User-Friendly Approach to Capacity Planning Studies with Java Modelling Tools

Marco Bertoli
Politecnico di Milano
Dip. di Elettr. e Informazione
Via Ponzio 34/5
Milano, Italy, 20133
bertoli@elet.polimi.it

Giuliano Casale
College of William and Mary
Dept. of Computer Science
140 McGlothlin-Street Hall
Williamsburg, VA, 23187-8795
casale@cs.wm.edu

Giuseppe Serazzi
Politecnico di Milano
Dip. di Elettr. e Informazione
Via Ponzio 34/5
Milano, Italy, 20133
giuseppe.serazzi@polimi.it

ABSTRACT

We present the Java Modelling Tools (JMT) suite, an integrated framework of Java tools for performance evaluation of computer systems using queueing models. The suite offers a rich user interface that simplifies the definition of performance models by means of wizard dialogs and of a graphical design workspace.

The performance evaluation features of JMT span a wide range of state-of-the-art methodologies including discrete-event simulation, mean value analysis of product-form networks, analytical identification of bottleneck resources in multiclass environments, and workload characterization with fuzzy clustering. The discrete-event simulator supports several advanced modeling features such as finite capacity regions, load-dependent service times, bursty processes, fork-and-join nodes, and implements spectral estimation for analysis of simulative results. The suite is open-source, released under the GNU general public license (GPL), and it is available for free download at http://jmt.sourceforge.net.[1]

1. INTRODUCTION

Java Modelling Tools (JMT) is a suite of free open-source Java applications for researching, teaching, and applying performance evaluation methodologies based on queueing models[5, 6]. JMT supports a wide-range of activities that are common in performance evaluation, such as solution of capacity planning models via simulation or analytical algorithms, feature extraction and pre-processing of log traces, clustering algorithms for selection of the most significant workloads to be modeled, determination of the optimal load conditions, and automatic identification of performance bottlenecks. The focus of JMT is particularly on queueing systems and queueing networks models [18], in which requests travel through a number of stations suffering queueing delays because of service contention. Usually, the analysis focuses on estimating performance indices such as mean throughputs and response times of requests. Queueing models are wide-spread in the performance evaluation community because they can capture very well the trend of performance indices of real systems including, but not limited to, multi-tier architectures [26, 9], storage arrays [14], and communication networks [13]. A comprehensive documentation of JMT and related case studies are available at [16].

The JMT suite started as a project of the Politecnico di Milano - DEI in 2002 with the aim of integrating into a single portable Java application pre-existing tools for workload characterization and queueing analysis, namely Win Modelling Tools [21]. Later, it evolved into a more sophisticated and integrated open-source application that runs on Windows, Linux, and MacOS. In particular, from April 2006 the tool has adopted a distribution and maintenance model based on sourceforge.net, which has quickly boosted the diffusion of the tool through practitioners, students, and researchers. Currently, the application has been downloaded about ten thousand times and has a solid base of users contributing to the JMT development forums.

A distinguishing feature of JMT is that its design has enhanced the usability and accessibility of the tool through a rich graphical user interface, rather than exposing the technical detail. In particular, the tool is able to guide users through simple wizard interfaces. Further, the parameterization of experiments requires minimal interaction with the user. In this way, JMT hides the complexity of the core algorithm implementations, thus significantly reducing the learning curve of inexperienced users. This feature also makes the tool of special interest for teaching purposes. However, interface simplifications do not penalize the technical level of the tool. JMT implements state-of-the-art methods for discrete-event simulation and analytical evaluation of queueing models as we describe in the remainder of this paper. The integration of the different applications that compose the tool as well as the communication between the graphical user interface and the underlying algorithms is based on XML. This, together with the open source development model, makes it simple to interface JMT algorithms with external software. For instance, OPEDo [2] uses the JMT analytical algorithms to optimize queueing models. Furthermore, JMT can execute parametric what-if analyses which are useful to evaluate the sensitivity of performance estimates to changes in the model characteristics.

[1]A short version of this paper titled "JMT - Performance Engineering Tools for System Modeling" will appear in the ACM Sigmetrics PER newsletter. ACM PER papers are not copyrighted and the authors retain the copyright.

SIMUTools 2009, Rome, Italy

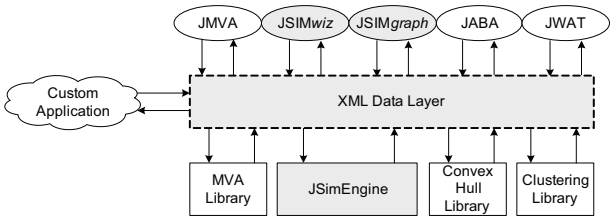

Figure 1: JMT - General Architecture

Parametric what-if curves can be plotted by JMT to display trends of performance indices as well as confidence intervals determined by the simulation engine.

The remainder of the paper is as follows. In Section 2, we introduce the tools that compose the suite and describe the general architecture of JMT. Section 3 describes discrete-event simulation of queueing models. Analytical evaluation of queueing models and bottleneck identification methods are described in Section 4 and Section 5, respectively. We discuss the features of JMT for workload characterization and traffic analysis in Section 6. In Section 7, we present a case study of admission control analysis in multi-tier systems which uses several tools of the JMT suite. Finally, Section 8 summarizes the paper and outlines future work.

2. JMT SUITE ARCHITECTURE

The architecture of JMT (version 0.7.4) consists of a set of loosely coupled graphical applications that communicate using XML with a core algorithmic module composed by the simulation engine (JSIMengine) and by a library of analytical functions, see Figure 1. Applications are selected through the main JMT interface, see Figure 2. JSIMgraph and JSIMwiz are graphical and wizard-based design environments for queueing models, respectively. The two applications provide a front-end interface for the underlying discrete-state simulation engine. The focus of JSIMgraph and JSIMwiz is on generating XML specifications of simulation models, pretty-print visualization of complex networks, automatic model debugging, and dynamical presentation of the current simulation state and of the current estimates of performance metrics and related confidence intervals. JSIMengine supports the evaluation of the most important classes of queueing models that cannot be solved with exact analytical techniques. These include, among others, multiclass queueing networks with blocking, priorities, fork-and-join elements, burstiness, and state-dependent routing schemes.

JMVA is a graphical user interfaces for the analytical evaluation of queueing network models. The tool relies on an implementation of the Mean Value Analysis (MVA) algorithm for closed networks [24], together with similar algorithms described in [7] for open and mixed networks. Compared to JSIMgraph and JSIMwiz, JMVA relies on a much simpler description of the model based only on the mean service demands of the different classes of requests and on the workload intensities (arrival rates, population sizes).

JABA is an analytical tool for automatic identification of performance bottlenecks in multiclass closed queueing networks. The tool receives in input a set of service demands specifying the speed of each server in processing requests of the different classes. Using the geometrical approach in [10],

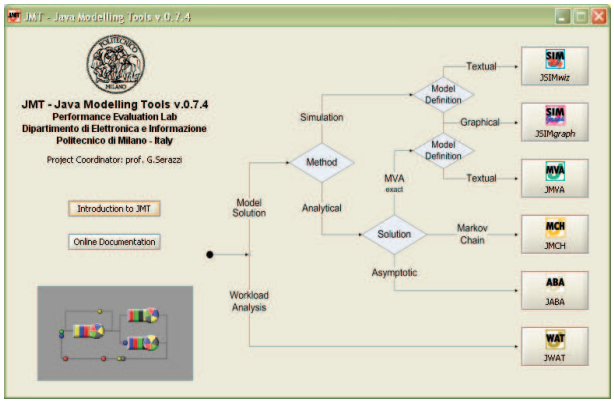

Figure 2: JMT - Tool Selection Interface

JABA efficiently identifies under which mixes of requests a server can become the most congested in the network. This saves the computational costs of a long simulative analysis over different mixes of requests.

The JWAT tool offers a stand-alone environment for log file analysis and preprocessing, identification of relevant workloads in multiclass performance data using clustering techniques, and primitives for characterization of time-varying workloads (e.g., burstiness analysis). The tool is used in the derivation of the parameters needed to define the queueing models analyzed by the other applications of the JMT suite, such as the mean service demands used in JMVA and JABA.

Finally, the JMCH application is a graphical simulator of M/M/1 and M/M/1/K queues. The simulation state is visualized both on the queue buffer and on a Markov model representing the system. The tool has the pedagogic purpose of showing to students the very basic ideas underlying queueing system analysis. Because this tool has only graphical capabilities, we do not discuss it in the rest of the paper.

3. JSIMENGINE, JSIMGRAPH, AND JSIMWIZ

JSIMgraph is a graphical design environment for queueing network models which is tightly coupled to the JSIMengine for running discrete-event simulation, see Figure 3. JSIMwiz gives the same functionalities of JSIMgraph, but replaces the graphical framework with a set of wizards that guide the user through the definition of a queueing model as depicted in Figure 5. In the simulation of queueing network models with JSIMgraph and JSIMwiz, stations are represented as a composition of three objects: input section, service section, and output section. An input section can be either a finite buffer that receives jobs from other stations or from one or more infinite sources that generate new jobs. The other sections specify scheduling disciplines (e.g., First-Come First-Served - FCFS, Last-Come First-Served - LCFS) and routing schemes (e.g., Join the Shortest Queue - JSQ). Currently, there is support only for non-preemptive scheduling disciplines. The simulation engine works on the basis on a number of "distributed protocols" which specify the interaction of different sections belonging to the same or to different stations. For example, in a queueing model with blocking, when a job departs from a queue, its routing section and the input section of the destination queue decide without the help of a centralized controller if the job can be accepted in the destination buffer. This simulation style

has been implemented to allow in future releases of JMT the migration to a multi-processor or distributed simulation environment, see [6] for additional details.

The simulator supports a variety of distributions for arrival and service processes, including Exponential, Erlang, Hypo- and Hyper-Exponential, Pareto, and Gamma. These are obtained as usual by evaluating the inverted cumulative distribution function (CDF) on a random number sampled from a uniform distribution. High-quality pseudo-random numbers are generated using the Mersenne Twister engine, which has low computational requirements [19]. Samples can be also read directly from an external log file. We have also added to the simulator the capability of generating samples from correlated processes, which are important to model *burstiness* in arrival or service times. In particular, we have implemented in the simulator support for the burstiness model of [20], we point to Section 6 for further details. Another important feature of JSIMengine is that the simulator provides native mechanisms for the definition of *load-dependent service times*. Load-dependence is useful to represent certain systems, e.g., disk drives, where the latency of service can be a function of the local number of enqueued jobs. Load-dependence is also important in hierarchical modelling and parametric analysis [11]. In JSIMgraph and JSIMwiz the user is given the flexibility to specify load-dependent service rates by associating a different service distribution to a specific range of populations in the local queue. Further, the parameters of each distribution can be expressed as an arbitrary mathematical function of the current population.

JSIMgraph and JSIMwiz allow the definition of a number of stations and attributes resulting in models that are not easy to evaluate analytically and typically require simulation to be solved. First, there is support for *fork* and *join* nodes, in which a request is first forked into a user-specified number of sibling tasks which are later synchronized at the join node. These models are important to describe parallel systems where a request is decomposed into a number of processing units for load-balancing reasons. The simulator also supports *finite capacity regions* which impose constraints on the maximum number of jobs accessing a local subnetwork of servers. These are extremely useful to capture the behavior of performance saturation effects which are imposed by admission control policies or memory constraints. Finite capacity regions can include multiple stations (see the blue area in Figure 3), they can put limitations only on the population of selected classes or on the aggregate number of jobs in the subnetwork, and they can either drop incoming jobs or place them in a hidden admission buffer at the boundary of the region. Currently, the support for finite capacity regions is meant as a simple high-level approximation of admission control, since JSIMengine does not allow to specify dynamic admission rules that depends on the whole state of the network. We remark that statistics about the behavior of the finite capacity region can be collected by specific performance indexes, e.g., request drop rates at the boundary of the region. Other important modeling features supported by JSIMengine include support for *priority classes* in scheduling disciplines (namely priority FCFS and priority LCFS), *open and closed populations*, and *state-dependent routing strategies* which direct jobs to stations with either the least utilization, the smallest expected response time,

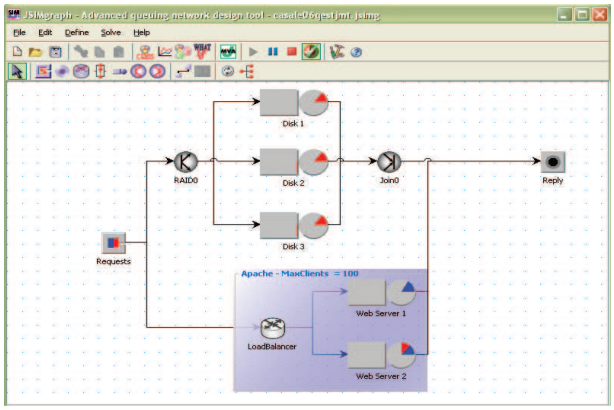

Figure 3: JSIMgraph - Graphical design workplace

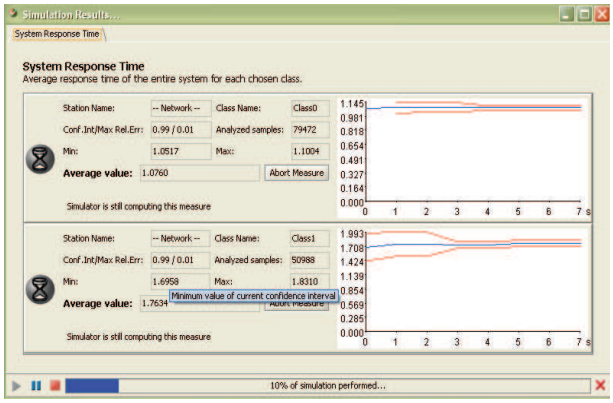

Figure 4: JSIMgraph - Online computation of confidence intervals by the spectral method

the fastest mean service time, or the the shortest queue length (i.e., JSQ load-balancing).

In the remainder of this section, we complete the presentation of the JMT simulation capabilities by discussing some important features related to the statistical analysis of simulation results and to the control of simulation experiments.

3.1 Analysis of Simulation Results

JSIMengine supports the estimation of several reward measures that describe the performance of the simulation models. These include *station-level* metrics such as queue length, response time over a single or multiple visits, utilization, throughput, and drop rate for queues with a finite buffer. There is also a support for *system-level* aggregates like the total number of jobs in the simulation model or within a finite capacity region, as well as the throughput and response time of jobs in traversing (or cycling for closed classes) the network. Each performance index can be estimated for a particular job class or as an aggregated measure over all classes.

Each station in the simulation model is associated with an object called *JobInfoList* that, for each performance index requested by the user, feeds online a statistical analyzer with samples of that particular metric. Simulation results are analyzed using transient detection and confidence interval estimation algorithms. These techniques are executed online and are responsible to stop the simulation when it

is conveyed that all performance metrics can be estimated with sufficient accuracy. Confidence intervals are generated using the spectral method presented in [15], see Figure 4 for their online representation in JSIMgraph. The effectiveness of this technique depends on the stationarity of the sample distribution. If the series of samples shows trends or pronounced non-stationary behaviors, the estimate of its power spectrum can be unreliable and this degrades the quality of the confidence intervals generated by the spectral method. Therefore, techniques for determining if a group of samples is stationary are important to maximize accuracy, since detecting and removing non-stationary data can greatly improve the quality of power spectrum estimates. For this purpose, we have implemented transient detection using the R5 heuristic [12] and the MSER-5 stationarity rule [23]. We point to [6] for a flowchart that explains how these two techniques have been combined in JSIMengine.

After completing the transient detection phase, the statistical analyzer runs the spectral method of Heidelberger and Welch [15], which is a stable and computationally efficient method for computing simulation confidence intervals using variable batch sizes and a fixed amount of memory. JSIMengine runs the spectral analysis periodically until the confidence intervals are generated with enough accuracy. Nonetheless, one run is often sufficient to accurately estimate confidence intervals. The spectral analysis proceeds as follows. Given a stationary sequence of samples

$$X(1), \ldots, X(N)$$

of a performance measure X, the spectral analysis obtains an estimate of the power spectrum $p(f)$ at the frequency $f = 0$. The value $p(0)/N$ is an unbiased estimator of the sample variance, which can be used to generate confidence intervals on X also if the sequence of sample is correlated. The advantage of the technique with respect to the standard variance estimator is that the latter does not hold if the samples are correlated. However, measured of performance metrics in queueing network models are often correlated and this is an important motivation to use spectral analysis methods in JSIMengine. The spectral analysis requires computation of the sample periodogram $I(n/N)$, $0 < n < N/2$, and then generate a certain function $J(n)$, which is the logarithm of the mean of two consecutive values of $I(n/N)$. The most important property of $J(n)$ is that its functional shape is a low-order polynomial in a neighborhood of a zero frequency, thus the value of $I(0)$, from which we compute the unbiased estimator of the sample variance, can be estimated by inexpensive regression of $J(n)$. In JSIMengine, we implemented the regression technique using singular value decomposition and we fit the data to a polynomial of order two. This is consistent with [15] which shows that $J(n)$ is often linear or quadratic in a neighborhood of $n = 0$. We point to [6] for additional details on the simulation engine and on the analysis of simulation results.

3.2 Control of Simulation Experiments

JSIMengine offers several options for controlling simulation experiments. Besides defining long-run simulations, the most interesting features are the possibility of executing an automatic stop as a function of the quality of confidence interval estimates and to run parametrically a sequence of "what-if" experiments. The automatic stop is based on the concept of maximum relative error of the confidence interval

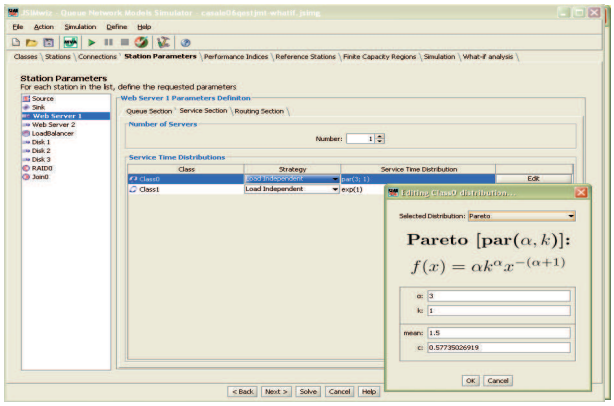

Figure 5: JSIMwiz - Wizard-guided analysis

[22] and stands for maximum acceptable ratio ϵ between the half-width of the confidence interval measure α and the estimated mean. For instance, setting $\epsilon = 0.07$ and $\alpha = 0.05$ imposes a simulation stop when the half-width of the 95% confidence interval is no more than 7% of the sample mean.

JSIMgraph and JSIMwiz integrate native support for "what-if" analyses, which are repeated execution of simulation experiments for different values of a control parameter. Parametrical experiments can be controlled by changing the value of the input traffic rate, of the number of jobs for closed classes, of the percentage of jobs belonging to a closed class, of the mean service times, or for different initialization seeds of the random number generator.

4. JMVA

JMVA is a tool for computing mean performance indexes of the class of product-form queueing network models defined in [4]. The tool uses an efficient implementation of the Mean Value Analysis (MVA) algorithm of Reiser and Lavenberg [24] for closed multiclass networks. However, JMVA can also compute performance indexes for models with open and mixed workload classes. The implementation of MVA is based on the algorithms described in [7] which support both constant-rate and load-dependent queues. The specification of load-dependent rates uses the same approach described for JSIMgraph and JSIMwiz.

Compared to JSIMengine, JMVA has much lower execution times on models where the number of workload classes is not too large, usually less than three or four classes. However, JMVA can have a larger memory requirement than JSIMengine if the populations of closed classes are composed by several tens or hundreds of requests, because of the well-known inefficiency of the MVA algorithm in this case. However, the increased speed of JMVA is an important motivation to use this tool, especially in what-if analysis which can be very time-consuming if simulation is used. JMVA offers the same what-if analysis functionalities of JSIMengine (see Figure 6) but, because of the properties of the MVA algorithm, a restricted set of performance indexes can be computed. These are mean response times, mean throughputs, utilizations, and mean queue-lengths.

An interesting function of JMVA is the capability of importing model files saved by the JSIMgraph/JSIMwiz; also the JSIM application can read models saved by JMVA. This strengthen the inter-operability of the tools of the suite.

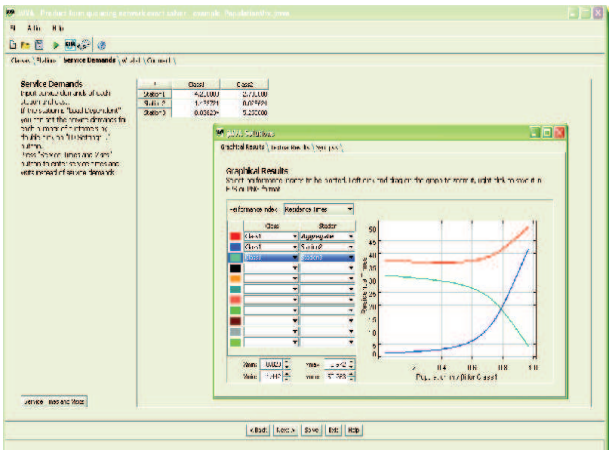

Figure 6: JMVA - What if analysis

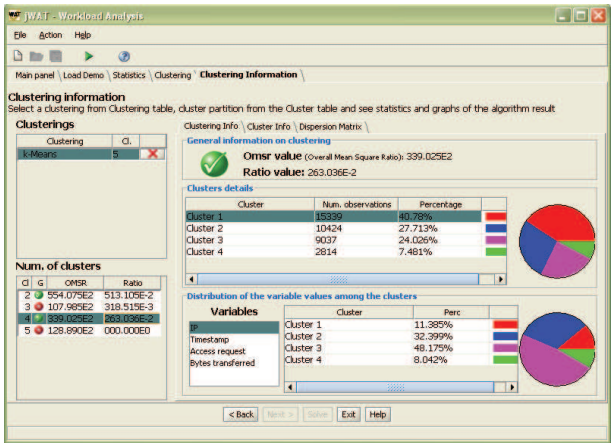

Figure 7: JWAT - Clustering Result Exploration

When an analytical model saved with JMVA is opened in JSIMgraph, the graphical user interface creates a graphical arrangement of the model based on automatic layout techniques and arbitrarily assigns service distributions to the stations consistently with the product-form assumptions [4] and with the mean service demands specified in JMVA. JSIMwiz behaves similarly to JSIMgraph, but without the need of a graphical arrangement. Instead, when a simulation model is imported in JMVA the situation is completely different, since the features of a simulation models are a superset of the ones support by JMVA. In this case, the tool reports warnings on properties of the model that are incompatible with product-form assumptions and tries to generate a product-form model that is the closest possible, but not always identical, to the input simulation model.

5. JABA

Java Asymptotic Bound Analysis (JABA) is a tool for the automatic identification of bottleneck stations in multiclass queueing networks. To understand the application of this tool, consider the following problem. A multi-tier system processes transactions belonging to several classes and, because the web server limits the maximum number of simultaneous connections, only up to a predefined number requests can be processed at a given instant. In general, the particular mix of requests served determines the bottleneck resource, i.e., a certain mix in which requests to pictures are the majority of the incoming traffic may overload the Web server, another mix with several queries may place the bottleneck at the database server, and a combination of the two may stress both servers simultaneously. The knowledge of the potential bottlenecks of an architecture is very important for resource provisioning, system sizing, and performance tuning. JABA has been designed with the focus on this type of performance analysis and optimization studies.

JABA searches for potential bottlenecks in a network with arbitrarily large number of stations and two or three workload classes. Given a description of server speeds, i.e., mean service demand for each class at each server, JABA determines for two class models the group of potential bottlenecks as a function of the mix of requests expressed as a vector $\vec{\beta} = (\beta_1, \beta_2)$, where β_1 is the percentage of requests of class one out of the total requests in the system and, similarly, $\beta_2 = 1 - \beta_1$ is the fraction of requests of class two. A graphical representation of the functional dependence on $\vec{\beta}$ is given based on diagrams similar to Figure 10 given in the case study. A similar functionality exists also for models with three workload classes, in this case the mix vector is $\vec{\beta} = (\beta_1, \beta_2, \beta_3)$ which can represented graphically as the triangular section of the plane $\beta_1 + \beta_2 + \beta_3 = 1$ where all β_rs, $1 \leq r \leq 3$, are non-negative.

In addition to mapping the potential bottleneck set as a function of the mix $\vec{\beta}$, JABA offers a convenient interface to explore graphically how this set changes as a function of the service demands of each server. The user is provided with a two-dimensional plot of points, each representing a server, with coordinates given by the service demands of the two workload classes at that server. A convex hull technique is used to compute the yellow polytope in Figure 11 reported at the end of the paper. According to the results in [10], points falling on the boundary of the polytope are potential bottlenecks. In a well-balanced system, all points would have similar coordinates and would concentrate on the same edge of the polytope; thus the number of edges of the polytope quantifies the level of balance of the system and the likelihood of bottleneck shifting between resources.

In JABA, the user can also consider what-if scenarios by changing the service demands of a server and seeing graphically how the polytope and the potential bottleneck sets are modified. This is supported by a drag-and-drop interface for the points of the polytope in Figure 11. This can be useful to investigate the robustness of an architecture under slight variations of the workload characteristics as well as to investigate the sensitivity of the results if measurement errors have affected the service demand estimates.

6. JWAT

The Java Workload Analyzer Tool (JWAT) is an application for exploratory analysis of performance data and for generation of static and dynamic workload characterizations that can be used within JSIM, JABA, and JMVA models. JWAT is based on the workload characterization methodology presented in [8]. This starts with a *static analysis* phase, in which JWAT extracts samples from the original log file according to various criteria, computes the distribution and moments of the measures after a number of pre-

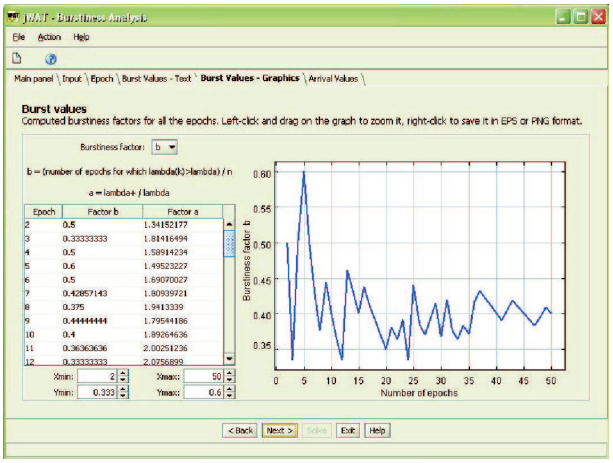

Figure 8: JWAT - Traffic Burstiness Analysis

processing and transformation steps, and then moves to a *dynamic analysis* in which time-varying properties of the trace, such as burstiness, are assessed.

Before starting the static analysis phase, JWAT applies preliminary filtering and sampling on the raw trace to reduce, upon loading in memory, the size of the data. Here, the user can formulate Perl5 regular expressions for specifying the columns of interests in the trace and check the correctness of the entries. We also provide predefined templates to import performance data from Apache web server log files. After selecting the metrics of interests, the user can apply further filtering and transformations to the data (e.g., variable standardization, logarithmic transformation) guided by graphical aids such as cross-correlation plots, histograms, scatter-plots, and QQ-plot diagrams. This stage of the analysis is followed by cluster analysis to partition the workload into representative classes. These classes can be used directly in JSIM, JABA and JMVA models as workload classes. For instance, the centroid of a cluster may be considered as a representative value of the service demands of a workload class. JWAT implements two clustering techniques: k-means and fuzzy k-means. Both techniques require the specification of the maximum number of clusters that JWAT can create and of the maximum number of iterations allowed. The last parameter controls the trade-off between clustering accuracy and computational costs. While k-means assigns an observation to only one cluster, fuzzy k-means computes for each observation a stochastic vector describing the probability of being a member of a certain cluster. Fuzzy characterization of multiclass workload can be more robust to outliers than traditional approaches. Clustering results can be inspected for both k-means and fuzzy k-means using the graphical interface, (see Figure 7).

JWAT also helps in identifying the optimal number of workload classes to be used in the cluster analysis. The optimality of a number of clusters is evaluated using different metrics for k-means and fuzzy k-means: for k-means, JWAT seeks the maximum *overall mean square ratio* (OMSR) value, which is the ratio between the squared sums of intra-cluster and inter-cluster similarities between samples. For fuzzy k-means, the internal consistency of each cluster is evaluated using the *entropy function*, which quantifies the amount of information carried by each cluster.

Dynamic modeling is still an experimental feature of the JWAT, aimed at the generation of models that can characterize the burstiness of a time series. JWAT uses the characterization proposed in [20]. This is a compact representation where the trace is first divided in intervals called epochs and then modeled by two parameters a and b that account for the percentage of epochs where the arrival rate is observed to be greater than the mean arrival rate and for the mean arrival rate associated with these traffic surges.

JWAT supports the computation of the parameters a and b using the traffic analysis functions. In the current implementation, there is support for investigating the sensitivity of a and b with respect to the user-defined number of epochs n. For instance, the graph plotted in Figure 8 shows the estimated value of b as a function of n for an HTTP trace of a university Web server. By progressively increasing the number of epochs, we see that the value of b converges to a value close to 0.40. However, the point-wise estimation of this value for small values of n ($n < 15$) suggests very different values, up to 0.60. This shows the practical advantage of sensitivity analysis in the estimation of the parameters a and b.

7. CASE STUDY

In this section, we illustrate the performance optimization methodology supported by JMT using a case study that shows how it is possible to use several tools (i.e., the simulator and the asymptotic bound analyzer) in an integrated way to find the optimal load of a network. We consider a multi-tier web system composed by a front server (FS), a storage server (SS), and a database server (DB). The application running on the multi-tier system exports two web services (WS1 and WS2) with the characteristics of mean service times and number of visits to the servers reported in Table 1. Due to the characteristics of the analyzed system, both requests of type WS1 and WS2 have been modelled as open classes with exponentially-distributed inter-arrival times $\lambda_{WS1} = 20\ req/s$ and $\lambda_{WS2} = 30\ req/s$, respectively.

Mean Service Times		
	WS1	WS2
FS	$28.48ms$	$68.07ms$
SS	$314.36ms$	$62.01ms$
DB	$111.37ms$	$126.90ms$

Number of Visits		
	WS1	WS2
FS	1.00	1.00
SS	0.22	0.89
DB	0.78	0.11

Table 1: Case study characteristics: mean service time and number of visits of service requests

We focus on the following scenario: the administrator wants to restrict the maximum number of concurrently served requests to $N_{max} = 100$ and needs to decide the best mix of requests of type WS1 and of type WS2 that should be admitted in concurrent execution into the system by the admission control algorithm. Indeed, the definition of best may depend on the design objectives, here we consider as best the one that generates the maximum throughput. In JMT, admission control can be modeled as a finite capacity region, see the shaded area in Figure 9; we assume that admission control redirects unadmitted requests elsewhere, which can be modeled in the queueing network as a request drop at the border of the finite capacity region.

A classical approach to find the best mix of services in execution is the exhaustive evaluation of all possible com-

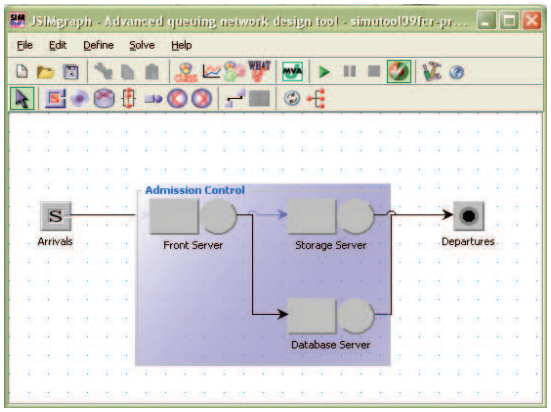

Figure 9: Case study: multi-tier architecture with admission control system

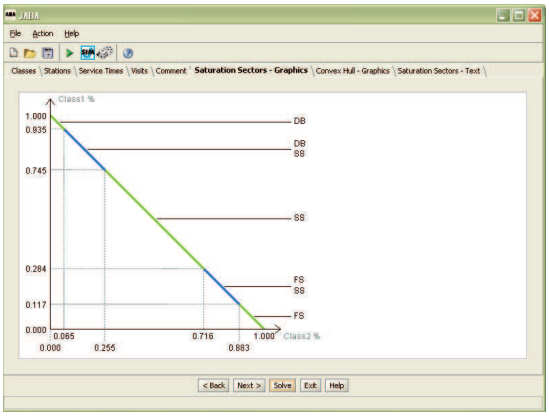

Figure 10: Case study: potential bottlenecks of the web architecture

binations of requests of the different types WS1 and WS2. However, this is not a scalable approach because the number of request types grows because the number of possible workload mixes grows combinatorially. In this section, we describe a solution approach supported by JABA that applies also to models with more than two classes, although for simplicity we consider only WS1 and WS2, and provides solutions in a computationally efficient manner.

In the first step of this methodology, the values in Table 1 are inserted in JABA to determine the graph in Figure 10. The figure shows the bottlenecks in heavy-load of the web architecture as a function of the percentage of WS1 requests (Class 1, vertical axis) and WS2 requests (Class 2, horizontal axis) while the total population of requests in the system is kept constant. The segments in green represent mixes of WS1 and WS2 requests where only one server saturates, the blue segments are mixes of requests for which two resources saturate concurrently. With the service demands of the considered system, no mix exists that produces saturation on all three servers concurrently. For instance, if the admission control system allows to enter the system 50% of requests of each type, the saturation condition is given by the middle sector of the figure, which tells us that the storage server (SS) will be the bottleneck.

Saturation is known to be problematic for response times, but in presence of an admission control scheme the maximum latency is externally controlled by the maximum number of concurrently served requests N_{max}. After this value has been fixed, one tries to maximize the utilization of the resources to increase the throughput. Clearly, two servers that work in parallel are better than a bottleneck (saturated)

experiment name	*max reqs* WS1	*max reqs* WS2	*expected bottleneck(s)*
I	95	5	DB
II	80	20	DB and SS
III	50	50	SS
IV*	20	80	FS and SS
V	5	95	FS
legend: * = mix with expected best performance			

Table 2: Experiment characteristics. Each experiment targets a different set of bottleneck resources.

resource coupled with an idle server. Thus, a logic choice is to configure the admission control algorithm with a target mix of requests within the blue segments in Figure 10. Let $\beta_1 \in [0, 1]$ be the fraction of WS1 requests admitted to service out of the total admitted; according the values given by the JABA algorithm and shown in Figure 10, for $\beta_1 \in [0.117, 0.284]$ front server and storage server saturate together, while for $\beta_1 \in [0.745, 0.935]$ database server and storage server are bottlenecks. To gain intuition on if it is better to have the front server or the database server saturating together with the storage, we consider the JABA polytope in Figure 11. Here, by clicking on the edge connecting front server and storage server, JABA displays the expected throughputs of the different classes according to theoretical formulas in [3]. When front server and storage server saturate, JABA expects a total throughput of 171 requests per unit of time; on the other hand, when database server and storage server saturate (case not displayed in Figure 11), they have an expected throughput of 147 requests per unit of time. Thus, our best guess for the optimal mix to be assigned to the admission control algorithm is any value within $\beta_1 \in [0.117, 0.284]$ which we expect to make front server and storage server saturate with maximal throughput.

Note that the estimates of JABA are obtained under product-form assumptions that do not apply theoretically to the model in Figure 9, however we have verified that these are often robust approximations. To confirm this assumption, we undertake a simulation study to validate the effectiveness of our prediction. We consider simulation experiments where we impose to the finite capacity region constraints on the maximum population of each class according to the segments in Figure 10. Table 2 summarizes simulation experiments together with the limits imposed in the finite capacity region. Each simulation is expected to evaluate the performance of the system in a different sector of Figure 10, i.e., by shifting the group of bottlenecks in all possible ways, we try to verify that the joint saturation of FS and SS gives the best performance, i.e., in our case the maximum throughput.

Simulation results have been obtained by long run simulation with 0.95 confidence interval, 0.05 maximum relative error, and a sample space of approximately 10, 000, 000 samples. Automatic stop functions called by the JSIMengine are disabled in all experiments. Numerical results are shown

in Figure 12 which shows the throughput for the different mixes. Note that the ratio between the throughput of experiment II and experiment IV is not too far from the 147/171 predicted theoretically and the individual values differ substantially in magnitude only because JABA assumes infinite requests processed, while here we have only one hundreds requests. For higher population values it is possible to see that the throughputs closely approach the theoretical values. Also the other conjectures formulated with JABA are all validated by the simulation. In fact, the throughput in experiment IV associated with saturation of FS and SS is indeed the maximum among the considered ones. The results for the response times are qualitatively similar, with experiment IV having a response time estimate for admitted jobs equal to $6.026s$, which is much better than the $7.345s$ achieved by experiment I; the other experiments have all response times between $6.390s$ and $6.547s$. Finally, we have also evaluated the server utilization in the five experiments finding that the largest utilization was consistent with the estimates in Table 2. For instance, in experiment IV, utilizations are 0.8478 for the front server, 0.9973 for the storage server, and 0.7227 for the database server, showing that the front server and the storage server are the closest to the saturation. A reason why the front server utilization is slightly less than the expected should be found in the nature of the JABA bottleneck evaluation technique, which is exact for product-form models with infinite populations, but which reduces to an approximate estimate for the models with finite capacity regions considered here.

Summarizing, we have illustrated with an example some of the performance evaluation studies that can be performed with JMT. In particular, we have shown that having different complementary tools in the suite can be helpful to validate the robustness of the predictions and to afford also optimization studies and not only simulations. Note also that one could have explored the expected throughputs and response times for the web system under the different mixes by means of JMVA. In this case, the approximation would be to evaluate a closed model with populations equal to the mix of requests of class WS1 and WS2 used in the five experiments. For example, setting the closed populations to $N_{WS1} = 20$ and $N_{WS2} = 80$ would have provided from JMVA a throughput estimate of $17.04req/s$ in the model of experiment IV, which is close to the simulation results and gives an accurate scaling of the $171req/s$ asymptotic throughput estimate given by JABA.

8. CONCLUSIONS

Java Modelling Tools (JMT) is an integrated environment for workload characterization and performance evaluation based on queueing models. This paper has summarized the features of the main applications that compose the suite, and has provided intuition on the versatility of JMT in dealing with the different aspects of the performance evaluation and optimization process.

There are several possible lines of extension of the tool. First, we plan to integrate approximate analytical methods (e.g., Bard-Schweitzer's Approximate MVA or the balanced job bounds [25, 18]) to evaluate queueing network models. These are considerably faster and less memory consuming than MVA on multiclass models. We also wish to add in the simulator native support for processes with burstiness characterized by Markov modulated processes. JMCH will also

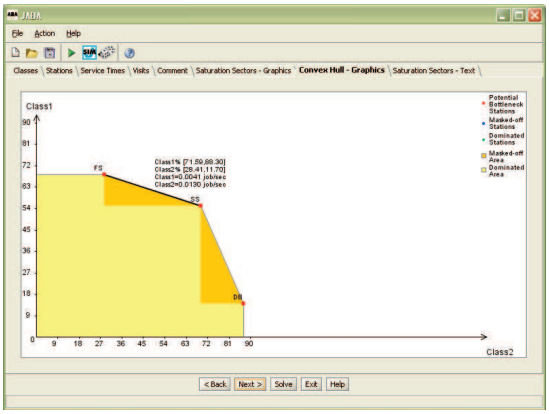

Figure 11: Case study: estimation of maximum throughput under saturation of multiple servers

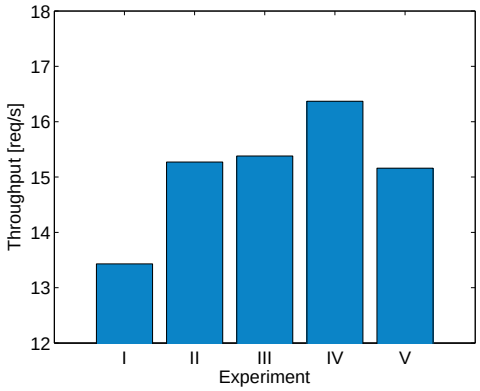

Figure 12: Case study: throughput of the experiments in Table 2. As expected, experiment IV has the best mix for admission control.

be extended to include other types of stations. We will also work toward a better integration of the different tools of the suite. Regarding possible extensions of the simulation algorithms, we will evaluate random number generators with multiple streams and substreams [17], as well as methods to estimate steady-state performance alternative to the spectral approach [1].

JMT is open source software released under the GNU general public license (GPL) and can be downloaded for free from the JMT web page http://jmt.sourceforge.net. We point to the web page for additional material.

9. ACKNOWLEDGEMENT

The authors wish to thank Richard R. Muntz for his comments on an earlier version of this manuscript which helped in improving the quality of this work.

10. REFERENCES

[1] C. Alexopoulos. Statistical Estimation in Computer Simulation. in *Handbooks in Operations Research and Management Science: Simulation*, S. G. Henderson and B. L. Nelson Eds., Elsevier, 2006, Chapter 8.

[2] M. Arns, P. Buchholz, and D. Müller. OPEDo: A Tool for the Optimization of Performance and Dependability

Models. *ACM Performance Evaluation Review*, 36(4), to appear in 2009.

[3] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. *Performance Evaluation*, 30(3):115–152, 1997.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[5] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proc. of the 3rd Conf. on Quantitative Evaluation of Systems (QEST)*, 119–120. IEEE, 2006.

[6] M. Bertoli, G. Casale, and G. Serazzi. The JMT simulator for performance evaluation of non-product-form queueing networks. In *Proc. of the 40th Annual Simulation Symposium (ANSS)*, 3–10, 2007.

[7] S. C. Bruell and G. Balbo. *Computational Algorithms for Closed Queueing Networks*. North-Holland, 1980.

[8] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *Proc. of the IEEE*, 81(8):1136–1150, 1993.

[9] G. Casale, N. Mi, L. Cherkasova, and E. Smirni. How to parametrize models with bursty workloads. *ACM Perf. Eval. Rev., Special Issue on the 1st HOTMETRICS Workshop*, 36(2):38–44, 2008.

[10] G. Casale and G. Serazzi. Bottlenecks identification in multiclass queueing networks using convex polytopes. In *Proc. of IEEE MASCOTS Symposium*, 223–230. IEEE Press, 2004.

[11] K. M. Chandy, U. Herzog, and L. Woo. Parametric analysis of queueing networks. *IBM J. Res. Dev.*, 19(1):36–42, 1975.

[12] G. S. Fishman. Statistical analysis for queueing simulations. *Management Science*, 20, 3:363–369, 1973.

[13] M. Garetto and D. Towsley. Modeling, simulation and measurements of queuing delay under long-tail internet traffic. In *Proc. of ACM SIGMETRICS*, 47–57. ACM Press, 2003.

[14] P. Harrison and S. Zertal. Queueing models of raid systems with maxima of waiting times. *Performance Evaluation*, 64(7-8):664–689, 2007.

[15] P. Heidelberger and P. D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Comm. of the ACM*, 24(4):233–245, 1981.

[16] JMT Documentation and Case Studies: http://jmt.sourceforge.net/Documentation.html

[17] A.M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 2000.

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

[19] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Comput. Simulation*, 8(1):3–30, 1998.

[20] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods.* Prentice-Hall, 2002.

[21] M. Pantano. Un'interfaccia grafica per modelli a reti di code e per la caratterizzazione del carico. Master Thesis (in Italian), Universita' Statale di Milano, Italy, July 1990.

[22] K. Pawlikowski. Steady-sate simulation of queueing processes: A survey of problems and solutions. *ACM Computing Surveys*, 22(2):123–168, 1990.

[23] K. Preston White Jr., M. J. Cobb, and S. C. Spratt. A comparison of five steady-state truncation heuristics for simulation. In *Proc. of the 32nd Winter simulation Conference*, 755–760. SCS, 2000.

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

[25] P. J. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. of the Int'l Conf. on Stoch. Control and Optim.*, pages 25–29, Amsterdam, 1979.

[26] 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*, 291–302. ACM Press, 2005.