temporal and spatial locality effects which introduce correlation and burstiness in the service demands [34]. It is much harder to model and predict system performance for workloads with service demand burstiness than for traditional workloads [12]. This stresses the need for benchmarking tools that support analytic and simulation techniques to study the performance impact of service demand burstiness. To the best of our knowledge, we are not aware of any benchmarking tool specifically focused on generating service demand burstiness.



(a) deterministic demand (b) variable demand

Fig. 1. Information loss due to CPU sampling. Average utilization is identical in the two samples for $[T_0, T_1]$.

3 Demand Characterization

The goal of the characterization step of BURN is to model for each test suite the probability of placing a given demand on a resource (e.g., on a CPU). Demand distributions are characterized for the resources of interest belonging to all server tiers. Specifically, BURN collects coarse-grained system measurements for each test suite and fits a set of Markov models, known as phase-type distributions [31], to model the measurements. The phase-type distributions obtained in this step, called *test suite demand models*, are inputs for the burstiness generation programs in Section 4.

3.1 Request and Session Characterization

Let us begin by considering as a resource the CPU of a server hosting a component of the multi-tier application, ignoring its dependence on other system components. Figure 1 illustrates two possible examples of CPU utilization measurements within a sample period of length $\Delta = T_1 - T_0$ seconds. In both examples, four requests of the same type are served within the time period at instants a, b, c, and d. The white boxes illustrate the demands caused by each request and their busy time within the sample period. Busy time divided by the duration of the sample period is defined as utilization.

When considering the general case of R request types, it is routine to estimate the mean service demand $E[D_{req,r}]$ of type-r requests using the count of the number of type-r requests served in the jth sample period,

$n_r^{(j)}$, and the sampled utilization $U^{(j)}$ in that period. In fact, assuming that all sample periods have identical length Δ, then utilization and the number of completed requests are related in the sample period by

$$E[U^{(j)}\Delta] = \sum_{r=1}^{R} n_r^{(j)} E[D_{req,r}], \tag{1}$$

where $E[U^{(j)}]$ is computed over all available samples $j = 1, \ldots, J$, and $U^{(j)}\Delta$ is the total busy time of the CPU during the jth sample period and thus is a summation of the service demands of the requests completed in that interval. In the above expression, the average service demands $E[D_{req,r}]$ are unknowns to be estimated. Estimates for such averages can be readily obtained using multivariate linear regression [5]. Note that we ignore the contribution to the busy period of requests that start in a sample period j and conclude execution in a different sample period $j' > j$ which is reasonable if $\Delta >> D_{req,r}$, otherwise a larger sampling period Δ should be considered.

Figure 1 illustrates the fundamental difficulty of estimating the request service demand distribution from utilization measurements. The two diagrams show different busy times imposed by the requests. Although the distribution of the service demands in the two cases shows different variabilities, the total busy time $U^{(j)}\Delta$ of the CPU in Figure 1(a) and 1(b), which is represented by the sum of the bin widths, is identical. That is, the measurement of the utilization values may result in information loss with respect to the distribution of the request service times. Unfortunately, while samples of $U^{(j)}$ are provided by standard performance monitoring tools, very long traces may be needed for inferring the second-order or third-order moments of the service demands $D_{req,r}$ by linear regression due to the difficulty in inferring the variance of the busy time reliably [7]. This poses a major challenge to modeling burstiness and injecting it in a controlled manner into synthetic workloads, which requires detailed knowledge of variability and skewness of the service demands in addition to the mean values.

The above results generalize immediately to sessions. Assume S session types, where a session is a fixed sequence of semantically-correct requests, and consider the characterization of the service demand $D_{sess,s}$ for session type $s = 1, \ldots, S$. Since the variance of the busy times is difficult to estimate, only the average service demand $E[D_{sess,s}]$ may be computed by linear regression similarly to $E[D_{req,r}]$ in (1).

3.2 Test Suite Characterization

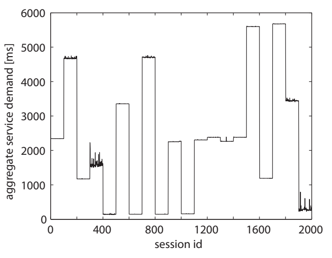
To overcome the difficulties of variance and higher-order moment estimation, we propose to evaluate the service demand distribution of a test suite, instead of each individual session or request type. Test suites may be pre-existing benchmarks that are already implemented for a system (e.g., the TPC-W workload mixes [15]) or user-defined sessions that are replayed or programmed using a workload generator (e.g., HP LoadRunner [19]). We assume in the rest of the paper that G test suites are in place for the system and that the heterogeneous sessions[1] in a suite are sent to the system in a random order.

The fundamental idea behind the test suite characterization approach is to assume that the variability of resource consumption is driven by the heterogeneity of the sessions the suite is made of, rather than due to the individual demand variability of each of them. This concept is illustrated by an experiment. We have submitted to the TPC-W testbed described later in Section 5 a mix of $2,000$ sessions drawn from the browsing mix of TPC-W. Sessions of the same type are sent serially to the TPC-W testbed with a closed-loop workload generation mechanism and without think times[2]. During the experiment, we have limited the concurrency level to a single request in the system at a time. Figure 2(a) shows the *aggregate* service demand of the $2,000$ sessions belonging to the TPC-W session types, which is the sum of the service demands at the front server and database servers, the front server being the machine that hosts both the web server and the application server. We have generated 100 sessions for each session type in the browsing mix. Consequently, each level in Figure 2(a) corresponds to a set of aggregate demands measured for sessions of the same type. Steps in the levels correspond to changes in session type.

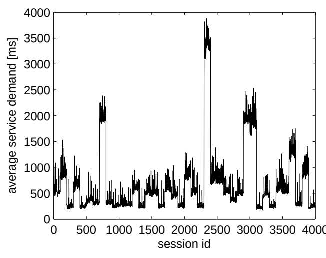
The important point in Figure 2(a) is that the extra variability shown by sessions of the same type (e.g., in the range $[300, 400]$) is negligible when compared to the total variance of the mean service demands due to the alternation of the session types, i.e., the staircase shape of the figure. This is also confirmed by Figure 2(b) which shows a similar plot obtained for $4,000$ sessions generated from the TPC-W shopping mix: the much increased variance in the demands of this test suit remains very small compared to the variability due to the staircase effect. The discussed properties of the aggregate demand can be translated, at least approximately, into similar features for the service demands. This qualitatively motivates the concept that the mix itself creates significantly larger resource consumption variability than each individual session and can be summarized in the following principle: *the main driver of the fluctuations in the resource consumption due to a mix of sessions is the variability in the*

1. For benchmarks such as TPC-W that randomize the behavior of each individual client, we consider two sessions to be identical if and only if they have the same service demand on all resources. Higher-level semantics are not considered in this paper, e.g., business semantics.

2. That is, a test suite submits the nth session only upon completion of the last request of the $n-1$th submitted session. The test suite does not wait between completion of the nth session and sending out the first request of the $(n+1)$th session. Similarly, it does not wait in between submission of consecutive requests of the same session.



(a) Browsing Mix



(b) Shopping Mix

Fig. 2. Aggregate session service demands of a TPC-W benchmark of sessions for the browsing and shopping mixes of TPC-W. The workload generator submits sessions of same type consecutively without think times; hence, each level is composed of aggregate service demand measurements of sessions of the same type.

mean service demand placed on the resources by the mix[3].

This principle leads to the following more practical approach to service demand characterization. Suppose that a characterization of the mean session service demand $E[D_{sess,s}]$ has been obtained, as discussed in the previous subsection, for each type of session s. Let $D_{tst,g}$ be the service demand of a random session drawn from the test suite g. Then the mean service demand of a random session in the test suite g is

$$E[D_{tst,g}] = \sum_{s \in g} \pi_{s,g} E[D_{sess,s}]$$

where $\pi_{s,g}$ is the probability of drawing a session of type s from test suite g. We approximate higher-order moments of $D_{tst,g}$ with those of a continuous-time Markov chain jumping randomly between values in the set $E[D_{sess,1}], \dots, E[D_{sess,S}]$. That is, we assume to characterize session service demands only by their means, thus ignoring the contribution of higher-order moments. The properties of this special class of continuous-time Markov chains are overviewed in Appendix B of [6]. This approach provides the following estimator for the variance of $D_{tst,g}$:

$$Var[D_{tst,g}] \approx \sum_{s=1}^{S} \pi_{s,g} (E[D_{sess,s}])^2 - E[D_{tst,g}]^2. \qquad (2)$$

and more generally for its moments

$$E[D^k_{tst,g}] \approx \sum_{s=1}^{S} \pi_{s,g} (E[D_{sess,s}])^k. \qquad (3)$$

Based on $E[D_{tst,g}]$ and $Var[D_{tst,g}]$ one can immediately fit, according to the procedure discussed in the next subsection, a service demand model that describes the resource consumption of test suite g. Approximation (2) applies to any resource in a multi-tier architecture and enables the characterization of resource consumption of a workload by an arbitrary number of moments $E[D^k_{tst,g}]$; in this paper we consider $k = 1, 2, 3$ which is equivalent to using mean, variance, and skewness of the distribution.

3. Note that different approaches may be used, such as characterizing the median service demand instead of the mean of each session type. However, for analytical modelling purposes, the mean is often easier to compute and characterize than median or other statistics.

3.3 Test Suite Demand Model

We are now in position to derive test suite demand models using phase-type distributions [31], which are a class of absorbing continuous-time Markov chains (CTMC) used to model probability density functions. A phase-type distribution with n states is described by a pair $(\boldsymbol{\pi}, \mathbf{A})$, where $\boldsymbol{\pi} = [\pi_1, \pi_2, \dots, \pi_n]$ is an initialization vector and

$$\mathbf{A} = \begin{bmatrix} -a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & -a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & -a_{n,n} \end{bmatrix}$$

is the CTMC infinitesimal generator satisfying $-\mathbf{A1} \geq 0$ and $a_{i,j} \geq 0$ for all elements, where $\mathbf{1}$ is a column vector of all ones. Examples of popular distributions expressed with the phase-type notation are given below.

Example 1: An exponential distribution with sample mean λ^{-1}, thus the service demand model has a single state such that

$$\boldsymbol{\pi} = [1] \qquad \mathbf{A} = \begin{bmatrix} -\lambda \end{bmatrix} \qquad (4)$$

Example 2: A three-stage Erlang distribution with mean $3\lambda^{-1}$ is obtained by summing the samples of three exponential distributions with same mean λ^{-1}, thus

$$\boldsymbol{\pi} = [1, 0, 0] \quad \mathbf{A} = \begin{bmatrix} -\lambda & \lambda & 0 \\ 0 & -\lambda & \lambda \\ 0 & 0 & -\lambda \end{bmatrix} \qquad (5)$$

Example 3: A two-stage hyper-exponential distribution with mean $p\lambda_1^{-1} + (1-p)\lambda_2^{-1}$ is obtained by a mixture of two exponential distributions having mean λ_1^{-1} and λ_2^{-1} respectively. This is modelled by a phase-type distribution

$$\boldsymbol{\pi} = [p, 1-p] \quad \mathbf{A} = \begin{bmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{bmatrix} \qquad (6)$$

In a phase-type distribution, the service time T of a session is modeled as a cumulative time to traverse a sequence of states each having exponentially-distributed sojourn time. That is, with probability π_i the session starts execution in state i, where it waits for an exponentially distributed time $t_1 \sim a_{i,i} e^{-a_{i,i} t_1}$. Next, with probability $a_{i,j}/a_{i,i}$ the state changes to j where the session sojourns for time $t_2 \sim a_{j,j} e^{-a_{j,j} t_2}$, and so forth. In each state i, there exist a probability $p_i^{abs} = 1 - \sum_{j \neq i} a_{i,j}/a_{i,i}$ that the session is "absorbed", i.e., it terminates execution upon the next jump. If a session terminates after the kth sojourn time, its cumulative passage time, i.e., sum of sojourn times, $T = t_1 + t_2 + \dots + t_k$ is recorded and the

CTMC is reinitialized according to the probability vector $\boldsymbol{\pi}$ to generate the next sample.

Phase-type distributions are useful in representing real computer system workloads because they provide a compact representation of a distribution of values using a small vector-matrix pair. Heavy-tail distributions, where very large observations occur with non-negligible probability, can be easily described by a phase-type model using a few states, which is often more convenient than an explicit histogram representation. In fact, transformations of phase-type probability distributions, such as addition or linear mixtures of two or more random variables, calculations of moments, superposition or modulation of independent flows, can be performed using basic matrix operations rather than integration or convolution of histograms. This is very useful for efficient integration of workload models into numerical optimization programs, where it would be very inefficient to compute integrals or convolutions at each iteration of a search algorithm.

By properly assigning the values in $\boldsymbol{\pi}$ and $\mathbf{A}$ one can constrain the service demand distribution of T to fit a sample distribution. In particular, the expression of the cumulative distribution function of a phase-type distribution is

$$\Pr[T \leq t] = \boldsymbol{\pi} e^{\mathbf{A}t}\mathbf{1},$$

where

$$\boldsymbol{e}^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{(\mathbf{A}t)^k}{k!} \tag{7}$$

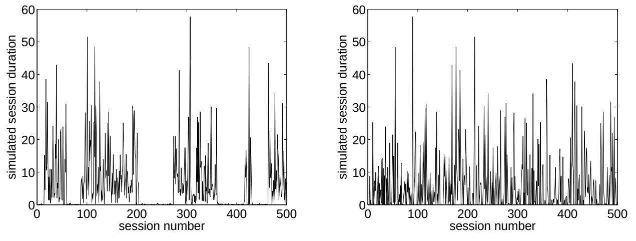
denotes the matrix exponential function evaluated at $\mathbf{A}t$. The moments of T are given by

$$E[T^k] = k!\boldsymbol{\pi}(-\mathbf{A})^{-k}\mathbf{1},$$

thus it is possible to fit $\boldsymbol{\pi}$ and $\mathbf{A}$ by the above formula to match given moments $E[D^k_{tst,g}]$. In particular, by applying standard moment matching fitting techniques, one can quickly obtain a set of phase-type distributions $(\boldsymbol{\pi}^i_g, \mathbf{A}^i_g)$, that describe the service demand moments $E[D^k_{tst,g}]$ at each resource. Throughout this work, we have fitted distributions with variability less than an exponential distribution using the GFIT tool [33]. High-variability distributions are instead fitted using the moment matching algorithms in the KPC-Toolbox [9], [11].

4 BURN Methodology

This section presents the BURN methodology for automatic generation of a benchmark B with controlled burstiness in service demands. We assume that the user provides in input a test suite mix vector $\boldsymbol{\gamma} \equiv (\gamma_1, \gamma_2, \ldots, \gamma_G)$, $\sum_g \gamma_g = 1$, where $\gamma_g \in [0,1]$ describes the percentage of sessions to be drawn from test suite g. The SWAT workload generator [22] mentioned previously can be easily adapted to determine a $\boldsymbol{\gamma}$ that also matches a desired request mix $\boldsymbol{\rho}$ and other properties such as a target session length distribution. Other inputs are the test suite demand models and a specification of the target burstiness according to the overdemand metric introduced in this section. The aim is to find a *session submission policy* $\mathbf{P}$ resulting in a benchmark B with the requested properties of burstiness and workload mix.



(a) Policy with burstiness (b) Traditional policy

Fig. 3. Generation of burstiness using the session submission policy $\mathbf{P}$. Careful selection of the state transition probabilities in the Markov chain $\mathbf{P}$ enables burstiness properties in the session demand.

4.1 Session Submission Policy

The session submission policy $\mathbf{P}$ determines the sequence of sessions submitted to the system, both in terms of their type and relative ordering. The policy works as follows. $\mathbf{P}$ is a discrete-time Markov chain

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,G} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,G} \\ \vdots & \vdots & \ddots & \vdots \\ p_{G,1} & p_{G,2} & \cdots & p_{G,G} \end{bmatrix}$$

such that after generating a session from test suite g there is probability $p_{g,h}$ that the following session will be submitted by test suite h. The challenge is how to define these probabilities to match the desired test suite mix vector $\boldsymbol{\gamma}$ and target burstiness. It is simple to show using equilibrium theory for discrete-time Markov chains [5], that if the policy satisfies

$$\boldsymbol{\gamma} = \boldsymbol{\gamma}\mathbf{P}, \tag{8}$$

then $\mathbf{P}$ generates exactly the desired test suite mix $\boldsymbol{\gamma}$. Our observation is that (8) leaves considerable flexibility in the definition of $\mathbf{P}$, which we can use to generate controlled burstiness in the demands. Specifically, (8) imposes $\boldsymbol{\gamma}$ to be the left eigenvector associated to the unit eigenvalue of $\mathbf{P}$, however the other $G-1$ eigenvectors and eigenvalues are degrees of freedom. We now present an illustrative example of this concept.

We have considered two policies $\mathbf{P}^{trad}$ for a traditional workload generation model and $\mathbf{P}^{burst}$ for a bursty benchmark. These are defined upon $G = 2$ test suites and both satisfy the same requirement $\boldsymbol{\gamma} = (0.5, 0.5)$ such that 50% of the sessions are drawn from test suite 1 and 50% from test suite 2. Consider the following definition for the two policies:

$$\mathbf{P}^{trad} = \begin{bmatrix} 0.50 & 0.50 \\ 0.50 & 0.50 \end{bmatrix}, \quad \mathbf{P}^{burst} = \begin{bmatrix} 0.99 & 0.01 \\ 0.01 & 0.99 \end{bmatrix}$$

which both satisfy (8) and differ for the fact that in $\mathbf{P}^{burst}$ there is 99% probability of consecutively sampling sessions from the same test suite g. This is a critical difference because the high probability of sampling sessions from the same state in $\mathbf{P}^{burst}$ implies the formation of bursts, as shown by the synthetically generated data in Figure 3(a). Bursts of similar magnitude do not appear in the traditional workload generation model results shown in Figure 3(b), which shows fluctuations but not a significant serial correlation. Therefore, by carefully selecting the transition probabilities in the submission policies we can create radically different behaviors in terms of service burstiness.

4.2 Bottleneck Migrations

Before stepping further into the technical details of BURN, we discuss the relation between generation of burstiness and bottleneck migrations. Bottleneck switches due to multiclass service demands have been studied theoretically [2], [3], [8], and applied to process migration in [25]. Recently, the topic has been the subject of intensive research in multi-tier applications [26], [28], [29], [38].

Stemming from the theory in [3], it is easy to show that the basic requirement for a benchmark B to create dynamic bottleneck switches between two resources i and $j \neq i$ is that there exist at least two test suites g and h that saturate i and j, respectively. Formally

$$E[D^i_{tst,g}] > E[D^i_{tst,h}] \text{ and } E[D^j_{tst,h}] > E[D^j_{tst,g}] \quad (9)$$

If (9) is satisfied, the session submission policy can alternate periods where resource i is congested with periods where j is congested. For the general case with several resources, linear programming may be needed to establish the potential bottlenecks for a system [8]. In BURN, the intensity of each burst can be controlled, at least in a statistical sense, using the overdemand metric introduced later in the paper. For example, (24)-(26), given later in the paper, can be used to control the duration of a transient bottleneck at a resource under a heavy-load regime assumption.

4.3 Composition Step and Benchmark Model

This subsection describes the composition step of BURN which predicts the service demand burstiness created at the different resources by a benchmark B that submits sessions using a policy P. A limitation of phase-type distributions is that they can be used only to generate independent and identically distributed (i.i.d) samples, i.e., a sequence of uncorrelated values that follow a given distribution. This is therefore not sufficient for prediction and control of burstiness at system resources. In order to tackle this problem, we now introduce Markovian arrival processes (MAPs) [32] to predict the service demand burstiness caused by a benchmark B under policy $\mathbf{P}$. Differently from a phase-type distribution, a MAP can describe both i.i.d and non-i.i.d. traces where there is a time-varying pattern in the samples. For example, the time series in Figure 3(b) may be represented both by MAPs and phase-type models. Instead, the temporal dependent workload in Figure 3(a) can only be described by a MAP. This illustrates why MAPs are needed for workload generation in BURN.

A MAP is defined by a pair of matrices $(\mathbf{D}_0, \mathbf{D}_1)$, where $\mathbf{D}_0$ has the same structure and interpretation of matrix $\mathbf{A}$ for phase-type distributions, while $\mathbf{D}_1 = \{d_{i,j}\}$ is a non-negative matrix such that $d_{i,j}$ is the rate of absorptions in state i after which the underlying Markov chain is reinitialized in state j. Thus absorption probabilities assume now the more general form $p^{abs}_{i,j} = d_{i,j}/a_{i,i}$. The ability of reinitializing the chain in different ways according to the last visited state allows one to create temporal dependence and serial correlation in the samples. As an example, this can be done by alternating reinitialization in states with short and long sojourn times. Such a temporal dependence is instead impossible in phase-type distributions where reinitialization is done randomly according to vector $\boldsymbol{\pi}$.

Stemming from the above definitions, the moments of a MAP for the service demand T are computed as

$$E[T^k] = k!\boldsymbol{\pi}(-\mathbf{D}_0)^{-k}\mathbf{1}, \quad (10)$$

where $\boldsymbol{\pi}$ is[4] the left eigenvector $\boldsymbol{\pi} = \boldsymbol{\pi}(-\mathbf{D}_0^{-1}\mathbf{D}_1)$. Joint moments are given by

$$E[T^k T^h_{+j}] = k!h!\boldsymbol{\pi}(-\mathbf{D}_0^{-k})(-\mathbf{D}_0^{-1}\mathbf{D}_1)^j(-\mathbf{D}_0^{-h})\mathbf{1}, \quad (11)$$

where T^k is the kth power of a service demand sample T and T^h_{+j} indicates the hth power of the jth service demand value observed after demand T.

In BURN, MAPs are useful for assessing if a given $\mathbf{P}$ is a good candidate for generating the requested level of burstiness in the system. Given a policy $\mathbf{P}$ and the test suite demand characterization obtained in Section 3, we define a burstiness model for a benchmark B relative to resource i (e.g., front server CPU, database CPU, ...) as the MAP that defines the *benchmark model*

$$(\mathbf{D}^i_0, \mathbf{D}^i_1) = compose(\boldsymbol{\pi}^i_g, \mathbf{A}^i_g, \mathbf{P})$$

where the function $compose(\boldsymbol{\pi}^i_g, \mathbf{A}^i_g, \mathbf{P})$ maps the test suite demand models $(\boldsymbol{\pi}^i_g, \mathbf{A}^i_g)$, $i = 1, \ldots, M$, and the

4. We deliberately use the symbol $\boldsymbol{\pi}$ since for a MAP that creates i.i.d. samples, this vector corresponds to the $\boldsymbol{\pi}$ vector in phase-type distributions [32].

policy $\mathbf{P}$ into the following matrices for each resource i:

$$\mathbf{D}_0^i = \begin{bmatrix} \mathbf{A}_1^i & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2^i & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_G^i \end{bmatrix} \tag{12}$$

$$\mathbf{D}_1^i = \begin{bmatrix} \mathbf{A}_{1,1}^i & \mathbf{A}_{1,2}^i & \dots & \mathbf{A}_{1,G}^i \\ \mathbf{A}_{2,1}^i & \mathbf{A}_{2,2}^i & \dots & \mathbf{A}_{2,G}^i \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{G,1}^i & \mathbf{A}_{G,2}^i & \dots & \mathbf{A}_{G,G}^i \end{bmatrix}. \tag{13}$$

where $\mathbf{A}_{g,h}^i = p_{g,h}(-\mathbf{A}_g^i)\mathbf{1}\boldsymbol{\pi}_h^i$ and $p_{g,h}$ is the probability in $\mathbf{P}$ of sampling a service demand from test suite h immediately after sampling a service demand from test suite g. The probabilistic interpretation of the blocks $\mathbf{A}_{g,h}^i$ is that they describe the rate of absorption for a sample taken from test suite g such that the Markov chain is then reinitialized with probability $\boldsymbol{\pi}_h^i$ into test suite h. This embeds the obtained phase-type distributions into a MAP. Summarizing, the $\mathbf{D}_0^i$ matrix specifies that the demand of a session generated by a test suite g follows its distribution model $\mathbf{A}_g^i$, while the $\mathbf{D}_1^i$ matrix describes the modulation due to the session submission policy $\mathbf{P}$ across the different test suites. Such a modulation determines the order of session submission and it is the action performed by the policy $\mathbf{P}$ to match target workload mix and burstiness.

Example 4: Consider a benchmark that modulates $G = 2$ test suites having service times that are exponential with mean λ_1^{-1} and Erlang-2 with mean $2\lambda_2^{-1}$, respectively. Then, recalling examples 1 through 3:

$$\boldsymbol{\pi}_1 = [1], \quad \mathbf{A}_1^i = \begin{bmatrix} -\lambda_1 \end{bmatrix},$$
$$\boldsymbol{\pi}_2 = [1, 0], \quad \mathbf{A}_2^i = \begin{bmatrix} -\lambda_2 & \lambda_2 \\ 0 & -\lambda_2 \end{bmatrix}.$$

For resource i, the benchmark demand model is thus

$$\mathbf{D}_0^i = \begin{bmatrix} -\lambda_1 & 0 & 0 \\ 0 & -\lambda_2 & \lambda_2 \\ 0 & 0 & -\lambda_2 \end{bmatrix}, \quad \mathbf{D}_1^i = \begin{bmatrix} p_{1,1}\lambda_1 & p_{1,2}\lambda_1 & 0 \\ 0 & 0 & 0 \\ p_{2,1}\lambda_2 & p_{2,2}\lambda_2 & 0 \end{bmatrix},$$

Depending on $\mathbf{P}$, the above example model can describe different correlations between the demands of the two test suites.

The fundamental result achieved by the composition step described above is that the benchmark burstiness model $(\mathbf{D}_0^i, \mathbf{D}_1^i)$ relative to the resource i allows us to estimate by closed-form analytical expressions the moments and the burstiness in the service demands for resource i that would result from the execution of benchmark B on the system. For instance, let T_i be the average session demand at resource i, then $E[T_i] = \boldsymbol{\pi}_i(-\mathbf{D}_0^i)^{-k}\mathbf{1}$, represents the average service demand for a session generated according to policy $\mathbf{P}$ in benchmark B. This may be readily used to infer the maximum session arrival rate λ in the benchmark that does not saturate any resource, i.e., such that utilization $E[U_i] = \lambda_i E[T_i] < 1$, $i = 1, \dots, M$, where $\lambda_i = V_i \lambda$ is the throughput that flows into resource i, i.e., the incoming throughput of the benchmark scaled by the number of times V_i that a session visits resource i [20]. Formulas for computing resource burstiness estimators by MAPs are given in the next section.

4.4 The Overdemand Metric

Burstiness for a service demand T can be characterized in the following ways according to the needs of the analysis:

Autocorrelation function: the autocorrelation coefficients $\rho(k)$ are standard second-order burstiness descriptors used in several studies, e.g. [29]. Formally,

$$\rho(k) = \frac{E[TT_{+k}] - E[T]^2}{E[T^2] - E[T]^2},$$

where T_{+k} is the kth service demand seen after observing T, which follows readily for MAPs using (10)-(11). To simplify interpretation, the lag-1 autocorrelation coefficient $\rho(1)$ is often used in place of $\rho(k)$, $k \geq 1$, to summarize burstiness levels.

Index of Dispersion [17], [27]: Under positive autocorrelations, burstiness levels for a resource are summarized by the *index of dispersion*

$$ID = CV^2(1 + 2\textstyle\sum_{k=1}^{\infty} \rho(k)),$$

where $CV^2 = Var(T)/E[T]^2$ is the squared coefficient of variation of the service demands. The index of dispersion of resource i for the service demand model $(\mathbf{D}_0^i, \mathbf{D}_1^i)$ is given by

$$ID(i) = 1 + 2\left(\boldsymbol{\alpha}_i \mathbf{D}_1^i \mathbf{1} - \boldsymbol{\pi}_i (\mathbf{1}\boldsymbol{\alpha}_i + \mathbf{Q}^i)^{-1} \mathbf{D}_1^i \mathbf{1}\right),$$

where $\mathbf{Q}^i = \mathbf{D}_0^i + \mathbf{D}_1^i$ can be shown to be an infinitesimal generator of a continuous time Markov chain. Here, $\boldsymbol{\alpha}_i$ (resp. $\boldsymbol{\pi}_i$) is the equilibrium probabilities of the continuous time (resp. discrete time) Markov chain $\mathbf{Q}^i$ (resp. $(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i$). That is, $\boldsymbol{\alpha}_i \mathbf{Q}^i = \mathbf{0}$, $\boldsymbol{\alpha}_i \mathbf{1} = 1$, and $\boldsymbol{\pi}_i = \boldsymbol{\pi}_i(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i$, $\boldsymbol{\pi}_i \mathbf{1} = 1$. The index of dispersion for a sample trace of demands can be estimated with several techniques [17], [27].

Overdemand: We propose in this paper a new metric, termed overdemand, for estimating the properties of bursts in workloads generated by the target benchmark B. For a given resource, its overdemand ϕ is defined as the conditional probability of consecutively generating two requests that both cause a service demand exceeding the mean. Formally,

$$\phi = \Pr[X_1 > E[T] \,|\, X_0 > E[T]],$$

where X_0 and X_1 are service times of any two consecutive requests submitted by the benchmark. Intuitively, ϕ grows with the probability of observing bursts of requests having long service demand, which are the most deleterious for performance.

We now derive the following expression for the overdemand in a MAP.

Theorem 1: In a Markovian Arrival Process $(\mathbf{D}_0, \mathbf{D}_1)$*, the overdemand is computed as*

$$\phi = \frac{\boldsymbol{\pi} e^{\mathbf{D}_0 E[T]}(-\mathbf{D}_0)^{-1}\mathbf{D}_1 e^{\mathbf{D}_0 E[T]}\mathbf{1}}{\boldsymbol{\pi} e^{\mathbf{D}_0 E[T]}\mathbf{1}}, \tag{14}$$

where $e^{\mathbf{D}_0 E[T]}$ *denotes the matrix exponential function (7) evaluated at* $\mathbf{D}_0 E[T]$*.*

The proof is given in the final appendix. The overdemand ϕ has a number of appealing properties that make it a practical descriptor of burstiness:

Computational Efficiency. First, overdemand enables a search for a policy $\mathbf{P}$ that achieves a target duration and intensity of bursts using linear programming, instead of non-linear optimization techniques such as those used in our previous work using the index of dispersion [7]. This makes the benchmark generation process computationally efficient. Furthermore, infeasible benchmark requirements are easily detected since they result in infeasible linear programs. We present in Section 5.2 experiments that establish the computational efficiency of overdemand-based benchmark generation.

Finite Range. Another useful property of the overdemand is that, being a probability value, its range is always bounded. This is in contrast with burstiness descriptors based on variance of the samples, such as the index of dispersion, which have an unbounded range [7]. In practice, the overdemand lies in a finite range where ϕ belongs to $\phi \in [\phi^{min}, \phi^{max}]$; $\phi^{min} \geq 0$ and $\phi^{max} \leq 1$ are nontrivial functions of the distribution of the test suite mix $\boldsymbol{\gamma}$ and of the session service demands for each test suite. In practice, the values of ϕ^{min} and ϕ^{max} are easy to obtain by linear programming. For example, ϕ_i^{min} is given by

$$\phi_i^{min} = \min_{\mathbf{P}} \phi_i \quad s.t. \tag{15}$$

$$(\mathbf{D}_0^i, \mathbf{D}_1^i) = compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}); \tag{16}$$

$$\boldsymbol{\pi}_i = \boldsymbol{\pi}_i(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i \tag{17}$$

$$\phi_i = \frac{\boldsymbol{\pi}_i e^{\mathbf{D}_0^i E[T_i]}(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i e^{\mathbf{D}_0^i E[T_i]}\mathbf{1}}{\boldsymbol{\pi}_i e^{\mathbf{D}_0^i E[T_i]}\mathbf{1}} \tag{18}$$

$$E[T_i] = \boldsymbol{\pi}_i(-\mathbf{D}_0^i)^{-1}\mathbf{1} \tag{19}$$

$$\boldsymbol{\gamma}\mathbf{P} = \boldsymbol{\gamma} \tag{20}$$

$$\boldsymbol{\pi}_i\mathbf{1} = 1 \tag{21}$$

$$\mathbf{P1} = \mathbf{1} \tag{22}$$

$$\mathbf{P} \geq \mathbf{0} \tag{23}$$

where (16) defines at each iteration of the linear programming solver the MAPs $(\mathbf{D}_0^i, \mathbf{D}_1^i)$ associated to the policy $\mathbf{P}$ currently being evaluated, (18) states that ϕ_i is a linear function of the unknown probabilities in $\mathbf{P}$ which define the MAP $(\mathbf{D}_0^i, \mathbf{D}_1^i)$, (20) imposes the test suite mix $\boldsymbol{\gamma}$ to be the equilibrium distribution of $\mathbf{P}$, and (22)-(23) require $\mathbf{P}$ to be a valid submission policy. Changing the objective function to a maximization problem provides ϕ_i^{max}. The resulting range $[\phi_i^{min}, \phi_i^{max}]$ is a quantitative characterization of the achievable burstiness for resource i under a mix $\boldsymbol{\gamma}$.

Achievable Burstiness. The overdemand also allows evaluation of whether a test suite mix is a good candidate for generating service demand burstiness. Specifically, an uncorrelated session submission policy $\mathbf{P}^{noburst} = \mathbf{1}\boldsymbol{\gamma}$, which generates workloads with i.i.d. service demands and mix $\boldsymbol{\gamma}$, can be used to compute the corresponding $\phi_i^{noburst}$ value from the MAP $(\mathbf{D}_0^i, \mathbf{D}_1^i) = compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}^{noburst})$. For the mixes $\boldsymbol{\gamma}$ that can generate very limited burstiness, this would be immediately reflected in a very narrow range $[\phi_i^{min}, \phi_i^{max}]$ where $\phi_i^{noburst} \approx \phi_i^{max} \approx \phi_i^{min}$. For example, this would be the case when modulating test suites with very similar mean service demands and low variability, which makes it hard to create bursts. Notice that the above observations lead in a natural way to defining the *normalized overdemand*

$$\psi_i = 100 \times \frac{\phi_i - \phi_i^{min}}{\phi_i^{max} - \phi_i^{min}}, \quad \psi_i \in [0, 100],$$

which is interpreted as the fraction of the maximum burstiness at resource i achieved by benchmark B.

Burst Length. An advantage of the overdemand with respect to other descriptors, noticeably the lag-1 autocorrelation, is also that it helps in understanding intensity and duration of a burst. Since ϕ represents the probability of the session submission policy to continue to feed a burst, burst lengths can be estimated by a geometric distribution with probability $1 - \phi_i$ of burst termination. This implicitly assumes that requests are continuously submitted, hence this is a heavy-load approximation. The choice of approximating the burst length by a geometric distribution is instead motivated by the use of a session submission policy $\mathbf{P}$ that is a discrete-time Markov chain and thus memoryless. We have observed this approximation to be a faithful characterization of the bursts created by the benchmark model B using the BURN approach. Based on this approximation and denoting by the random variable L_i the burst length for a resource i, the probability of having a burst of length k is

$$\Pr[L_i = k] \approx \phi_i^k(1 - \phi_i), \tag{24}$$

and its expected value is

$$E[L_i] \approx \frac{\phi_i}{1 - \phi_i}. \tag{25}$$

The last formula can be used for deciding the appropriate value to assign to an overdemand ϕ_i to generate a burst with an expected number $E[L_i]$ of sessions completed in it. Additionally, it provides an estimate for the minimum time required to process a burst ignoring contention, i.e.,

$$E[T_i] \approx E[L_i]E[T_L], \; E[D_{over}] = \left(\frac{\boldsymbol{\pi}\mathbf{e}^{\mathbf{D}_0 E[T]}(-\mathbf{D}_0)^{-1}\mathbf{1}}{\boldsymbol{\pi}\mathbf{e}^{\mathbf{D}_0 E[T]}\mathbf{1}}\right), \tag{26}$$

where $E[D_{over}]$ is the expected value of a service demand exceeding the mean (overdemanding sessions).

Limitations. A limitation of the overdemand metric is that it can be used to compare only workloads which have the same mix $\boldsymbol{\gamma}$. This is because the overdemand does not depend on properties such as the tail of the distribution function, which affects $e^{\mathbf{D}_0^i E[X_i]}$ and hence the absolute values of the ϕ_i. However, since the search techniques proposed in the following sections assume a user-specified $\boldsymbol{\gamma}$, the mix is always fixed and so is $E[D_{over}]$[5]. Thus this is not a limitation for the specific application of overdemand considered in BURN.

Example 5: Consider the following MAPs:

$$\mathbf{D}_0^a = \begin{bmatrix} -2.7963 & 0.2785 \\ 0.0975 & -50.4543 \end{bmatrix} \quad \mathbf{D}_1^a = \begin{bmatrix} 0.8147 & 1.7031 \\ 0.9058 & 49.4509 \end{bmatrix}$$

$$\mathbf{D}_0^b = \begin{bmatrix} -34.2766 \end{bmatrix} \quad \mathbf{D}_1^b = \begin{bmatrix} 34.2766 \end{bmatrix}$$

The first MAP $(\mathbf{D}_0^a, \mathbf{D}_1^a)$ *has* $CV^2 = 8.22$ *and lag-1 autocorrelation* $\rho_1 = 0.121$*; the second MAP* $(\mathbf{D}_0^b, \mathbf{D}_1^b)$ *is instead exponential, thus* $CV^2 = 1$ *and* $\rho_1 = 0$*. Indeed, the exponential MAP is expected to be less bursty than the first MAP, but the original overdemand cannot be used to highlight this since* $\phi(E[X^a]) = 0.2629$ *and* $\phi(E[X^b]) = 0.3679$*, where in this example* $E[X^a] = E[X^b] = 1$*. However, if we look at the tail of the distribution, say* $t = 5E[X^a] = 5E[X^b]$*, we find that* $\phi(5E[X^a]) = 0.1906$ *and* $\phi(5E[X^b]) = 0.0067$*, which readily shows that the first MAP has much higher likelihood to generate long-lasting bursts than the exponential distribution.*

4.5 Search Step: Matching Burstiness Levels

Based on the results of the previous subsection, we use an optimization program to search for a policy $\mathbf{P}$ that provides the desired level of burstiness in the benchmark service demands.

Additionally, we constrain the benchmark B to generate a target test suite mix $\boldsymbol{\gamma}$. It is useful to observe that, given two different submission policies $\mathbf{P}_1$ and $\mathbf{P}_2$ with identical equilibrium mix $\boldsymbol{\gamma}$, the values of $\boldsymbol{\pi}_i$, $\mathbf{D}_0^i$, and $E[T_i]$ for a given resource i are identical for both benchmark burstiness models $(\mathbf{D}_0^i, \mathbf{D}_1^i) = compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}_1)$ and $(\mathbf{D}_0^i, \mathbf{D}_1^i) = compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}_2)$. Hence one can compute initially $\boldsymbol{\pi}_i$, $\mathbf{D}_0^i$, and $E[T_i]$ for a random policy and reuse these values during the iterations of the solver. Thus $\boldsymbol{\pi}_i$, $\mathbf{D}_0^i$, $E[T_i]$, and related functions (e.g., matrix exponential) are constant in the optimization programs reported below. Under such assumptions, linear equations involving $\mathbf{D}_1^i$ remain linear functions of $\mathbf{P}$ which is the only unknown in the linear program.

5. Note that a possible extension to this work could search over different mixes. Metrics similar to the overdemand can then be easily defined to take into account the effects of the distribution on $E[D_{over}]$, thus making the case for the flexibility of this burstiness index. For example,

$$\phi(t) = \Pr[X_1 > t \,|\, X_0 > t] = \frac{\boldsymbol{\pi} e^{\mathbf{D}_0 t}(-\mathbf{D}_0)^{-1}\mathbf{D}_1 e^{\mathbf{D}_0 t}(-\mathbf{D}_0)^{-1}\mathbf{1}}{\boldsymbol{\pi} e^{\mathbf{D}_0 t}\mathbf{1}},$$

has the same properties of the overdemand, but it also considers the benchmark demand distribution up to an arbitrary quantile t. Note that the original definition of the overdemand implies $\phi \equiv \phi(E[T])$.

We now describe a technique for matching a target overdemand level. For a set of M resources, we can generate a submission policy $\mathbf{P}$ that matches the target burstiness according to the overdemand descriptors for all resources using the following linear program

$$\min_{\mathbf{P}} z = \sum_{i=1}^{M} s_i^+ + \sum_{i=1}^{M} s_i^- \quad s.t. \tag{27}$$

$$(\mathbf{D}_0^i, \mathbf{D}_1^i) = compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}); \tag{28}$$

$$\phi_i = \phi_i^{target} + s_i^+ - s_i^- \tag{29}$$

$$\boldsymbol{\pi}_i = \boldsymbol{\pi}_i(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i \tag{30}$$

$$\phi_i = \frac{\boldsymbol{\pi} e^{\mathbf{D}_0^i E[T]}(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i e^{\mathbf{D}_0^i E[T]}\mathbf{1}}{\boldsymbol{\pi} e^{\mathbf{D}_0^i E[T]}\mathbf{1}} \tag{31}$$

$$\boldsymbol{\gamma}\mathbf{P} = \boldsymbol{\gamma} \tag{32}$$

$$\boldsymbol{\pi}_i \mathbf{1} = 1 \tag{33}$$

$$\mathbf{P1} = \mathbf{1} \tag{34}$$

$$\mathbf{P} \geq \mathbf{0} \tag{35}$$

$$s_i^+ \geq 0 \tag{36}$$

$$s_i^- \geq 0 \tag{37}$$

where constraints are for $i = 1, \ldots, M$. Here the s_i^+ and s_i^- variables minimize the distance from the objectives using a linear function, while the other constraints are similar to the ones considered to determine ϕ_{min}. We stress again that all terms involving $\boldsymbol{\pi}$ and $\mathbf{D}_0^i$ are constant throughout the optimization, thus they can be evaluated initially for any feasible policy, e.g. from the non-bursty model $compose(\boldsymbol{\pi}_g^i, \mathbf{A}_g^i, \mathbf{P}_{noburst})$ where $\mathbf{P}_{noburst} = \mathbf{1}\boldsymbol{\gamma}$. Based on this property, the entries of $\mathbf{D}_1^i$ are the only unknowns and, by the given definitions, these have the form $p_{g,h}c_{g,h}^{i,j}$, where $c_{g,h}^{i,j}$ is the (i,j) entry of $-\mathbf{A}_g^i\mathbf{1}\boldsymbol{\pi}_h$ that can be computed prior to execution of the linear solver. Thus, the $p_{g,h}$ values are the only unknowns and the optimization program is a linear program.

5 Validation Experiments

To show that BURN is effective in creating controlled burstiness, we present experiments that consider combinations of the browsing, ordering, and shopping mix test suites of TPC-W. The matrix $\mathbf{P}$ generated by BURN is used to construct a trace of 10,000 sessions with a desired mix and burstiness and that combines the three test suites. The resulting benchmark B is run on a real testbed. In BURN, we consider CPU usage at the different computing nodes as resources. The testbed consists of a front server, a database server, and a client machine connected by a non-blocking Ethernet switch that provides a dedicated 1 Gbps connectivity between any two machines in the setup. The front server and database nodes are used to execute the TPC-W bookstore application implemented at Rice University [1]. The client node is dedicated for running the httperf Web request generator [30] under the Linux operating system. This was configured to emulate multiple concurrent

TABLE 2
Test suite service demand models for CPU. Mean is expressed in seconds.

	front server demand			*DB server demand*		
shopping	*mean*	CV^2	*skew*	*mean*	CV^2	*skew*
measured	0.290	0.575	2.671	0.097	7.590	4.509
phase-type	0.290	0.575	1.665	0.097	7.590	4.509
ordering	*mean*	CV^2	*skew*	*mean*	CV^2	*skew*
measured	0.131	0.805	1.797	0.623	1.761	2.530
phase-type	0.131	0.806	1.797	0.623	1.761	2.531

sessions in the experiments. The httperf generator has features such as non-blocking socket calls that allow it to stress servers and sustain overloads in a scalable manner [18]. All nodes in the setup contain an Intel 2.66 GHz Core 2 CPU and 2 GB of RAM. The Windows perfmon utility is used to gather CPU, disk, memory, and network usage at the server nodes; httperf provides detailed logs of end user response times. In all experiments we noticed very little disk, paging, and network activity.

5.1 Validation of Demand and Burstiness Models

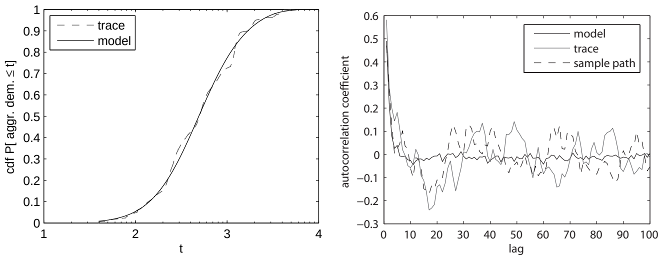
We first consider a benchmark B defined only by a mix of sessions of shopping (shp) and ordering (ord) types. The mix is balanced with test suite mix $\gamma_{shp} = \gamma_{ord} = 0.50$, and we use a session submission policy

$$\mathbf{P} = \begin{bmatrix} p_{shp,shp} & p_{shp,ord} \\ p_{ord,shp} & p_{ord,ord} \end{bmatrix} = \begin{bmatrix} 0.995 & 0.005 \\ 0.005 & 0.995 \end{bmatrix}$$

The aim of this case study is to validate the prediction accuracy of the models proposed in Sections 3 and 4. Since it is difficult to directly measure the service demands, we use for validation the prediction of utilization and aggregate service demand measurements for the sessions executed in isolation on the system.

For the test suite service demand model definition, we have run in isolation the shopping and ordering benchmarks and estimated the mean session demands for each of the session types used. Table 2 presents results. The table shows the estimated moments for the different CPU service demands and the respective moments of the phase-type distributions $\mathbf{A}^i_g$ we have fitted. The number of states used in the phase-type models is always 2. The results indicate that the phase-type distributions match nearly all moments of the measured test suite service demands. The demands for shopping at the front server underestimate skewness because it is difficult for phase-type models to match this parameter under low variability conditions. We have obtained more accurate fitting of this value with larger number of states but this did not significantly improve the precision of our predictions, so we chose to use models with fewer states. The phase type distributions fitted are used later in this subsection to predict the CPU utilization properties at the front server and database.

Additionally, for validation purposes we have fitted into phase-type distributions $\mathbf{A}^{aggr}_{shp}$ and $\mathbf{A}^{aggr}_{ord}$ the



(a) CDF Aggregate Demand (sec) (b) Front Server Utilization

Fig. 4. Experimental results for the mix of shopping and ordering sessions.

measured aggregate demand of shopping and ordering sessions when run in isolation, i.e., the results of the experiments shown in Figure 2 for shopping and ordering mixes. Moments were again matched very closely. These aggregate demands represent the end-to-end latency of sessions including CPU, disk, and I/O response times. The resulting MAP is denoted by $(\mathbf{D}_0, \mathbf{D}_1)$ where

$$\mathbf{D}_0 = \begin{bmatrix} \mathbf{A}^{aggr}_{shp} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{aggr}_{ord} \end{bmatrix}$$

$$\mathbf{D}_1 = \begin{bmatrix} p_{shp,shp}(-\mathbf{A}^{aggr}_{shp})\mathbf{1}\pi^{aggr}_{shp} & p_{shp,ord}(-\mathbf{A}^{aggr}_{shp})\mathbf{1}\pi^{aggr}_{ord} \\ p_{ord,shp}(-\mathbf{A}^{aggr}_{ord})\mathbf{1}\pi^{aggr}_{shp} & p_{ord,ord}(-\mathbf{A}^{aggr}_{ord})\mathbf{1}\pi^{aggr}_{ord} \end{bmatrix}$$

Figure 4(a) compares the cumulative distribution function (CDF) for the aggregate service demand of the benchmark B sessions with the ones predicted by the benchmark model $(\mathbf{D}_0, \mathbf{D}_1)$. The distribution of the MAP matches the empirical distribution of the aggregate service demand very well, thus confirming the predictive capabilities of BURN.

To further validate accuracy, we now focus on the burstiness model. We performed a trace-driven analysis of the system to compare the properties of the measured utilizations with those predicted by the MAP model using the autocorrelation function as a descriptor of burstiness. We denote the four phase-type distributions in Table 2 as $\mathbf{A}^{fs}_{shp}$, $\mathbf{A}^{fs}_{ord}$, $\mathbf{A}^{db}_{shp}$, and $\mathbf{A}^{db}_{ord}$. We generated for each resource a model similar to the one used used before for the aggregate service demands, i.e., $\mathbf{A}^{aggr}_{shp}$ and $\mathbf{A}^{aggr}_{ord}$ are there replaced by $\mathbf{A}^{fs}_{shp}$ and $\mathbf{A}^{fs}_{ord}$.

Figure 4(b) show the autocorrelation function of the measured and modeled CPU utilizations for the front server. The autocorrelations of the model are estimated by averaging the autocorrelations over 100 random experiments; conversely, the sample path curve shows a representative example of autocorrelation estimate for one of these random experiments. For low lags, model and sample path autocorrelations are in very good agreement with the TPC-W trace. Low lags are often the most significant for burstiness, as they measure the similarities of consecutive sessions that pack into bursts. The autocorrelation coefficient values for lags greater than 10 seem instead to suffer significant noise due to the limited size of the measurement. Yet, the good agreement of the sample path with the trace shows that sample paths of the MAP model can represent real system behavior observed in experiments.

5.2 Composition Step Performance

We now present experiments that establish the computational efficiency and scalability of benchmark generation owing to the use of overdemand as a descriptor of burstiness. Specifically, we compare the overdemand based linear programming formulation presented in this work with the nonlinear programming formulation based on the index of dispersion proposed in [7]. We illustrate the performance of the optimization program used in composition step using a set of $1,000$ random benchmark problems. We have set the number of test suites to $G = 2, 3, 4$, the phase-type distribution of each test suite is randomly generated with a random choice of the number states in $n \in [1, 3]$. We consider a single resource, i.e., $M = 1$. Phase-type distributions can have hypo-exponential, exponential, or hyper-exponential distribution. To explore a larger set of feasible burstiness levels, we have scaled the mean demand of the gth test suite to $E[D^i_g] = 2^g$ and explored the achievable burstiness of a given problem using 10 increasing equi-spaced values of $\phi \in [\phi_{min}, \phi_{max}]$. We have compared the execution of the optimization program that matches a target overdemand ϕ with the nonlinear optimization proposed in [7] that matches a target index of dispersion $ID(i)$ for resource i.

To compare equivalent results, we find the entries for the policy $\mathbf{P}$ that minimize the difference between the target index of dispersion and the one estimated for the service demands of resource i based on the $compose(\boldsymbol{\pi}^i_g, \mathbf{A}^i_g, \mathbf{P})$ mapping. For the ease of comparing results, we find the value of the index of dispersion that corresponds to a given ϕ by computing the value of ID from the MAP that matches ϕ. Furthermore, for both optimization programs we begin by using the MATLAB nonlinear solver fmincon. In this case, the reported performance for matching ϕ is a worst-case scenario and occasionally requires multiple solver runs to achieve the optimum. Average results are reported in Table 3. The number of runs indicates the number of independent executions of the optimization solver needed to achieve the target objective function within a tolerance of 10%. We also report the number of iterations and objective function evaluations per solver run, and the total execution time summed over the runs.

From Table 3 we see that in all cases the generation of a target benchmark requires less than a half second on average. The time requirements for matching a target index of dispersion is approximately 150% larger than for the overdemand. This is not explained by the results for the number of runs, which shows that typically one or two runs of the optimization solver are sufficient to match the target. Conversely, it is found that the number of iterations and the corresponding evaluation

TABLE 3
Overdemand ϕ versus Index of Dispersion ID - Composition step for 1000 random instances

method	G	*runs*	*iter/run*	*func/run*	fmincon *cpu* [s]
ϕ	2	1.58	4.17	30.02	0.09
ID	2	1.94	12.54	71.67	0.20
ϕ	3	1.15	6.68	85.02	0.18
ID	3	1.23	14.10	163.44	0.45
ϕ	4	1.04	6.08	118.28	0.36
ID	4	1.22	19.88	382.02	1.73

of the objective function to determine gradients are about 100% larger for the index of dispersion than for the overdemand approach. This is interpreted as a clear indication that the optimization program that matches the index of dispersion is harder than the one used for the overdemand, and this can be explained by the non-convex characteristics, with respect to the unknown $\mathbf{P}$, of the expression to compute the index of dispersion.

We have also implemented the BURN composition step using the GNU Linear Programming Kit (GLPK, http://www.gnu.org/software/glpk/) to explore the scalability of the technique as the number of test suites G grows. Each test suite is characterized by a 2 state service demand model. Table 4 reports results for increasing values of G on an Intel Core Duo 2.16 GHz machine; note that the results are better than in Table 3 due to increased performance of GLPK over MATLAB. The results indicate that the composition technique requires a few seconds for generation of a benchmark when the number of test suites considered is less than a 256. As the number of test suites increases, the quadratic growth in the number of elements of the $\mathbf{P}$ matrix makes the computational cost increase quadratically. The number of non-zero elements in the linear program coefficient matrix is growing quadratically; the matrix is not sparse. Nevertheless, the technique is able to find a benchmark with the desired target burstiness in a few minutes for hundreds of different test suites which demonstrates this is a very practical technique.

The previous validation proves that the overdemand approach would remain scalable even under such an increased complexity. Conversely, our initial experiments indicate that this would not be the case for the index of dispersion approach. For example, by generating random cases with 2-state test suite demand models, for $G = 8$ the fmincon CPU time is of the order of 100s, for $G = 16$ it grows up to about 500s, for $G = 32$ it grows up to about 8000s. This is clearly not scalable, whereas the overdemand solution by GLPK takes less than 0.1s in all of these cases.

It should be noted that some benchmarks consider small number of test suites, for example TPC-W considers 3 mixes, while commercial benchmark usually consider $6-7$ mixes per application [37]. However, the rapid growth of service-oriented computing and the development of compositional environments may raise in the near future the need for more complex workloads.

TABLE 4
Scalability of Composition Step Linear Program

test suites	*GLPK solver*		*linear program size*		
G	*cpu [s]*	*mem [MB]*	*rows*	*cols*	*non-zeros*
2	0.0	0.1	6	6	15
8	0.0	0.2	18	66	195
16	0.0	0.4	34	258	771
32	0.0	1.4	66	1,027	3,078
64	0.1	4.9	130	4,098	12,291
128	1.0	19.1	258	16,386	49,155
256	10.7	75.7	514	65,539	196,614
512	104.7	301.8	1,027	262,147	786,438
1024	1,149.8	1,205.3	2,051	1,048,579	3,145,734

TABLE 5
Performance Indexes for the Experiments in Section 5.3

λ	no burstiness $2.22 sess/s$	burstiness $2.22 sess/s$	burstiness $2.38 sess/s$
mean U^{fs}	0.2393	0.1753	0.2006
std.dev. U^{fs}	0.0509	0.1337	0.1421
mean U^{db}	0.6215	0.6651	0.7086
std.dev. U^{db}	0.1292	0.3755	0.3829
mean R [msec]	51715	83546	86483
std.dev. R [msec]	32302	86475	83920
95th prct. R [msec]	109185	239452	244928

For example, the SAP ByDesign system has thousands of business process variants [36], thus benchmarking of application of this kind may potentially require tens or even hundreds of test suites. Defining such benchmarks in a representative way is still an open challenge for the software engineering of large enterprise applications.

5.3 Generation of Burstiness and its Impact

We now consider a mix of ordering, browsing and shopping sessions, but we now focus on generating benchmarks to evaluate the performance of the TPC-W system under burstiness conditions. The results presented in this section prove that this can be done successfully with the proposed methodology and show the existence of scalability problems for systems that are not revealed by executions of traditional benchmarks without burstiness in the service demands. The experiments reported here show that BURN is a practical technique for benchmarking real-world software systems.

Using BURN, we have created two session submission policies $\mathbf{P}_{non-burst}$ and $\mathbf{P}_{burst}$ that both combine browsing (*bro*), shopping (*shp*) and ordering (*ord*) sessions with test suite mix $\gamma_{bro} = 0.014$ and $\gamma_{shp} = \gamma_{ord} = 0.493$. The test suite mix is essentially the same $10,000$ sessions as in Section 5.1, but there is an additional component of about $100 - 200$ browsing sessions. We use a small fraction of browsing sessions because the browsing mix is already known to impose bottleneck migrations between front server and database server [27]. Hence the browsing mix would not be helpful to show BURN's ability to create bursts in resource consumption.

For the generation of burstiness we used BURN to compute the following policy:

$$\mathbf{P}_{burst} = \begin{bmatrix} 0.001257500 & 0.000260220 & 0.998480000 \\ 0.000148940 & 0.999750000 & 0.000101650 \\ 0.030116000 & 0.000242700 & 0.969640000 \end{bmatrix}$$

which satisfies an initial requirement of normalized overdemand at the front server $\psi_{fs} = 99\%$ of the maximum overdemand and similarly achieves $\psi_{db} = 95\%$ of the maximum overdemand at the database server. Specifically, the expected overdemand for the front server CPUs is $\phi^{fs} = 0.4584410$ and for the database CPUs is $\phi^{db} = 0.3857909$; the feasible overdemand range for the front server CPU is $\phi^{fs} \in [0.339967, 0.460905]$ and for the database server CPU is $\phi^{db} \in [0.149827, 0.39704]$. Thus, from the overdemand ranges we learn that that there is little room for further increasing the burstiness on this workload. We report that we have tried to run benchmarks that achieve values of ϕ^{fs} and ϕ^{db} that are very close to the maximum possible, however the measured performance did not differ significantly from the results presented in this section. The non-bursty policy is defined as $\mathbf{P}_{non-burst} = \mathbf{1}\boldsymbol{\gamma}$. The overdemands for $\mathbf{P}_{non-burst}$ at the front server CPUs and at the database server CPUs are $\phi^{fs} = 0.375222$ and $\phi^{db} = 0.266628$, respectively, which are only 50% and 47% of the achievable burstiness.

Note that the same experimental results shown in this section for arrival rate $\lambda = 2.22\ sess/s$ appear in [7] for the policy obtained from the nonlinear optimization program based on the index of dispersion. In fact, we have chosen the overdemand values such that BURN and the index of dispersion optimization programs provide the same policy for this parameterization. This implies that the overdemand can return qualitatively similar answers to the index of dispersion approach, but further enjoying the additional properties discussed in Section 4.4. For example, we have found the overdemand to be far more informative than the index of dispersion. In fact, in [7] we could not anticipate that the target index of dispersion was nearly an upper bound for the achievable burstiness and we had to run additional experiments to verify this. Instead, the overdemand range and the maximum overdemand can be deduced at the benchmark design phase. This is a major advantage of the overdemand over the index of dispersion, since it avoids the trial-and-error approach needed to determine the maximum achievable burstiness.

5.3.1 Experimental Results

Figure 5 compares the performance impact of the two benchmarks on the TPC-W system for an experiment with Poisson session arrivals with rate $\lambda = 2.22\ sess/s$ and multiple concurrent sessions in execution. Even though the non-bursty and bursty benchmarks have the same test suite mix $\boldsymbol{\gamma}$, session inter-arrival time distribution, and similar CPU utilizations, the bursty benchmark stresses the system differently compared to the non-bursty benchmark. Table 5 reports the numerical values of the performance indexes for these experiments and for the one with increased rate with rate $\lambda = 2.38\ sess/s$ discussed later in this section; in the table U^{fs} and U^{db} are utilizations[6] respectively at the front server and database server, R is the session response time. From Figures 5(a)-(b), it is immediately evident that for the bursty benchmark the front and database server CPU utilizations are not stable around an equilibrium as in the original non-bursty workload. In all plots, the sampling granularity is 15 seconds for each period. The bursty workload adversely impacts the responsiveness and throughput of the system. From Figures 5(a)-(b), the maximum utilization of the database server is higher for the bursty workload than the non-bursty workload. From Figure 5(b) it can be observed that the database server is saturated from sample 0 to sample 100 and from sample 200 to sample 300. This behavior is absent in the non-bursty workload. The heightened contention for the database server causes the number of concurrent sessions in the system to raise beyond the limit imposed by the sizes of the front server thread pool and listen queue. As a result, the front server drops several connections leading to a 25% drop in request throughput relative to the non-bursty case.

Figure 5(c) compares request response times under the bursty and non-bursty benchmarks: most of the requests without burstiness are served in less than 10 seconds, however more than 20% of the requests in the bursty benchmark require at least 100 seconds to be completed, thus making the point that our approach is better suited than standard approaches for stress testing.

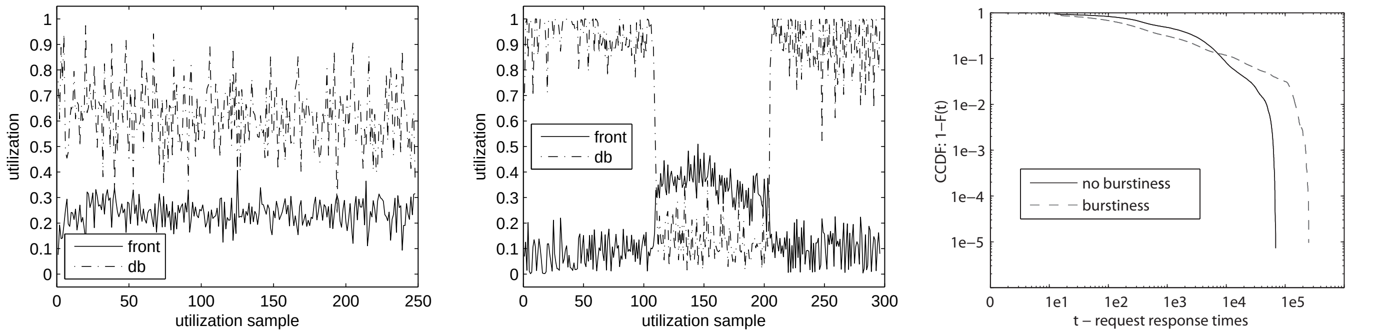
The experiments also reveal that burstiness can cause bottleneck switches that introduce complex transient behavior into the system. From Figure 5(b), the system bottleneck switches from the database server to the front server near the 100th sample. In contrast, from Figure 5(a), there is no bottleneck switch with the non-bursty workload. The bottleneck switch is caused by a transition from the database intensive browsing and ordering sessions to the front server intensive shopping sessions in the bursty workload. It can be observed from Figure 6(b) that there is a significant accumulation of shopping sessions in the system exactly at the time of the bottleneck switch, visible around sample 110 with the decreasing number of ordering sessions. This large accumulation of shopping sessions is caused because of these sessions being delayed by the last of the browsing and ordering sessions at the database server. As a result of this dynamic, it takes the system about 12 minutes spanning the period from sample 110 to sample 160 to significantly reduce the backlog of shopping sessions. This type of unstable transient behavior was not observed with the non-bursty workload and represents a unique feature of burstiness that cannot be exposed with non-bursty submission policies or by running the two benchmarks in isolation. Specifically, such techniques would result in no backlog for the shopping sessions and therefore would never exhibit the properties highlighted in Figure 6(b).

We call the time before reaching the steady state behavior around 160 the *relaxation time* of the application, since it describes the ability of an application to react to transient overloads of a workload. We see that the relaxation time in Figure 6(b) is about 60 periods, since at 160 the application enters the steady-state for the shopping workload. We also conducted in Figure 6(c) a related experiment where we study the behavior of the relaxation time under an increased arrival rate $\lambda = 2.38\ sess/s$ for the same sequence of session types used in Figure 6(b). We notice that the application does not cope with this transient as well as in Figure 6(b), since the equilibrium is reached near sample 180, i.e., the relaxation time is about 33% longer under just a 7% increase of the arrival rate. This information could be exploited in several ways, e.g., by defining reactive policies that allocate more resources to an application as the load increases, driven by the worst case scenarios outlined by the BURN benchmark.

Furthermore, the accumulation of the backlog due to bottleneck switch dominates the response time results presented in Figure 7. Figure 7(a)-(b) show the mean response times of session over time. Specifically, Figure 7(b) shows many important points for the analysis. First, ordering and browsing sessions have considerably larger response times when executed on the system from sample 0 to sample 110 and from sample 210 to 300 than in the corresponding periods of the non-bursty benchmark in Figure 7(a). This suggests that burstiness is more critical for the response times of browsing and ordering sessions which are both database intensive and hence place a strongest congestion level if they are executed in the system without shopping sessions interleaved between them that can alleviate the bottleneck by shifting more load on the front server. The second fundamental observation is that, from sample 110 to 130, the first shopping sessions entering into the system receive dramatically large response times due to the bottleneck switch phenomena. Progressively, as the backlog is flushed response times display a reducing trend. From sample 160 the response times of shopping sessions is lower than in the no-burstiness case, suggesting that the front server can cope well with this level of parallelism for shopping sessions. Finally, Figure 7(c) shows that the observations for Figure 7(b) are even more extreme under an increased arrival rate $\lambda = 2.38\ sess/s$.

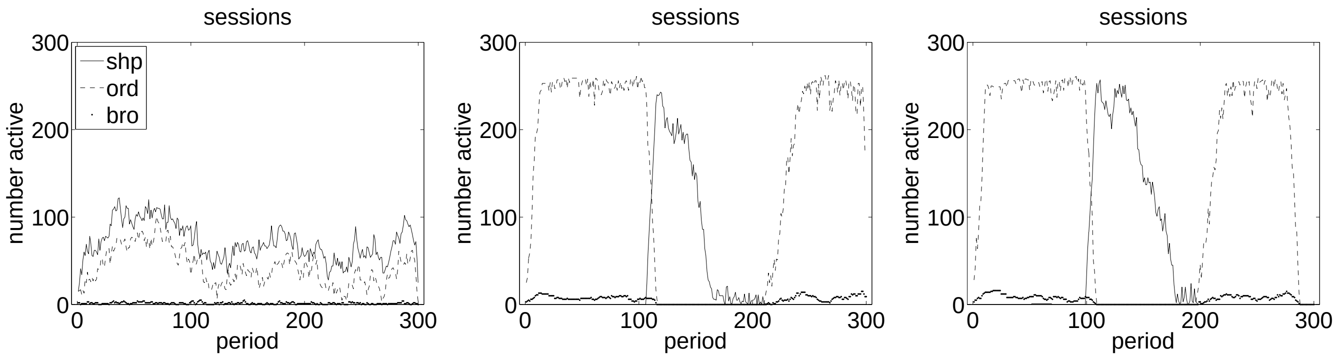
Figure 8 plots for both the bursty and non-bursty benchmark for $\lambda = 2.22\ sess/s$ the number of database queries that are waiting to acquire locks for rows in the database. From the figure the bursty workload causes heightened contention for locks after the first bottleneck switch. Recall that the bottleneck switch was caused by the arrival of a burst of shopping sessions. Furthermore, shopping sessions rely on a common set of data. Conse-

6. Note that small perturbations in the average utilization arise due to the randomness of the experiment.



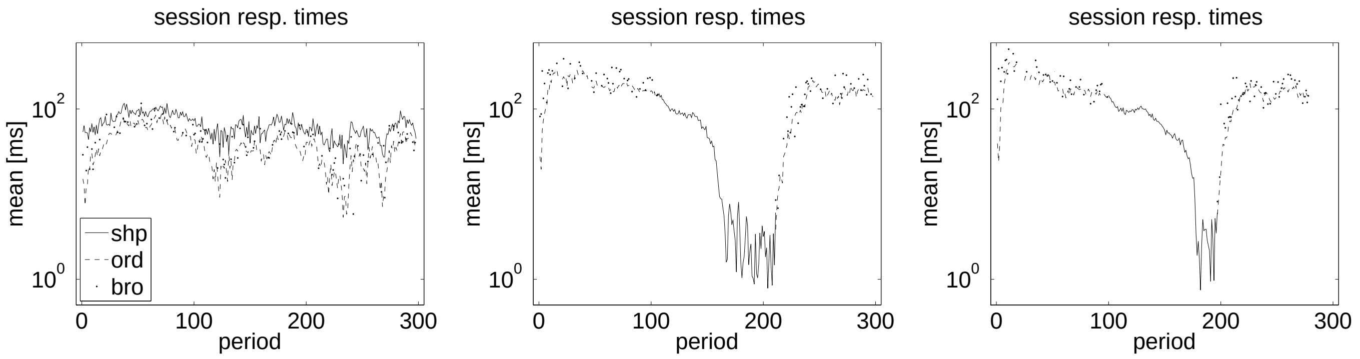
(a) no Burstiness (b) burstiness, $\lambda = 2.22$ sess/s (c) $Pr(RT > t)$, t in msec, $\lambda = 2.22$ $sess/s$.

Fig. 5. Experimental results - CPU utilization and response times, $\lambda = 2.22$.



(a) no burstiness, $\lambda = 2.22$ sess/s (b) burstiness, $\lambda = 2.22$ sess/s (c) burstiness, $\lambda = 2.38$ sess/s

Fig. 6. Experimental results - number of active sessions



(a) no burstiness, $\lambda = 2.22$ sess/s (b) burstiness, $\lambda = 2.22$ sess/s (c) burstiness, $\lambda = 2.38$ sess/s

Fig. 7. Experimental results - mean slowdown of session response times

quently, the burst of shopping sessions causes increased locking activity in the system while accessing this common data set. In contrast, the non-bursty workload does not contain significant bursts of sessions of similar type and as a result does not expose heightened lock contention and the associated performance implications.

6 Conclusion

We have proposed BURN, a model-based methodology for the automatic generation of benchmarks with customizable levels of burstiness in the service demands. BURN generalizes existing approaches for automatic synthesis of benchmarks such as SWAT [22]. Experiments on a real TPC-W testbed have shown that the proposed models are very accurate in predicting the service demands and their burstiness at the different tiers of the architecture. We have shown a case where the ordering and shopping mixes of TPC-W have been combined to inject controlled burstiness in the demands resulting in stress conditions for performance that are not shown by non-bursty combinations of the two mixes. Furthermore, BURN is simple to implement since it only requires a linear programming solver in order to define the session submission policy, e.g., the GNU Linear Programming

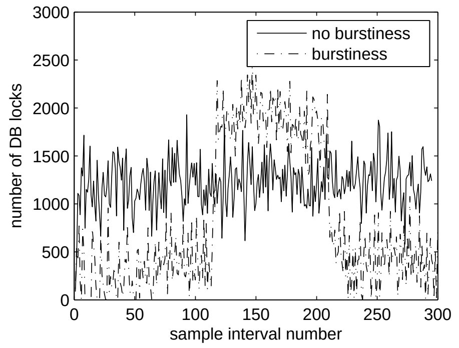


Fig. 8. Number of active locks at the database server - $\lambda = 2.22$ sess/s

Kit (GLPK, http://www.gnu.org/software/glpk/), a workload generator, e.g., httperf, and utilization measurement tools.

It should be remarked that the proposed approach is useful to generate bursts only if the available workloads have sufficient variability between their mean service demands. If this condition is not verified, the proposed overdemand metric is capable of diagnosing this issue and additional workloads should be defined by the benchmark user.

Another limitation is that the methodology is that it ignores queueing effects at the different resources. It may be useful to take into account response times seen by the final users in the benchmark generation technique, e.g., to generate experiments that break in a controlled manner a service-level agreement. More importantly, it may help address cases where the system uses admission control to schedule jobs on a small pool of server threads, e.g., in enterprise-resource planning (ERP) applications. We aim to extend BURN for such scenarios in future work.

Appendix

Proof of Theorem 1. By definition

$$\phi = \frac{\Pr[X_1 > E[T], X_0 > E[T]]}{\Pr[X_0 > E[T]]},$$

where

$$\Pr[X_0 > E[T]] = \boldsymbol{\pi} \boldsymbol{e}^{\mathbf{D}_0 E[T]} \mathbf{1}$$

since the cumulative distribution of a MAP is, similarly to phase-type distributions,

$$\Pr[X \leq x] = 1 - \boldsymbol{\pi} \boldsymbol{e}^{\mathbf{D}_0 x} \mathbf{1}.$$

Consider now the joint probability expression for two consecutive samples of a MAP which is [10]

$$\Pr[x_1, x_0] = \boldsymbol{\pi} \boldsymbol{e}^{\mathbf{D}_0 x_0} \mathbf{D}_1 \boldsymbol{e}^{\mathbf{D}_0 x_1} \mathbf{D}_1 \mathbf{1},$$

and the joint distribution

$$\Pr[X_1 \leq x_1, X_0 \leq x_0] = \int_0^{x_0} \int_0^{x_1} \Pr[t_1, t_0] dt_0 dt_1$$

We get

$$\begin{aligned}\Pr[X_1 > x_1, X_0 > x_0] &= 1 - \Pr[X_1 \leq x_1, X_0 \leq x_0] \\ &= \int_{x_0}^{\infty} \int_{x_1}^{\infty} \Pr[t_1, t_0] dt_0 dt_1, \\ &= \boldsymbol{\pi} \int_{x_0}^{\infty} \boldsymbol{e}^{\mathbf{D}_0 t_0} \mathbf{D}_1 \int_{x_1}^{\infty} \boldsymbol{e}^{\mathbf{D}_0 t_1} \mathbf{D}_1 \mathbf{1} dt_0 dt_1 \\ &= \boldsymbol{\pi} \int_{x_0}^{\infty} \boldsymbol{e}^{\mathbf{D}_0 t_0} \mathbf{D}_1 (-\boldsymbol{e}^{\mathbf{D}_0 x_1})(-\mathbf{D}_0)^{-1} \mathbf{D}_1 \mathbf{1} dt_0 \\ &= \boldsymbol{\pi} \boldsymbol{e}^{\mathbf{D}_0 x_0} (-\mathbf{D}_0)^{-1} \mathbf{D}_1 \boldsymbol{e}^{\mathbf{D}_0 x_1} \mathbf{1}.\end{aligned}$$

where $\lim_{t \to +\infty} \boldsymbol{e}^{\mathbf{D}_0 t} = \mathbf{0}$ and $(-\mathbf{D}_0)^{-1}\mathbf{D}_1$ is a stochastic matrix. Note that the matrix $(-\mathbf{D}_0)^{-1}\mathbf{D}_1$ is known in MAP theory to admit the following probabilistic interpretation: the element in row i and column j stands for the conditional probability of starting in state j to compute a sample given that for the previous sample the underlying chain was started in state i.

References

[1] C. Amza, A. Ch, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. *Proc. of WWC 2002*.

[2] G. Balbo, G. Serazzi. Asymptotic Analysis of Multiclass Closed Queueing Networks: Common Bottleneck. *Perform. Eval.*, 26(1):51–72,, 1996.

[3] G. Balbo, G. Serazzi. Asymptotic Analysis of Multiclass Closed Queueing Networks: Multiple Bottlenecks. *Perform. Eval.*, 30(3):115–152,, 1996.

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

[5] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi. *Queueing Networks and Markov Chains*. Wiley, 2006.

[6] G. Casale, A. Kalbasi, D. Krishnamurthy, and J. Rolia. Automated Stress Testing of Multi-Tier Systems by Dynamic Bottleneck Switch Generation. University of Calgary Technical Report SERG-2009-02, April 2009. http://people.ucalgary.ca/∼dkrishna/SERG-2009-02.pdf.

[7] G. Casale, A. Kalbasi, D. Krishnamurthy, and J. Rolia. Automated Stress Testing of Multi-Tier Systems by Dynamic Bottleneck Switch Generation. In *Proc. of Middleware*, LNCS 5896, 393-413, Springer, 2009.

[8] G. Casale, G. Serazzi. Bottlenecks Identification in Multiclass Queueing Networks using Convex Polytopes. In *Proc. of MASCOTS*, 223–230, Oct 2004, IEEE Press.

[9] G. Casale, E. Z. Zhang, and E. Smirni. KPC-Toolbox: Simple Yet Effective Trace Fitting Using Markovian Arrival Processes. In *Proc. of QEST*, 83–92, Sep 2008, IEEE Press.

[10] G. Casale, E. Z. Zhang, and E. Smirni. Trace Data Characterization and Fitting for Markov Modeling. *Perf. Eval.*, vol. 67, Issue 2, 61-79, Feb 2010.

[11] G. Casale, E. Z. Zhang, and E. Smirni. KPC-Toolbox: Best Recipes for Automatic Trace Fitting Using Markovian Arrival Processes . *Perf. Eval.*, vol. 67, Issue 9, 873-896, Sep 2010.

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

[13] M. Crovella, L. Lipsky. Long-Lasting Transient Conditions in Simulations with Heavy-Tailed Workloads. In Proc. of Winter Sim. Conf., 1005-1012, 1997.

[14] J. J. Dujmovic, Universal benchmark suites, In *Proc. of MASCOTS*, 197-205, 1999.

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

[16] R. Grace, *The benchmark book.* Prentice Hall, 1996.

[17] R. Gusella, Characterizing the Variability of Arrival Processes with Indexes of Dispersion. *IEEE Journal on Selected Areas in Communications*, 9(2):203–211, 1991.

[18] R. Hashemian and D. Krishnamurthy and M. Arlitt. Web Workload Generation Challenges - An Empirical Investigation, HPLabs Technical Report HPL-2010-163, 2010.

[19] HP LoadRunner, http://www.hp.com/go/loadrunner.

[20] H. Kobayashi, B. L. Mark. System Modeling and Analysis: Foundations of System Performance Evaluation. Pearson, 2008.

[21] K. Kant, V. Tewary, and R. Iyer. An Internet Traffic Generator for Server Architecture Evaluation. In *Proc. CAECW*, Jan 2001.

[22] D. Krishnamurthy, J. A. Rolia, and S. Majumdar. A synthetic workload generation technique for stress testing session-based systems. *IEEE Trans. Soft. Eng.*, 32(11):868–882, Nov 2006.

[23] U. Krishnaswamy, D. Scherson, A framework for computer performance evaluation using benchmark sets. *IEEE Trans. on Computers*, 49(12):1325-1338, Dec 2000.

[24] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Trans. on Networking*, 2(1):1–15, 1994.

[25] M. Litoiu, J. A. Rolia, G. Serazzi Designing Process Replication and Activation: A Quantitative Approach. *IEEE Trans. Software Eng.* 26(12): 1168-1178 (2000).

[26] Simon Malkowski, Markus Hedwig, Calton Pu, Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks, *Proc. of IEEE IISWC*, 118-127, 2009.

[27] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proc. of Middleware*, LNCS 5346, 265–286, 2008.

[28] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting Realistic Burstiness to a Traditional Client-Server Benchmark. In *Proc. of ICAC*, June 2009, 149–158.

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

[30] D. Mosberger and T. Jin. httperf - A Tool for Measuring Web Server Performance. Hewlett Packard Laboratories Tech. Rep. HPL-98-61, 1998.

[31] M. F. Neuts. Renewal processes of phase type. *Nav. Res. Log. Quart.*, 25:445–454, 1978.

[32] M. F. Neuts. A Versatile Markovian Point Process. *J. App. Prob.*, 16(4):764–779, Dec 1979.

[33] A. Panchenko and A. Thümmler Efficient phase-type fitting with aggregated traffic traces. *Perf. Eval.*, 64(7-8):629–645, 2007.

[34] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *Proc. of QEST*, 41–50, 2006.

[35] J. Rolia, D. Krishnamurthy, G. Casale, and S. Dawson. BAP: A Benchmark-driven Algebraic Method for the Performance Engineering of Customized Services, Invited Paper, in *Proc. of WOSP/SIPEW*, Jan 2010.

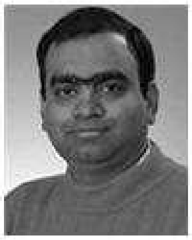
[36] J. Rolia, D. Krishnamurthy, A. Kalbasi, and S. Dawson. Resource demand modeling for complex services, in *Proc. of WOSP/SIPEW*, Jan 2010.

[37] SAP Standard Application Benchmarks http://www.sap.com/solutions/benchmark/index.epx

[38] J. W.J. Xue, A. P. Chester, L. He, S. A. Jarvis. Dynamic Resource Allocation in Enterprise Systems. In *Proc. of ICPADS*, 203–212, 2008.

Giuliano Casale received the M.Eng. and Ph.D. degrees in computer engineering from Politecnico di Milano, Italy, in 2002 and 2006 respectively. He joined the Department of Computing at Imperial College London in 2010 where he is currently an Imperial College Junior Research Fellow in the AESOP group. His research interests include performance modeling, workload characterization, stochastic scheduling, simulation, and resource consumption estimation, topics on which he has published more than 60 research papers. Prior to joining Imperial College London, he was a full-time researcher at SAP Research UK in 2009, and a postdoctoral research associate at the College of William and Mary, Virginia, between 2007 and 2008. In Fall 2004 he was a visiting scholar at UCLA. He has published more than 50 papers in journals, conferences, and book chapters. He is a member of the ACM, the IEEE, and the IEEE Computer Society. He is also a coauthor of the Java Modelling Tools performance evaluation suite (JMT, http://jmt.sf.net). Personal page: http://www.doc.ic.ac.uk/ gcasale/

Amir Kalbasi is a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Calgary. He received his M.Sc. degree from the same department in 2010. His research interests are focused on software performance evaluation and modeling.



Diwakar Krishnamurthy is an assistant professor with the department of electrical and computer engineering at the University of Calgary in Calgary, Alberta, Canada. He received his PhD from Carleton University, Ottawa in 2004. His research interests are focused on software performance evaluation. Diwakar is currently involved in projects related to performance testing and performance modeling of enterprise application systems.

Jerry Rolia is a Principal Scientist in the Services Research Laboratory of Hewlett-Packard Labs. His research interests include smart-grids, performance modelling for situational awareness, software performance engineering, and data centre resource management. Jerry received his Ph.D. from the University of Toronto '92, was an Associate Professor in the department of Systems and Computer Engineering at Carleton University in Ottawa, Canada until 1999, and has been with HP Labs since.