Dealing with Burstiness in Multi-Tier Applications: Models and Their Parameterization

Giuliano Casale*, Ningfang Mi*, Ludmila Cherkasova*, and Evgenia Smirni
*Member, IEEE

Abstract—Workloads and resource usage patterns in enterprise applications often show burstiness resulting in large degradation of the perceived user performance. In this paper, we propose a methodology for detecting burstiness symptoms in multi-tier applications but, rather than identifying the root cause of burstiness, we incorporate this information into models for performance prediction. The modeling methodology is based on the index of dispersion of the service process at a server, which is inferred by observing the number of completions within the concatenated busy times of that server. The index of dispersion is used to derive a Markov-modulated process that captures well burstiness and variability of the service process at each resource and that allows us to define queueing network models for performance prediction. Experimental results and performance model predictions are in excellent agreement and argue for the effectiveness of the proposed methodology under both bursty and non-bursty workloads. Furthermore, we show that the methodology extends to modeling flash crowds that create burstiness in the stream of requests incoming to the application.

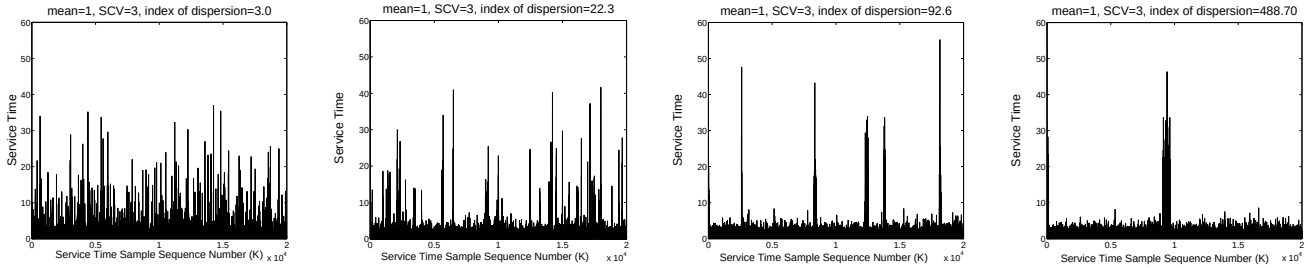
Index Terms—Capacity planning, multi-tier applications, bursty workload, bottleneck switch, index of dispersion.

1 Introduction

The performance of a multi-tier application is determined by the interactions between the incoming requests and the different resources and software components that serve them. In order to model these interactions for capacity planning, a detailed characterization of the workloads and of the application is needed, but a white-box analysis may be very time consuming, error-prone, and simply infeasible for complex commercial applications. An alternative approach is to rely on live system measurements and to assume that the performance of each software or hardware resource is completely characterized by its *mean* service time, a quantity that is easy to obtain with simple estimation procedures, e.g., by multivariate linear regression of utilization against throughput [19]. The mean service times of different classes of transactions together with the transaction mix can be used as inputs to the widely-used Mean Value Analysis (MVA) models [10], [21], [24] to predict the overall system performance under various load conditions. The popularity of MVA models, also called in the literature product-form queueing networks, is due to their simplicity and their ability to capture complex systems and workloads in a straightforward manner. In this paper, we present strong evidence that MVA models of multi-tier architectures can be unacceptably inaccurate if the processed workloads exhibit *burstiness*, i.e., uneven spikes of congestion that are observed during the lifetime of the system. Motivated by this problem, we define a new methodology for effective capacity planning for systems processing workloads with burstiness.

Internet flash-crowds are familiar examples of burstiness and are characterized by periods of continuous peak arrival rate that significantly deviate from the average traffic intensity. Similarly, a footprint of burstiness in system workloads is the presence of peaks in utilization measurements, which indicates that periodically the server has no spare capacity. In multi-tier systems, congestion may arise by the super-position of several events including database locks, variability in service time of software or database operations, memory contention, caching, and/or due to the characteristics of the scheduling algorithms used. The above events interact in a complex way both with the underlying hardware/software systems and with the incoming requests, often resulting in periods where the entire architecture is significantly slowed down. For example, consider a multi-tier system with an over-sized database server, a locking condition on a database table may still slow down the service of multiple requests that try to access the same critical section making the database the bottleneck server for a time period. During that period of time, the database congestion dominates the performance of the overall system, even though most of the time another resource, e.g., the application server, may be the primary cause of delays in the system *in the long run*. Thus, the performance of the multi-tier system can vary in time depending on which is the current bottleneck resource and can be significantly conditioned by *dependencies* between servers that cannot by captured by MVA models. Still, no simple

- *G. Casale is with Imperial College London, Department of Computing, 180 Queen's Gate, London SW7 2AZ, UK; email: g.casale@imperial.ac.uk.*
- *N. Mi is with Northeastern University, Boston, MA 02115; email: ningfang@ece.neu.edu*
- *L. Cherkasova is with Hewlett-Packard Laboratories, Palo Alto, CA 94304, US; email: lucy.cherkasova@hp.com*
- *E. Smirni is with the College of William and Mary, Computer Science Department, 23187 Williamsburg, VA, US; email: esmirni@cs.wm.edu*
- *This work has been partially supported by NSF grants CNS-0720699 and CCF-0811417, a gift from HPLabs, and the Imperial College JRF.*
- *A preliminary conference version of this paper titled appeared in [15].*



(a) (b) (c) (d)

Fig. 1. Four workload traces with identical hyper-exponential distribution (mean $\mu^{-1} = 1$, $SCV = 3$), but different burstiness profiles. The index of dispersion I grows with the tendency of the trace to aggregate into large bursts.

methodology exists that captures in a simple way this time-varying *bottleneck switch* in multi-tier systems and its performance implications across the tiers.

In this paper, we present a methodology to integrate workload burstiness in performance models, which relies on server busy times and measurements of request completions within these busy times. Busy periods are obtained from server utilization measurements across time. We show that burstiness in the service process can be inferred from these traces using the *index of dispersion* [5] of completed requests, a measure of burstiness frequently used in the analysis of time series and network traffic. Using the index of dispersion, we show that the accuracy of performance model predictions can be increased by up to 30% compared to standard queueing models parameterized only with mean service demands. Exploiting basic properties of bursty processes, we are also able to use for parameterization the 95th percentile of service times, which is widely used in computer performance engineering to quantify the peak-to-mean ratio of service demands. Furthermore, we include in the analysis also the estimated median of service times. In this way, our performance models are specified by only four parameters for each server: mean, index of dispersion, median, and 95th percentile of service demands. To the best of our knowledge, this paper makes a first case in the use of a new practical modeling paradigm for capacity planning that encompasses workload burstiness. Theoretical foundations for some of the models used in this paper can be found in [3].

After proposing a solution for service burstiness modeling, we generalize the methodology to modeling flash crowds, which are often observed in a system with burstiness in the request arrival process. The approach leverages on a recent generalization of the class of queueing models used in the present paper, called MAP queueing networks [3]. Using experimental data collected from a modified TPC-W benchmark proposed in [16], we show that our models are effective in capturing also burstiness in the request inter-arrival times.

The rest of the paper is organized as follows. In Section 2, we introduce service burstiness using illustrative examples and we present the methodology for the estimation of the index of dispersion. In Section 3, we discuss the multi-tier architecture and the TPC-W workloads used in experiments. We show in Section 4 that existing queueing models are unreliable if bottleneck switch exists in the system. The proposed modeling paradigm that integrates burstiness in performance models is then presented in Section 5. Section 6 shows the experimental results that validate the accuracy of the methodology. Section 7 overviews related work. Finally, Section 8 draws conclusions.

TABLE 1
$M/Trace/1$ queue with the service times traces in Figure 1.

	Resp. Time (util=50%)		*Resp. Time (util*=80%)		
	mean	95th prct	mean	95th prct	I
Fig. 1(a)	3.02	14.42	8.70	33.26	3.0
Fig. 1(b)	11.00	83.35	43.35	211.76	22.3
Fig. 1(c)	26.69	252.18	72.31	485.42	92.6
Fig. 1(d)	120.49	1132.40	150.32	1346.53	488.7

2 Burstiness in Performance Models: Do We Really Need It?

Let us consider the four workloads shown in Figure 1. Each plot represents a sample of $20,000$ consecutive service times X drawn from a set of four traces generated from the same hyper-exponential distribution with mean $E[X] = \mu^{-1} = 1$ and squared coefficient-of-variation $SCV = Var[X]/E[X]^2 = 3$. The only difference between the workloads is that we impose to each trace a unique burstiness profile. In Figure 1(b)-(d), the traces are ordered such that large service times progressively aggregate in bursts, while in Figure 1(a) they appear in random points of the trace.

What is the performance implication on systems of the different burstiness profiles in Figure 1(a)-(d)? Is it relevant for modeling purposes to consider burstiness in the description of a system? Assuming that the request inter-arrival times to the server follow an exponential distribution with mean $\lambda^{-1} = 2$ and 1.25, a simulation analysis of the $M/Trace/1$ queue at 50% and 80% utilization, respectively, provides the response times (service time + queueing time) shown in Table 1.

The results in Table 1 suggest the conclusion that burstiness should be accounted for in performance models. For instance, at 50% utilization the mean response

time for the trace in Figure 1(d) is approximately 40 times larger than with the service times in Figure 1(a) and the 95th percentile of the response times is nearly 80 times longer. In general, the performance degradation is monotonically increasing with the observed burstiness; therefore it is important to distinguish the behaviors in Figure 1(a)-(d) with a quantitative index. If burstiness is ignored, then predictions on the scalability of a system may be overly optimistic. The index of dispersion introduced next captures the difference in the burstiness profiles of different workloads.

2.1 Characterization of Burstiness: the Index of Dispersion

We use the *index of dispersion* I to characterize the burstiness of service times [5]. This is a standard burstiness index used in networking [5], which we here apply to the characterization of burstiness in multi-tier applications.

The index of dispersion of a service process is a measure defined on the squared-coefficient of variation SCV and on the lag-k autocorrelation coefficients ρ_k, $k \geq 1$, of the service times as follows:

$$I = SCV \left(1 + 2 \sum_{k=1}^{\infty} \rho_k \right). \quad (1)$$

The dependence of I on both SCV and autocorrelations allows one to discriminate traces like those in Figure 1(a)-(d): e.g., for the trace in Figure 1(a) the correlations are statically negligible, since the probability of a service time being small or large is unrelated to its position in the trace. However, for the trace in Figure 1(d) consecutive samples tend to assume similar values, either small or large, which results in positive autocorrelation that increase the sum in (1). The last column of Table 1 reports the values of I for the four example traces, which indicate that I is able to characterize the increasing burstiness levels in Figure 1(a)-(d).Note that for exponential service times it is $I = 1$, thus the index of dispersion may be interpreted qualitatively as the ratio of the observed service burstiness with respect to a Poisson process.

2.2 Measuring the Index of Dispersion

For estimation of I, we use an alternative definition of the index of dispersion of a service process considered in the literature [5]. Assume stationarity for the time series under study. Let N_t be the number of requests completed in a time window of t seconds, where the t seconds are counted *ignoring* the server idle time, that is, by conditioning on the period where the system is busy. If we regard N_t as a random variable, that is, if we perform several experiments by varying the time window placement in the trace and obtain different values of N_t, then the index of dispersion I is the limit:

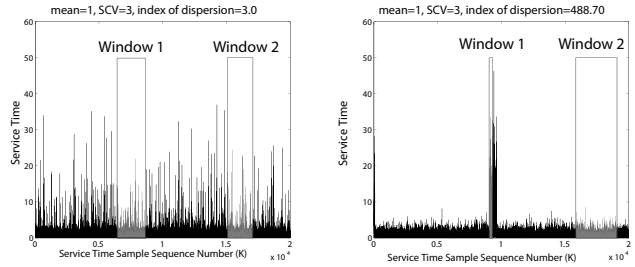
$$I = \lim_{t \to +\infty} \frac{Var(N_t)}{E[N_t]}, \quad (2)$$

where $Var(N_t)$ is the variance of the number of completed requests in a window of t time units and $E[N_t]$ is the mean number of service completions within the window. Thus, the value of I depends on the number of completed requests in an asymptotically large time scale. For example, suppose that the sampling resolution is $T = 60s$, and assume to approximate the asymptotically large time scale as $t \approx 120T$, then N_t is computed by summing the number of completed requests in 120 consecutive samples. By changing the initial position of the window used to compute N_t, we obtain a sequence of random variables N_t^1, N_t^2, N_t^3, ..., that are used to compute $Var(N_t)$ and $E[N_t]$ appearing in (2).

Figure 2 illustrates why the ratio $Var(N_t)/E[N_t]$ in (2) captures burstiness. Consider two windows of t time units, denoted by "Window 1" and "Window 2". Let us focus on the estimation of the index of dispersion I for the service time traces shown in Figure 1(a) (no burstiness) and Figure 1(d) (high burstiness). In the first case (without burstiness) shown in Figure 2(a), the service times of the requests are variable but still rather homogeneous throughout the trace, thus the placement of the two windows in different positions does not affect significantly the number of completed requests N_t within the t seconds. In our representation, the two shaded areas define the subset of requests completed in t seconds for "Window 1" and "Window 2", respectively. Clearly, the number N_t of completed requests in the two windows in Figure 2(a) is very similar because of the similar horizontal width. However, in presence of bursts such as in the trace in Figure 2(b), the placement of the two windows in different positions can strongly affect the number of completed requests within them: e.g., "Window 1" in Figure 2(b) shows a case where N_t is small because all service times falling in the window are large and hence only very few requests can be completed in t seconds, while "Window 2" in Figure 2(b) illustrates the opposite case where all service times are small and many requests can be served within the time window.

As a result of the above observations, Figure 2(a) is a case where $Var(N_t)$ is very small, while in Figure 2(b) is very large. This is immediately reflected by the value $Var(N_t)/E[N_t]$ of the index of dispersion. We describe below a simple algorithm to estimate I. We point to [5] for a review of estimation techniques for the index of dispersion which may be alternatively used.

In this work, we use the pseudo-code in Figure 3 to estimate I directly from (2). The pseudo-code is a straight-forward evaluation of $Var(N_t)/E[N_t]$ for different values of t which returns an estimate of I if the ratio converge or an estimate on I if the number of available samples is not large enough to obtain a reliable estimate of $Var(N_t)/E[N_t]$. Intuitively, the algorithm in Figure 3 calculates I of the service process by observing the completions of jobs in concatenated busy time samples. Because of this concatenation, queueing is masked out and the index of dispersion of job completions serves as a good approximation of the index of dispersion of the



(a) (b)

Fig. 2. Graphical interpretation of $Var(N_t)/E[N_t]$

Input
T, the sampling resolution (e.g., $60s$)
K, total number of samples, assume $K > 100$
U_k, utilization in the kth period, $1 \leq k \leq K$
n_k, completions in the kth period, $1 \leq k \leq K$
tol, convergence tolerance (e.g., 0.20)
Estimation of the Index of Dispersion I
1. busy time in the kth period $B_k := U_k \cdot T$, $1 \leq k \leq K$
2. initialize $t = T$ and $Y(0) = 1$
3. do
   a. for each set $(B_k, B_{k+1}, \ldots, B_{k+j})$
      such that $\sum_{i=0}^{j} B_{k+i} \approx t$,
      aa. compute $N_t^k = \sum_{i=0}^{j} n_{k+i}$.
   b. if the set of values N_t^k has less than 100 elements.
      the trace is too short:
      stop and collect new measures.
      Return $I = Y(t - T)$
   c. $Y(t) = Var(N_t^k)/E[N_t^k]$
   d. increase t by T
   until $|1 - (Y(t)/Y(t - T))| \leq tol$
4. return $I = Y(t)$

Fig. 3. Estimation of I from utilization samples.

service process.

3 Burstiness in Multi-Tier Applications: Symptoms and Causes

Today, a multi-tier architecture has become an industry standard for implementing scalable client-server enterprise applications. In our experiments, we use a testbed of a multi-tier e-commerce site that is built according to the TPC-W specifications [4]. This allows us to conduct experiments under different settings in a controlled environment, which then enables the evaluation of the proposed modeling approach.

3.1 Experimental Environment

TPC-W is a widely used e-commerce benchmark that simulates the operation of an online bookstore [4]. Typically, this multi-tier application uses a three-tier architecture paradigm, which consists of a web server, an application server, and a back-end database. A client communicates with this web service via a web interface, where the unit of activity at the client-side corresponds to a web page download. In general, a web page is composed by an HTML file and several embedded objects such as images. In a production environment, it is common that the web and the application servers reside on the same hardware, and shared resources are used by the application and web servers to generate main HTML files as well as to retrieve page embedded objects. We opt to put both the web server and the application server on the same machine called the front server. We use terms "front server" and "application server" interchangeably. A high-level overview of the experimental set-up and specifications of the software/hardware used can be found in [15].

Since the HTTP protocol does not provide any means to delimit the beginning or the end of a web page, it is very difficult to accurately measure the aggregate resources consumed due to web page processing at the server side. There is no practical way to effectively measure the service times for *all* page objects, although accurate CPU consumption estimates are required for building an effective application provisioning model. To address this problem, we define a *client transaction* as a combination of *all* the processing activities at the server side to deliver an entire web page requested by a client, i.e., generate the main HTML file as well as retrieve embedded objects and perform related database queries.

Typically, a continuous period of time during which a client accesses a Web service is referred to as a *User Session* which consists of a sequence of consecutive individual transaction requests. According to the TPC-W specification, the number of concurrent sessions (i.e., customers) or emulated browsers (EBs) is kept constant throughout the experiment. For each EB, the TPC-W benchmark defines the user session length, the user think time, and the queries that are generated by the session. In our experimental environment, two Pentium D machines are used to simulate the EBs. If there are m EBs in the system, then each machine emulates $m/2$ EBs. One Pentium D machine is used as the back-end database server, which is installed with MySQL 5.0 and with 10,000 items in inventory.

There are 14 different transactions defined by TPC-W. In general, these transactions can be roughly classified of "Browsing" or "Ordering" type. Furthermore, TPC-W defines three standard transaction mixes differing for the weight given to each type (i.e., browsing or ordering) in the particular transaction mix:

- *browsing mix*: 95% browsing and 5% ordering;
- *shopping mix*: 80% browsing and 20% ordering;
- *ordering mix*: 50% browsing and 50% ordering.

One way to capture the navigation pattern within a session is through the *Customer Behavior Model Graph (CMBG)* [14], which describes patterns of user behavior, i.e., how users can navigate through the site, and where arcs connecting states (transactions) reflect the probability of the next transaction type. TPC-W defines the set of probabilities that drive user behavior from one state to another at the user session level. During each session, each EB cycles through a process of sending a transaction request, receiving the response web page and selecting the next transaction request.

Typically, a user session starts with a Home transaction request. The TPC-W implementation is based on the J2EE standard – a Java platform which is used for web application development and designed to meet the computing needs of large enterprises. For transaction monitoring we use the HP (Mercury) Diagnostics [7] tool which offers a monitoring solution for J2EE applications. The Diagnostics tool collects performance and diagnostic data from applications without the need for application source code modification or recompilation. It uses bytecode instrumentation. This instrumentation enables a tool to record processed transactions and their database calls over time as well as to measure their execution time (both transactions and their database calls). We use the Diagnostics tool to measure the number of completed requests n_k in the kth period with a granularity of $T = 5$ seconds. Such value can be obtained with negligible overhead, even for a high load and small interval it is less than 5% utilization overhead. We use the sar command to obtain the utilizations of two servers across time with a 1 second granularity; this is later aggregated with the same 5 seconds granularity of n_k.

3.2 Bottleneck Switch in TPC-W

For each transaction mix, we run a set of experiments with different numbers of EBs ranging from 25 to 150. Each experiment runs for 3 hours, where the first 5 minutes and the last 5 minutes are considered as warm-up and cool-down periods and thus omitted in the analysis. User think times are exponentially distributed with mean $E[Z] = 0.5s$. Figure 4 presents the overall system throughput, the mean system utilization at the front server and the mean system utilization at the database server as a function of EBs. Figure 4(a) shows that the system becomes overloaded when the number of EBs reaches 75, 100, and 150 under the browsing mix, the shopping mix, and the ordering mix, respectively. The system throughput then remains asymptotically flat with higher EBs. One reason for this is the "closed loop" aspect of the system, i.e., the fixed number of EBs, that is effectively an upper bound on the number of jobs that circulate in the system at all times. Another reason is that the bottleneck switch phenomenon, described in the rest of the section, can make transactions suffer joint congestion at multiple resources.

The results from Figures 4(b) and 4(c) show that under the shopping and the ordering mixes, the front server is a bottleneck, where the CPU utilizations are almost 100% at the front tier, but only 20-40% at the database tier. For the browsing mix, we see that the CPU utilization of the front server increases very slowly as the number of EBs increases beyond 75, which is consistent with the very slow growth of throughput. For example, when the front server is already 100% utilized under the shopping and the ordering mixes, the utilization of the front server for the browsing mix is around 80%. Meanwhile, for the browsing mix, the CPU utilization of the database server increases quickly as the number of EBs increases. When the number of EBs is beyond 100, it becomes not obvious which server is responsible for the bottleneck: the average CPU utilizations of two servers are about the same, differing by a statistically insignificant margin. In presence of burstiness in the service times, this may suggest that a phenomenon of *bottleneck switch* occurs between the front and the database servers *across time*. This phenomenon is not specific to the testbed described in the current work. In an earlier paper [25], a similar situation was observed for a different TPC-W testbed. That is, a server may become the bottleneck while processing consecutively large requests, but be lightly loaded during other periods. In general, additional investigation to determine the existence of bottleneck switch is required when the average utilizations are relatively close or when the workloads are known to be highly-variable.

To confirm our conjecture about the existence of bottleneck switch in the browsing mix experiment, we present CPU utilizations of the front and the database servers across time for the browsing mix, as well as the shopping and the ordering mixes with $N = 100$ EBs, see Figure 5. A bottleneck switch occurs when the database server utilization becomes significantly higher than the front server utilization, as clearly visible in Figure 5(a) under the browsing mix workload. As shown in Figures 5(b) and 5(c), there is no bottleneck switch for the shopping and the ordering mixes, although these two workloads are also highly variable.

The bottleneck switching is a characteristic effect of burstiness in the service times, but is hard to model. Later, in Section 6.1, we will show that the browsing mix exhibits a significantly higher index of dispersion for both the front and database server compared to the shopping and ordering mixes.

3.3 The Analysis of Bottleneck Switch

Now, we focus on the burstiness in a multi-tier application to further analyze the symptoms and possible causes of bottleneck switching. Indeed, for a typical request-reply transaction, the application server may issue multiple database calls while preparing the reply of the web page. This cascading effect of various tasks breaks down the overall transaction service time into several parts, including the transaction processing time at the application server as well as all related query processing time at the database server. Therefore, the application characteristics and the high variability in database server may lead to the burstiness in the overall transaction service times.

To verify this, we record the queue length at the database server at each instance that the database request is issued by the application server and the prepared reply is returned back to the application server. Figure 6 presents the queue length at the database server (see solid lines in the figure) as well as the CPU utilizations of the database server (see dashed lines in the figure) across time for all three transaction mixes.

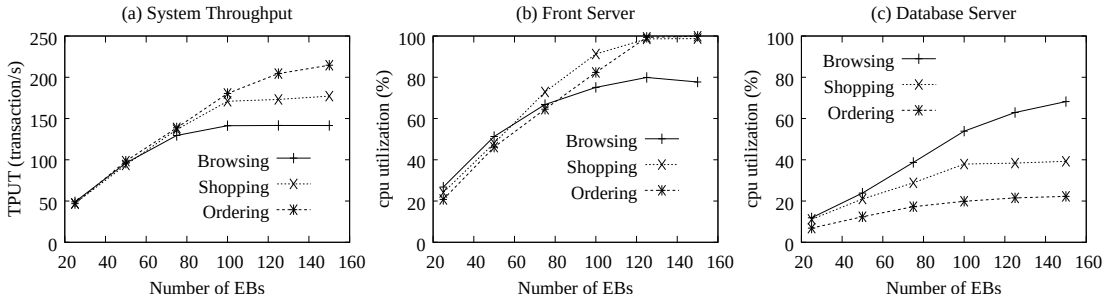


Fig. 4. Illustrating (a) system throughput, (b) average CPU utilization of the front server, and (c) average CPU utilization of the database server for three TPC-W transaction mixes. The mean think time $E[Z]$ is set to 0.5 seconds.

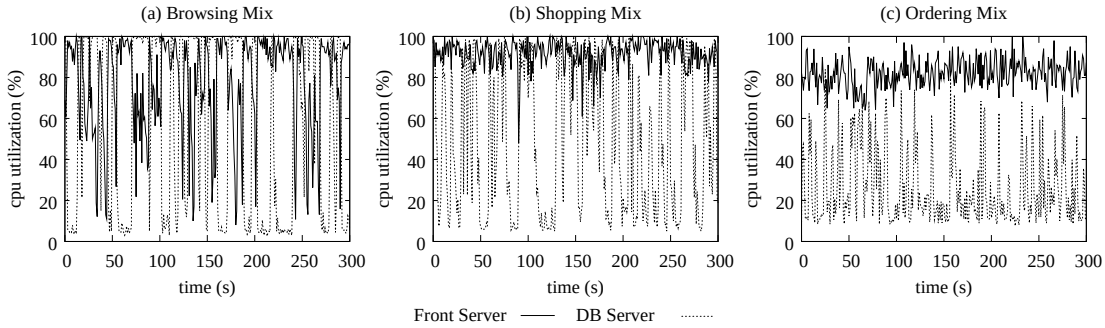


Fig. 5. The CPU utilization of the front server and the database server across time with 1 sec granularity for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix with $N = 100$ EBs. The monitoring window is 300 seconds.

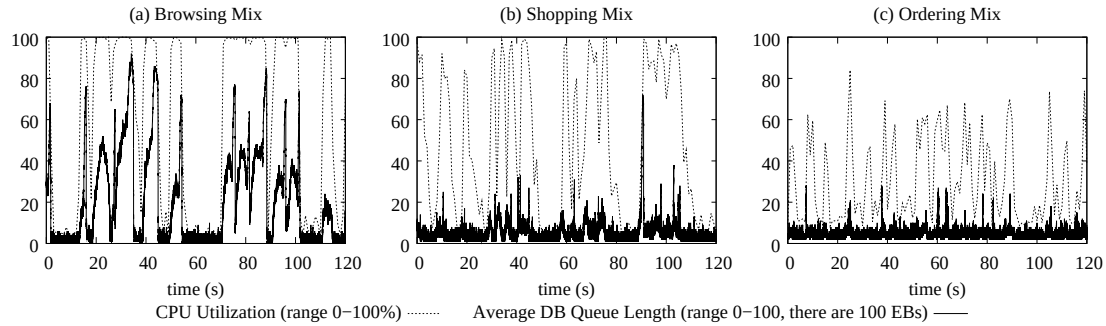


Fig. 6. The CPU utilization of the database server (dashed lines) and the average queue length at the database server (solid lines) across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix. In this figure, the y-axis range of both performance metrics is the same because there are $N = 100$ EBs (clients in the system). The monitoring window is 120 seconds.

Here, in order to make the figure easy to read, we show the case with $N = 100$ EBs such that the y-axis range for both performance metrics (i.e., queue length and utilization) is the same. First of all, the results for the browsing mix in Figure 6(a) verify that burstiness does exist in the queue length at the database server, where the queue holds less than 10 jobs for some periods, while sharply increases to as high as 90 jobs during other periods. More importantly, the burstiness in the database queue length exactly matches the burstiness in the CPU utilizations of the database server. Thus, in some periods almost all the transaction processing happens either at the application server (with the application server being a bottleneck) or at the database server (with the database server being a respective bottleneck). This leads to the alternating bottleneck effect between the application and the database servers.

In contrast, no burstiness can be observed in the queue length for the shopping and the ordering mixes, although these two workloads have also high variability in their utilizations, see Figures 6(b) and 6(c). These results are consistent with those shown in Figures 5(b) and 5(c), where the application server is the main system bottleneck.

According to the TPC-W specification, different transaction types may have different number of outbound database queries. For example, the *Home* transaction has one or two database queries for each transaction

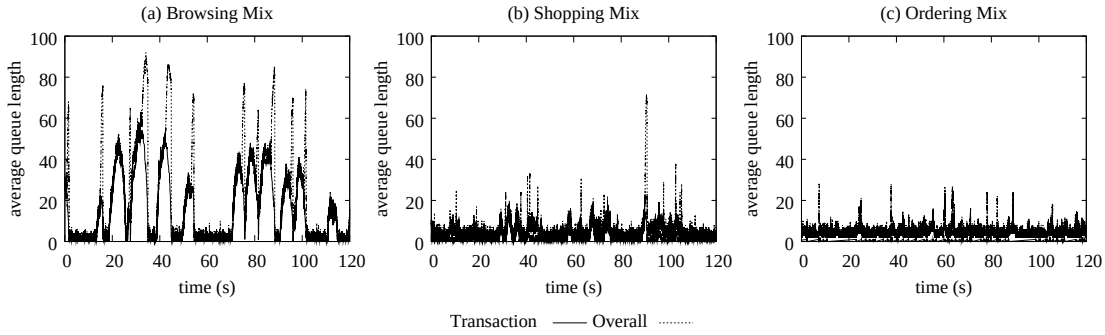


Fig. 7. The overall queue length at the database server (dashed lines) and the number of current requests in system for the *Best Seller* transaction (solid lines) across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix, when there are $N = 100$ EBs and the mean think time is $0.5s$. The monitoring window is 120 seconds.

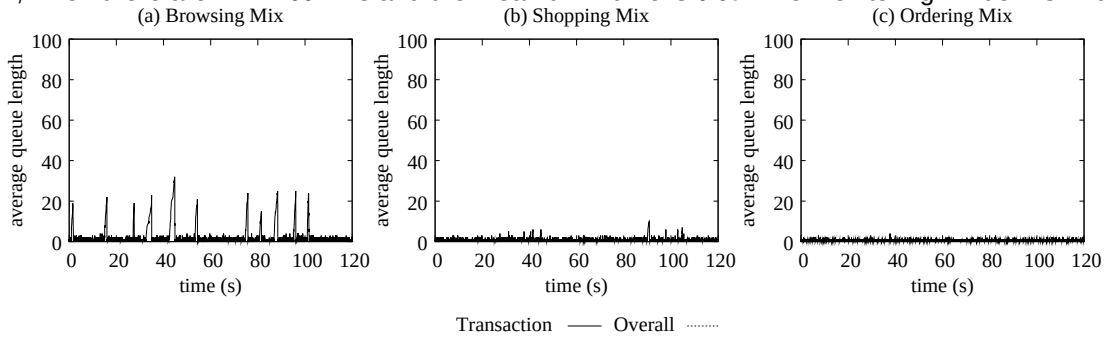


Fig. 8. The number of current requests in system for the *Home* transaction across time for (a) the browsing mix, (b) the shopping mix, and (c) the ordering mix, when there are $N = 100$ EBs and the mean think time is $0.5s$. The monitoring window is 120 seconds.

request while the *Best Seller* transaction always has two outbound database queries per transaction request. To analyze whether the burstiness in database queue length comes from some particular transaction types, we have measured the number of current requests for each transaction type over time. After revisiting all 14 transaction types, we find that the source of this burstiness is due to specific transaction types. Figures 7 and 8 show the results for two representative transaction types, the *Best Seller* transaction and the *Home* transaction, under three transaction mixes. The overall database queue length across time is also plotted as a baseline.

As shown in Figure 7(a), although in the browsing mix only 11% of requests belongs to the *Best Seller* transaction type, the number of these requests dominates the overall database queue length: the spikes in the overall queue length in the database clearly comes from this particular transaction type. Furthermore, there exists burstiness in the number of requests for this transaction type and this burstiness matches that in the overall queue length in database server very well. In addition, for some extremely high spikes, e.g., at the 40 timestamp in Figure 7(a), the requests of another popular transaction type, the *Home* transaction, also contribute to burstiness, see Figure 8(a). These figures indicate that *Best Seller* and *Home* transactions share some resources required for their processing at the database server, and it leads to extreme burstiness during such time periods.

For the shopping and the ordering mixes, there is no visible burstiness in either the queue length at the database server or in the number of current requests for each transaction type, as shown in Figure 7(b)-(c) and Figure 8(b)-(c), respectively.

Summarizing, we showed that

- burstiness in the service times can be a result of a specific workload combination (mix) in the multi-tier applications (e.g., burstiness in the service times may exist under the browsing mix in TPC-W);
- burstiness in the service times can be caused by a bottleneck switch between the tiers, and can be a result of "hidden" resource contention between the transactions of different types and across tiers.

Such instability in systems with burstiness is hard to characterize and model. The super-position of several events, such as database locking conditions, variability in service time of software operations, memory contention, and/or characteristics of the scheduling algorithms, may interact in a complex way. The question is whether instead of identifying the low-level *exact* causes of burstiness as traditional models would require, one can provide an effective way to infer this information using live system measurements in order to capture burstiness into new capacity planning models. We investigate this in the next sections.

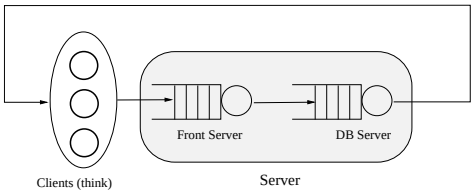


Fig. 9. Closed queueing network model of a multi-tier application composed by a front server and a DB server.

4 Inapplicability of MVA Predictions

In this section, we use standard MVA-based performance evaluation methodologies to define an analytical model of the multi-tier architecture presented in Section 3.1. Our goal is to show that existing queueing models can be largely inaccurate in performance prediction if the real system is subject to bottleneck switch. We show in Section 5 how performance models can be generalized to correctly account for burstiness and bottleneck switches.

4.1 Mean Value Analysis

We first model the multi-tier architecture considered in the experimental analysis by the closed queueing network composed by two queues and a delay center shown in Figure 9. The two queues are first-come first-serve and model the front server and the database server, respectively. The delay station models the average user think time $E[Z]$ between receiving a Web page and submitting a new page download request. The tandem topology is explained as follows. In the real application, the servlet code that implements a transaction type is a mix of instructions at the front server and the database server. An expensive (or infeasible) analysis of the source code is required to characterize the switch of the execution from the front server to the database server and back. Thus, we make a simplification by assuming that requests first execute at the front server without any interruption and then they are processed at the database server. This approach is classic in performance modeling, where it is common to re-scale the service times received at a queue by the average number of visits at the station in order to use a simpler cyclic topology [10]. Yet, general topologies can be defined in the MAP queueing network models used in this paper [3]. Furthermore, this approximation implicitly assumes that the burstiness at small time scales has a negligible impact on mean performance, which is dominated by fluctuations that are visible at large time scales. This is consistent with [11] that shows that queue behavior can be quite insensitive to high-frequency spectral components, such as fluctuations at small timescales. This is consistent with the definition of index of dispersion, which only characterizes burstiness at large time scales.

We here consider the two cases where the workload is assumed to be composed of requests of a single type (MVA) or of multiple types (multiclass MVA). In multiclass models, the analysis distinguishes the workload into transaction types (classes) and it is possible to specify different mean service times and user think times for each class. In the single class cases one chooses to ignore the partitioning across classes to simplify the model parameterization and evaluation. The MVA model is parameterized by the number of clients N and by the following values:

- the mean service time $S_{FS,r}$ of class r transactions at the front server;
- the mean service time $S_{DB,r}$ of class r transactions at the database server;
- the mean think time $E[Z]_r$ between submission of two successive class r transactions by a same EB;
- the probability p_r that an EB sends a class r transactions to the system.

for each class $r = 1, \ldots, R$. From the above parameters, the mean number of class r transactions circulating in the system is immediately given by the product $N_r = Np_r$. In TPC-W, clients are called emulated browsers (EBs) and, for multiclass models, there are $R = 14$ transaction classes. We parameterize the service demands by multivariate linear regression of the utilization samples against the number of completed transactions for each class in each sample period [24]. Think times are set to $E[Z_r] = 0.5s$ for all transaction classes. A similar parameterization is obtained for the single class model using univariate linear regression.

Figure 10 shows the results of the single class and multiclass MVA model predictions versus the actual measured throughput (TPUT) of the system as a function of the number of EBs. The three plots in Figure 10 illustrate the different model accuracy under the browsing, shopping, and ordering mixes. The results show that the single and multiclass MVA model prediction are quite similar and accurate for the shopping and the ordering mixes, while there exists a large deviation between the predicted and the measured throughputs for the browsing mix. In particular, the maximum error in Figure 10(a) is as large as 36%. This suggests that MVA models can deal very well with models *without* burstiness (e.g., the ordering mix in Figure 10(c)) and with models where burstiness does *not* result in a bottleneck switch (e.g., the shopping mix in Figure 10(b)). However, the fundamental and most challenging case of burstiness resulting in bottleneck switches shown in the browsing mix in Figure 10(a) reveals the limitation of the MVA modeling assumptions, which do not contemplate burstiness.

5 Integrating Burstiness in Performance Models

In this section, we use the measurement of burstiness for the parameterization of the multi-tier system model presented in Figure 9. We first present the methodology for integrating the burstiness in queueing models in Section 5.1 and then discuss the modeling approach in

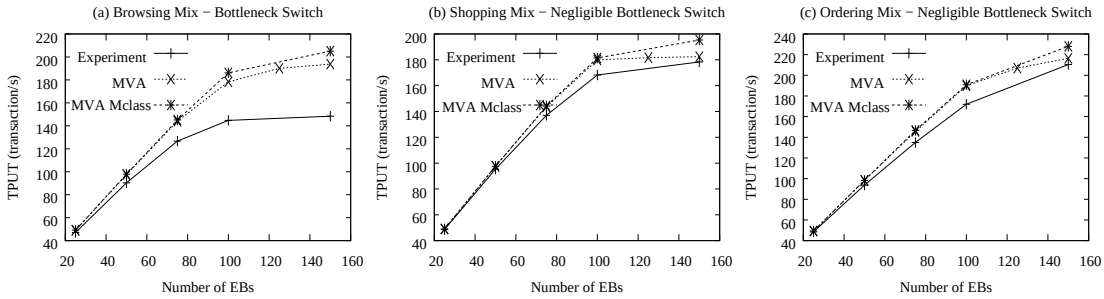


Fig. 10. Single-class and multi-class MVA model predictions versus measured throughput.

Section 5.2. Finally, we discuss the generalization to flash crowds analysis in Section 5.3.

5.1 Modeling Bursty Service Times

In order to integrate the index of dispersion in queueing networks, we model service times as a Markovian Arrival Process (MAP) [6], [9] with 2 states (MAP(2)). A MAP(2) may be seen as a continuous-time Markov chain that jumps between two states and the active state determines the current rate of service. Thus it is a mathematical model used to describe the variability over time of the service rate for a system. If properly fitted, the MAP(2) can be integrated into a queueing network model to describe service times characterized by burstiness. This is discussed in Section 5.2, while here we focus on the fitting of a MAP(2) from coarse-grained utilization and throughput measurements.

The state jump behavior of a MAP(2) is defined by a pair of matrices $(\mathbf{D}_0, \mathbf{D}_1)$ where

$$\mathbf{D}_0 = \begin{bmatrix} -\lambda_{1,1} & \lambda_{1,2} \\ \lambda_{2,1} & -\lambda_{2,2} \end{bmatrix}, \quad \mathbf{D}_1 = \begin{bmatrix} \lambda^*_{1,1} & \lambda^*_{1,2} \\ \lambda^*_{2,1} & \lambda^*_{2,2} \end{bmatrix}, \tag{3}$$

in which $\lambda_{i,i} > 0$, $\lambda_{i,j} \geq 0$, $i \neq j$, $\lambda^*_{i,j} \geq 0$, and the sum of terms of $\mathbf{D}_0$ and $\mathbf{D}_1$ on the same row is zero. This notation states that, after initialization, the MAP spends in state 1 an exponentially distributed sojourn time $T \sim \lambda_{1,1} e^{-\lambda_{1,1} t}$, after which with probability $p_{1,2} = \lambda_{1,2}/\lambda_{1,1}$ jumps to state 2, and with probability $1 - p_{1,2}$ it completes service for a job. In particular, $1 - p_{1,2} = q_{1,1} + q_{1,2}$ where $q_{1,2} = \lambda^*_{1,2}/\lambda_{1,1}$ is the probability of completing a job and simultaneously jumping to state 2, while with probability $q_{1,1} = \lambda^*_{1,1}/\lambda_{1,1}$ the MAP(2) remains in state 1 after the job completion. Similar interpretations hold for the rates of state 2. Additional details on the MAP(2) model can be found in [16].

For our purposes, we need to assign the rates $\lambda_{i,j} \geq 0$ and $\lambda^*_{i,j}$ such that the time between completing jobs in the MAP(2) has the same distribution and burstiness of the service times measured on the multi-tier application. Denote by X the service time of a request modeled by the MAP(2). Also, let $\boldsymbol{\pi}$ be the eigenvector of matrix $(-\mathbf{D}_0)^{-1}\mathbf{D}_1$ for the unit eigenvalue, let $\boldsymbol{\alpha}$ be the eigenvector of matrix $\mathbf{Q} = \mathbf{D}_0 + \mathbf{D}_1$ for the zero eigenvalue, and let $\mathbf{1}$ be a column vector of ones of proper size. Then it can be shown by standard arguments for MAPs [9] that these expressions hold for the moments of X:

$$E[X^k] = k!\boldsymbol{\pi}(-\mathbf{D}_0)^{-k}\mathbf{1} \tag{4}$$

The joint moments are instead given by

$$E[X^k X^h_{+j}] = k!h!\boldsymbol{\pi}(-\mathbf{D}_0)^{-k}\left((-\mathbf{D}_0)^{-1}\mathbf{D}_1\right)^j(-\mathbf{D}_0)^{-h}\mathbf{1} \tag{5}$$

where X^h_{+j} denotes the hth power of the jth sample drawn at equilibrium after sample X. In addition, the kth percentile is computed as

$$p_k = \text{argmin}_x|(1 - \boldsymbol{\pi}e^{\mathbf{D}_0 x}\mathbf{1}) - k/100| \tag{6}$$

and the index of dispersion as

$$I = 1 + 2\left(\boldsymbol{\alpha}\mathbf{D}_1\mathbf{1} - \boldsymbol{\pi}(\mathbf{1}\boldsymbol{\alpha} + \mathbf{Q})^{-1}\mathbf{D}_1\mathbf{1}\right) \tag{7}$$

The challenge is to fit, based on the above formulas, a MAP(2) that represents the service times based only on measurements of busy times and throughputs. We assume to know for each servers the busy times B_k and the number of completed request n_k in B_k, for all sample periods $k = 1, \ldots, K$. From basic operational analysis [10, Chap 3] it is

$$E[X] = \frac{\sum_{k=1}^K B_k}{TPUT}, \quad TPUT = \frac{\sum_{k=1}^K n_k}{T}$$

where $TPUT$ is the average throughput of the multi-tier application and T is the time interval between collection of two samples. The index of dispersion I for the service times is instead estimated using the pseudo-code in Figure 3. The 2 remaining degrees of freedom for the MAP(2) are spent to fit in a best-effort way the median p_{50} and the $95th$ percentile p_{95} of the service times. We have chosen to focus on these two descriptors because they are well-understood both in academia and in industry and are often fundamental quantities used in the capacity planning process.

We estimate the pth percentile of the service times as the pth percentile of the measured busy times B_k scaled by the median number of requests processed in the sample periods. If the trace has high dispersion, say $I >> 100$, bursts tend to be longer than the sample period T and the n_k jobs that are served within B_k may be assumed to have similar statistical properties. The busy time is therefore $B_k \approx n_k E[X]$, being $E[X]$

the average service time. Our approximation consists in assuming that n_k is independent of k and equal to its median value $med(n_k)$. Under this assumption, the pth percentile of B_k is simply $med(n_k)$ times the pth percentile of X. Conversely, if the trace has low dispersion, then our approach works as an approximation.

Let the notation $\mathbf{A}[i,j]$ indicate the element in row i and column j of matrix $\mathbf{A}$. The fitting of mean, median, 95th percentile, and index of dispersion is obtained by the following nonlinear optimization program

$$\min ||(1-\boldsymbol{\pi} e^{\mathbf{D}_0 p_{50}}\mathbf{1}) - 0.50||_2 + ||(1-\boldsymbol{\pi} e^{\mathbf{D}_0 p_{95}}\mathbf{1}) - 0.95||_2$$

subject to

$$\begin{aligned}
E[X] &= \boldsymbol{\pi}(-\mathbf{D}_0)^{-1}\mathbf{1} \\
I &= 1 + 2\left(\boldsymbol{\alpha}\mathbf{D}_1\mathbf{1} - \boldsymbol{\pi}(\mathbf{1}\boldsymbol{\alpha}+\mathbf{Q})^{-1}\mathbf{D}_1\mathbf{1}\right) \\
\mathbf{D}_0[i,j] &\geq 0 && \forall i,j : i \neq j \\
\mathbf{D}_0[i,i] &= -\textstyle\sum_{j:j\neq i}\mathbf{D}_0[i,j] - \sum_k \mathbf{D}_1[i,k] && \forall i \\
\mathbf{D}_1[i,j] &\geq 0 && \forall i,j
\end{aligned}$$

where in the objective function $||\cdot||_2$ is the 2-norm and the last three constraints ensure that $(\mathbf{D}_0, \mathbf{D}_1)$ is a valid MAP. Note that the above optimization program returns in general a best-effort solution to the estimation problem, which yet ensures that I and $E[X]$ are always matched exactly.

5.2 Queueing Network Model

In order to integrate burstiness in our analysis, we parameterize the model in Figure 9 as a single class MAP queueing network [3] with the service processes that are fitted by MAP(2)s and with exponential think times. A MAP queueing network is a very general class of queueing models which allows us to describe as a MAP both the service processes at queueing stations and the client think times at delay stations. Here, we discuss the efficient solution of such models for all values of the number of clients N.

5.2.1 MAP Flow Equivalent Server

Closed MAP queueing networks for large number of users have been mainly studied by bounds on performance metrics based on linear programming [3]. We here introduce a new approximate technique with computational requirements that are much cheaper than a direct numerical evaluation of the underlying Markov chain. For instance, with the proposed approximation we are able to solve using MATLAB on a Intel Xeon 2x2.53 GHz machine, models with $N = 500$ clients in 10.35 sec and $N = 1,000$ clients in 24.17 sec which grows almost linearly with the total population. Using a direct numerical method for Markov chains we were instead unable to solve these models. The case $N = 500$ was still executing after more than 6 hours. This is because a significant fraction of the time requirements is due to the cost of the state space generation which grows exponentially with the number of queues. Conversely, the computational costs of the approximate method proposed in this section are independent of the number of queues in the model.

The approximate technique we propose is a generalization of the classic Norton's theorem for hierarchical modeling of queueing networks [10]. The main idea is to recursively replace pairs of resources, either queues or delays, by a special station with load-dependent behavior, called a *flow equivalent server*. The approach is first illustrated using the case of Figure 9, and then generalized to a model with M stations and arbitrary topology. We begin with considering the subnetwork composed only by the front server and by the database server station. The subnetwork is populated by n client EBs and considered in isolation from the rest of the model; the routing topology is cyclic. The aim is to characterize the distribution and burstiness of the throughput $TPUT(n)$ of this subnetwork as a function of all possible numbers of EBs $n = 1,\ldots,N$ served simultaneously by the subnetwork at a given time. Such a throughput is measured at the output link of the subnetwork that feeds the rest of the model, in the example at the output of the database server. Based on $TPUT(n)$, $n = 1,\ldots,N$, we then replace the subnetwork with a *MAP flow equivalent server*, which is a queue that dynamically changes its MAP service process depending on the number of jobs n being served. That is, the flow equivalent server is a load-dependent queue which uses a MAP $(\mathbf{F}_0^n, \mathbf{F}_1^n)$, $n = 1,\ldots,N$, to model the service rate when there is a population of n jobs being served in the subnetwork. After replacing the front server and database server by a single queue, one is left with a model having only the delay and the MAP flow equivalent server. Models with two queues are easy to handle having a state space that grows only as $O(N)$ with the total number of client EBs. Thus, they can be solved very efficiently either by direct numerical methods or by specialized techniques for finite quasi-birth death processes [9].

Note that for models with more than $M = 3$ stations the technique applies recursively. One first computes the flow equivalent server for the subnetwork corresponding to stations labelled 1 and 2. Then a sequence of subnetworks composed by station $i = 3,\ldots,M$ and the MAP flow equivalent server generated at the previous step is evaluated. At each step the pair composed by station i and the flow equivalent server is replaced by a new flow equivalent server. Finally, the model is reduced again to a network with station M and the flow equivalent server obtained from the previous iteration, and the model is then solved numerically. Note that for models with arbitrary topology the above steps apply in a similar way, but superposition and splitting of job flows due to the routing should be taken into account [8]. From such solution, one can obtain the mean throughput $TPUT$ which provides also the mean end-to-end response time $RT = N/TPUT - E[Z]$, being $E[Z]$ the mean think time.

5.2.2 Parameterization of MAP Flow Equivalent Server

The MAP flow equivalent server approach is applicable only if we can compute the moments and burstiness of the job inter-departure times from the subnetwork into the rest of the model. These moments are the input parameters needed to fit $(\mathbf{F}_0^n, \mathbf{F}_1^n)$ for all possible populations n in the subnetwork. In this section, we assume that the number of states of $(\mathbf{F}_0^n, \mathbf{F}_1^n)$ is 2, thus the first three moments $E[X], E[X^2], E[X^3]$ and the index of dispersion I are sufficient to fit the MAP(2) [6]. Note that in this case the autocorrelation structure admits a simple form [6] and thus the index of dispersion simplifies to $I = SCV + (SCV - 1)\gamma(1-\gamma)^{-1}$, where

$$SCV = \frac{E[X^2]}{E[X]^2} - 1, \quad \gamma = 2\frac{E[XX_{+1}] - E[X]^2}{E[X^2] - 2E[X]^2}$$

Hence if we know how to compute the moments $E[X], E[X^2], E[X^3]$ and the joint moment $E[XX_{+1}]$ the MAP(2) can be readily fitted.

We propose to compute the moments and joint moments of the job inter-departure times as follows. Consider the subnetwork composed by a queue with MAP service $(\mathbf{D}_0, \mathbf{D}_1)$ and a MAP flow equivalent server $(\mathbf{F}_0^n, \mathbf{F}_1^n)$. Then, inter-departure times are known to admit a MAP $(\mathbf{T}_0, \mathbf{T}_1)$ with

$$\mathbf{T}_0 = \begin{bmatrix} \mathbf{D}_0 \otimes \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} \otimes \mathbf{F}_1^1 & \mathbf{D}_0 \oplus \mathbf{F}_0^1 & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{I} \otimes \mathbf{F}_1^{n-1} & \mathbf{D}_0 \oplus \mathbf{F}_0^{n-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \otimes \mathbf{F}_1^n & \mathbf{I} \otimes \mathbf{F}_0^n \end{bmatrix},$$

$$\mathbf{T}_1 = \begin{bmatrix} \mathbf{0} & \mathbf{D}_1 \otimes \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_1 \otimes \mathbf{I} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{D}_1 \otimes \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

and $\oplus$ and $\otimes$ are the Kronecker sum and Kronecker product operators, respectively. The above formula restricts to the finite state-space case MAP descriptions of the inter-arrival time processes studied in works such as [8], [20]. Thus, we have reduced the analysis of the subnetwork to the problem of computing moments and joint moments of the MAP $(\mathbf{T}_0, \mathbf{T}_1)$.

A direct evaluation of (4)-(5) can be very expensive computationally, since the order of the above matrices is $4(N+1)$ which can be of the order of thousands in practice. In particular, the matrix inversion for the terms $(-\mathbf{T}_0^{-1})$ does not preserve the sparsity of the $\mathbf{T}_0$ matrix, hence the cost of storing the resulting matrix in memory may be prohibitive. Thus, we need a specialized technique that enables the efficient computation of moments and joint moments taking into account sparsity. This technique is then applied to all populations $n = 1, \ldots, N$ to compute the MAPs $(\mathbf{F}_0^n, \mathbf{F}_1^n)$ by fitting of the moments and index of dispersion of $(\mathbf{T}_0, \mathbf{T}_1)$. Note that in practice it is often simpler to consider a subset of values in the range $[1, N]$ since MAPs for similar values of n tend to be similar. In all experiments in this paper we considered the first 10 population values and a set of 10 equi-spaced points in $[11, N]$.

We have found that the following integral-based approach allows us to solve the issue. Consider the following exact expression

$$\boldsymbol{\pi}(-\mathbf{T}_0^{-1}) = \boldsymbol{\pi} \int_0^\infty \mathbf{e}^{\mathbf{T}_0 t}\, dt. \tag{8}$$

such that $E[X] = \boldsymbol{\pi}\mathbf{T}_0^{-1}\mathbf{1}$. We can integrate the above expression and compute the moment $E[X]$ using the trapezoid rule and Euler's approximation: $e^{\mathbf{D}_0 t} \approx I + \mathbf{T}_0 t + O(t^2)$. A good integration step is often $\Delta < |d_{max}^{-1}|$, being d_{max} the diagonal element of $\mathbf{T}_0$ with the largest absolute value, see the uniformization method for a probabilistic interpretation of d_{max} [2]. Note that the integration is efficient thanks to Euler's approximation, as it involves products of a vector of order $4(N+1)$ with the sparse matrix $\mathbf{T}_0$.

Using the same approach we can easily compute the other moments. For instance, the moment $E[X^2]$ (resp. $E[X^3]$) is computed by replacing in (8) $\boldsymbol{\pi}$ with $\boldsymbol{\pi}' = \boldsymbol{\pi}(-\mathbf{T}_0^{-1})$ (resp. $\boldsymbol{\pi}'' = \boldsymbol{\pi}'(-\mathbf{T}_0)^{-1}$) and then computing $E[X^2] = 2\boldsymbol{\pi}'(-\mathbf{T}_0)^{-1}\mathbf{1}$ (resp. $E[X^3] = 6\boldsymbol{\pi}''(-\mathbf{T}_0)^{-1}\mathbf{1}$). Similarly, the joint moment $E[XX_{+1}]$ is obtained by first computing the vector $\boldsymbol{\theta} = (\boldsymbol{\pi}\mathbf{T}_0^{-1})\mathbf{T}_1$ and then applying (8) with $\boldsymbol{\theta}$ that replaces $\boldsymbol{\pi}$. Once that $E[X]$, $E[X^2]$, $E[X^3]$, and $E[XX_{+1}]$ are computed, the MAP(2) is immediately fitted using the formulas reported earlier. For some combinations of these four parameters, the MAP(2) may be infeasible, e.g., some rates in the defining matrices might be negative. We solved this problem using an exponential distribution when $SCV \leq 1$ or $I < SCV$, since in both cases the burstiness is negligible, or, when the value of $E[X^3]$ is responsible for the infeasibility, we set $E[X^3] = (3/2)E[X^2]^2/E[X] + \epsilon$, for small $\epsilon > 0$, which assumes the largest possible heavy-tail decay of the distribution. We found this approximation to have little impact on the overall precision of the models.

5.3 Analysis of Flash Crowds

As mentioned in the introduction, burstiness in the inter-arrival times of requests is another common source of performance degradation. We describe two alternative approaches that may be used to model flash crowds.

5.3.1 Bounded Flash Crowds

The first method applies to systems where the number of users generating the flash crowd is limited by a known upper bound. We now assume that flash crowds can be generated by a pool of N users, where N may be a very large number (e.g., hundreds or thousands). This case is useful to model the performance of enterprise systems where the number of remote clients that can request an operation is limited by design (e.g., financial applications

queried by servers at bank branches) or constrained (e.g., admission control).

In [16] it is shown that such workloads can be effectively represented as closed systems where there is burstiness in the think times in between submission of consecutive requests, see that paper for fitting formulas that allow to model arrival burstiness using a MAP(2). Such a flash crowd model can be analyzed in our MAP queueing network methodology. It is sufficient to replace the exponential distribution used for the think times with the MAP $(\mathbf{Z}_0, \mathbf{Z}_1)$. The new station is treated as a load-dependent server, where the service rates when the current population at the delay is n are specified by a MAP with rates $(n\mathbf{Z}_0, n\mathbf{Z}_1)$. Indeed, since the rates are scaled by n, the service rate grows proportionally to the number of queued customers, thus the apparent response time at the resource becomes equivalent to a station without queueing. An illustrating example that applies this methodology in shown in Section 6.3.

5.3.2 Unbounded Flash Crowds

In some systems there might be uncertainty regarding the maximum number of users that populate a flash crowd, thus it can be easier to specify burstiness in terms of a variation of the inter-arrival times of requests at the multi-tier application. Let $(\mathbf{A}_0, \mathbf{A}_1)$ denote a MAP(2) fitted to describe distribution and burstiness in the inter-arrival times of requests. A number of methods have been proposed in the literature to solve the related open queueing network of MAP stations and may be applied directly, e.g., [8]. These methods apply effectively to open MAP queueing networks, but they have not been generalized yet to include load-dependent stations (e.g., the delay servers studied in this paper). Instead, our approach applies also in the load-dependent case.

Let λ be the average arrival rate of requests to the system and denote by $E[X_i]$ the average service time of requests at queue i, already scaled by the average number of visits at the resource [10]. The utilization at queue i is $U_i = \lambda E[X_i]$, thus $\lambda < 1/\max_i E[X_i]$ to guarantee stability of the system. Suppose now that the multi-tier system is described by a queueing network with M queues, then if we add a MAP queue with average service rate λ this will become the long-term bottleneck of the system for large populations, since the stability bound implies that this station must be the one with the slowest rate. It is therefore simple to conclude that, for large enough population N, any closed MAP queueing network will converge to a system where the slowest queue behaves as a bottleneck. When the utilization of such queue is sufficiently close to $U_{max} = 1$, its service process will be continuously operating and thus, by all means, become equivalent to an open job source that spawns requests according to the MAP(2) process $(\mathbf{A}_0, \mathbf{A}_1)$ specified for that bottleneck resource.

Summarizing, closed models can approximate for large enough population N the behavior of any open model. The precision of such approximation depends only on the utilization of the bottleneck station being large enough. Recall that for general queues the utilization is lower bounded by

$$U_i \geq \frac{NE[X_i]}{E[Z] + N\sum_j E[X_j]}$$

which is known as the lower ABA bound [10]. This expression can be used to determine a population value N that ensures the bottleneck station to be above a desired utilization threshold $1 - \epsilon$, for small $\epsilon > 0$.

6 Validation of Prediction Accuracy

6.1 Closed Transaction Mixes

We have solved the MAP(2) fitting program using the interior point algorithm in MATLAB's fmincon function with tolerance $\epsilon_{tol} = 10^{-8}$ and a maximum of 200 iterations. In all examples considered in this paper the execution time of fmincon never exceeded 30 seconds and always returned a solution prior to reaching the maximum number of iterations.

Figure 11 compares the analytical results for the resulting MAP queueing network model with the experimental measurements of the real system for the three transaction mixes. The values of the index of dispersion for the front and the database service processes are also shown in the figure. Throughout all experiments, the mean user think time is set to $E[Z] = 0.5s$. The MAP(2)s are estimated from experimental data collected with $E[Z] = 14s$, whereas the original experiments in [15] collected data for MAP estimation with $E[Z] = 7s$ and $E[Z] = 0.5s$. We point to [15] for a sensitivity analysis showing that larger think times tend to provide more reliable estimates of I. Notice that for $E[Z] = 7s$ and $E[Z] = 0.5s$ the different index of dispersion estimates result in small errors within $5\% - 10\%$ on the throughput predictions. To understand better this phenomenon, we discuss in Section 6.2 sensitivity analysis of our results with respect to the estimated value of I. Furthermore, as the load is increased, the mean service time of each resource for both MVA and MAP queueing network models is scaled according to the observed value in the corresponding experiment. This value is estimated by linear regression of resource utilization samples against throughput [19].

Figure 11 gives evidence that the new analytic model based on the index of dispersion achieves can explain observations much better than MVA models across the workload mixes, since it is reliable also when the workloads are not bursty. In the browsing mix, the index of dispersion enables the queueing model to effectively capture *both* burstiness and bottleneck switch. The results of the proposed analytic model match closely the experimental results for the browsing mix, while remaining robust in all other cases.

The shopping mix presents an interesting case: as observed in Section 4, the MVA model performs well

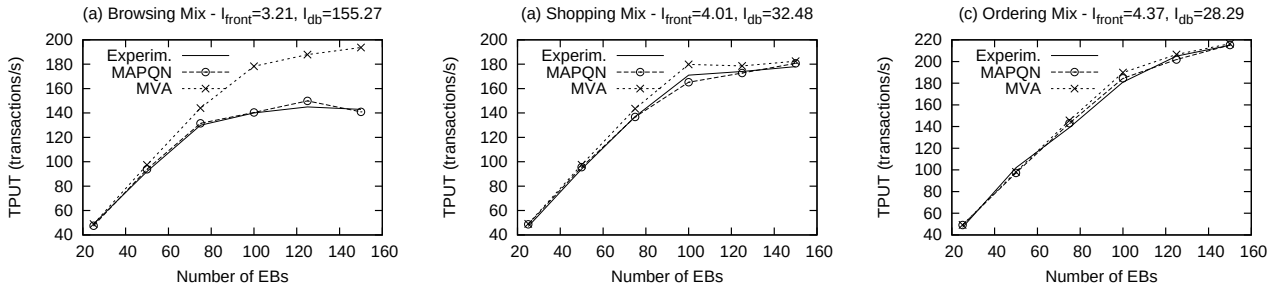


Fig. 11. Modeling results for three transaction mixes as a function of the number of EBs.

on the shopping mix despite the burstiness because, regardless of the variation of the workload at the database server, the front server remains the major source of congestion for the system and the new model behaves similarly to a MVA model (no bottleneck switch). This stresses the fact that our modeling methodology provides the largest improvements when the system exhibits bottleneck switches that cannot be modeled, even as approximations, by MVA models.

In the ordering mix, the feature of workload burstiness is almost negligible and the phenomenon of bottleneck switching between the front and the database servers cannot be easily observed, see Section 3.2. For this case, MVA yields prediction errors up to 5%. Yet, as shown in Figure 11(b) and 11(c), our analytic model further improves MVA's prediction accuracy.

6.2 Sensitivity Analysis

We illustrate the robustness of the proposed capacity planning methodology by studying how the prediction accuracy of the capacity planning models we have proposed in the previous sections changes due to estimation noise on the index of dispersion. We focus on the most challenging TPC-W browsing mix and investigate how noise in the I value at the database server that is affected by burstiness impacts on the mean throughput value of the queueing network. This is an important issue in practice, since inferring the characteristics of burstiness from limited measurements can introduce some errors on the I value, thus sensitivity analysis is needed.

To perform sensitivity analysis, we have considered the same MAP queueing network model used in Figure 11, but we have parameterized the MAP at the database server with a value of the index of dispersion equal to $k \cdot I$, where I is the index of dispersion value used in Figure 11 and k is a scaling factor representing the noise on the index of dispersion estimate. Note that changes in the value of k impact on the burstiness of the database service times, but not on their distribution. Figure 12 shows sensitivity analysis on the throughput prediction accuracy of the queueing network for different values of the scaling factor k. The value $k = 1.0$ corresponds to the analysis in the experiment in Figure 11, whereas the limit case $k = 0$ is a model without burstiness. The bars indicate the maximum absolute through-

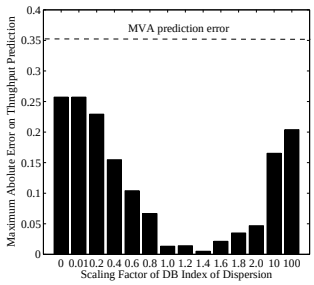


Fig. 12. Result sensitivity to estimated value of the index of dispersion I. The MVA prediction error is independent of I.

put prediction error on experiments with number of EBs from 1 to 200; the think times are $Z = 0.5s$. The figure shows that the best value of the index of dispersion is for $k = 1.4$, where the throughput prediction is in near perfect agreement with the experimental data. In particular, in the range $k \in [0.6, 2.0]$ the prediction of the model is relatively stable with respect to the results in Figure 11 and the maximum absolute relative error is of 10% for $k = 0.6$.

Outside this interval, the MAP queueing network model shows significant deviations up to $20\% - 25\%$ of the experimental throughput. However, it is still very interesting to note that these results remain more accurate than those of the basic MVA model, which has a maximum absolute error of about 35%.

Summarizing, the experiments in this section suggest that the estimate of the index of dispersion I do not need to be very accurate to lead to significant improvements compared to MVA models. It is found that errors up to $50 - 100\%$ on the I estimate can be still tolerated.

6.3 Flash Crowd Model

We conclude the experimental validation by considering the case of a flash crowd model. Due to the technical difficulties of implementing a benchmark with unbounded flash crowds, we have focused on modeling the bounded case using the experimental data collected using a modified TPC-W benchmark [16]. In that paper, we considered the same testbed of the present work and analyzed the fluctuations in the resource consumption

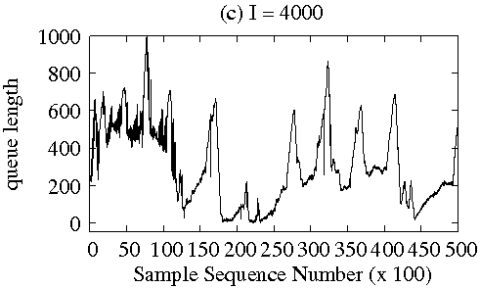


Fig. 13. Flash crowd experiment studied in [16], $I = 4000$, $E[Z] = 7s$, Browsing mix. Sum of queue lengths at the front and the database servers.

when *the think times* are characterized by an extremely large burstiness equal to $I = 4,000$. We have shown that this parameterization yields frequent flash crowds of many hundreds users, see the browsing mix experiment in Figure 13. Populations of this level are $N = 1,000$ which are about 10 times larger than in the configuration examined in the previous experiments, thus they are representative of large flash crowd events and may be seen as an open-type workload, since the population is very large. To avoid instability of the system, we have increased the average think time to $E[Z] = 7$ seconds as in the standard TPC-W specification. The data indicates that the departure process from the front server back to the clients is very bursty, with an index of dispersion equal to $I = 1985$.

In order to model the system shown in Figure 13, we have parameterized a MAP(2) $(\mathbf{Z}_0, \mathbf{Z}_1)$ using the same algorithm described in [16]. The MAP has then been integrated into the queueing network as the load-dependent service process $(n\mathbf{Z}_0, n\mathbf{Z}_1)$ of the delay server. We have then computed the expected throughout and end-to-end response time of requests using simulation and the Norton's equivalent method.

Model results are a predicted throughput of 99.96 job/s (Norton's approximation) and 104.44 job/s (simulation) against an observed 102.55 job/s; for response time the model returns 3.00s (Norton's approximation) and 2.61s (simulation) against the measured 2.642s. The results indicate that our modeling approach works well also when the system deals with flash crowds. Still, the good agreement of the experimental results with model predictions suggest that MAP queueing networks can be a versatile tool for modeling multi-tier application performance in a variety of contexts.

7 Related Work

Capacity planning of multi-tier systems is a critical part of the architecture design process and requires reliable quantitative methods, see [14] for an introduction. Queueing models are popular for predicting system performance and answering what-if capacity planning questions [14], [21]–[23]. Single-tier queueing models focus on capturing the performance of the most-congested resource only (i.e., bottleneck tier): [23] describes the application tier of an e-commerce system as a M/GI/1/PS queue; [18] abstracts the application tier of a N-node cluster as a multi-server G/G/N queue.

Mean Value Analysis (MVA) queueing models that capture all the multi-tier architecture performance have been validated in [21], [22] using synthetic workloads running on real systems. The parameterization of these MVA models requires only the mean service demand placed by requests at the different resources. In [19] the authors use multiple linear regression techniques for estimating from utilization measurements the mean service demands of applications in a single-threaded software server. In [13], Liu et al. calibrate queueing model parameters using inference techniques based on end-to-end response time measurements. A traffic model for Web traffic has been proposed in [12], which fits the real data using the mixture of distributions.

However, the observations in [17] show that autocorrelation in multi-tier systems flows, which is ignored by standard capacity planning models, must be accounted for accurate performance evaluation. Indeed, [1] presents that burstiness in web traffic and related application peaks the load of the Web server beyond its capacity, which results in the significant degradation of the actual server performance. In this paper, we have proposed solutions for capacity planning under workload burstiness. The class of MAP queueing networks considered here can capture the effects of burstiness and have been studied in [3] together with a bounding technique for approximate model solution. In this paper, we have proposed a parameterization of MAP queueing networks using for the service process of each server its mean service time, the index of dispersion, and the 95-th percentile of service times. The index of dispersion has been frequently adopted in the networking literature for describing traffic burstiness [5]; in particular, it is known that the performance of the G/M/1/FCFS queue in heavy-traffic is completely determined by its mean service time and the index of dispersion. Recently, in [16] a new benchmarking technique based on the index of dispersion has been proposed for multi-tier applications motivated by the measurements in [15].

8 Conclusion

In this work, we have presented a solution to the problem of modeling burstiness in enterprise applications by inferring essential service process information from utilization and throughput measurements. The parameterized queueing model can thus be used to closely predict performance in systems even in the challenging case where there are bottleneck switches among the various servers. Detailed experimentation on a multi-tiered system using the TPC-W benchmark validates that the proposed technique offers a robust solution to predicting performance of systems subject to burstiness, bottleneck switching conditions, and flash crowds.

REFERENCES

[1] G. Banga and P. Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

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

[3] G. Casale, N. Mi, and E.Smirni. Model-Driven System Capacity Planning Under Workload Burstiness. *IEEE Trans. on Computers*, 59(1):66–80, Jan 2010.

[4] D. Garcia and J. Garcia. TPC-W E-commerce benchmark evaluation. *IEEE Computer*, pages 42–48, Feb. 2003.

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

[6] A. Heindl. Correlation bounds for second-order MAPs with application to queueing network decomposition. *Perf. Eval.*, 63(6):553-577, June 2006 .

[7] http://www.mercury.com/us/products/diagnostics

[8] A. Horváth and G. Horváth and M. Telek. A joint moments based analysis of networks of MAP/MAP/1 queues. *Perf. Eval.*, 67(9):759-778, 2010.

[9] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. ASA-SIAM, 1999.

[10] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[11] S. Li and C. Hwang. Queue Response to Input Correlation Functions: Discrete Spectral Analysis. *IEEE/ACM Trans. Networking*, 1(6), pp. 678-692, 1993.

[12] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perf. Eval.*, 46(2-3), 2001.

[13] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang. Parameter inference of queueing models for it systems using end-to-end measurements. *Perf. Eval.*, 63(1):36–60, 2006.

[14] D. A. Menascé and V. A. F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice-Hall, Inc., 2000.

[15] N. Mi, G. Casale, L. Cherkasova, E. Smirni. Burstiness in Multi-Tier Applications: Symptoms, Causes, and New Models. *Proc. of Middleware*, Springer LNCS 5346, 265-286, Dec 2008.

[16] N.Mi, G.Casale, L.Cherkasova, E.Smirni. Sizing Multi-Tier Systems with Temporal Dependence: Benchmarks and Analytic Models. *Springer J. Internet Services and App.*, 1(2): 117-134, Aug 2010.

[17] 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.

[18] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *Proc. of IWQoS*, 2002.

[19] J. Rolia and V. Vetland. Correlating resource demand information with arm data for application services. In *Proc. of WOSP*, pages 219–230. ACM, 1998.

[20] R. Sadre, B. R. Haverkort. Flows in Networks of MAP/MAP/1 Queues. In *Proc. of MMB*, pages 195 208, 1998.

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

[22] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Proc. of ICAC*, 217–228, 2005.

[23] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Technol.*, 7(1):7, 2007.

[24] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Proc. of Middleware*, 244–265, Newport Beach, CA, 2007.

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

Giuliano Casale 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 in 2010 the Department of Computing at Imperial College London where he holds a Junior Research Fellowship. His research interests include performance modeling, workload characterization, and resource management. He is co-author of the Java Modelling Tools performance evaluation suite (http://jmt.sf.net). From 2011, he serves as secretary/treasurer of the IEEE STC on Sustainable Computing and as editor of ACM Performance Evaluation Review. He currently serves as program co-chair for QEST 2012 and for ACM SIGMETRICS/Performance 2012. He is a member of ACM, IEEE, and IEEE Computer Society.

Ningfang Mi Ningfang Mi is an Assistant Professor at Northeastern University, Department of Electrical and Computer Engineering, Boston, MA 02115 (ningfang@ece.neu.edu). She received her Ph.D. degree in Computer Science from the College of William and Mary, VA in 2009; her doctoral dissertation was on dependence-driven techniques in system design. She received her M.S. in Computer Science from the University of Texas at Dallas, TX in 2004 and her B.S. in Computer Science from Nanjing University, China, in 2000. Her research area mainly focuses on capacity planning, resource management, performance evaluation, simulation, virtulization, and cloud computing.

Lucy Cherkasova Dr. Ludmila Cherkasova is a principal scientist at HP Labs, Palo Alto, USA. Her current research interests are in developing quantitative methods for the analysis, design, and management of concurrent and distributed systems (such as emerging systems for "Big Data" processing, internet and enterprise applications, virtualized environments, and next generation data centers). She is the ACM Distinguished Scientist and is recognized by the Certificate of Appreciation from the IEEE Computer Society. She has authored over 80 referred publications and more than 70 patent applications. Her mostrecent works were on the design of new technologies for efficient management and capacity planning of internet and enterprise systems with emphasis on performance and scalability issues.

Evgenia Smirni Evgenia Smirni received the diploma degree in computer engineering and informatics from the University of Patras, Greece, in 1987, and the PhD degree in computer science from Vanderbilt University in 1995. Currently she is a Professor of Computer Science at the College of William and Mary, Williamsburg, Virginia. Her research interests include analytic modeling, stochastic models, Markov chains, matrix analytic methods, resource allocation policies, Internet and multi-tiered systems, storage systems, workload characterization, and modeling of distributed systems and applications. She has served as a program cochair of QEST 2005, of the ACM SIGMETRICS/Performance 2006, and of HotMetrics'10. She has also served as a general cochair of QEST'10. She is a member of the ACM, the IEEE, and the Technical Chamber of Greece.