Automatically Generating Bursty Benchmarks for Multi-Tier Systems

Giuliano Casale
SAP Research
CEC Belfast, UK
giuliano.casale@sap.com

Amir Kalbasi
University of Calgary
Calgary, AB, Canada
akalbasi@ucalgary.ca

Diwakar Krishnamurthy
University of Calgary
Calgary, AB, Canada
dkrishna@ucalgary.ca

Jerry Rolia
Automated Infrastructure Lab
HP Labs, Bristol, UK
jerry.rolia@hp.com

ABSTRACT

The performance of multi-tier systems is known to be degraded by burstiness. Therefore methods are needed to create benchmarks with controllable levels of burstiness for stress testing and system validation. This paper proposes a novel model-based technique to construct such benchmarks in an automated manner. The technique permits the simultaneous matching of request mix and session service time burstiness. Case studies on a TPC-W testbed demonstrate the effectiveness of the approach.

1. INTRODUCTION

Burstiness refers to temporally dependent workload request patterns that cause serial correlations in service demands at various system resources. Recent work has suggested that burstiness is prevalent in multi-tier systems [5]. Furthermore, bursty workloads are known to stress such systems more than workloads with random patterns. For example, burstiness can trigger frequent bottleneck switches between system resources that limit scalability and make performance prediction a challenging task [4]. Consequently, techniques are needed to incorporate burstiness in a controlled manner within the synthetic workloads used for stress testing and system sizing exercises. Unfortunately, due to the session-oriented nature of workloads, this is a challenging task for multi-tier systems. A synthetic workload for such systems must simultaneously match many different characteristics such as request mix and session inter-arrival time distribution while using only semantically correct request sequences. Furthermore, the creation of controlled burstiness requires understanding of the specific characteristics of the service demand placed by the requests at each tier of the architecture. In this paper, we propose an automated approach to solve the problem.

The problem under study can be formulated as follows. Consider a multi-tier system with a pre-existing set of test suites $B_1, \ldots, B_j, \ldots, B_J$. Each test suite is a stress testing application that can submit to the system a sequence of sessions and is characterized by a particular mix of session types. Each *session type* is a semantically correct sequence of requests. Each request is submitted after completion of the previous request in the session, possibly with a think time between the requests. We want to generate a benchmark B^* that submits a sequence of sessions to the system by modulating the session generation of the pre-existing test suites. The goal of the benchmark B^* is to generate sessions selected from the pre-existing test suites such that a user-specified session type mix is achieved while simultaneously causing user-specified levels of burstiness in resource consumption at the system's different tiers.

The methodology we propose to solve this problem involves analytical modeling of the resource consumption of each session type and the formulation of an optimization program that seeks for the best combination of pre-existing test suites that can generate the desired session type mix and burstiness. Our approach can be used to support performance sizing, controller tuning, and performance debugging exercises. Sizing requires representative burstiness in synthetic test workloads to ensure a system can handle its load with appropriate response times. In shared virtualized environments application resource allocations may be governed dynamically by automated controllers. Our approach can be helpful in ensuring that these controllers are tuned appropriately to react effectively to bursts in application workloads. Performance debugging also benefits from fine control over burstiness. For example, it can help determine the impact of burstiness on cache misses and virtual memory swapping effects that are not visible with random workloads.

The approach presented in this paper is motivated by the previous work of Krishnamurthy *et al.* on synthetic workload generation for session-based systems [3]. The work 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 sessions from a session based system, each with a particular URL request mix, and computes a ratio of sessions to achieve specific workload characteristics. For example, the technique can reuse the existing sessions to simultaneously match a new URL request mix and a particular session length distribution and to prepare a corresponding synthetic workload to be submitted to the system. Our current work exploits and extends these concepts. SWAT can be used to find a mix of session types that matches a request mix and other session properties. The approach presented in this paper then matches a desired level of burstiness for session service demands.

Summarizing, the proposed methodology has the following main advantages: (i) It can cause controlled levels of burstiness in service demands. To the best of our knowledge we are not aware of any other benchmarking approach that supports the ability to explicitly control the level of burstiness in service demands. (ii) It is automated, combining existing non-bursty, semantically correct sessions of benchmarks

for the definition of benchmarks with burstiness based on the solution of an optimization program; (iii) It has wide applicability since it only requires information about mean demands of sessions within each benchmark. Such demands can be deduced directly or using techniques such as linear regression [1], operational analysis [1].

2. METHODOLOGY

This paper employs a composite approach to construct the benchmark B^*. A PH-type *group service demand model* characterizes the expected service demand of a random session for each test suite and for each server in the system. The term *group* denotes the set of sessions generated by a given test suite. A discrete-time Markov chain $\mathbf{P}$, called the *session submission policy*, determines the mix and ordering of sessions. A Markovian Arrival Process (MAP) [6] specifies the service demand *burstiness model* used to predict the system's service demand burstiness as caused if the sessions were generated by $\mathbf{P}$. Finally, a nonlinear optimization program iteratively searches for a $\mathbf{P}$ to match desired mix and level of burstiness. The $\mathbf{P}$ matrix obtained in this manner is used to drive the session generation from B^*.

2.1 Group Service Demand Model

The group service demand model uses resource utilization measurements to estimate the service demand of sessions generated by each test suite B_j. First, we estimate the mean service demand of each session type in the group. Next, we derive a compact analytical model that summarizes information for each test suite. The class of analytical models we consider are PH-type distributions [6], which are generalizations of well-know models of service demands such as Erlang, hypo-exponential, and hyper-exponential distributions. The main difference between a PH-type distribution and these models is that it allows more flexibility and higher fitting accuracy, but at the expense of an increased number of states in the underlying Markov model.

Resource utilization measurements are obtained by running each test suite B_j in isolation and collecting traces of resource usage and session completion times for each server i. Let n be the number of completed sessions in the jth sample interval and assume that n_t of them are of type t. Then, the measured utilization samples U_i at server i during the jth sample period are related to the per-session resource consumption by the utilization law [1] $U_i = \sum_{t=1}^{T} m_{i,t} n_t / L$, where L is the time duration of the sample interval, T is the number of session types used in B_j, and $m_{i,t}$ is the mean service demand of session type t at server i. We can measure the mean service demands $m_{i,t}$ of the different sessions types of the test suite B_j directly or using a multivariate linear regression of samples obtained over the different intervals [1]. Based on the estimated mean service demands $m_{i,t}$, we fit a PH-type distribution $\mathbf{H}^{i,j}$ that describes the service demand placed by a random session of B_j on server i. If we indicate with $\beta_{j,t}$ the probability that B_j generates a session of type t such that the vector $\boldsymbol{\beta}_j = (\beta_{j,1}, \ldots, \beta_{j,t}, \ldots, \beta_{j,T})$, $\sum_{t=1}^{T} \beta_{j,t} = 1$, specifies the *mix* of sessions in the benchmark B_j, then the problem of modeling B_j session service demands amounts to defining a PH-type distribution that matches the moments (e.g., mean, variance, skewness, ...) of the $m_{i,t}$ service demands accounting for the mix $\boldsymbol{\beta}_j$. For instance, if X represents the service demand of a random session of B_j, then the mean is $E[X] = \sum_{t=1}^{T} \beta_{j,t} m_{i,t}$ and we approximate the variance as $Var(X) = \sum_{t=1}^{T} \beta_{j,t} (m_{i,t})^2 - E[X]^2$ which is exact if the real demands are deterministic. Based on mean and variance estimates[1], we describe compactly the distribution of session service demands for benchmark B_j at server i by a PH-type model $\mathbf{H}^{i,j}$ which fits these moments.

2.2 Service Demand Burstiness Model

We define Markovian arrival processes (MAPs) [6] to predict the service demand burstiness created by sessions that are randomly picked from the test suites B_1, ..., B_j, ..., B_J. The approach relies on the policy $\mathbf{P}$ that is defined further in the next subsection. The MAPs are used to assess whether a certain $\mathbf{P}$ is a good candidate for generating the desired level of burstiness for the benchmark. We use MAPs instead of PH-type models because the latter can only characterize the distribution, but not the time series of service demands as instead supported by MAPs. It would therefore be impossible to describe with a PH type model the bursts in service demands over time.

The service demand burstiness model is summarized by a function $f^i(\mathbf{H}^{i,j}, \mathbf{P})$ that, given the Markov chain $\mathbf{P}$ and the PH-type models $\mathbf{H}^{i,j}$ of the test suites B_1, ..., B_j, ..., B_J, returns a MAP to predict properties of the time series of session service demands of B^* at server i. The transition probability $p_{i,j}$ of the Markov chain $\mathbf{P} = \{p_{i,j}\}_{J \times J}$ is defined such that if the last session of B^* has been generated by the test suite B_i, then B^* uses with probability $p_{i,j}$ the test suite B_j to generate the subsequent session. Henceforth, we represent PH-type distributions using the $(\mathbf{D}_0, \mathbf{D}_1)$ notation of Markovian arrival processes [6] and we denote the $\mathbf{D}_0$ and $\mathbf{D}_1$ matrices of $\mathbf{H}^{i,j}$ by $\mathbf{H}_0^{i,j}$ and $\mathbf{H}_1^{i,j}$, respectively. The interpretation of $\mathbf{D}_0$ and $\mathbf{D}_1$ is as follows. The MAP describes the service times received by a job and its active state changes over time according to the state transition rates in $\mathbf{D}_0$ and $\mathbf{D}_1$. The transitions in $\mathbf{D}_1$ represent completion of the current job, while all remaining transitions are placed in $\mathbf{D}_0$.

The function $f^i(\mathbf{H}^{i,j}, \mathbf{P})$ that provides the service demand burstiness model of a random session of B^* at each server i can be defined using the compositional properties of Markov processes. By modulating the different test suites B_j with a Markov chain $\mathbf{P}$, the service demand model of B^* can still be described by a Markov process: this enables analytical tractability of burstiness properties and it is the main motivation behind our choice of using the Markov chain $\mathbf{P}$ for session modulation. In fact, from the definition of $\mathbf{D}_0$ and $\mathbf{D}_1$ given above, it follows that the demand of B^* at server i is described by the MAP with matrices

[1]While only a few cases such as deterministic and exponential distributions are fully described by the first two moments, we note that we are modelling session type demands and not per request demands. For our case study session demands were nearly deterministic.

$\mathbf{D}_0^i = diag(\mathbf{H}_0^{i,1}, \ldots, \mathbf{H}_0^{i,j}, \ldots, \mathbf{H}_0^{i,J})$ and

$$\mathbf{D}_1^i = \begin{bmatrix} p_{1,1}\mathbf{H}_1^{i,1} & p_{1,2}\mathbf{H}_1^{i,1} & \ldots & p_{1,J}\mathbf{H}_1^{i,1} \\ p_{2,1}\mathbf{H}_1^{i,2} & p_{2,2}\mathbf{H}_1^{i,2} & \ldots & p_{2,J}\mathbf{H}_1^{i,2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{J,1}\mathbf{H}_1^{i,J} & p_{J,2}\mathbf{H}_1^{i,J} & \ldots & p_{J,J}\mathbf{H}_1^{i,J} \end{bmatrix}. \quad (1)$$

The above equations uniquely specify the $f^i(\mathbf{H}^{i,j}, \mathbf{P})$ function that translates the service demands of sessions of the test suite B_j into those of the benchmark B^*. The $\mathbf{D}_0^i$ matrix specifies that the service demands of a session generated by B_j follow the PH-type distribution specified in $\mathbf{H}^{i,j}$; the $\mathbf{D}_1^i$ matrix definition, instead, imposes that a session of B_j is followed with probability $p_{j,k}$ by a session generated from B_k thus modeling the modulation of the Markov chain $\mathbf{P}$. The fundamental result achieved by this step is that, given the MAP process $(\mathbf{D}_0^i, \mathbf{D}_1^i)$, it is easy to evaluate burstiness in the demands for sessions of B^* as we explain below.

2.3 Searching for a P to Match Burstiness

Finally, we use a nonlinear optimization program to search for a policy $\mathbf{P}$ that provides the desired levels of burstiness in the service demands. Our approach is to evaluate iteratively the burstiness generated by several Markov chains $\mathbf{P}$ and choose the one that minimizes the distance between the predicted burstiness and the target one specified by the user. The optimization is constrained on the benchmark B^* generating a predefined mix $\boldsymbol{\beta}^* = (\beta_1^*, \beta_2^*, \ldots, \beta_J^*)$ of the test suites B_j producing a mix of sessions that is considered representative of the intended usage of the system. The $\boldsymbol{\beta}^*$ vector can be determined using, e.g., the SWAT tool [3] to match a customers required request mix. Different objective functions can be defined according to the preferred burstiness descriptor, e.g., index of dispersion[2], lag-1 autocorrelation coefficient, or the decay rate of the autocorrelations.

Due to limited space, we exemplify the generation of burstiness at a server i that matches a given index of dispersion value I_{target}. We use the following nonlinear optimization program:

$$\min_{\mathbf{P}} z = |I - I_{target}| \quad s.t. \quad (2)$$

$$(\mathbf{D}_0^i, \mathbf{D}_1^i) = f^i(\mathbf{H}^{i,j}, \mathbf{P}); \quad (3)$$

$$I - 1 = 2\left(\bar{S} - \bar{S}^{-1}\pi_e \mathbf{D}_1^i(\mathbf{D}_0^i + \mathbf{D}_1^i + \mathbf{e}\pi_e)\mathbf{D}_1^i\mathbf{e}\right); \quad (4)$$

$$\bar{S} = \pi_e(-\mathbf{D}_0^i)^{-1}\mathbf{e}; \quad (5)$$

$$\pi_e = \pi_e(-\mathbf{D}_0^i)^{-1}\mathbf{D}_1^i; \quad (6)$$

$$\mathbf{P}\mathbf{e} = \mathbf{e}; \quad (7)$$

$$\mathbf{P} \geq \mathbf{0}; \quad (8)$$

$$\boldsymbol{\beta}^*\mathbf{P} = \boldsymbol{\beta}^*; \quad (9)$$

where $\mathbf{e}$ is a column vector of size J composed of all ones. The search is on the entries of the matrix $\mathbf{P}$ that specifies B^*. The program tries to minimize the absolute difference between the target index of dispersion and the one estimated for the service demands at server i based on the $f^i(\mathbf{H}^{i,j}, \mathbf{P})$ mapping, where $\mathbf{P}$ indicates the current estimate of the nonlinear program for the Markov chain transition matrix. The constraints are of three types: (4)-(6) are standard formulas for computing the index of dispersion I applied to the Markovian arrival process $(\mathbf{D}_0^i, \mathbf{D}_1^i)$; (7)-(8) impose that $\mathbf{P}$ is a stochastic matrix; finally, (9) imposes the session type mix β^* by constraining the steady-state of $\mathbf{P}$.

The nonlinear program (2)-(9) returns a Markov chain $\mathbf{P}$ that achieves the stated goal of this paper of creating a benchmark B^* with controlled burstiness in a tier i. Although the optimization program is nonlinear, we have found it in practice easy to solve. In the experiments reported in Section 3, we obtained good solutions in less than one minute for all experiments including the time for restarting the optimization program from a different random initialization point when it fails to return a $\mathbf{P}$ that matches requirements. We remark that if one is interested in generating controlled burstiness in all tiers simultaneously, it is possible either to consider multiple objective functions, each representing the index of dispersion of a different server, or to insert burstiness into the *aggregate service demand* of the sessions, i.e., the round-trip time of a session when executed in isolation on the system. For instance, for a system with a front server and a database server, the aggregate service demand is $R = S_{fs} + S_{db}$, where S_{fs} and S_{db} are service demands at the two servers. We illustrate the effectiveness of this approach in the first case study.

3. VALIDATION EXPERIMENTS

To show that our benchmark generation approach is effective in creating controlled burstiness, we present a case study that considers a particular combination of the ordering and shopping mix benchmarks of TPC-W. The resulting benchmark B^* is run on a real testbed. The testbed consists of a front server node, a database server node, and a client node 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 server nodes are used to execute the TPC-W bookstore application implemented at Rice University. The client node is dedicated for running the httperf Web request generator. 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 our experiments we noticed very little disk, paging, and network activity at the server nodes.

Throughout the experiments, we have used two pre-existing test suites created to follow the shopping and ordering mixes specified by TPC-W. The matrix $\mathbf{P}$ that results from the benchmark generation step is used to construct a trace of 10,000 sessions with desired mix and burstiness and that combines the shopping and ordering sessions. Finally, httperf is used to submit the session trace to the system. Due to limited space, we report below only two validation experiments, but we remark that we have considered several other experiments resulting in qualitatively similar results to those reported below.

[2]Under positive autocorrelations, burstiness levels can be summarized by the *index of dispersion* $I = CV^2(1 + 2\sum_{k=1}^{\infty}\rho(k))$, where CV is the coefficient of variation of the service demands, $\rho(k)$ is the lag-k autocorrelation coefficient of the service demands, see [4] for further information.

	front server demand			*DB server demand*		
shopping	*mean*	*CV*	*skew*	*mean*	*CV*	*skew*
measured	0.290	0.575	2.671	0.097	7.590	4.509
PH-type	0.290	0.575	1.665	0.097	7.590	4.509
ordering	*mean*	*CV*	*skew*	*mean*	*CV*	*skew*
measured	0.131	0.805	1.797	0.623	1.761	2.530
PH-type	0.131	0.806	1.798	0.623	1.761	2.531

Table 1: Benchmark service demand models. Mean values are expressed in seconds.

3.1 Validation of Service Demand and Burstiness Models

We consider a benchmark B^* defined by a mix of sessions of shopping (s) and ordering (o) type. The mix is balanced with $\beta_s^* = \beta_o^* = 0.50$, and we assume the Markov chain $\mathbf{P}$ assigned such that shopping (resp. ordering) sessions have a probability $p_{s,s} = 0.995$ (resp. $p_{o,o} = 0.995$) that the next session generated after them will be again of shopping (resp. ordering) type. The aim of this case study is to validate the prediction accuracy of the models proposed in Sections 2.1 and 2.2. Since it is hard to obtain direct measurement of the service demands, we focus on utilization and aggregate demand measurements for the sessions executed in isolation.

For the group service demand model definition, we have run in isolation the shopping and ordering test suites and estimated the mean session demands $m_{i,t}$ for each of the session types used in the these mixes. Table 1 presents results. The table shows the estimated moments for the different service demands and the respective moments of the PH-type distributions $\mathbf{H}^{i,j}$ we have fitted; the number of states we have used in the PH-type models is no greater than 5. The results indicate that the PH-type distributions match very well mean and CV of the measured service demands, while they slightly underestimate the value of the skewness probably due to the difficulty in modeling in a Markovian setting the nearly-deterministic demand of individual session types. Using the PH-type distributions $\mathbf{H}^{i,j}$ and the Markov chain $\mathbf{P}$, we have then defined the MAPs that describe the service demands at the front server $(\mathbf{D}_0^{fs}, \mathbf{D}_1^{fs})$ and at the database server $(\mathbf{D}_0^{db}, \mathbf{D}_1^{db})$. We have also defined a MAP to describe the aggregate service demand of the sessions: assuming that each session visits once the front server and database server before completing execution, the MAP that captures the aggregate demands has $(\mathbf{D}_0, \mathbf{D}_1)$ representation, denoted by $(\mathbf{R}_0, \mathbf{R}_1)$, which is a combination of $\mathbf{H}^{fs,s}$, $\mathbf{H}^{db,s}$, $\mathbf{H}^{fs,o}$, $\mathbf{H}^{db,o}$ weighted by the probabilities $p_{o,o}$ and $p_{s,s}$ as in (1).

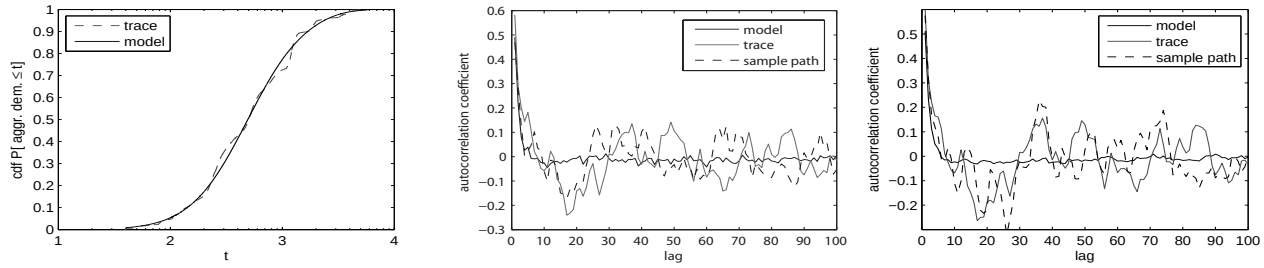
Figure 1(a) compares the cumulative distribution function (CDF) for the aggregate service demands of the sessions with the ones predicted by the $(\mathbf{R}_0, \mathbf{R}_1)$ model. The distribution of the MAP matches very accurately the empirical distribution of the aggregate service demand, thus suggesting the effectiveness of our benchmark service demand models in capturing the distribution of the service demands. Using $(\mathbf{R}_0, \mathbf{R}_1)$, we have also compared the burstiness of the aggregate service demands predicted by the model with the one measured on the real system using the autocorrelation function as a descriptor of burstiness [5]. The result (not shown in the figure due to limited space) indicates good prediction accuracy, with the aggregate service demand autocorrelation coefficients quickly decaying to zero for both the model and the measurements, and with the lag-1 coefficient being $\rho(1) = 0.028$ for the measured aggregate service demands and $\rho(1) = 0.039$ for the $(\mathbf{R}_0, \mathbf{R}_1)$ model.

The results of the aggregate service demand analysis indirectly suggest that the models developed in Section 2 capture per-tier service demands, otherwise it would be hard to predict accurately aggregate service demands distribution and burstiness. To further validate accuracy, we have also performed a trace-driven analysis of the system to compare the properties of the measured utilizations with those predicted by the MAP models. Figure 1(b)-(c) show the autocorrelation function of the measured and modeled utilization for the front server and database server, respectively. 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. The results are qualitatively similar for both servers suggesting that the $\mathbf{P}$ modulation impacts equally on the two tiers. For low lags, model and sample path autocorrelations are in very good agreement with the TPC-W trace. Low lags are the most significant for burstiness, as they measure the similarities of consecutive sessions to pack into bursts, while high lags are mostly related to the length of these bursts. The autocorrelation coefficient values for lags greater than 10 seem instead to suffer significant noise due to limited measurements available from utilization sampling; the presence of noise is proved by the difference between the sample path curve and the model results averaged over 100 experiments. Yet, the good agreement of the sample path with the trace proves that sample paths of the MAP model are representative of system behavior observed in real experiments. Summarizing, the experiment in this section suggest that the proposed PH-type and MAP model can summarize and predict effectively the properties of the demands in both the pre-existing benchmarks and in the composed benchmark B^*. The next case study focuses instead on the quality and practical impact of the burstiness generation methodology.

3.2 Generation of Burstiness and its Impact

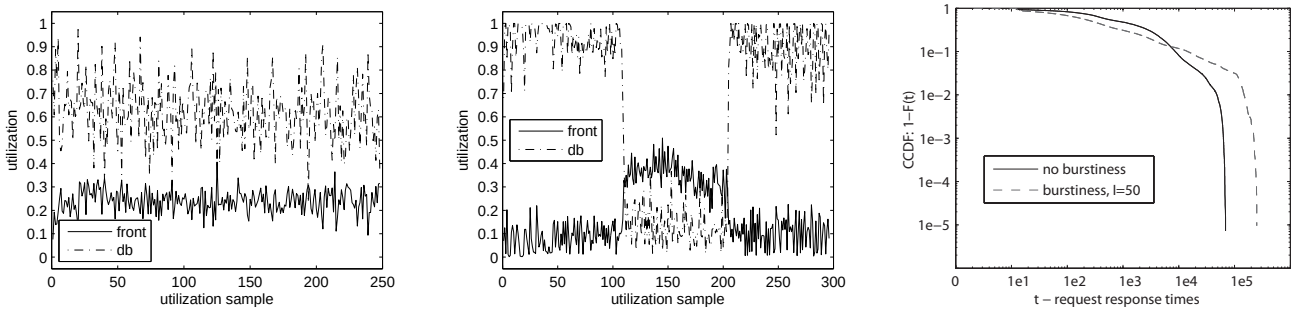
We consider the same mix of ordering and shopping sessions evaluated in Section 3.1, but we now focus on generating benchmarks to assess the performance under burstiness conditions and compare it with respect to the non-bursty case.

Solving the nonlinear optimization algorithm defined in Section 2.3 with the fmincon function of MATLAB 7.6.0, we have obtained two policies $\mathbf{P}_{non-bursty}$ and $\mathbf{P}_{high-bursty}$ that both combine shopping and ordering sessions with mix $\beta_s^* = \beta_o^* = 0.50$. The two benchmarks differ only for the index of dispersion values in the aggregate demand. The *non-bursty benchmark* has index of dispersion $I = 1.14$, which is a case corresponding to the removal of burstiness from the aggregate demand by imposing a zero value for all the autocorrelation coefficients. This also results in negligible burstiness in the service demands of the servers: the expected index of dispersions at the front-server and at the database server are $I_{fs} = 0.82$ and $I_{db} = 2.11$, respectively. Note that the scale of the index of dispersion is comparable to the scale of the squared coefficient of variation CV^2. The *high-burstiness benchmark* has index of dispersion $I = 50$,



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

Figure 1: Experimental results for the mix of shopping and ordering sessions.



(a) $I = 1.14$, No Burstiness (b) $I = 50$, High Burstiness (c) CCDF Response Times

Figure 2: Impact of burstiness on the performance of the TPC-W testbed.

which creates large burstiness both in the aggregate and per-server service demands. The expected dispersion at the front server is $I_{fs} = 597.34$ and at the DB is $I_{db} = 2242.9$.

Figure 2 compares the performance impact of the two benchmarks on the TPC-W system for an experiment with Poisson session arrivals and multiple-concurrent sessions in execution. Even though both benchmarks have the same session type mix, session inter-arrival time distribution and approximately the same server utilizations, the bursty benchmark stresses the system differently from the non-bursty benchmark. From Figures 2(a)-(b), the front and DB server CPU utilizations display a more non-uniform pattern for the bursty benchmark. In fact, Figure 2(b) shows that the bursty benchmark causes a bottleneck switch around the 100th utilization sample resulting in the front server becoming the bottleneck, a behavior absent in Figure 2(a). Bottleneck switch is typical of burstiness, but it has never been found in the ordering and shopping mixes of TPC-W [4]. It is also remarkable that the heightened contention in the first part of the experiment among sessions of ordering type caused the server to drop several connections leading to a 25% drop in throughput relative to the non-bursty case. From Figure 2(c), the tail of the response time distribution for those requests that were not dropped is heavier for the bursty benchmark proving burstiness degradations. Additional plots providing further detail on the response times received by each session type are given in [2]. Summarizing, this experiment validates the capability of our approach to insert burstiness in workloads and help uncover bottlenecks that are not exposed with non-bursty benchmarks.

4. CONCLUSION

We have proposed a model-based methodology for automatic generation of benchmarks with customizable levels of burstiness in the service demands. Our methodology extends existing approaches for benchmark synthesis such as SWAT [3]. Experiments on a real TPC-W testbed have shown that our models are very accurate in predicting service demands and their burstiness at the different tiers. We have shown a case where the ordering and shopping mixes of TPC-W have been combined to insert controlled burstiness in the demands resulting in stress conditions for performance that are not shown by non-bursty combinations of the two mixes.

We plan to further develop and validate this new approach within a framework that aims to characterize, synthesize and predict the impact of burstiness for multi-tier systems. We also plan to investigate the various kinds of system level degradations that can be caused by burstiness.

5. REFERENCES

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

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

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

[4] 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.

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

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