BAP: A Benchmark-driven Algebraic Method for the Performance Engineering of Customized Services

Invited Paper

Jerry Rolia
Automated Infrastructure Lab
Hewlett Packard Laboratories
Bristol, U.K.
jerry.rolia@hp.com

Diwakar Krishnamurthy
Department of Engineering
University of Calgary
Calgary, Alberta, Canada
dkrishna@ucalgary.ca

Giuliano Casale
SAP Research
Belfast, Northern Ireland
giuliano.casale@sap.com

Stephen Dawson
SAP Research
Belfast, Northern Ireland
stephen.dawson@sap.com

ABSTRACT

This paper describes our joint research on performance engineering methods for services in shared resource utilities. The techniques support the automated sizing of a customized service instance and the automated creation of performance validation tests for the instance. The performance tests permit fine-grained control over inter-arrival time and service time burstiness to validate sizing and facilitate the development and validation of adaptation policies. Our novel research on sizing also takes into account the impact of workload factors that contribute to such burstiness. The methods are automated, integrated, and exploit an algebraic approach to workload modelling that relies on per-service benchmark suites with benchmarks that can be automatically executed within utilities. The benchmarks and their performance results are reused to support a Benchmark-driven Algebraic method for the Performance (BAP) engineering of customized services.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; D.2.8 [**Metrics**]: Performance Measures

General Terms

Performance, Measurement, Design

WOSP/SIPEW'10, January 28–30, 2010, San Jose, California, USA.

1. INTRODUCTION

Virtualization has enabled the dynamic provisioning of Software as a Service (SaaS) in large shared resource utilities. Service instances can be configured in portals and automatically rendered onto shared resource pools of computing, network, and storage resources. In such utilities capacity sizing for a service is less risky than when provisioning dedicated hardware because the capacity offered to a service can be adjusted quickly on the fly post-deployment. However significant performance risks still remain. Each instance of a service may be customized to use a subset of features so that its performance behaviour is unique. A utility may support hundreds or thousands of service instances so performance management methods must be automated, the methods must satisfy each instance's performance requirements, and the methods must be accurate enough to make efficient use of resources. This is a challenge because the relationship between the performance of each service instance and its capacity can be complex, particularly in multi-tier applications with many potential bottlenecks. Furthermore utilities may dynamically vary the capacity associated with services to make more efficient use of resources and to decrease energy costs. Our joint research has focused on the development of automated techniques to support performance management of customized service instances in shared resource utilities. This paper describes our progress and experiences to date.

The methods we are developing exploit an algebraic approach to workload modelling. Basically, each service is characterized using a number of service-specific benchmarks that can be run and measured in an automated manner within a utility environment. Each benchmark acts as a semantically correct workload with a particular business object mix, where business objects of the service contain logic that causes demands on the system. It is important that such workloads are semantically correct, i.e., that they use business objects in a correct sequence. A customized service instance also has a particular business object mix that is likely different from the mix of any of the benchmarks. Thus it is likely to cause different demands on the system. However the customized business object mix may be estimated as a linear combination of the pre-existing business

object mixes of the benchmarks. Thus we can reuse a subset of benchmarks in certain combinations and in certain orders to match the customized workload mix and other workload properties. The subset of benchmarks acts as an algebraic basis [16]. This enables an algebraic and automated approach to performance engineering for customized service instances.

We have found that by matching a business object mix using the algebraic approach we are able to estimate the resource demands of a customized service instance. This provides workload specific demand values for parameters of a performance model for the customized service instance. We assume that each service has a pre-existing performance model that has been prepared by a performance analyst. The automation step saves effort by assigning appropriate parameter values for the model for each separate customized service instance. The same linear combination of benchmarks can also be used to automatically prepare customized performance validation tests for a service instance. We have enhanced the tests to consider various aspects of inter-arrival time and service demand burstiness. The tests are used to study the sensitivity of system performance to such workload features. Finally, the burstiness examples shed light on the complexity of the relationship between response times and capacity. We describe a technique we have developed to take into account such burstiness to improve the predictive accuracy of performance models so they can be used to better support performance management.

Section 2 gives an overview of related work. Section 3 describes a method for customizing service instances of SAP ERP systems in a service on demand environment. The method provides a practical motivation for our algebraic approach for workload modelling. Sections 4 and 5 describe recent work on demand estimation and synthesizing performance validation tests. Section 6 demonstrates the use of a layered queueing model (LQM) to predict the behaviour or a SAP ERP system for a sales and delivery workload. It also presents a study for a TPC-W system that shows how a recently developed technique named the Weighted Average Method (WAM) can use traces from performance validation tests with burstiness to improve the accuracy of predictive models. Summary and concluding remarks are offered in 7.

2. RELATED WORK

This section describes key works that are related to or have contributed directly to this paper. We refer to reader to more complete related works in [13][28][6][27][14].

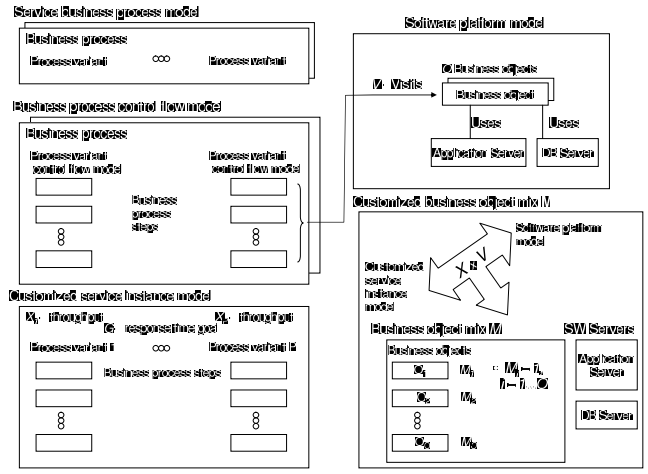
This paper presents techniques we have developed using the notion of an algebraic approach to workload modelling. Litoiu et. al [18][19] consider mixes of workloads that cause combinations of bottlenecks in distributed systems to saturate. They employ a linear programming based approach to find combinations of workloads that achieve certain properties. Dujmovic [8] describes benchmark design theory that models benchmarks using an algebraic space and minimizes the number of benchmark tests needed to provide maximum information. Dujmovic's work also informally describes the concept of interpreting the results of a ratio of different benchmarks to better predict the behaviour of a different but unmeasured system however no formal method is given to compute the ratio. Krishnaswamy and Scherson [15] also model benchmarks as an algebraic space but also do not consider the problem of finding such a ratio.

Krishnamurthy *et al.* [12][13] introduce the Synthetic Web Application Tester (SWAT) which includes a method that automatically selects a subset of pre-existing user sessions from a session based e-commerce system, each with a particular URL 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 mix and a particular session length distribution and to prepare a corresponding synthetic workload to be submitted to the system. They showed how such workload features impact the performance behaviour of session based systems. We consider results from [13] in Sections 5.1 and 5.2. BAP exploits the ratio computation technique in this work to automatically find a basis of benchmarks and to compute a ratio of the benchmarks that enables the creation of customized performance model and performance validation test for a service instance.

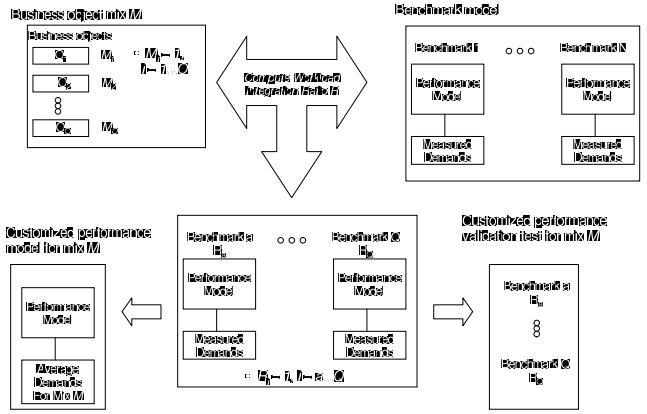
Bard and Shatzoff studied the problem of characterizing the resource usage of operating system functions in the 1970's [3]. The system under study did not have the ability to measure the resource demands of such functions directly. The execution rates of the functions and their aggregate resource consumption were measurable and were recorded periodically. This data served as inputs to a regression problem. Bard used the least squares technique to successfully estimate per-function resource consumption for the study. We summarize results for a new Demand Estimation method with Confidence intervals (DEC) that solves the same problem [28] in Section 4. DEC has certain advantages. We refer to that paper's related work for more on the topic.

Burstiness in service demands has recently been shown to be responsible for major performance degradations in multi-tier systems [23]. 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 [17] and there are many benchmarking tools that can shape correlations between arrivals [4, 11, 22, 13]. In contrast, service demand burstiness can be seen as the result of serially correlated service demands placed by consecutive requests on a hardware or software system [26, 23, 21], rather than a feature of the inter-arrival times between requests. It is much harder to model and predict system performance for workloads with service demand burstiness than for traditional workloads [7]. This stresses the need for benchmarking tools that support analytic and simulation techniques to study the performance impact of service demand burstiness. Casale et. al [6] present an approach for supporting the generation of performance validation tests that control service demand burstiness for session based systems. We refer to this approach here as a Burstiness eNabing method (BURN). A summary of results from the approach are considered in Section 5.3.

Queueing Network Models (QNMs) [5] and Mean Value Analysis (MVA) [25] have been used to study computer system performance since the early 1970's. Researchers have recently applied them directly to the study of multi-tier systems [32]. Layered Queueing Models (LQM) [29][33] are based on QNMs, and were developed starting in the 1980's to consider the performance impact of software interactions in multi-tier software systems, e.g., systems that have contention for software resources such as threads. Tiwari *et al.*[31] report that layered queuing networks were more appropriate for modeling a J2EE application than a Petri-



(a) Customized Business Object Mix



(b) Customized Model and Performance Valiation Test

Figure 1: Customized Service Business Object Mix, Performance Model, and Validation Test

Net based approach because they better addressed issues of scale. Balsamo *et al.* conclude that extended QNM-based approaches, such as LQMs, are the most appropriate modeling abstraction for multi-tiered software environments [2].

Rolia *et al.* describe the validation of an LQM for a SAP ERP system [27]. A summary of results are offered in Section 6.1. The Weighted Average Method (WAM) was recently proposed as a method for taking into account the impact of inter-arrival time burstiness in predictive analytic models [14]. A summary of results of [14] is presented in Section 6.2 and we refer to that paper for more extensive related work.

3. CUSTOMIZING SERVICE INSTANCES

This section gives an example of a method for customizing a service instance for an on-demand business process platform. The example motivates the algebraic approach to workload modelling.

Figure 1 illustrates the method. Figure 1(a) shows a service that supports business processes for customers. Examples of business processes include customer relationship management, supply chain management, and sales and distribution [30]. These can be complex processes that implement how a business interacts with its partners and customers.

The service in Figure 1(a) has many business processes each with many process variants such as order entry, sale, and return. A service could have hundreds or thousands of process variants. Each process variant has a number of process steps each of which uses business objects[1] in application servers and database servers. The business objects implement application and database logic that determine resource demands on the system. Process variants can be profiled to discover the average number of visits by a variant to each business object. A customized service instance defines which process variants are to be used by a customer. This determines which software modules are available to the end users of the service instance. From a performance perspective, it is also necessary to specify the throughput for each of the variants, e.g., the number of times per hour the variant is started, and a goal for mean response time for interactions with the system. A simple cross product of the throughputs specified when customizing the system and visit values found when profiling the system gives the expected number of visits to each business object, e.g., per hour, and when normalized with respect to one to gives the probability of invocation for each business object. This is the customized business object mix M for the customized service instance. It specifies the relative proportion of system functions for the customized service instance in terms of business objects.

Figure 1(b) illustrates how the customized business object mix M can be used to automatically compute parameters for a customized performance model and to automatically prepare a performance validation test. Our approach assumes that the service has a performance model that is prepared by an analyst. Further, each service has a suite of benchmarks that can be automatically executed and measured in the utility. For example, there may be N benchmarks. Each benchmark is executed and measured to obtain its business object mix and resource demand values, e.g., CPU time per request, for the service's performance model. It is possible to automate the execution of such benchmarks in shared resource utilities during times when resources are underutilized. The measured values can be reused to support performance engineering for different customized service instances.

An Algebraic Approach

The customized business object mix M and benchmark business object mixes can be thought of as vectors. A linear programming algorithm can be used to find a basis Q of the N benchmarks. The basis enables the customized business object mix M to be estimated as a linear combination R of the object mixes of the Q benchmarks. For example, a customized service instance with business object mix M may be similar to a workload with $R = 50\%$, 30%, and 20% of sessions from $Q = 3$ benchmarks named Ben_1, Ben_2, and Ben_3, respectively. The SWAT algorithm [13] can be used to find the Q benchmarks and to estimate R. SWAT also permits the specification of a matching tolerance for each

[1]Our use of business object refers to a class of business object rather than any instance of the class of business object.

business object so that those that affect performance most can be matched more exactly.

Once the ratio R of benchmarks is known, R can be used to estimate the demands for the customized performance model corresponding to the customized business object mix. The demands are simply a linear combination, using R, of the measured benchmark demands. Furthermore, replaying benchmark sessions in proportion R synthesizes the customized object mix. Therefore the benchmarks can also be reused to automatically create performance validation tests. The following two sections give case study results for this approach.

4. DEMAND ESTIMATION WITH CONFIDENCE (DEC)

Demand estimation in software performance is often conducted using regression techniques. However, regression makes certain assumptions that can lead to inaccuracies in demand predictions. For example, in the current context, regression assumes that visits to objects are not correlated. Yet software systems typically use objects in correlated ways. For example, functions such as add to cart and buy are often used together. This introduces the problem of multi-colinearity into the regression problem which can lead to inaccurate estimates for per-object demands. Furthermore, for confidence interval calculations, regression assumes that the demand caused by a particular mix of objects is deterministic, any variation is assumed to be due to measurement error. The variation is allocated to an error term that is assumed to have values that are Normally distributed and are used to compute a confidence interval. Yet computing system resource demands are rarely deterministic and the corresponding errors are not always Normally distributed. Thus confidence interval estimates from regression for computing system applications are not always statistically sound or reliable.

We have proposed a Demand Estimation method with Confidence intervals (DEC) for estimating demands that overcomes these problems [28]. DEC uses the ratio of benchmarks R to estimate demands as illustrated in Figure 1(b). With DEC, confidence interval calculation is straightforward. Several independent experiment replications are conducted for each benchmark. Each benchmark replication yields a benchmark replication demand. The mean of the benchmark replication demands is computed as the overall benchmark mean demand. From the central limit theorem a benchmark mean demand is Normally distributed if the number of replications is large [10]. Furthermore, a linear combination of Normally distributed independent random variables results in a random variable that is also Normally distributed. Since DEC uses a linear combination of measured benchmark mean demands to predict the resource demand of a business object mix, the predicted demand is also Normally distributed. This allows confidence intervals to be computed with certainty for the predictions from DEC.

Figure 2 compares DEC with the Least Squares (LSQ) and Least Absolute Deviation (LAD) regression techniques that are often cited in the literature [28]. The results were obtained from a multi-tier TPC-W system deployed at the University of Calgary that consists of a Web 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 Web and database server nodes are used to execute the TPC-W bookstore application. The client node is dedicated for running the httperf [24] Web request generator that was used to submit benchmarks to the TPC-W system. All nodes in the setup contain an Intel 2.66 GHZ Core 2 CPU and 2 GB of RAM. We used the Windows perfmon utility to collect resource usage information from the Web and database server nodes using a sampling interval of 1 second. The CPU demands are much larger than disk and network demands for this system so we focus on demand estimation for these values.

Figures 2(a) and 2(b) give cumulative distribution functions (CDF) for estimated CPU demands as compared to measured CPU demands for customized business object mix for the server and database nodes. The figures present results for thirty cases. Figures 2(c) and 2(d) give the CDF of two-sided confidence intervals for the same cases.

In Figure 2(a), the results of DEC, LSQ and LAD are very similar with a maximum error below 15%. It must be noted that for all the cases the server node demand only changed by a factor of 1.5. There is no clear advantage for any technique.

Figure 2(b) gives the corresponding results for the DB CPU demand estimates. The demand value varied by a factor of over 200 for these cases and were important for the performance of the system. The figure shows that DEC offered much lower errors for 30% of the cases.

Figures 2(c) and (d) show that DEC's confidence interval predictions are more in line with measured confidence intervals than those predicted by the LSQ and LAD methods. DEC's confidence interval calculations are statistically sound. Note that LAD's reported confidence intervals are tighter than LSQ confidence intervals due to the nature of the LAD algorithm.

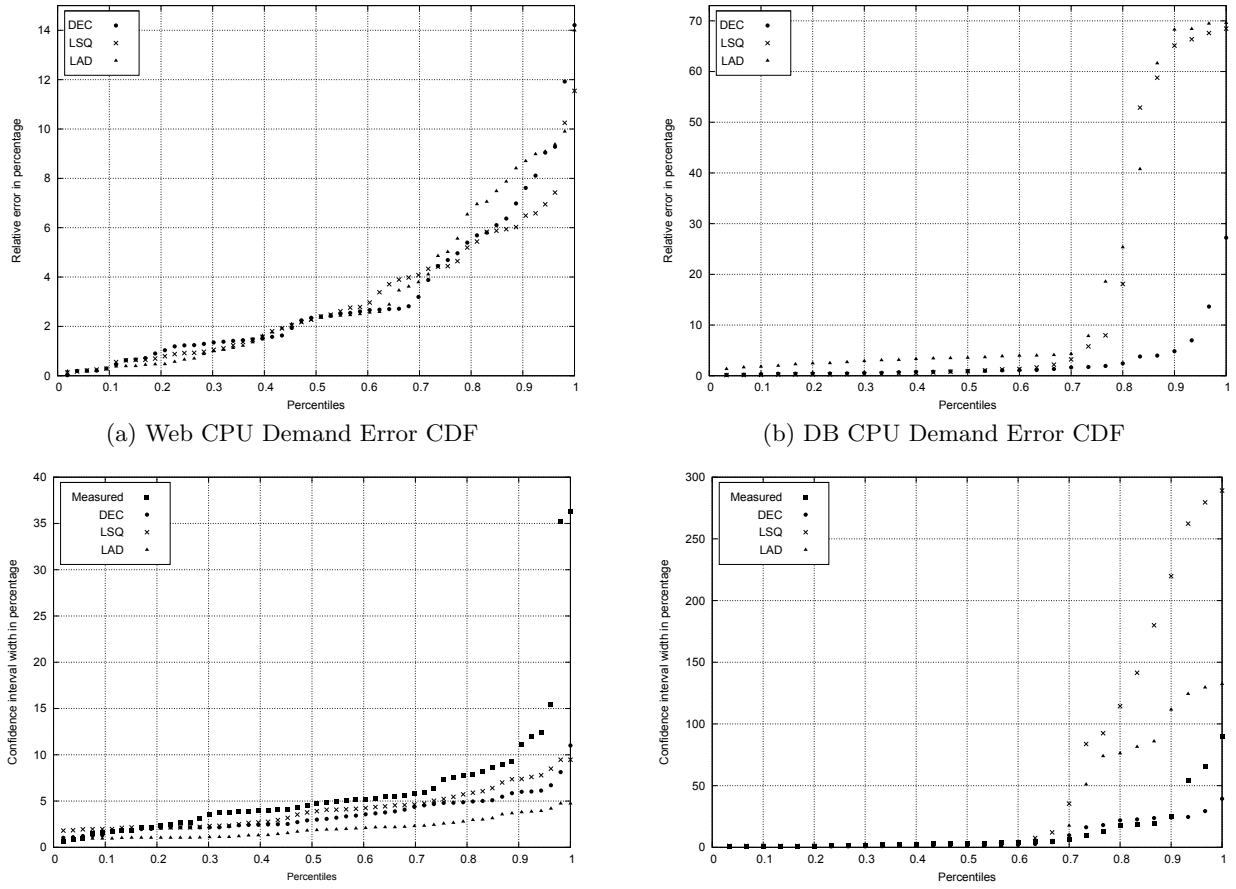
To summarize, DEC is a new technique for estimating resource demands that relies on the algebraic approach to workload modelling. It does not suffer from the problem of multi-colinearity and provides statistically sound confidence interval calculations that appear to be tighter and closer to measured confidence intervals than the regression based techniques. The demands predicted by DEC can be used in customized performance models. The next system considers the creation of performance validation tests.

5. CUSTOMIZED PERFORMANCE VALIDATION TESTS

Just as it is possible to automatically render service instances into shared resource pools, it is also possible to automatically generate and execute performance tests that validate the performance of such instances [9]. Customized performance validation tests can be used to stress such systems to verify that they are able to satisfy performance requirements.

The section gives examples of performance validation tests that match a customized workload mix, that match a mix and session length distribution, that match a mix and think time distribution, and that match a mix and service demand burstiness. The techniques are based on the algebraic approach to workload modelling.

Sections 5.1 and 5.2 rely on case study data from a multi-tier TPC-W system at Carleton University as described



(a) Web CPU Demand Error CDF

(b) DB CPU Demand Error CDF

(c) Web CPU Demand CI CDF

(d) DB CPU Demand CI CDF

Figure 2: Comparison of Demand Prediction and CI Errors between DEC and Regression Techniques

in [13]. The Carleton system had topology similar to the University of Calgary system described in Section 4 but with different software and hardware. Section 5.3 relies on case study data from the multi-tier TPC-W system at the University of Calgary.

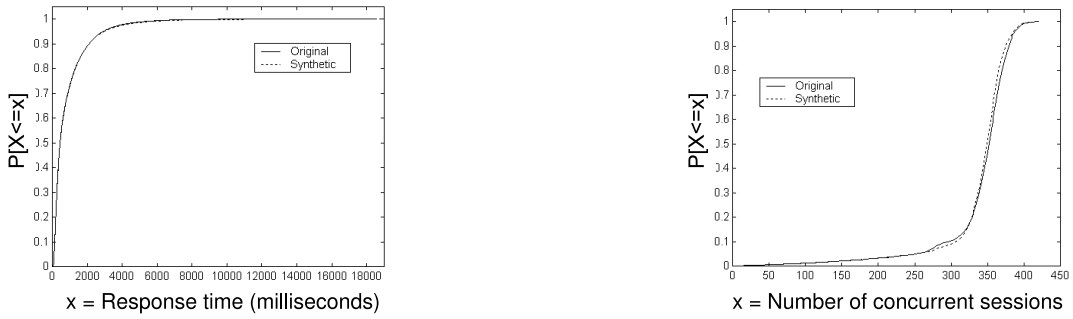
5.1 Matching Functional Mix with SWAT

Once computed, the ratio R of the Q chosen benchmarks can be used to automate the creation of a performance validation test. As illustrated in Figure 1(b), sessions in the test are simply drawn from the Q benchmarks according to the probabilities in R. Figure 3 compares two experiment runs for the TPC-W system. The original workload is generated using the TPC-W emulated browser workload generator. Forty thousand sessions were generated with 838 unique business object mixes and a certain overall business object mix. Next we treated the 838 unique sessions from the original workload, i.e., sessions with different object mixes, as our set of benchmarks. Using the linear programming method, a subset of $Q = 420$ of the benchmarks were chosen as the basis along with the ratio R. A new trace of fouty thousand sessions was generated using the Q benchmarks and the ratio R. This corresponds to the synthetic workload in the figure, and corresponds to a performance validation test.

Figure 3(a) shows the response time distributions for the original and synthetic workloads [13]. They are nearly overlapping. Figure 3(b) shows the distribution for the number of concurrent users. Again they are almost identical. The synthetic workload matched the business object mix M, the response time distribution, device utilization values, and session population distribution. A smaller set of benchmarks, i.e., fewer than Q benchmarks, would approximate the business object mix and performance measures.

5.2 Inter-arrival Time Burstiness with SWAT

The approach of Section 5.1 has been enhanced in two ways to evaluate the impact of inter-arrival time burstiness on the TPC-W system. For the first enhancement, burstiness due to session length distribution was considered. To support this, additional constraints were added to the linear programming formulation so that the sessions for an experiment satisfied both a customized business object mix and a desired session length distribution as specified using a probability distribution function (pdf). Each point in the sample space of the pdf corresponds to a session length range. For example, the first point may have probability 0.1 and correspond to sessions with length 1 to 3. The second point may have probability 0.9 and correspond to sessions with length 4 to 12. The linear programming solution was repeated for each different point in the sample space. The mix that corresponds to each sample point has its selected benchmarks weighted by both the corresponding point's probability from the pdf and the benchmark's corresponding R value from its solution of the linear program. The benchmarks from all



(a) Response Time CDF

(b) Concurrent Session Population CDF

Figure 3: Comparison of Original Workload and Performance Validation Test

the points contribute to an integrated workload mix for the whole session length distribution.

For the second enhancement, we considered burstiness due to think time distributions. Table 1 shows the impact of mean response time and the 95-percentile of response time for several experiments [13]. The empirical workload had think times and session lengths that corresponded to data collected from a large e-commerce system [1]. Exponential corresponds to the exponential distribution for think times and session lengths. Heavy-tailed session length used a bounded Pareto distribution for session lengths and the empirical distribution for think times. Heavy-tailed think time used a bounded Pareto distribution for think times and empirical distribution for session lengths. In all cases, the mean session length and mean think time was the same. Furthermore the per resource utilizations and throughput were the same for each of the runs. The Table shows that heavy tailed session lengths could increase mean response time by over 25% and the 95-percentile of response time by nearly 50% even though the overall resource utilizations and request throughput were the same for each run. Heavy tailed think times also had a significant impact. This demonstrates aspects of the complex relationship between software performance and capacity.

Figure 4 compares the response time CDF and CDF for the number of concurrent sessions for the empirical and heavy-tailed session length distribution cases. Figure 4(a) shows more larger response times and fewer small response times for the heavy-tailed case. Figure 4(b) suggests that the increases in response times are because arriving sessions have a higher probability of arriving when there are more concurrent sessions being served than for the empirical case. Similar results are available for heavy tailed think time distributions [13] but are omitted here due to space limitations.

This subsection has shown that the algebraic approach to workload modelling can be used to create performance validation tests that match workload mix as well as customizable session length and think time distributions. The methods can be employed to evaluate the sensitivity of a system to such workload features using automatically generated performance validation tests.

5.3 Service Demand Burstiness with BURN

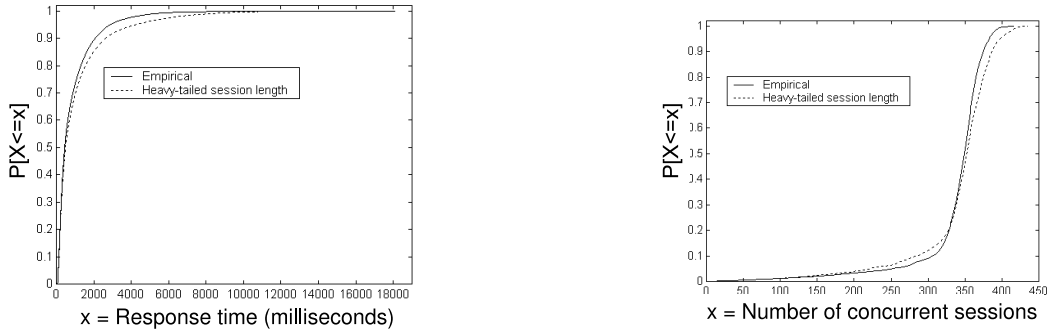
Service demand burstiness is an interesting feature of systems [23]. Changes in workload mix can cause bottleneck shifts in systems with potentially significant impact on response times. We have extended the linear programming approach further to control bottleneck switching in multi-tier systems. The extension takes as input a measure of burstiness called the index of dispersion [21]. The index of dispersion is the product of the coefficient of variation squared of service demands and the lag-k autocorrelation coefficient of the service demands. We use the index of dispersion to control, in a statistical manner, the ordering of sessions [6]. We refer to this as a BURstiness eNabling method (BURN). A low index of dispersion behaves like the algorithm of Section 5.1 with sessions chosen from benchmarks at random using the ratio R. A high index of dispersion results in the same business object mix but increases the likelihood of sessions from the same benchmark appearing in sequence. If different benchmarks have different bottlenecks then this can cause switches in bottlenecks. The full algorithm for controlling service demand burstiness is given in [6].

Figure 5 demonstrates the impact of service demand burstiness [6]. Two tests were run against the TPC-W system as described in Section 4. The first test had an index of dispersion of 1.14 which corresponds to low burstiness, with the squared coefficient of variation of service demands approaching a value of one. The second case had an index of dispersion of 50 which corresponds to high burstiness. Figures 5(a) and (b) show examples of time-varying CPU usage for the front server and database server. For the low burstiness case resource usage is consistent over time. For the high burstiness case we see heavy use of the server node CPU and then a switch in bottleneck to the database node CPU. Note that for both the server node CPU and database node CPU, the average CPU utilization over the entire run was the same for the two tests.

Figures 5(c) and (d) give the mean number of requests over time for the three types of benchmark sessions that make up the workload. For the low burstiness case the numbers are consistent over time. For the high burstiness case

Table 1: Effects of Session Length and Think Time Distributions

Workload	Mean R	95p R	$\frac{95p\ R \text{ Workload}}{95p\ R \text{ Empirical}}$
Empirical	0.86	2.90	1.0
Exponential	0.83	2.83	0.98
Heavy-tailed session length	1.09	4.27	1.47
Heavy-tailed think time	0.96	3.40	1.17



(a) Response Time CDF

(b) Concurrent Session Population CDF

Figure 4: Comparison of Empirical Workload and Inter-arrival Burstiness due to Session Length Distribution

we see the proportion of requests of each type changing, which explains the shift in bottleneck to the DB CPU.

Figures 5(e) and (f) illustrate mean response times for the three types of benchmark sessions for the low and high burstiness cases. The high burstiness scenario causes much greater variability in response times for all classes of requests. Finally, Figure 5(g) shows the CDF of response times over all requests for the two tests. Even thought the device utilizations and request throughputs were the same, burstiness has a significant impact on the distribution of response times.

To summarize, the algebraic approach to workload modelling can help to control service demand burstiness in automated performance validation tests. Such tests can be used to evaluate the sensitivity of a system to such workload features.

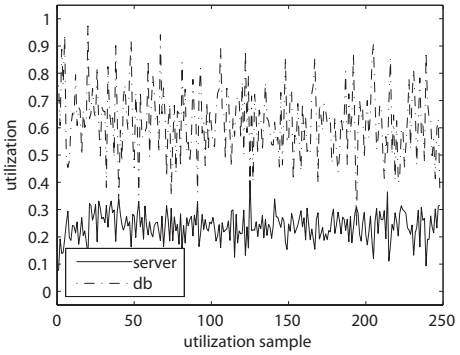
6. PREDICTIVE MODELS

The previous sections considered computing the parameters of a performance model for a customized service instance and the automated creation of performance validation tests. This section focuses on examples of performance models and their predictive accuracy. The first example demonstrates a validated model for a deployment of the SAP ERP platform for a sales and distribution workload. SAP ERP is a complex business process execution environment used by many of the world's medium and large sized businesses. Though validated, the workload for the system is relatively simple. Such models can yield inaccurate results for systems subject to more complex bursty workload behaviour. Our second example is a model for the TPC-W system deployed at Carleton University. For this system we demonstrate that a recently developed method can use burstiness information derived from performance validation tests along with the performance model to improve the accuracy of the model's performance predictions. By taking into account such burstiness with performance models we are better able to use performance modelling to help guide choices of more time consuming measurement based validation tests.

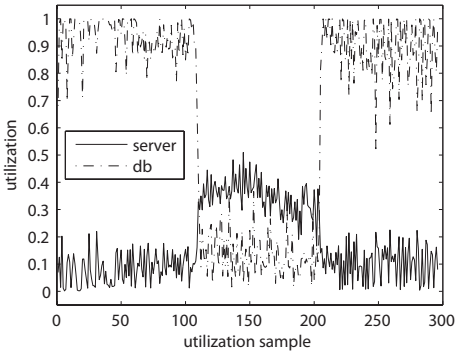
6.1 LQM for an SAP ERP System

Figure 6 illustrates a layered queueing model [29] (LQM) for a SAP ERP system [27]. The process for customizing an instance of a SAP ERP service, as described in Section 3, leads to specific resource demand parameters for the model as determined by a customer's choices of business process variants and business process variant throughputs.

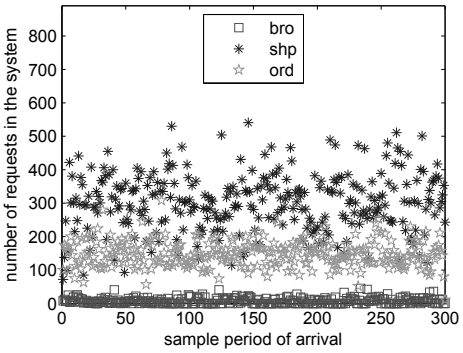
Figure 6 illustrates a LQM for the SAP ERP system under study. The system node had two virtual CPUs each with its own physical CPU. An application server and database server were both assigned to the system node. The application server is composed of a dispatcher and pools of separate operating system processes, called work processes, which serve requests forwarded by the dispatcher [30]. The dispatcher assigns each request to an idle work process. Service within a work process is non-preemptive. Work processes may access the database in a synchronous manner during which the work process remains unavailable to serve other requests due to a lack of preemptive capabilities. Requests are served by three pools of work processes. The most important pool serves dialog step requests, which are responsible for processing and updating information and data that are displayed on the client-side through the graphical user interface; this information is exchanged between client and ERP application by means of a proprietary communication protocol. Each dialog step request alternates cycles of CPU-



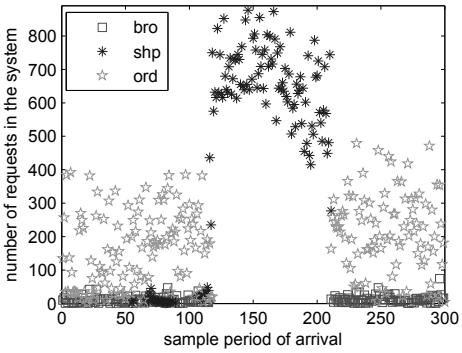
(a) $I = 1.14$, CPU U No Burstiness



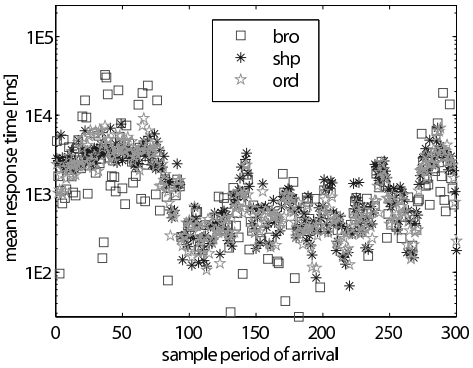
(b) $I = 50$, CPU U High Burstiness



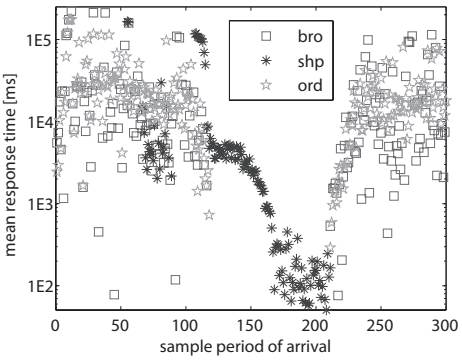
(c) Mean Num Requests No Burstiness



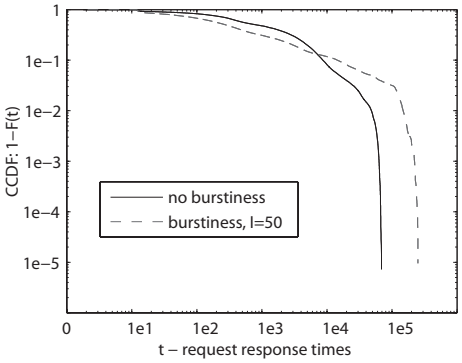
(d) Mean Num Requests High Burstiness



(e) Mean R Request Type No Burstiness



(f) Mean R Request Type High Burstiness



(g) CDF Response Times

Figure 5: Performance effects service demand burstiness causing dynamic bottleneck switch

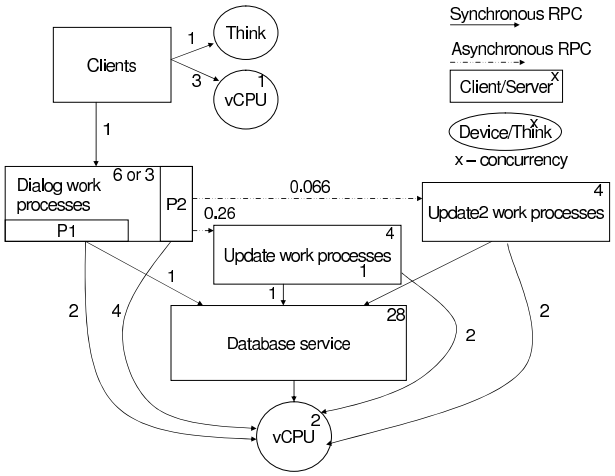


Figure 6: LQM of SAP ERP System

intensive activities and synchronous calls to the database for data retrieval. Additionally, dialog work processes generate asynchronous calls to the database that start being served only after completion of the dialog step which requests them. These asynchronous calls are of type update and update2 and are handled by separate pools of work processes. Update2 requests are processed at a lower priority by the platform.

Tables 2 and 3 offer measured and predicted results for the SAP ERP system for a sales and distribution workload [27]. The tables show the measured CPU utilization, measured dialog request response time, and predicted mean dialog response time for the six and three work process cases respectively. The results show that the performance model is well able to predict the behaviour of the system. For example, measured and predicted mean dialog response times are generally affected in the same proportion as utilizations increase and threading levels increase.

In [27], we showed that features including multi-threading for software servers, multiple processors per server, synchronous and asynchronous remote procedure calls, multiple phases of processing, and priority of access to CPU resources were needed to validate a model for the system. We note however that the workload for the system was not complex. It did not involve different types of workloads. Session lengths were always the same. Session inter-arrival and departure times are assumed to be exponentially distributed. Previous sections showed that both inter-arrival and service demand burstiness can have a significant impact on performance. That burstiness affects the relationship between response time and utilization. Predictive models such as LQMs and QNMs do not have the ability to take into account such burstiness. The next section describes a technique developed to take into account such burstiness to improve the accuracy of performance predications.

6.2 WAM: Performance Modelling with Complex Workloads

This section describes the Weighted Average Method (WAM) for taking into account burstiness in predictive performance models [14]. To date we have used the technique to study inter-arrival time burstiness in systems where per request think times are an order of magnitude or more greater than per-request response times.

WAM takes as input a performance model, e.g., an LQM that models customer sessions competing for system resources, and a trace of sessions, e.g., as generated for a performance validation test. WAM walks over the trace of sessions and uses the performance model to estimate the session population distribution for the test. Initially WAM assumes there are zero sessions in the system – as is the case at the start of a performance test. As requests from new sessions arrive WAM increments its count of the number of concurrent sessions. For each request in a session, WAM estimates the response time for the request as the mean request response time for the request, as found using a performance model with the current count for the number of concurrent sessions as the customer population in the model. The request's subsequent think time is taken as an input from the trace. When the estimated response time of the last request in a session completes WAM decreases the count of number of concurrent sessions by one. When all the sessions in the trace are processed WAM outputs its estimate for session population distribution. This is used with the performance model to predict system measures such as the mean request response time.

To demonstrate the effectiveness of WAM at improving the accuracy of predictions of performance models with bursty workloads, we conducted a case study with data from the TPC-W system deployed at Carleton University [14]. The case study considers various scenarios with inter-arrival time burstiness. We now summarize results that employ a LQM alone (LQM), without taking into account burstiness, an LQM paired with a Markov Chain Birth Death process (Markov Chain Birth Death LQM) that models session population distribution but assumes session arrival and departure times are exponentially distributed [20], WAM with the session population distribution computed empirically (Empirical WAM LQM) using measured response times from the trace but that estimates overall mean response time using the corresponding measured session population distribution with the LQM, and WAM with the session population distribution computed using an LQM for request response time estimates and the overall mean response time calculation (Monte Carlo WAM LQM).

Empirical WAM LQM gives the best possible results for the approach. Monte Carlo WAM LQM offers a constructive method that approximates Empirical WAM LQM and can also be used to conduct sensitivity analyses by either modifying think time distributions or using re-generated traces with different session length distributions. Clearly, in such cases measured response time values are no longer known so Empirical WAM cannot be applied.

Table 4 summarizes the error measures for mean response times for thirty nine experiments, including seventeen experiment replications that employed bounded Pareto session length and think time distributions [14]. Table 5 includes only the results for the seventeen cases with inter-arrival time burstiness. The results offer the mean absolute error, maximum error and trend error for each approach. The mean absolute error is a weighted measure of error, the trend error is the greatest difference between error values (including sign). A larger trend error suggests less confidence in the ability of the model to predict the impact of changes to the system.

Table 2: Measured and predicted mean dialog response times for 6 dialog workprocess

Pop	Measured CPU U	Measured DialogResp	LQM DialogResp
10	0.08	142.12	139.67
50	0.36	151.77	149.14
75	0.49	157.15	158.42
100	0.60	165.36	170.63
150	0.76	209.62	241.19
175	0.85	293.11	329.66

Table 3: Measured and Predicted mean dialog response times for 3 dialog work process

Pop	Measured CPU U	Measured DialogResp	LQM DialogResp
10	0.08	142.08	135.62
50	0.36	154.55	154.96
75	0.49	164.23	167.94
100	0.60	190.56	180.59
150	0.76	320.32	322.84
175	0.84	473.23	586.71

Table 4 shows that the LQM alone has the greatest errors. While the Markov Chain Birth Death approach does model session population distribution, it does not model burstiness so it does not improve the accuracy of predictions. The Empirical WAM approach decreases all measures of errors by 10% or more by using the empirical session population distribution along with LQM. Monte Carlo WAM LQM offers results that are nearly as accurate as those of Empirical WAM. Table 5 gives similar results but only includes the cases with inter-arrival time burstiness.

Figure 7 illustrates some interesting results for four replicates of a case with heavy tailed session length distributions. For this case system behaviour was different for each of the four runs. Figures 7(a) and (b) illustrate session population pdfs for two of the four runs. Clearly the pdfs are very different. The pdfs for the other two runs were also different. This was despite the fact that the runs were for many hours each. Significant burstiness can cause a range of system behaviours such that there is no steady state behaviour. Figure 7(c) shows the measured and estimated mean request response times for the LQM, Markov Chain Birth Death LQM, and Monte Carlo LQM approaches for the four runs. The Monte Carlo LQM approach outperforms the other approaches. The Markov Chain Birth Death LQM approach models session population distribution but not the run specific impact of burstiness on the distribution.

To summarize, this section described the WAM method. It has been used with traces of sessions, similar to those from performance validation tests, along with performance models to predict the impact of inter-arrival time burstiness on systems. The approach is trace based and as a result is compatible with the algebraic method for workload modelling.

7. SUMMARY AND CONCLUSIONS

This paper describes our progress to date on BAP, a Benchmark-driven Algebraic method for the Performance engineering of customized services in shared resource utilities. We have described a customization process for service instances that enables parameters for performance models and performance validation tests to be automatically generated. An algebraic approach is used for workload modelling. This enabled an integrated set of methods named SWAT, DEC, BURN, and WAM that support performance modelling and performance testing and that take into account important features such as inter-arrival time and service demand burstiness.

The examples presented in the paper have shown that the relationship between software performance, in terms of response times, and capacity, in terms of utilizations, is complex. There can be many different behaviours for systems with the same throughputs and utilization values. While this has certainly been demonstrated in other related fields, we are not aware of other comprehensive methods such as those presented here for studying these issues for multi-tier software systems.

Our future work includes further case studies that apply the methods to enterprise systems. We intend to further develop WAM to support more kinds of burstiness, and to support the development and validation of adaptation policies for complex applications, to predict the behaviour of services in adaptive environments, and to explore the notion of benchmark design to better choose sets of benchmarks to characterize services.

8. ACKNOWLEDGMENTS

The authors thank Amir Kalbasi and Min Xu of the University of Calgary, Stephan Kraft of SAP Research, and Sven Graupner of HP Labs for their collaboration and support of this work.

9. REFERENCES

[1] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[2] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[3] Y. Bard and M. Shatzoff. Statistical methods in computer performance analysis. In K. Chandy and R. Yeh, editors, *Current Trends in Programming*

Table 4: WAM vs BirthDeath for Modelling Session Population Distribution for All Experiments

Approach	Abs Error %	Max Error %	Trend Error %
LQM	15.2	32.4	42.5
Markov Chain Birth Death LQM	13.7	32.6	45.1
Empirical WAM LQM	5.79	15.5	28.2
Monte Carlo WAM LQM	7.2	18.4	33.2

Table 5: WAM vs BirthDeath for Modelling Session Population Distribution for Bursty Experiments

Approach	Abs Error %	Max Error %	Trend Error %
LQM	21.2	32.4	28.0
Birth Death LQM	19.2	32.6	30.1
Empirical WAM LQM	4.93	13.8	25.5
Monte Carlo WAM LQM	7.1	18.4	31.2

Methodology, Volume III, Software Modelling. Prentice-Hall, Englewood Cliffs, N.J., 1978.

[4] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *ACM Performance Evaluation Review*, 26(1):151–160, 1998.

[5] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. of the ACM*, 16(9):527–531, 1973.

[6] G. Casale, A. Kalbasi, D. Krishnamurthy, and J. Rolia. Automatic stress testing of multi-tier systems by dynamic bottleneck switch generation. In *To appear in Proc. of USENIX Middleware*, December 2009.

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

[8] J. J. Dujmovic. Universal benchmark suites. In *In Proc. of MASCOTS*, pages 197–205, 1999.

[9] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia. Vats: Virtualized-aware automated test service. In *In Proc. of Quantitative Evaluation of Systems*, 2008.

[10] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[11] K. Kant, V. Tewary, and R. Iyer. An internet traffic generator for server architecture evaluation. In *In Proc. CAECW*, January 2001.

[12] D. Krishnamurthy. *Synthetic Workload Generation for Stress Testing Session-Based Systems*. Ph.D. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, January 2004.

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

[14] D. Krishnamurthy, J. Rolia, and M. Xu. Wam: The weighted average method for predicting the performance of systems with bursts of customer sessions. *To appear in IEEE Transactions on Software Engineering*, 2009.

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

[16] P. Lax. *Linear Algebra*. John Wiley & Sons, New York, 1997.

[17] 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.

[18] M. Litoiu, J. Rolia, and G. Serazzi. Designing process replication and threading policies: A quantitative approach. In *In Proc. 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, LNCS*, volume 1469, pages 15–26, 1998.

[19] M. Litoiu, J. Rolia, and G. Serazzi. Designing process replication and activation, a quantitative approach. *IEEE Transactions on Software Engineering*, 26(12):1168–1178, 2000.

[20] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.

[21] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In V. Issarny and R. E. Schantz, editors, *Middleware*, volume 5346 of *Lecture Notes in Computer Science*, pages 265–286. Springer, 2008.

[22] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *In Proc. of ICAC*, June 2009.

[23] 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.

[24] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *In Proc. of Workshop on Internet Server Performance*, June 1998.

[25] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):312–322, 1980.

[26] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *Proc. of the 3rd Conf. on Quantitative Evaluation of Systems (QEST)*, pages 41–50. IEEE Press, 2006.

[27] J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, and S. Kraft. Predictive modelling of sap erp applications: Challenges and solutions. In *To Appear in Proc. of the Workshop on Run-time Models for Self-managing Systems and Applications*, Oct 2009.

[28] J. Rolia, D. Krishnamurthy, A. Kalbasi, and S. Dawson. Resource demand modeling for multi-tier services. In *To Appear in Proc. of the Workshop on Software and Performance*, Jan 2009.

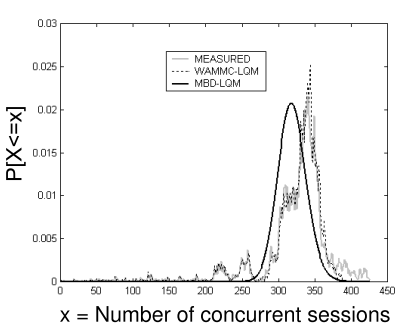
[29] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. on Software Engineering*, 21(8):689–700, 1995.

[30] T. Schneider. *SAP Performance Optimization Guide*. SAP Press, 2003.

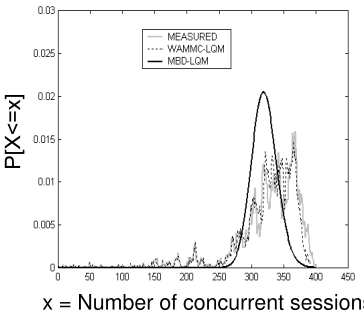
[31] N. Tiwari and P. Mynampati. Experiences of using lqn and qpn tools for performance modeling of a j2ee application. In *In Proc. of Computer Measurement Group (CMG) Conference*, pages 537–548, 2006.

[32] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. of ACM SIGMETRICS*, pages 291–302. ACM Press, 2005.

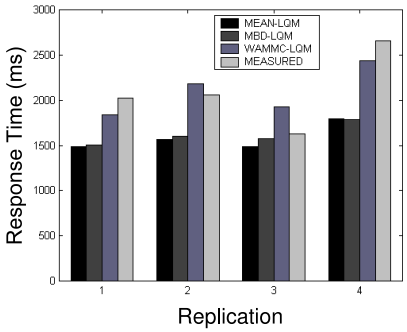
[33] M. Woodside, J. E. Nielsen, D. C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, 44(1):20–34, January 1995.



(a) Session Pop Distribution Example 1



(b) Session Pop Distribution Example 2



(c) Mean Response Time Comparison for Four Runs

Figure 7: Predicted and Measured Mean Response Times with Heavy Tail Distribution