An Overview of the JMT Queueing Network Simulator

M.Bertoli, G.Casale, G.Serazzi

Politecnico di Milano - DEI

Via Ponzio 34/5, I-20133, Milan, Italy

{bertoli, casale, serazzi}@elet.polimi.it

Abstract

This paper describes JSIM: the simulation module of the Java Modelling Tools (JMT), an open-source fully-portable Java suite for capacity planning studies. The simulator has been purposely developed to help both unexperienced and advanced users. Most of the difficult decisions that are needed in order to run simulations properly, such as the detection of the transient part of samples to be discarded, have been automated. The tool also provides guidance over the graphical design of the network and over the analysis and the plot of the results. What-if parametric analyses for parametric evaluation of complex systems are supported. Several features that increase the generality of the applications to capacity planning studies are provided, among them fork-join service centers, regions with finite capacity, state-dependent routing algorithms, priority classes and import of real workload distributions from log files.

1. Introduction

The availability of several simulation packages, either commercial or free, makes simulation one of the most commonly used techniques for performance evaluation of computer systems and networks. In general, evaluation techniques, that are methods by which performance evaluation indices are obtained, can be subdivided into two main categories: measurement (or empirical) techniques and modeling techniques. Empirical techniques require that the system or network to be evaluated exists and direct measurements of the evaluation target have to be taken. On the other hand, modelling techniques only require a model of the system. Modeling techniques are of two types: simulation and analytic. Among them, simulation is the most popular, as it applies to a wider variety of systems and does not require restrictive assumptions. However, in spite of their generality and ease of use, simulation models may fail or produce non-accurate results. A first source of errors is related to the statistical techniques implemented in the simulator engine, e.g., the quality of the random number generator, the detection and removal of the transients, the algorithms used for confidence intervals and variance estimation. A second source of errors comes from users' mistakes, such as inadequate level of detail adopted to describe the target system, too short simulation time, errors in input parameter values and distributions, errors in output data interpretation and incorrect modeling of the characteristics of the target system. JSIM, the software package described in the paper, is a simulator that aims to minimize common mistakes in simulation studies by helping the average user in two ways. Firstly, critical statistical decisions, such as transient detection and removal, variance estimation, and simulation length control, have been completely automated, thus freeing the user from taking decisions about parameters s/he may not be familiar with. Secondly, a user-friendly graphical interface allows the user to describe, both the network layout and the input parameters. Furthermore, the graphical interface also provides support for advanced features like fork and join of customers, blocking mechanisms, regions with finite capacity constraints on population, state-dependent routing strategies, user-defined general distributions and import of log data. A module for what-if analyses with several types of control parameters, particularly useful in capacity planning, tuning and optimization studies is also provided. JSIM is fully implemented in Java and is available for download at the URL `http://jmt.sourceforge.net`. Since it is distributed under GNU GPL as an open source code, users can easily add new modules or integrate existing ones to customize the tool according to their needs.

The paper is organized as follows. An overview of the simulation engine is given in Section 2. The implemented statistical techniques are reviewed in Section 3. The most important supported features for non-product-form modelling are described in Section 4. A case study focusing on the asymptotic behavior of queueing networks with finite capacity regions is described in Section 5. Conclusions and future developments are discussed in Section 6.

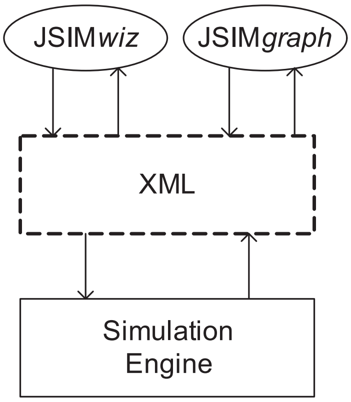


Figure 1. Simulator architecture.

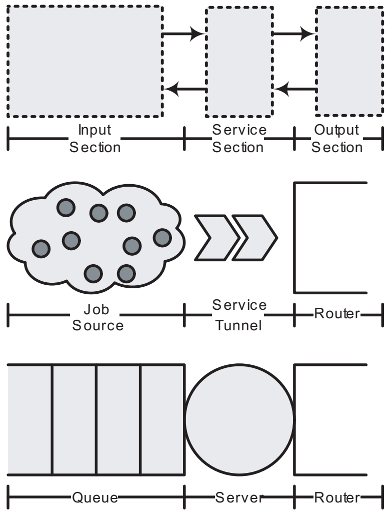


Figure 2. Service center sections.

2 Simulator Architecture

A graphical illustration of simulator architecture is given in Figure 1. The underlying design principle is to obtain a complete separation of the presentation and computational layers using XML. This approach has several benefits. For instance, it is possible to reuse the simulation engine within external applications; further, it simplifies the implementation of different graphical user interfaces. Concerning the last point, the JMT suite offers two simulation interfaces, called JSIM*wiz* and JSIM*graph*, which are discussed later. In the rest of this section, we give an overview of the main functionalities of each architectural layer.

2.1 Discrete-Event Simulation Engine

The core module of the simulation engine is a discrete event calendar [2] that acts as a message broker, dispatching messages to simulation entities. Each significant event, e.g., the arrival of a new job to a queue or the departure of a job after service completion, is represented by a message with a specific identification code. When all current events have been processed, the simulation current time is moved forward to the next instant with an event in the calendar.

In the simulated network, each service center is composed of three entities, called *sections*, as shown in the topmost diagram of Figure 2. The engine demands to the user to specify through the graphical user interface the input, service and output sections to be considered in each service center involved in the simulation. The bottommost diagram shows the instantiation of the three sections in a queueing center.

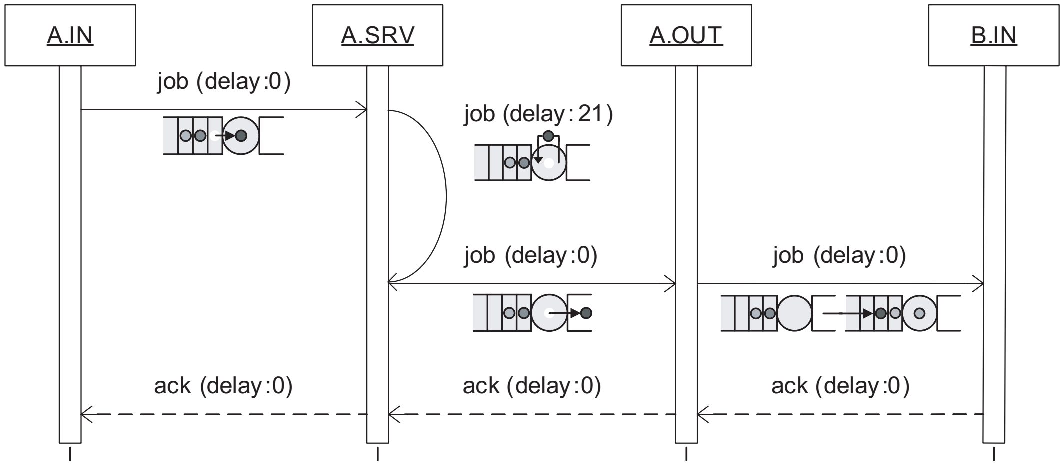


Figure 3. UML sequence diagram of an example message flow. The $(delay : D)$ labels indicate the elapsed simulation time before message delivery.

The queue *input section* is used in this example to receive incoming jobs. It implements the queueing buffer and the queueing discipline, e.g., first-come first-served (FCFS) or last-come first-served (LCFS), which selects the jobs to be processed by the *service section*. The service section simulates the service process, e.g., a single server with user-specified service time distribution. After service completion, jobs are forwarded to a *output section* called *Router*, which sends them to the input section of another service center according to a user-specified routing strategy, e.g., Markovian routing.

As another example of application of this modular structure, we discuss the diagram in the middle of Figure 2, which represents a *source* service center. A source models the job arrival process for *open workloads*, i.e., flows of requests that arrive to the network according to a user-specified inter-arrival time distribution. The input section, called *job source*, is a pool of job objects that are sent, according to the selected distribution, to a fictitious server section, called *service tunnel*. This immediately forwards the job, with no delays, to the Router output section, which finally sends them into the network.

The main benefit of this modular representation of service centers is the ability of reusing the code of the same section in several service centers. For example, the $M/M/1/\infty$ and $M/M/1/k$ queues differ only for the input sections that have different buffer sizes.

2.1.1 Simulation Coordination

The communication between entities is fundamental to coordinate simulations. In order to explain the message flows, we give an illustrative example. Let the service center A be a $M/M/1/\infty$ FCFS queue initially with three enqueued jobs. We denote the input, service and output sections of A respectively by $A.IN$, $A.SRV$, and $A.OUT$. Further, we call *immediate* a message that is processed by the simulator without increasing the simulation time. The example we describe is illustrated in the UML sequence diagram in Figure 3.

Message Flow Example

1. Initially, $A.IN$ has been informed by $A.SRV$ that the server is idle. Then, $A.IN$ selects the next job to be serviced according to the implemented queueing strategy, and sends an immediate notification to $A.SRV$. The job moves to $A.SRV$;

2. $A.SRV$ receives the message and determines the job service time, e.g., $S_A = 21$, according to the user-specified service time distribution. Then, it sends to itself a notification with delay S_A;

3. After S_A simulation time units, $A.SRV$ receives the message and sends a new message to $A.OUT$ to notify the service completion. The job moves to $A.OUT$;

4. $A.OUT$ receives the messages, selects a routing destination, e.g., queue B, and notifies it to $B.IN$;

5. $B.IN$ can either accept the new job or refuse it if the input buffer is full. The latter case is useful for models with blocking or finite capacity regions. If the job is accepted, $B.IN$ sends an immediate acknowledgement to $A.OUT$. The job is now in $B.IN$;

6. $A.OUT$ propagates the acknowledgement to $A.SRV$;

7. $A.SRV$ informs $A.IN$ with the acknowledgement that it is returned to the initial idle state.

Our messaging paradigm, which maintains a complete separation between sections, allows external contributors to develop new sections without the knowledge of other internal implementation details. In other words, in order to specify a new section it is sufficient to implement the correct messaging behavior.

2.1.2 Performance indices

The JMT simulator allows the computation of several performance indices. A comprehensive list with descriptions is given in Table 1. Here we distinguish between *open classes*, that may have an unbounded number of jobs in the network, and *closed classes*, which have a fixed job population. For a class r, we call *reference station* an arbitrarily-chosen service center i that is used for two purposes: first, its class-r throughput is conventionally assumed to be the system throughput of class r; second, for closed classes, a job passage at i denotes the completion of its activity cycle, such that the residence time at a service center $k \neq i$ is the average time spent by the job in the visit(s) to k before returning to i (e.g., average time spent on I/O operations before completion of a certain CPU activity). Each performance index can be estimated for a particular job class or as an aggregated measure over all classes.

In order to collect samples, we use a special data structure referred to as *JobInfoList*. This is associated to each entity of interest, i.e., a section, a service center, or the entire network. The *JobInfoList* logs the arrival time of jobs and then, for each index of interest, feeds a statistical analyzer with the collected data. Each statistical analyzer estimates the requested index using spectral analysis and transient removal methods described in Section 3.

2.1.3 Control of Simulation Experiments

The user has several options for controlling simulation experiments. By default, the simulation is automatically stopped when at least one of the following criteria is satisfied by *every* performance index:

- both confidence interval estimates satisfy user requirements. This is implemented in JSIM through the concept of *maximum relative error*, which is equivalent to "the relative precision of the confidence interval" in [18], and stands for maximum acceptable ratio ϵ between the half-width of the confidence interval estimate (for the requested significance level α) and the estimated mean. For instance, setting $\epsilon = 0.1$ and $\alpha = 0.05$ imposes a simulation stop when the half-width of the estimated 95% confidence interval is no more than 10% of the non-transient sample mean.
- the number of collected samples exceeds a user-specified maximum threshold;
- the elapsed time exceeds a user-specified maximum value;
- user aborts computation.

Performance Index	*Description*
Queue Length	Average number of jobs in the service center (queueing + in service).
Queue Time	Average time spent by a job before being served.
Response Time	Average time spent by a job before leaving the center.
Residence Time	Average time at a center before leaving the network (open classes) or before returning to the reference station (closed classes).
Utilization	Average number of jobs in service.
Throughput	Service center departure rate.
System Response Time	Average time between job arrival and departure from the network (open classes) or between consecutive visits to the reference station (closed classes).
System Throughput	Departure rate from the network (open classes) of from the reference station (closed classes).
Number of jobs	Average number of jobs in the network (open classes).

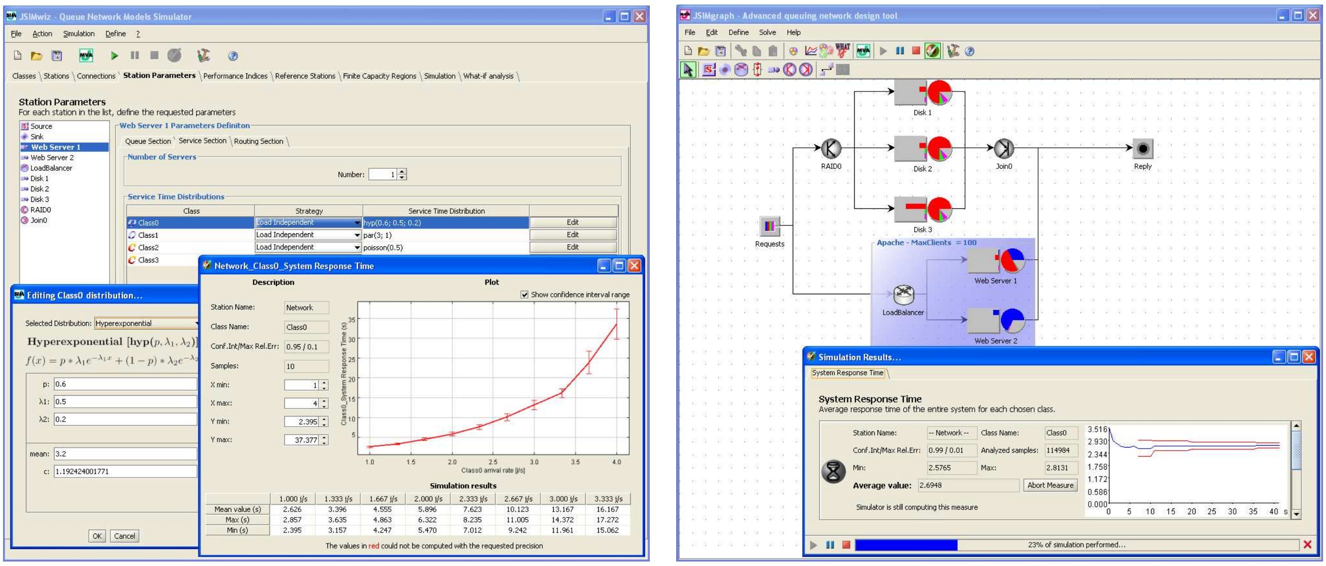
Table 1. Performance Indices.

However, in some cases the user may be interested in performing *long-run simulations*. This feature is useful, e.g., for models with heavy-tail distributions, where large delays occur with low probability, and thus may not be observed if the simulation stops too early. For these cases, JSIM gives the possibility of disabling the automatic stop feature.

Another important feature of the JMT simulator is the support for parametric analyses using *what-if simulations*. In practice, the analyst chooses a control parameter among the number of jobs for closed classes, the percentage of jobs belonging to a certain class, the service times for a service center, or the seed of the random number generator. In this case, JSIM performs a user-specified number of simulations varying the control parameter, and finally plots the results with confidence intervals over the different experiments. We show an example of this simulation feature in the final case study. A screenshot of a what-if plot is given in the right dialog in Figure 4(a), which is commented later.

2.2 XML data layer and Graphical User Interface

The simulator uses an XML data layer to invoke simulation engine, and to obtain simulation current estimates and final results. Furthermore, the XML format is used to save user-specified models: the output file format describes not only service center configurations and network topology, but, if required, also graphical positions of entities in the graphical interface and each Java class used to implement the different sections. This allows the introduction of new features using the Java Reflection API [9] without modifying the simulation engine code.



(a) JSIM*wiz*. (b) JSIM*graph*.

Figure 4. Screenshots of the graphical user interfaces.

The two graphical user interfaces to the simulator engine, called JSIM*wiz* and JSIM*graph*, are built on the top of the XML layer. JSIM*wiz* offers a simple wizard-based interface that guides through model parametrization. JSIM*graph*, instead, gives an easy-to-use graphical layout that enables to draw the network using, e.g., drag-and-drop of predefined service centers. Screenshots of the two interfaces are given in Figure 4. The JSIM*wiz* figure shows the wizard dialogs which let the user specify the structure of each service center. For each class, one can choose between the different available service time distributions, e.g., the hyperexponential distribution. The plot on the right represents the result of a what-if analysis where a class response time is evaluated for different job arrival rates. The JSIM*graph* figure, instead, shows the graphical interface for specifying network structure. During simulation, queues are animated in order to represent the current per-class queue-lengths and utilizations. The blue area delimits a region with finite capacity constraints (which are discussed later). The small dialog window shows the transient behavior of a performance measure estimate, with the estimated confidence intervals that progressively tighten as the simulation goes on.

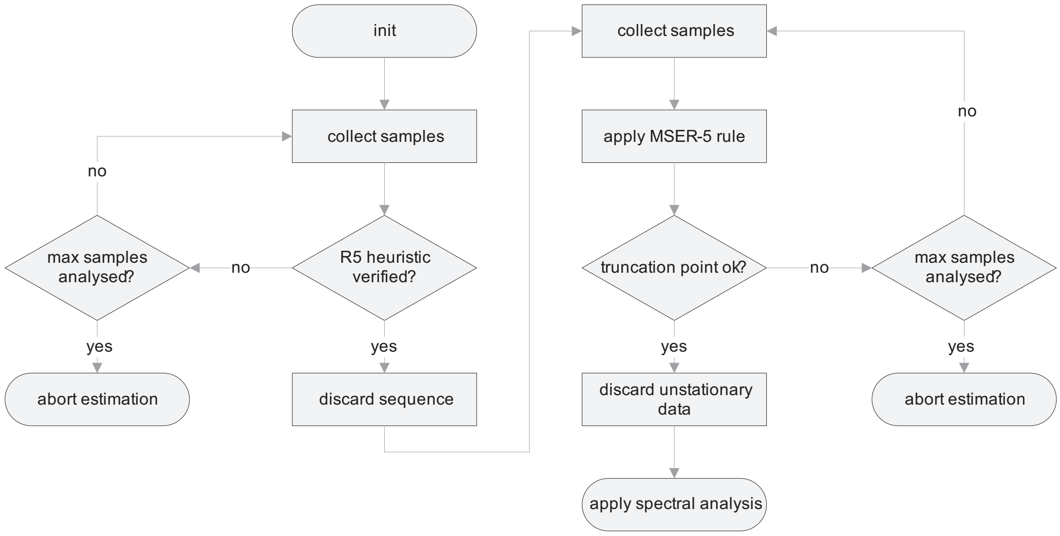


Figure 5. Transient detection and removal flowchart.

3 Statistical Analysis of Simulation Results

Simulation results are analyzed using transient detection and confidence interval estimation algorithms. These techniques are executed online, and can call for a simulation stop if all accuracy requirements are met. Confidence intervals are computed online using spectral methods [12]. However, their effectiveness depends on the stationarity of the sample distribution. In fact, unless long-run simulations are considered, transient effects can significantly affect simulation results. Therefore, discriminating if a group of samples describes transient or steady-state system performance is important to maximize accuracy, since transient data has to be discarded. We implemented transient detection using the R5 heuristic [8] and the MSER-5 stationarity rule [23]. The resulting transient detection and removal flowchart is shown in Figure 5. We now briefly review these methods and the most significant choices performed in their implementation.

3.1 R5 Heuristic

The R5 heuristic [8] detects the initial transient period by checking if the time series of samples crosses its mean value more than k times, where k is a user-specified parameter. We implemented this rule using the indications of Pawlikowski [18]. In order to limit time and space requirements, we periodically check transient termination using a period of 500 samples. If the new samples do not satisfy the crossing condition, 500 new samples are collected before a new check is performed. After matching the stopping criteria, the $R5$ heuristic is complete and the simulator performs a second transient detection using the MSER-5 rule. This is done because we noted that applying several rules empirically reduces the number of wrong detections.

Concerning the choice of the k parameter, Gafarian et al. [10] recommend $k = 25$ for $M/M/1/\infty$ models, while Wilson and Prisker [25] recommend $k = 7$ for $M/M/1/15$ models. We inspected more than 50 simulation models with randomly generated networks including up to 4 load-independent and load-dependent queues. We always considered a single workload. We checked the behavior of utilization, throughput, queue-length and response time metrics for each queue in the network. We observed that the values proposed in the literature did not correctly identified transients on some of the instances. In particular, there exist cases in which $k = 7$ gives an early detection during an unfinished initial transient ramp. Conversely, with the $k = 25$ value, the transient of the utilization metric tends to be very slow, and in some cases even hundreds of thousands of samples are discarded before moving to the MSER-5 rule. We found empirically that the $k = 19$ setting produces good results also on problematic instances.

3.2 MSER-5 Rule

We implemented the MSER-5 rule in the variant with 5 batches discussed in [23]. This is an identification method for the best truncation-point in a data sequence. Initially, we used Schruben's test for this purpose [21], but the results were not always satisfactory, as observed also in [13]. Since the MSER-5 rule is not generally meant for online analyses, but instead works on a fixed data set, we adopted the approach of Robinson [20] which consists in getting online new samples unless the truncation point is detected before the half of the analyzed sample set. Our implementation uses circular lists of 5000 batches (25000 samples), and has constant access and computation times. We observed that using structures with increased size had no benefit on accuracy, but imposed additional computational overheads. In the simulator, the MSER-5 rule is reapplied periodically until the detection of the optimal truncation-point. We set the period equal to the number of samples discarded by the R5 heuristic.

3.3 Spectral analysis

The spectral analysis of Heidelberger and Welch [12] is a stable and computationally efficient method for computing simulation confidence intervals using variable batch sizes and a fixed amount of memory.

In the JMT simulator, the method is run on the non-transient part of the sequence of samples. The spectral analysis is periodically run until the required confidence intervals are found. Nonetheless, it happens quite often that the portion of data immediately available after the MSER-5 rule is already sufficient to compute the required confidence intervals.

4 Non-Product-Form Modelling Features

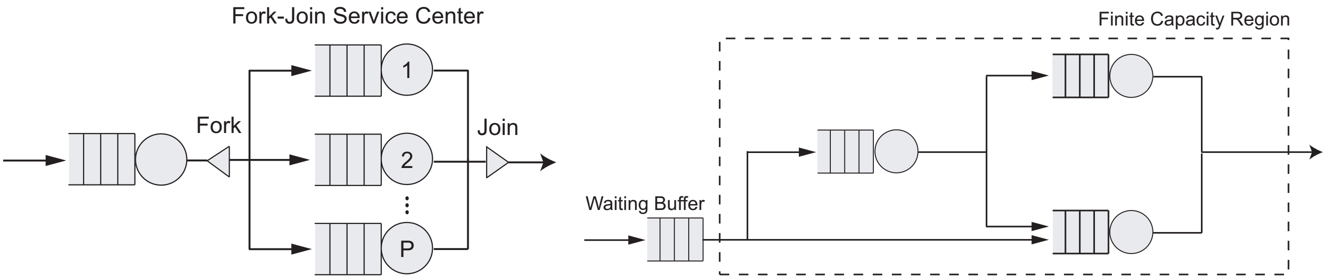
In this section we give an overview of the supported modelling features of the simulation engine. We give a description of significant design choices for each feature.

4.1 Arrival and Service Processes

Nowadays performance models require heavy-tail distributions for arrival and service processes which can be modelled using, e.g., Pareto distributions [7]. Exact results for models representing these effects are seldom available, and typically simulation is required. The JMT simulator supports several distributions for arrival and service processes. These include, among others, Pareto, Gamma, Hyperexponential and Erlang. Random number generation is based on the Mersenne Twister engine [17]. We also implemented a *LogReplayer* distribution that allows to import the random values from an external text file. This may be valuable for replaying in simulations real workload data collected in log files, or to include in simulations unsupported distributions. Load-dependent service processes of arbitrary type can also be specified. These are useful to model devices with service times depending on the current queue-length. Moreover, load-dependence is required for hierarchical modelling of large infrastructures [5] and for parametric analyses [6].

4.2 Fork-Join Service Centers

Fork-join service centers are employed to represent resources that can serve jobs in parallel [11], and are frequently used in storage, parallel and grid system modelling. Fork-join service centers are composed by $P > 1$ queues in parallel. Each time a job arrives to a fork-join service centers, it is split by a *fork* node into P sibling *tasks*. Each of them is assigned to a different parallel queue. After receiving service, jobs synchronize and merge at a *join* node before leaving the service center. Figure 6(a) shows an open network with a fork-join service center. Despite investigated for a long time, approximate solution techniques



(a) An open network with a fork-join service center (b) A finite capacity region

Figure 6. Some non product-form features supported by the JMT simulator.

for networks with fork-join service centers are limited by simplifying assumptions, e.g., exponentiality of service processes. Furthermore, the mean service times of the queues in parallel have to be identical. Clearly, such assumption is unrealistic when the parallel queues represent heterogenous systems having different processing speeds. In these cases, simulation is the only feasible technique.

In the JMT simulator, the user needs only to connect fork and join nodes to any subnetwork. The fork node is implemented as a service center with arbitrarily-chosen input section, with service section having service time always equal to zero (henceforth called a service *tunnel* section), and with a special output section that sends tasks into all outbound connections. Each task is associated to a data structure which identifies the original job arrived at the fork node, and the number P of sibling tasks. The *join* node has a special input section which waits for all P tasks before forwarding to a tunnel section the merged job. The output section of the join is defined by the user. Note that we allow that a job forked at an instant t_2 may be sent out from the join section before a job forked at $t_1 < t_2$. This is important to support discplines like last-come-first-served (LCFS) for the parallel queues.

4.3 Finite Capacity Regions

Models of simultaneous resource possession due to memory or software constraints often require *finite capacity regions* (see, e.g., [16]). These are subnetworks where the number of circulating jobs is constrained. *Shared* constraints impose an upper bound on the allowed number of jobs in the region regardless of their service class. *Dedicated* constraints, instead, limit the number of cycling jobs for a specific class. Jobs arriving to a full region enqueue in a *waiting buffer* outside the region. An illustration is given in Figure 6(b). The presence of the waiting buffer makes it difficult to obtain an analytical solution to models with finite capacity regions. Therefore, only approximation techniques have been developed [14]. However, simulation remains the most important analysis technique in presence of realistic workloads with multiple classes.

Finite capacity regions are implemented as follows. The waiting buffer is a service center with infinite capacity queue and tunnel service section. The output section implements the access control policy according to the user-specified shared and dedicated constraints. Waiting jobs are selected to enter the region according to a FCFS discipline. We point out that region access control is centralized, i.e., all arrivals are routed to the same waiting buffer. Currently, the simulator does not support multiple waiting buffers and nested or intersecting regions. We also remark that, when a region is full, the user can force the simulator to drop arriving jobs. This may be used to represent systems with losses, e.g., the $M/M/1/k$ queue.

4.4 Priority Classes

Priority modeling is required in numerous applications including, e.g., models of packet flows that are differentiated according to quality classes (e.g., [4]), or in the analysis of scheduling rules. Priority models have studied since the early years of performance modelling, and several analytical results of interest are available. Simple formulas for single queueing systems both with preemptive and non-preemptive policies are known [11]. Similarly, there are several analytical methods for queueing networks, such as Sevcik's *shadow server* technique [22], or the approximations for models with class switching and mixed priorities [11]. Typically, it is assumed that classes are served according to a FCFS discipline.

Despite the maturity of the field, there exists a large family of models which cannot be handled by analytical methods. Consider, for instance, emerging "smart" storage systems based on queueing and priority techniques, which requires the joint modeling of fork-join and priority features. Analytical methods cannot be easily, because they cannot handle simultaneous non-product-form features. Hence, simulation is the only available evaluation technique. In the JMT simulator, we support priority disciplines without preemption, namely priority FCFS and priority LCFS. The integration of preemptive methods is currently left as future work.

Routing Discpline	*Description*
RANDOM	Jobs are routed randomly to one of the centers connected to the current one. All routes have the same probability of being selected.
PROBABILITIES	Jobs are routed randomly to one of the centers connected to the current one according to user-specified routing probabilities.
ROUND ROBIN	Jobs are deterministically routed in a cyclic fashion to the centers connected to the current one.
SHORTEST QUEUE LENGTH	Jobs are routed to the center with the smallest current queue length.
SHORTEST RESPONSE TIME	Jobs are routed to the center with the smallest average response time, for the considered job class.
LEAST UTILIZATION	Jobs are routed to the center with the smallest average utilization.
FASTEST SERVICE	Jobs are routed to the center with the shortest average service time for the considered job class.

Table 2. Supported Routing Strategies.

4.5 State-Dependent Routing Strategies

A key assumption of product-form queueing networks is that the behavior of a service center depends only on its current state, and not on the current state of the other queues. Sophisticated generalizations for integrating features depending on network state have been presented in the literature [15, 24], but they are limited to specific classes of models. Thus, in general, one needs simulation to evaluate the impact of state-dependent routing techniques. A list of the routing disciplines available in the JMT simulator is given with descriptions in Table 2.

5 Case Study

In this section we present a case study that illustrates the potentialities of the JMT simulator. We give a comparative study on the asymptotic behavior of product-form versus certain non-product-form networks. Our analysis describes saturation effects that, to the best of our knowledge, have not been pointed out in the queueing network literature so far. While the distribution of jobs for asymptotically large populations is a topic frequently addressed for product-form models (e.g.,[1]), little is known about the asymptotic behavior of non-product-form networks, and especially in the multiclass case. This lack of results can be easily explained by observing that, because exact formulas for describing the steady-state distribution of jobs in non-product-form networks are not known, one may derive asymptotic behaviors only resorting on approximate formulas whose error cannot be bounded. Thus, asymptotic results derived from such approximations would be of little theoretical value. Our contribution is to show some interesting

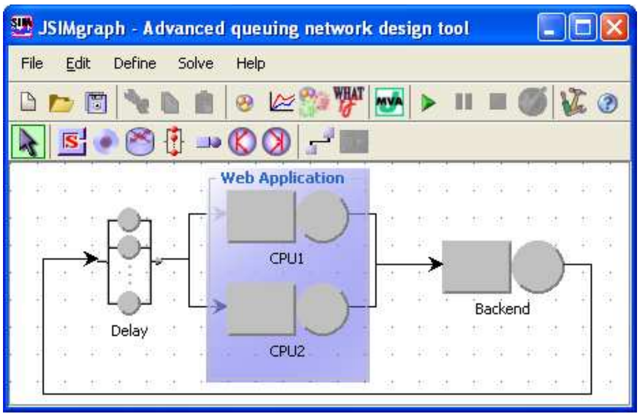


Figure 7. Case study: network topology.

differences in the asymptotic behavior of non-product-form finite capacity networks compared to the product-form one. Our results stress the importance of using simulation to verify the actual accuracy of predictions obtained with approximations based on multiclass product-form modelling.

Network Description We consider an example network composed by three queues and by a delay server, i.e., a service center with infinite processing capacity that models user think times between submissions of requests. The model has the topology shown in Figure 7. We model a bi-processor application server as a finite capacity region composed by two identical queues representing CPU1 and CPU2. The finite capacity regions accounts for the maximum number of incoming connections that can be accepted by the server. We assume that the Web application concludes its activity by placing an order or calling a service from a Backend server. Assuming as negligible the communication overheads after backend service completion, the requests flowing out from the Backend queue immediately return to the clients. For illustration purposes, we consider two *closed* workload classes, i.e., with constant population size, that place significantly different loads on the CPUs and on the backend. The average service time of requests (i.e., response time in absence of queueing) at the different service centers is given in Table 3 together with routing probabilities.

In our experiment, the population sizes for the two workloads are varied while keeping fixed their sum to the constant value $N_1 + N_2 = 1000$ jobs, where N_r stands for the fixed population of the closed class $r = 1, 2$. In other words, denoting by $\beta_1 = N_1/(N_1 + N_2)$ and $\beta_2 = N_2/(N_1 + N_2)$, such that $\beta_1 + \beta_2 = 1$, the percentage of jobs belonging to the classes, our study consists in a parametric analysis of

	Service Time		Routing Probability			
Center	Class-1	Class-2	CPU1	CPU2	Backend	Delay
CPU1	1	10	0.0	0.0	1.0	0.0
CPU2	1	10	0.0	0.0	1.0	0.0
Backend	10	2	0.0	0.0	0.0	1.0
Delay	10	25	0.5	0.5	0.0	0.0

Table 3. Service centers and routing parameters for the case study

the network behavior as a function of β_2. Note that assigning a specific value to β_2 immediately implies $\beta_1 = 1 - \beta_2$. We evaluated the network for $\beta_2 = 0.0, 0.1, 0.2, \ldots, 1.0$ collecting 100.000 samples for each server utilization measure in the following four scenarios:

- *Product-From Case.* The finite capacity region is disabled, and the number of jobs that can enter CPU1 and CPU2 has no constraints. In order to meet product-form assumptions, we assume that all queues have the processor-sharing scheduling rule. The results reported for this case are exact values computed using the MVA algorithm [19].

- *Shared Constraint Case.* In this case we assume that the aggregated number of jobs in CPU1 and CPU2 is less than or equal to 150, regardless of the class of membership. All queues have now the FCFS scheduling rule that usually considered in models with finite capacity regions.

- *Class-1 Dedicated Constraint Case.* This scenario is analogous to the Shared Constraint Case, but the 150-jobs bound is a limit on the class 1 population only.

- *Class-2 Dedicated Constraint Case.* This case is similar to the previous one, but the constraint is now placed on the class-2 population only.

Discussion of Experimental Results. The total mean utilizations of the CPUs and of the Backend servers in the different scenarios are given in Figure 8. It is well-known from the asymptotic theory of [1] that there exists a continuous interval of values for the population mix β_2 where both the CPUs and the Backend centers must reach maximum utilization $U_{CPU1} = U_{CPU2} = U_{Backend} = 100\%$. This interval, called *common saturation sector*, is highlighted in Figure 8(a)-(b), and is slightly smaller than the theoretical ones obtainable with the formulas in [1], which hold for asymptotically large populations, since we obviously need here to consider a *finite* total population[1].

By comparison with the other scenarios, we clearly see that the predictions of multiclass product-form models may be significantly different from those of models where we account also for the finite population constraints. In particular, we found that the shared and class-1 dedicated constraint case *do not exhibit common saturation sectors*. Thus, the CPUs and the Backend center simple interchange, for a certain value of β_2, the role of "bottleneck" center limiting network performance. In particular, for the class-1 dedicated constraint model, the system looks quite unstable for $\beta_2 \in [0.5, 0.7]$, since a minor change of the population mix produces a macroscopic change in the occupation of the CPUs. In fact, the estimated aggregated total population in $CPU1$ and $CPU2$ is ≈ 1 job for $\beta_2 = 0.5$, ≈ 14 jobs for $\beta_2 = 0.6$, and ≈ 843 jobs for $\beta_2 = 0.7$.

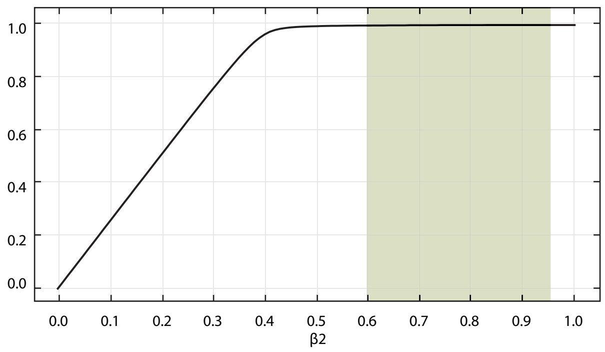
The scenario with the class-2 dedicated constraint, instead, shows in Figure 8(g)-(h) a behavior that is similar to the product-form case, despite the common saturation sector looks much smaller. It is possible to see from simulation results that the main difference of this model with respect to the others is that, for $\beta_2 \in [0.8, 1.0]$, the class-2 jobs saturate both the CPUs and the Backend server. The only other case where there is a large number of jobs inside the CPUs is in the shared constraint case, but in this scenario we have that most of class-2 jobs reside in the waiting buffer of the finite capacity region.

In conclusion, the case study outlines the importance of the detail level used to model the target system. Moreover, we have shown that the asymptotic theory developed for multiclass product-form models may indeed give a reasonable approximation of the actual performance of specific non-product-form models (the class-2 dedicated constraint case), a circumstance that has not been observed in previous work. However, simulations should be carried out to determine which classes of non-product-form networks admit saturation sectors that can be approximated well under product-form assumptions.

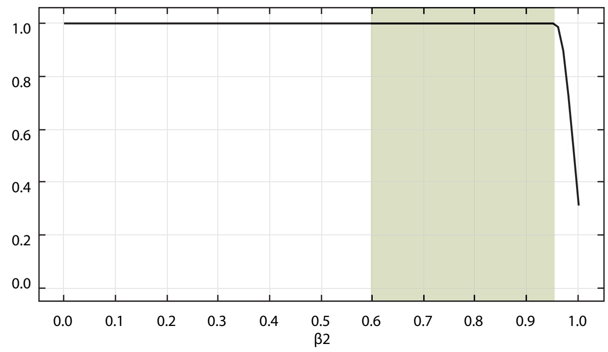
6 Conclusions and Future Work

In this paper, we presented JSIM, a simulator for queueing network models used in capacity planning, tuning and optimization studies. The simulator can be downloaded, with the other tools included in the JMT suite [3], from the project homepage at the URL `http://jmt.sourceforge.net/`. Since the tool is released as a free open source software, the participation to the project of performance analysts and

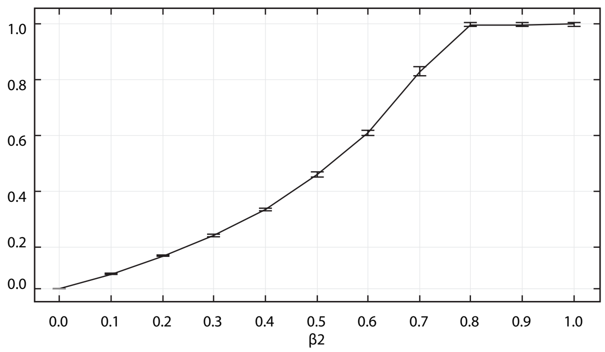
[1]The boundaries of the asymptotic common saturation sector can be determined with the JABA tool of the JMT suite [3].



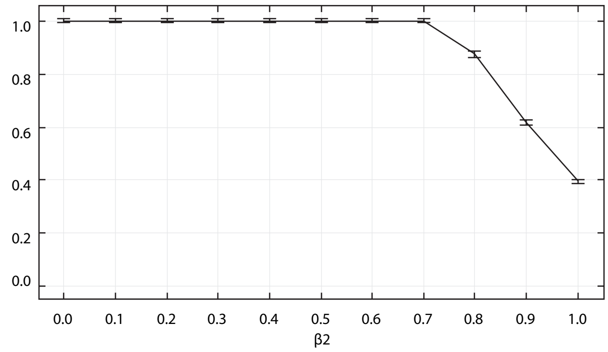
(a) Product-Form Case: U_{CPU1} ($\equiv U_{CPU2}$).



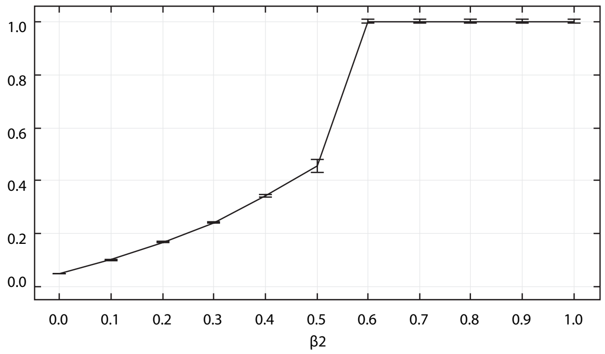
(b) Product-Form Case: $U_{Backend}$.



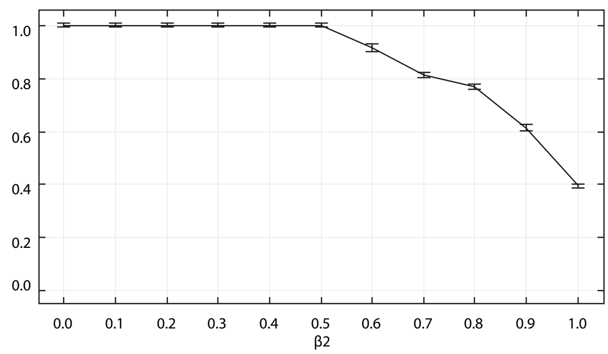
(c) Shared Constraint: U_{CPU1} ($\equiv U_{CPU2}$).



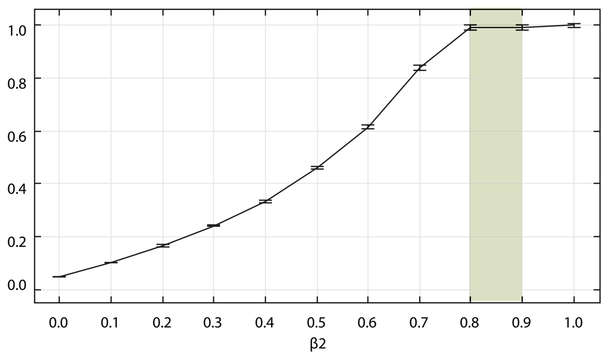
(d) Shared Constraint: $U_{Backend}$.



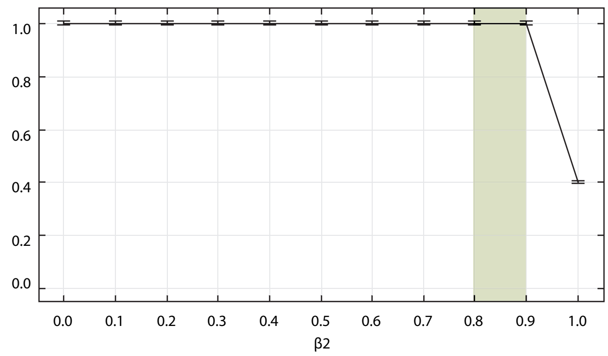
(e) Class-1 Dedicated Constraint: U_{CPU1} ($\equiv U_{CPU2}$).



(f) Class-1 Dedicated Constraint: $U_{Backend}$.



(g) Class-2 Dedicated Constraint: U_{CPU1} ($\equiv U_{CPU2}$).



(h) Class-2 Dedicated Constraint: $U_{Backend}$.

Figure 8. Server utilizations as a function of the population mix β_2 ($\beta_1 = 1 - \beta_2$).

simulation researchers is welcome.

Future work includes the definition of a library of case studies and the integration of new service time distributions.

References

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

[2] J. Banks and J. Carson. *Discrete-Event System Simulation.* Prentice-Hall, Inc., New Jersey, USA, 1984.

[3] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*, pages 119–120, Riverside, US, Sep 2006. IEEE Press.

[4] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Internet Request for Comment RFC 1633, Internet Engineering Task Force, June 1994.

[5] J. C. Browne, K. M. Chandy, R. M. Brown, T. W. Keller, D. F. Towsley, and C. W. Dissly. Hierarchical techniques for the development of realistic models of complex computer systems. In *Proceedings of the IEEE*, volume 63, pages 966–975, 1975.

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

[7] P. Cremonesi and G. Serazzi. End-to-end performance web services. In *Performance 2002.* LCNS 1786, Springer, 2002.

[8] G. S. Fishman. Statistical analysis for queueing simulations. *Management Science, Series A (Theory)*, 20, 3:363–369, 1973.

[9] D. Flanagan. *Java in a Nutshell.* O'Reilly and Associates, Sebastopol, CA, 1997.

[10] A. V. Gafarian, C. J. Ancker, and T. Morisaku. Evaluation of commonly used rules for detecting "steady state" in computer simulation. *Naval Research Logistics Quarterly, vol. 25, 511-530*, 1978.

[11] G.Bolch, S.Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains.* John Wiley and Sons, 1998.

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

[13] J. K. Preston White, M. J. Cobb, and S. C. Spratt. A comparison of five steadystate truncation heuristics for simulation. In *Proc. of the 32nd Winter Simulation Conference, SCS, Orlando, Florida, 2000.*, pages 755–760, 2000.

[14] A. E. Krzesinski and P. Teunissen. Multiclass queueing networks with population constrained subnetworks. In *SIGMETRICS*, pages 128–139, 1985.

[15] S. Lam. Dynamic scaling and growth behavior of queueing network normalization constants. *J.ACM*, 29(2):492–513, 1982.

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

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

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

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

[20] S. Robinson. Automated analysis of simulation output data. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 763–770. Winter Simulation Conference, 2005.

[21] L. Schruben. Detecting initialization bias in simulation output. *Operations Research*, 30:569–590, 1982.

[22] K. Sevcik. Priority scheduling disciplines in queueing network models of computer systems, 1977.

[23] S. C. Spratt. Heuristics for the startup problem, 1998.

[24] D. Towsley. Queueing networks models with state-dependent routing. *J.ACM*, 27(2):323–337, 1980.

[25] J. R. Wilson and A. A. B. Pritsker. A survey of research on the simulation startup problem. *Simulation*, 31, 2:55–58, 1978.