Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 275 (2011) 53–72

www.elsevier.com/locate/entcs

Stochastic Model for QoS Assessment in Multi-tier Web Services

Ricardo M. Czekster[1,6] Paulo Fernandes[2,7] Afonso Sales[3,8]
Thais Webber[4,9] Avelino F. Zorzo[5,10]

Pontifícia Universidade Católica do Rio Grande do Sul
Avenida Ipiranga, 6681 – Prédio 32
90619-900 – Porto Alegre – RS – Brazil

Abstract

Service Level Agreements (SLAs) are used to guarantee quality of service (QoS) between customers and service providers. In an SLA, parties establish a common set of rules and responsibilities. In this paper we propose a practical stochastic modeling of a multi-tier architecture considering SLAs for specific transactions. The model is parameterized with available performance testing data for a real web service, and with a testing environment having unpredictable and unknown external workloads of simultaneous execution. In addition, we present multiple scenarios of external applications impacting on the SLAs in our target architecture. Having a previous knowledge about the average time demanded by some external applications, our model results can provide evidences when the system under test will not respect the agreed-upon SLAs. Finally, we discuss possible model extensions towards further unknown workload characterizations and considerations about application execution profiling.

Keywords: Analytical Modeling, Stochastic Automata Networks, Service Level Agreements, Quality of Service, Performance Evaluation

1 Introduction

Service Level Agreements (SLAs) are contracts between service providers and customers and are used to ensure that an application will deliver a high *quality of*

[1] Corresponding author. The order of authors is merely alphabetical.
[2] Paulo Fernandes receives grants from CNPq-Brazil (PQ 307284/2010-7).
[3] Afonso Sales receives grants from CAPES-Brazil (PNPD 02388/09-0).
[4] Thais Webber receives financial support provided by PUCRS-Petrobras (Conv. 0050.0048664.09.9).
[5] Avelino F. Zorzo receives grants from CNPq-Brazil (PQ 307737/2010-1).
[6] ricardo.czekster@pucrs.br
[7] paulo.fernandes@pucrs.br
[8] afonso.sales@pucrs.br
[9] thais.webber@pucrs.br
[10] avelino.zorzo@pucrs.br

doi:10.1016/j.entcs.2011.09.005

service (QoS) in a timely manner. Ensuring SLAs to execute within healthy environments is important to performance testing since it enables the capacity planning process as well as scalability analysis due to predictable load increase, user growth or based on patterns of future use.

According to *Software Performance Engineering* (SPE) practices [1], performance analysts doing *Performance Testing* are expected to enforce that the *System Under Test* (SUT) is executed in isolation and following a specific test plan according to a well-defined test objective [2,3]. Respecting SPE practices adds a layer of responsibilities, new roles (human resources such as performance analysts and similar capacities) invariably leading to additional costs, and also new risks to software projects. However, it is not always possible to follow SPE guidelines rigorously due to the low priority (or impossibility) that is associated with *pure* performance testing in software projects. In contrast, *Functional Testing* is considered more important for stakeholders and thus, has a higher priority level. In software projects, one must balance the advantages of using SPE to devise responsive and performant applications in respect to the effort to be spent.

Both types of tests are equally important and usually executed in testing environments to comply with distinct objectives. The former is interested in testing the system for failures (according to the taxonomy proposed by Avizienis *et al.* [4]) with respect to the functional aspects of software, whereas the latter works with the non-functional requirements of software, *e.g.*, availability, usability, quality of service or compliance with design specifications, to name a few. Generally, functional testing is applied to validate or to verify software products and performance testing is used to attest quality attributes of systems under particular workloads. Despite the fact that performance testing is often considered after functional testing, this process is equally important because it helps improving overall quality of service.

One should remark the fact that perfect conditions for having a dedicated performance testing environment seldom exists. Problems such as servers belonging to remote locations, security constraints (*e.g.*, static firewall machines), unavailability of the maintenance team for emergency repairs and general updates, general project miscommunications on the test purpose (both for failure discovery and performance), among other factors contribute to degrade web services responsiveness. Web Servers, for instance, for small or large organizations are configured to run multiple types of services, *e.g.*, mail server, multiple instances of *java virtual machines* governed by application server options or external applications under software testing.

To enhance QoS, web services stipulate contracts to be executed within certain predefined amounts of time, *i.e.*, they should respect an SLA to avoid contractual penalties. In this context, we are interested to devise a stochastic model that describes the operation of web services under performance testing subjected to a testing environment running several external applications with unpredictable workload intensities. Note that external applications also share important resources that may deteriorate the response time of the application under performance testing, and sometimes contribute to break SLAs. Our proposed model is parameterized with

data obtained from a performance testing study of a critical application of a large software corporation (omitted here due to signed non-disclosure agreements). Our results demonstrate that characterizing a set of known external workloads intensities it is possible to devise maximum levels of response times given the service demands and verify if the SLA is not respected.

The remainder of this paper is organized as follows. Section 2 addresses software testing and performance testing generally. Our target application is explained on Section 3 as well as its architecture and internal operational details. In Section 4 we discuss stochastic modeling and the formalism known as Stochastic Automata Networks. Section 5 presents our model and an analysis of our results. Finally, in Section 6, we present our final considerations and future works.

2 Software Testing for Performance and Failure Analysis

Software Testing is a crucial task in current *Information Technology* (IT) organizations because it helps ensuring the delivery of high quality products to end customers [5]. There are two aspects of testing that must be considered for every type of software project (small, medium and huge sizes), *i.e.*, failure analysis (or functional software testing) and performance testing. *Functional Software Testing* (FST) is a process that follows a rigid set of rules allowing testers and developers to repeat error conditions and fix issues, hopefully, in a timely manner. The main interest of FST is to ensure if a functionality is producing the expected output for given input.

Performance Testing is an important component of *Software Performance Engineering* (SPE) practices [1]. In contrast to FST, it is directed towards the non-functional aspects of systems, *e.g.*, availability, security, reliability, responsiveness, among other attributes [6]. There are three major objectives to test the performance of an application:

(i) determine the load intensity at which the system fails;

(ii) discover bottlenecks that impairs operation; and

(iii) perform capacity planning [1,2].

The interest is to evaluate the *quality* of the product that is subjected for consideration. To test a given application in terms of performance, it is recommended to follow guidelines and principles according to a methodology. The methodology must relate to a precise objective, for instance, discover the major application bottleneck or assess the network impact on performance indices. It also describes the workload that must be characterized to assess the overall performance.

It is usual to set up the same environment for both FST and Performance Testing. Multi-tier applications, *e.g.*, web servers, application servers and database servers, are deployed in the same environment, causing overloads that diminish the server's original capacity. This excess of execution is attributed to the installation of multiple services with multiple workloads and executing profiles (*e.g.*, CPU or IO

bounded processes). The organizations choose a single server to act as the external accessible server, installing the firewall for security reasons. Therefore, the main reason for building such rigid and underachieving infrastructures is due to security concerns rather than performance. To test the performance of applications, one must be aware that external and unpredictable workloads will be present and eventually disturb the monitoring as all applications are sharing resources (processor time, memory, and so on).

Unfortunately, performance testing is expected to be carried out only by the conclusion of projects rather than simultaneously with the product development. Nevertheless, if development and performance teams somehow managed to work together, as recommended by experts and researchers, maybe the product would complete faster and with more quality [1]. An even more critical aspect to this is that the stakeholders already dispensed large sums of financial incentives to have a high quality software, ready to be used, responsive, and having reasonable resource allocation. Nevertheless, even if it runs in the very last phase of any given project, performance testing must be executed at least until resource usage and application's response time have acceptable levels according to the design specification. More importantly, the application must conform with the predefined SLAs stipulated with the clients.

SLAs are high-level contracts established by stakeholders, *e.g.*, service providers and customers. The main objective to set up an agreement stems from the need to guarantee that quality of service is present and ensured throughout a business relation. Defining SLAs between interested parties helps the understanding of responsibilities and conditions to deliver performant services. SLAs relates to non-functional application testing since it helps devising a compromise in terms of expected quality of service.

When applications are fully tested for both functional and non-functional specifications, they are ready to be deployed in a production environment. There is a huge research effort to characterize the behavior of applications and map to distributions in order to enhance the comprehension of how the system will behave under certain conditions, anticipating and efficiently reacting to problems. Next section discusses related works regarding stochastic models of multi-tier architectures and also some approaches where SLAs are under consideration.

2.1 Related works on modeling SLAs

SLAs were discussed in seminal works regarding *Service Oriented Computing* [7,8,9,10]. Dealing with SLAs and cost models associated with contracts was researched by several authors in a recent past. Ashok *et al.* [11] investigated location-aware SLA contracts and quality of service measurements whereas Liu *et al.* [12] build a cost model to analyze the impact of SLA to maximize profit. Cost models for SLAs are a hot topic for research as it was investigated by Ardagna *et al.* [13], where the authors designed a resource allocation scheduler to study SLAs presenting heuristics as to how maximize the associated profits. SLAs and *Queueing Networks* modeling has also been used before to represent and analyze the effects of service deadlines

in several domain applications. For instance, Abrahao *et al.* [14] devised a self-adaptive SLA capacity planner for Internet applications and Menascé *et al.* [15] discussed policies for managing web related resources for e-commerce servers using a *Customer Behavior Model Graph*. The technique used to extract performance indices was the development of a simulation model of an electronic bookstore as the main example. Ferrari *et al.* [16] used *Queueing Networks* (and also simulation for validation purposes) to model a tiered system comprising an Application Server and a Database Server. This work is closely related to the approach adopted in the present paper, where a simple stochastic model is presented and offer a fast approach to extract performance indices readily available to decision makers.

A more structured approach was presented by Clark *et al.* [17], using a formalism close to a *Process Algebra* known as *Markovian Calculus* as the main mechanism to study SLAs and quality of service. Clark and Gilmore [18] used *Performance Evaluation Process Algebra* (PEPA) to describe a stochastic model and then they converted it to a *Petri Nets* representation for analysis using a special set of compiling tools. The example considered an Automotive Crash Scenario where the deployment of certain car attributes (*e.g.*, the air bag) in combination with actions to be taken was studied as well as the modeling of event durations with uncertainty data (an aspect of special interest to the present work). A *Layered Queueing Network* was proposed in Diao *et al.* [19] to model differentiated services in multi-tier web applications. Once each tier is evaluated, the authors proceed analyzing per-tier concurrency limits and cross-tier interactions. Finally, Sauvé *et al.* [20] proposed a method to reduce costs in IT realities by defining *Service Level Objectives* with examples in the context of multi-tier architectures.

2.2 Discussion

The research presented here distinguishes itself from related works from previous authors on the description and analysis of a stochastic model specially tailored for applications under performance testing subjected to external workloads and observance of SLAs. It is important to map the amount of external influence or unknown workloads to predict if the contract established by the service agreement will not be met. Case the time to complete jobs in the main load balance servers is taking a time that is superior to the threshold computed by our stochastic model, given the external load, it indicates that the SLAs will probably never be met. It is reasonable to consider that external influences are the main cause for the test fail, not because of some bottleneck problem. After some time or if the main server experiences less amount of loads, one could restart the Performance Testing process and resume SUT operation.

In fact, the bottleneck for such types of environments is directed towards the main server acting as a dispatcher that distributes the workload among the remaining servers. As stated before, after the transactions pass this server, they are executed in a clean and dedicated environment, where more reliable usage statistics are enabled. Thus, it is possible to populate a stochastic model with parameters measured after the transaction passed the main server. One clear advantage of such

stochastic models is the possibility of computing the average time necessary to process a given transaction, using the SLA deadline to calculate the available time that can be spent in the main server. An online monitoring tool could keep observing the process that are under execution in the main server and decide whether or not the SLA will be met. If a break of contract is imminent, one could stop the performance test and resume afterwards, to continue searching for application bottlenecks or some other performance testing objective.

3 Overall System Architecture and Operation

This section presents the architecture of the target application. This is a common setup for an IT infrastructure having *Application Servers* and *Database Management Servers* (DBMS). The SUT runs in a shared execution environment with external and unpredictable service demands. All computational resources (*e.g.*, processor, disks, main memory, and network) are common in the environment. The DBMS was configured to behave as a *Storage Area Network* with high memory capacity (superior to one TByte).

The architecture is presented by Figure 1. It consists of a multi-tier architecture having web servers for presentation, application servers for running business logic and database servers for data storage, and a DBMS for query management. Transactions arrive with rate λ in the first web server called PRF01, which sole purpose relies on routing them to one of two other web servers, named PRF02 and PRF03. Both PRF02 and PRF03 consult one of the available DBMS in the system, named DBMS01 and DBMS02. The SUT was implemented in *Java* and uses a pool of threads of fixed capacity and two pools of connections to the DBMS for performance reasons. The main distinction between server PRF01 and its counterparts PRF02 and PRF03 is the fact that this is the only machine accessible externally, *e.g.*, there is also a firewall installed for security concerns. Because PRF01 is the only machine with a valid external IP address, every application is installed, *e.g.*, applications that must submit to software failure analysis before its deployment in the production environment. For this reason, PRF01's Processor Time is usually high due to unscheduled executions that systematically runs in the server in a daily basis.

Once the transaction is successfully processed by PRF01 and routed to PRF02 or PRF03, the execution becomes dedicated for the SUT. All three web servers run on a Pentium IV 2.66GHz machines with 16 GBytes of RAM, dualcore, running WebLogic 8.1 as application server and Windows 2003 Server Edition. The DBMS runs on a Pentium IV 3.2GHz, quadcore, with the same amount of RAM and running Linux RedHat Enterprise Edition and Oracle 10i with dedicated execution. Transactions that disrespects the SLA are stored for counting reasons (for subsequent quality assurance purposes) and exits the system. Ideally, no transaction ought to pass the higher limit of the agreement ensuring high QoS to the system and certainty that every transaction runs below specified thresholds.

Since PRF02 and PRF03 are dedicated, the best possible execution scenario is

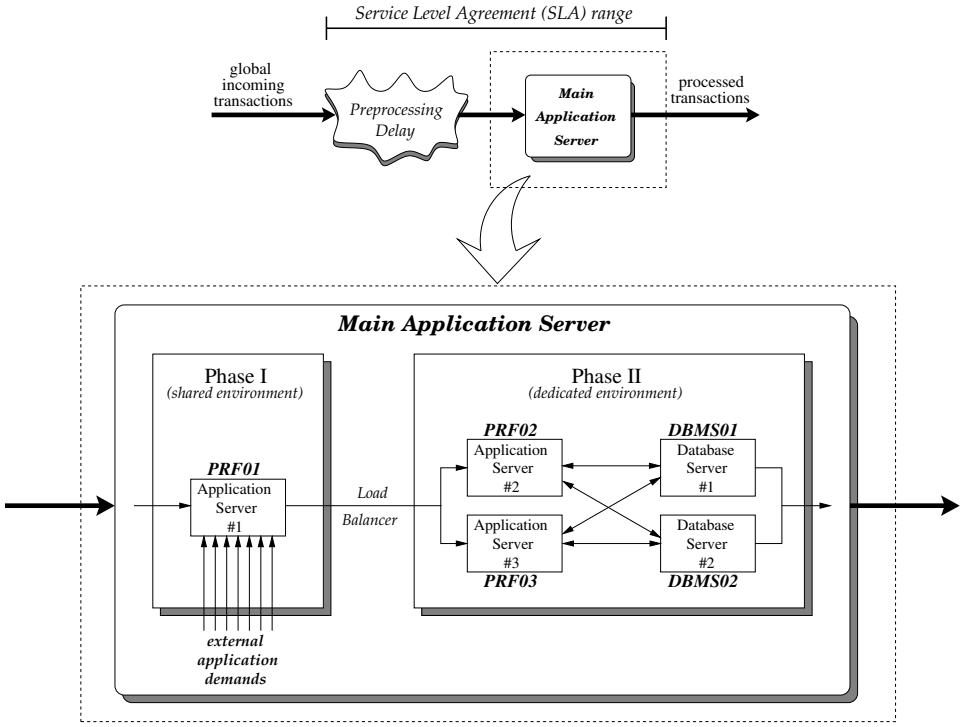


Fig. 1. Main architecture and details of the testing environment.

to enable transactions to bypass PRF01 and directly reach them without loosing time and disputing processing slices with every application present in the first server. However, due to security precautions this configuration is forbidden and PRF02 and PRF03 should remain hidden to the outside world, *e.g.*, configured with only local IP addresses. This decision also makes it very hard to install new software, update the hardware or perform other configurations on the main machine. Moreover, the machine was located outside the main development site, making maintenance very difficult.

The measurements ruled out the network as an application bottleneck since every server is present in the same environment and the monitoring tool was able to attest that the system was robust and healthy for the totality of the test plan execution. The critical aspect of the architecture is towards the PRF01 server since it encompasses every demand for every application that is running. Since the SLA must comply with a set of rules, our main strategy is to divide the system into two distinct blocks and analyze them separately, in order to isolate our problem into more manageable pieces.

The execution profile of the PRF01 machine showed that most of the time, its processor is running the Application Server and creating multiple instances of the *Java Virtual Machine*. Operating Systems naturally consume continuous (sometimes fixed) amounts of processing power. On our case, the servers are installed with the Windows 2003 and its operation does not compromise the access to resources (we are assuming that the machine is well configured and ready to execute

processes). Occasionally, a process is loaded to memory and remains executing indefinitely. This can be mitigated by a soft reset (only a few processes are destroyed) or, more extremely, a hard reset (the whole machine is reinitialized). In production, the same problems related to server instabilities may or may not appear, depending on the execution profile. Since we are testing the application in the most problematic situation, our effort in this work will present a worst case scenario analysis.

3.1 Transaction lifecycle

Transactions in the SUT follow a specific pattern from creation until they are processed and stored. Customers positioned around the globe start the whole process under the form of issues that must be solved, *e.g.*, repairments, questions, operation of devices, and so forth (in Figure 1, it corresponds to the *Preprocessing Delay*). All transactions are sent to a general purpose queue that follows a *First Come First Served* policy that translates them to a format understandable by the *Main Application Server*. The format uses an XML based structure with particular attributes such as general issue descriptions, time stamps, or customer identifier, to name a few. The next step for the transaction is to enter the queue of processes of the main application server, sharing execution and time with external processes from the point of view of the SUT. Once in the queue, operators accessing the system in other global locations are allowed to take ownership of the issue, taking an amount of time to either solve it or relaying it forward (also known as *thinking time*, associated with mouse clicks, click presses or general system operation). If this happens, the ownership is removed for that operator, and the transaction returns to the queue of transactions. Every interaction between operators and the system are accumulated to the overall time to solve the transaction, *i.e.*, under the time limit imposed by the existing SLAs. The time operators take to process transactions are based on averages readily accessible in the application's log files and available for us to use in our stochastic model.

3.2 Research opportunity

The architecture presented earlier (Figure 1) is common to many IT organizations. Evidently, SLAs must be met in production to avoid customer dissatisfaction or even legal problems. Performance Testing techniques are used to verify if SLAs are being respected, however, it can cloud the QoS assessment regarding the amount of time dispensed for each transaction. Due to this fact, applications having SLAs must be tested for performance in an environment that emulates production as closely as possible.

Our objective here is two-folded: firstly, we are profiting that testing environments have external demands just like happens in production, determining the conditions on which SLAs will not be met; and secondly, to enumerate the set of executing conditions that imbalance the time to process every transaction. As a positive side effect, the data available during testing will be used without loss of generality to parameterize our stochastic models.

The approach taken here innovates the way of mixing performance testing and stochastic modeling altogether, offering a way to match distinct problems in Software Performance Engineering: stochastic model parameterization, meeting SLAs in production and profiting of usual external workloads present in testing environments to study its influence on overall time to process transactions.

4 Stochastic Modeling and Stochastic Automata Networks

The objective of our model is to describe an IT infrastructure and compute, in average, the amount of time needed to process a transaction after it is routed by a server that executes important services (*e.g.* load balancing, firewall service, among other). The main characteristic of our environment is that we have a server that is overwhelmed processing various requests, acting as the only externally accessible machine. When transactions reach dedicated servers (just as presented in the architecture - Figure 1), they operate in full capacity, accessing resources located in the vicinity (without additional cost due to network exchanges). There is a deadline that a transaction must respect, defined between customers and service providers, measured since creation until it exits the system. The deadline is defined according to an agreed-upon SLA with the customer, and the main objective is to offer QoS and ensure that once a transaction is present in the system, it will have an associated time to completion, otherwise it will incur in losses (*e.g.* monetary).

4.1 Stochastic Models

For this particular problem, one could model the reality choosing standard *Queueing Networks* (QN) [21,22]. QN is a very powerful formalism to model systems and extract performance indices such as average state permanence probabilities, transient behavior or scalability analysis [23]. It has been used in the past in many applications with significant results, from economic models to distributed computing. However, our reality implements a unique behavior to balance transaction routing that is hard, if not impossible, to model with QNs. The problem under analysis needs a more sophisticated manner to convey the fact that transactions are routed following patterns that must know the apparent load within each server.

For that matter, we took the decision to model the reality with a structured formalism based on *Markov Chains* [24,25] named *Stochastic Automata Networks* (SAN) [26]. The reason to adopt such formalism stems from the fact that SAN allows easy definition of modular structures, also known as automaton, having states and transitions among states according to a list of events (one or more). Events can be defined as one of two types: *local events*, happening in the context of a single automaton; and *synchronizing events*, which needs to act accordingly to other automaton (or more) to be fired. Each event is mapped to a frequency of occurrence, termed a *rate*. Every rate in a SAN model is governed by a *constant* or a *functional* value [27,28]. Constants are based on pure observations of the reality, whereas functional rates are dynamically computed based on the states of other automata's

states.

SAN is used to model realities where parallel and synchronizing behavior (resource sharing, for instance) is expected to occur. It is specially suited for distributed systems but can be applied to several application domains. It has been successfully used to extract useful performance indices of *Global Software Development* realities [29], *Non-Uniform Memory Access* architectures [30], *Master/Slave* parallel computing platforms [31] and *Mobility patterns* [32] to name a few. In a mathematical point of view, SAN uses *Tensor Algebra* properties to compute the probability vector that withholds the performance indices. It basically multiplies a vector by a non-trivial structure called a *Markovian Descriptor*, *i.e.*, a list of small sized matrices that captures the occurrence of every event present in a given model. These matrices are operated with tensor sums for local events and tensor products for synchronizing behavior. One of the greatest advantages of using SAN to represent and solve stochastic models is due to its power of description and efficient storage mechanism.

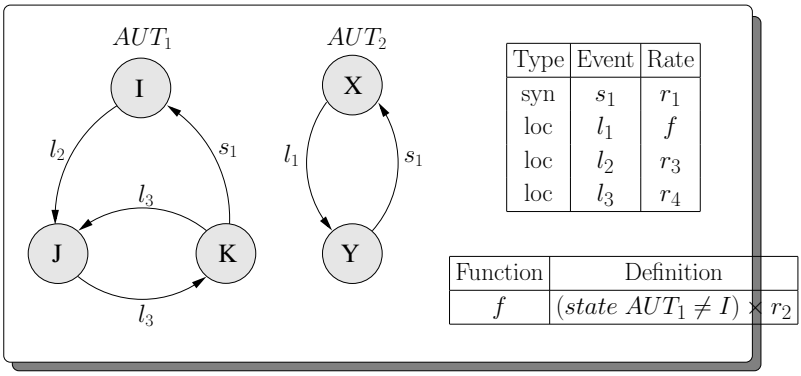


Fig. 2. SAN example having local and synchronizing events with constant and functional rates.

Figure 2 shows an example of a SAN model. It has two automata with states, transitions, events and associated rates. The automata are named AUT_1 and AUT_2, respectively with states I, J, K and X, Y. The cardinality of the state space set for this model is six. It is calculated by the product of local state spaces of each automaton, where the global states are IX, IY, JX, JY, KX, and KY. The transitions among states vary from each automaton, where a corresponding event is set and follow a given rate value. In this example, there are four events, one synchronizing (s_1) and three local (l_1, l_2, l_3), having constant rates and a functional rate for the event l_1. Due to the synchronization defined by the event s_1, the local states of both automata are changed simultaneously (from K to I in AUT_1, and from Y to X in AUT_2). Considering this global change in terms of the underlying Markov chain that is created for the model, the state combination of KY changes to IX with rate r_1. The functional rate verifies if the state of automaton AUT_1 is different of I. If this condition is true, the event can occur with a rate r_2 according to the function definition (f).

5 Numerical Analysis

This section presents our stochastic model and a numerical analysis to inspect the maximum amount of time that could be used on each phase (according to Figure 1). We begin our analysis presenting the results for the dedicated environment defined for Phase II because its behavior is more stable, in Section 5.1. In Section 5.2 we analyze Phase I in detail, where we calculate the response time of the main application server in observance to the SLA. The section also explains the model, its parameters and the numerical results. We finalize the numerical analysis discussing future model extensions in Section 5.3.

To model our reality we will assume a SLA for each transaction of 10 seconds. To our SUT, the SLA total time must be computed taking into account the time since the beginning of the processing until the transaction leaves the system. This time is divided by the time spent to preprocess the transaction (T_{proc}) plus the response time of Phase I (T_{phaseI}) and Phase II ($T_{phaseII}$). The value must be less than 10 seconds, otherwise the SLA will not be met. Our measured data on the preprocessing server accounted a time of 0.02 seconds for every transaction to be properly formatted to an XML definition and other modifications necessary to serve as valid input to the SUT. So, we have $T_{proc} = 0.02$ seconds, and the remaining available time to respect the SLA is equal to 9.98 seconds, distributed between Phases I and II.

5.1 Average time to completion analysis for Phase II

This section presents how we calculate the time needed to process the Phase II, *i.e.*, the total response time for this phase ($T_{phaseII}$). The main idea is to profit from the isolation of the application and database servers in terms of execution. Once the transaction arrives in this phase, it is processed in dedication with full use of available processing power and memory. We are assuming the servers with a high level of stability, *i.e.*, needing system restorations and management operations only occasionally.

Let N be the average queue length, X the throughput, S the service time, R the response time, and U the utilization. To compute the values we used well established QN formulas available in the theory [21]. We used Little's Law ($N = X \times R$), the Utilization Law ($U = X \times S$), average queue length derivation ($N = U/(1-U)$) and Response Time ($R = S/(1-U)$) which was derived from the previous formulas.

According to our performance testing data, the workload intensity is 50 *Transactions Per Second* (TPS), in average, corresponding to the *global incoming transactions* (Figure 1). The chart in Figure 3 shows the average response time for different values of U and S. For instance, if the utilization is equal to 90%, the best response time will be 1.1 seconds when the service time is equal to 0.12 seconds. However, as the time increases, it dramatically changes the response time from 1.1 seconds to the maximum value of 8.8 seconds *per* transaction, assuming the same utilization of 90%. This example shows the worst response time, *i.e.*, for higher utilization values. The figure shows that if the utilization decreases, the response time follows

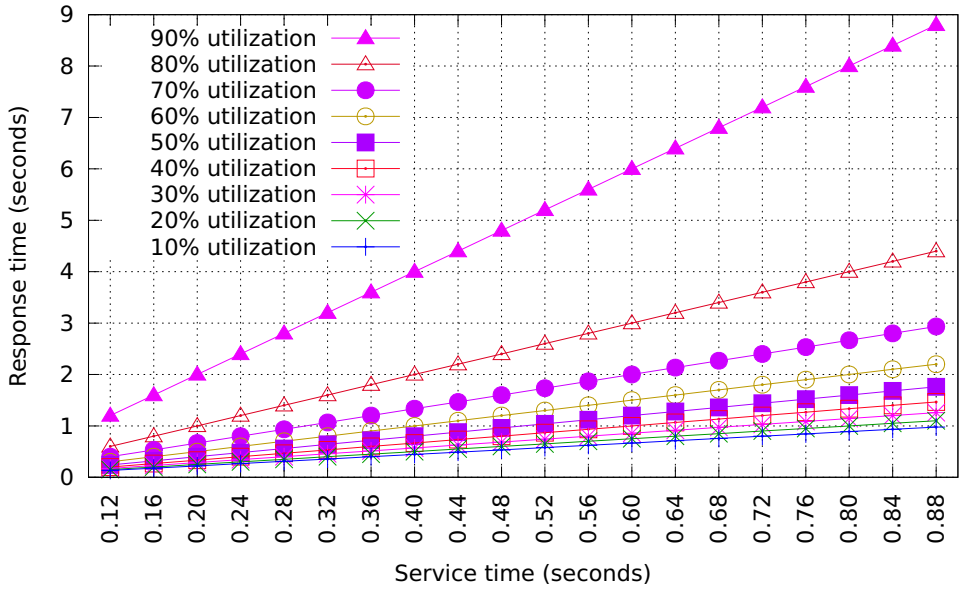


Fig. 3. Maximum Response Time according to utilization choice for Phase II.

the same pattern, *e.g.*, approximately one second for utilizations of 10% or 20%.

In this paper, once the main application server routes transactions forward, we assume that on the worst case scenario a maximum theoretic value for the response time of precisely 8.8 seconds, *i.e.*, we assume the worst case scenario, $T_{phaseII} = 8.8$ seconds. This value will be used to derive the final SLA time needed to complete the transaction.

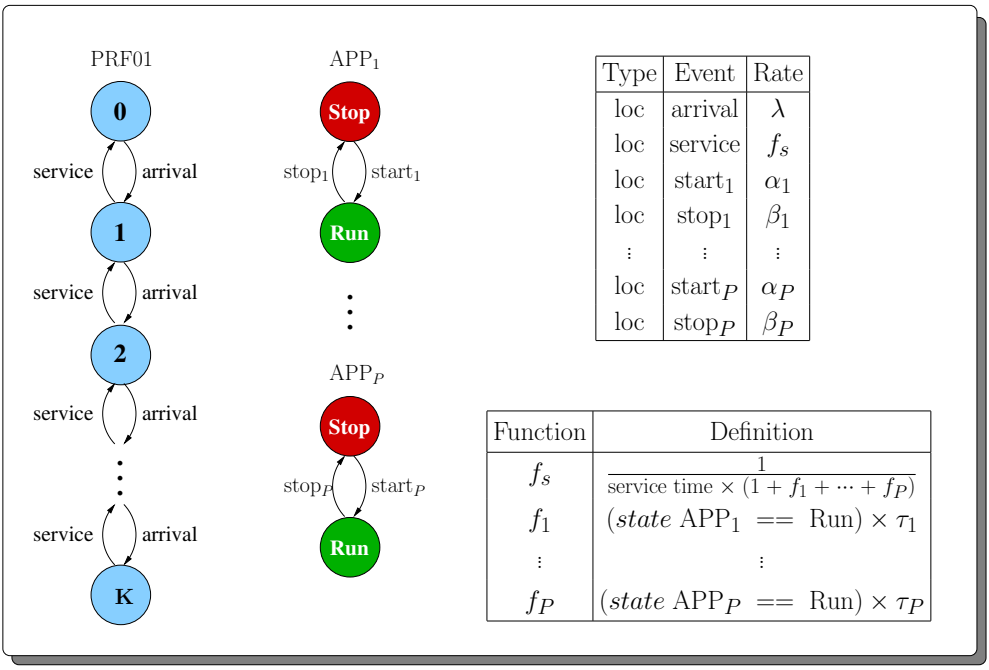
The real existing problem to meet the SLA is set by the amount of time needed to route the transactions in our reality due to external workloads with somewhat unpredictable behavior. If no process is executing, *i.e.*, just after a system reboot, every transaction is routed with full capacity. However, if some other process is running at the same time, the routing is impaired in direct proportion to the intensity of external applications that we must share resources with.

5.2 Maximum value for the response time of Phase I

We proceed our analysis on studying the response time for Phase I (T_{phaseI}). The model presented here was described by a SAN instead of a QN because our reality has several different behaviors that must be captured in terms of external influence modeling. Next, in Section 5.2.1 we explain our stochastic model in detail and its results are presented in Section 5.2.2.

5.2.1 Proposed model

We choose to model the transaction queue and several external processes that are either stopped or running within the application server. A depiction of our SAN based stochastic model is presented in Figure 4.



Type	Event	Rate
loc	arrival	λ
loc	service	f_s
loc	start_1	α_1
loc	stop_1	β_1
⋮	⋮	⋮
loc	start_P	α_P
loc	stop_P	β_P

Function	Definition
f_s	$\frac{1}{\text{service time} \times (1 + f_1 + \cdots + f_P)}$
f_1	$(state\ \text{APP}_1 == \text{Run}) \times \tau_1$
⋮	⋮
f_P	$(state\ \text{APP}_P == \text{Run}) \times \tau_P$

Fig. 4. SAN model proposed for Phase I.

The number of automata in the model varies, depending to the desired number of external processes that will be created. We defined the main automaton, named PRF01, that represents the transaction queue to be processed by the main application server. This queue has $K + 1$ positions where K is the queue capacity, representing that at least K transactions could be saved to be served by PRF01. Event *arrival* indicates the number of incoming transactions that arrives in the system, with constant rate λ. Event *service* sets the frequency on which transactions are processed, and it has a functional rate defined by f_s. The service time of this queue is influenced by the number of existing external applications, *i.e.*, its performance depends on the amount of work that needs to be processed by the main server.

Complementing the first automaton, we have created P other automata called APP_i where $i \in 1..P$ of two states. P represents the amount of different types of applications present in the system for execution, having distinct times and patterns of execution (more CPU bound or more IO bound processes, for instance). Each application type has a *weight* or a *proportion* of influence applied to the service time of the main server (defined by τ_i), slowing it down in this case. An automaton APP has two states: *Stop*, indicating that the application has been stopped; and *Run* otherwise. There are two events present, named *start* and *stop* with rates respectively equal to α and β. When either event is triggered, the application starts or stops its execution under different rates depending on the case.

As a final remark, we are abstracting the fact that the SUT has two cores of execution, modeling our main queue with a single server to process every transaction.

Our proposed model focus on the external execution aspect of applications, and its influence on the QoS requirements that must be enforced by the SLA.

5.2.2 Model results

Stochastic Automata Networks are solved using a software package called *GTAexpress* [33]. On the core of its implementation, the tool is equipped with the latest version of the *Vector-Descriptor Product* algorithm called Split [34], which depending on the model to be solved, accelerates convergence and allows modelers to assess performance faster. GTAexpress is also under new developments such as the addition of *Multi-valued Decision Diagrams* [35] and *Perfect Simulation* [36] techniques for state spaces that exceeds 65 million of states, *i.e.*, the current limit of the tool for machines with 4 GBytes of RAM.

To sum up the time spent until now, we have $T_{proc} = 0.02$ seconds spent in preprocessing, $T_{phaseII} = 8.8$ seconds to run the business logic of our application in a dedicated environment at most and 10 seconds to comply with the SLA. So, it remains 1.18 seconds to use in Phase I and still guarantee that the system will meet the time requirements, *i.e.*, T_{phaseI} must be inferior to 1.18 seconds.

We conducted several performance tests and monitored important resources such as *Processor Time* and *Available Memory* (only to name a few) for all application servers and the database servers. We used *LoadRunner* as the main tool to perform the load testing procedure on our SUT, where the verified throughput (X) was 50 TPS. Following a methodology, our performance testing objective was to verify that at least 90% of the total number of transactions were being met by the SLA.

We have measured that the main server (PRF01) had, in average, an utilization of 75% for the workload proposed in the performance test plan. Having the throughput ($X = 50$ TPS) and the utilization ($U = 0.75$), we used the *Utilization Law* to compute the service time S for this application. The value that we have obtained was 0.015 seconds to execute a transaction. It is worth mentioning that we ascertain that no external execution was present in PRF01 in order to determine the utilization, *i.e.*, the service time was not contaminated by external processes (when none external applications are present in PRF01, then all τ are equal to zero).

Using this service time of 0.015 seconds in function f_s of our model, we have a service rate of 66.67 TPS for an arrival rate of 50 TPS (assuming balance in the system, *i.e.*, the arrival rate equals the throughput). The average number of processes in the queue is $N = 3$ and the response time is $R = 0.06$ seconds, which is quite low and would ever respect the remaining possible time to ensure the SLA, calculated in 1.18 seconds (an order of almost 20 times as low as the possible value). However, this is also quite unrealistic to our testing environment, where it is almost impossible to test the system without interference. It is only natural to have multiple sources of interference, impairing the service time and thus increasing the response time.

To estimate what is this influence in the overall performance of our SUT we need to assign external workload profiles from different application types that are executed in the machine (also termed *application profiles*). We will assume $P = 5$,

e.g., five types of different applications such as scientific software, service daemons, firewall server, or other web services. We also choose to define fixed applications that corresponds to the machines' native operational system that is running continuously and constantly using computational resources (or any other application that endlessly runs within the testing environment).

Table 1 shows the average time that each application is under execution on PRF01 in a single day. Note that APP_5 represents the operating system of the machine, and we measured that, in the worst case in a one day time frame, at least one 10 minute full reboot happens. These data were obtained through average usage monitoring that was done prior and it was run according to a measurement methodology adapted from [2]. We used the usual performance counters (*%Processor Time*, *Available Memory*) for the Windows machines and a combination of monitoring scripts (e.g. *vmstat, iostat, etc.*) for the ones executing Linux. We have created several scripts to convert log files from the measurements information. We combine this file with the one created by the Windows machines, which allowed us to analyze the average values for each time frame of each experiment.

Table 1
Running time estimates for each application in a single day of operation.

	APP_1	APP_2	APP_3	APP_4	APP_5
Running time	18 hours	8 hours	20 hours	5 hours	23:50 hours

Table 2 shows the τ configured for each application. It basically states the weight associated to every external application that influences the service time of the main server. This factor is important to our model since it relates to the amount of external work that must be done by the server, usually having variable processing demands.

Table 2
Different execution scenarios based on application profiles.

Scenario	τ_1	τ_2	τ_3	τ_4	τ_5	Total
1	17%	13%	6%	4%	5%	45%
2	4%	2%	33%	1%	5%	45%
3	9%	16%	6%	14%	5%	50%
4	11%	3%	27%	4%	5%	50%
5	3%	19%	1%	32%	5%	60%
6	19%	4%	21%	11%	5%	60%
7	2%	11%	3%	49%	5%	70%
8	22%	17%	14%	12%	5%	70%
9	3%	12%	8%	52%	5%	80%
10	33%	2%	37%	3%	5%	80%
11	6%	8%	4%	67%	5%	90%
12	36%	28%	17%	4%	5%	90%

Next, in Table 3 we show our main performance indices in terms of the utilization, the average number of transactions in the queue, throughput and response time for the scenarios explained in Table 2.

Table 3
Performance indices in terms of N, X, U and R in relation to the proposed scenarios.

Scenario	U (%)	N (Trans.)	X (TPS)	R (seconds)
1	95.55	21.48	50.00	0.43
2	99.27	135.50	49.32	2.75
3	93.44	14.25	50.00	0.28
4	99.37	157.63	48.83	3.23
5	90.78	9.85	50.00	0.20
6	99.43	175.67	47.91	3.67
7	91.20	10.36	50.00	0.21
8	99.44	177.44	47.70	3.72
9	94.96	18.85	50.00	0.38
10	99.49	194.25	41.94	4.63
11	94.73	17.99	50.00	0.36
12	99.48	192.32	43.54	4.42

The results shows that the response time extracted from the model is varied and directly proportional to the workload intensity of the applications. However, it is interesting to verify that not necessarily decreased load levels impacts on equally decreased response times. For instance, in Scenarios 1 and 2, the total time added in the application's service time is up to 45%. The time spent by the external applications to execute presents different response times for every scenario. As an example of this behavior, Scenario 1 had a response time of 0.43 seconds, which is less than the available time to meet the SLA (estimated as 0.7 seconds), contrary to what is observed by the results showed by Scenario 2 that spent 2.75 seconds. The increase in time was due to the variation in terms of average queue length as well as a throughput decrease that happened accordingly. For these cases the arrival rate is greater than the service rate, bringing congestion, causing delays and queueing in the system. The same situation is verified in Scenarios $4, 6, 8$ and 12, where the workload has impacted the overall response time under different circumstances.

For the rest of our analyzed case scenarios where we varied the workload intensity, we were able to compute response times within the available SLA time, *i.e.*, cases where the external application influence over service time enables meeting the time constraint. Our stochastic model was able to help us understand the fact that depending on the application execution profile we can anticipate if the response time for our SUT will fall under the SLA threshold. This is very interesting because the model could forecast, for instance, the unfavorable or advantageous conditions present on the environment to allow the execution of performance testing. Case such conditions are causing external delays, it is safe to conclude that it is not our

SUT that have a problem, maybe our executing environment is not yet suitable to perform accordingly.

5.3 Model extensions discussion

Our model could be extended to compose other behavior arising in multi-tier architectures. For instance, one could consider studying the effect on end-to-end response time when the external applications are observed to have workload burstiness, *i.e.*, several transactions arriving almost instantaneously and long periods of idle time. We are aware of an efficient overload management in multi-tier environments having bursty workloads recently studied by Lu *et al.* [37]. In our model, this issue can be adapted by setting the parameters Stop-Run of the external loads accordingly. The analysis would show how the burstiness affects performance and SLA requirements.

Another interesting aspect to be further inspected concerns extracting performance indices when the external applications are executing stress testing within several contexts. During stress test, the limits for a given application are tested and usually it increases the amount of existing resource sharing and also the rate of failures (or even faults). Our model could define different application testing profiles to inspect its relation to the performance testing of our SUT.

Since we have half of our system under dedicated operation, we could adapt our environment to receive incoming transactions from external workloads as well. Then, we could assess and analyze the sharing resources (in this case, every server have shared execution) and the impact to meet the contract stipulated by the SLA.

6 Final Considerations

The present work proposed stochastic models and SLA assurance applied in the context of multi-tier web services with external workloads. There is an increasing interest for practical applications of stochastic modeling for performance evaluation. We modeled the reality and performed a worst case scenario analysis to verify if the contracts between users and service providers were being respected. Our performance indices computations presented means to decide if the SLA would be executed below its deadline and the impact of external workloads in this time.

The IT multi-tier infrastructure modeled in this work shares resemblance with many real web services deployed throughout the world. Many organizations use SLAs in their business operations to ensure high quality of service to their customers. The present paper proposed a stochastic model to represent and evaluate such architectures and studied the influence of external workloads to meet SLAs. The model was parameterized with data obtained from a testing environment, where performance testing processes were being executed with unpredictable workloads caused by the presence of external services. As mentioned before, a good side effect of this is that the same model could be adapted and parameterized with data obtained from a production environment. This will enable the verification of SLAs in customer side applications, assessing overall quality of service that is being provided for.

As future works, we consider applying the same model ideas for busy environments with more intense workload variation. Such work may demand a deeper analysis of stochastic distributions leading to the extensions of the model to consider more complex distributions. One option is the inclusion of phase-type transitions to approach some non-exponential phenomena like timeouts. In this case, some prior studies on phase-type representation for SAN formalism [38] could be used.

In another interesting future work we could also develop and install a daemon with both a stochastic model and a numerical solver (*e.g.*, GTAexpress [33] or similar) in the application servers of interest to monitor the execution and self-parameterize a model with the obtained data to decide whether or not the SLA will be met in a timely fashion. This research will allow decision makers to stop the execution of some external workloads or to control the incidence of external applications that are allowed to run. This will undoubtedly help to assure a higher level of availability to the web service, avoiding economical losses and ensuring user satisfaction.

Acknowledgments

The authors thank Dr. Alberto Avritzer from Siemens Corporate Research (SCR), Princeton/NJ/USA, for insightful discussions that led to the elaboration of this paper and the Performance Testing Research Team, a collaboration between the Faculty of Informatics at PUCRS and Dell Brazil. This work has also support from FAPERGS/CNPq.

References

[1] C. U. Smith, L. G. Williams, Performance Solutions: a practical guide to creating responsive, scalable software, Addison-Wesley, Boston, MA, USA, 2002.

[2] D. A. Menascé, V. A. F. Almeida, Capacity Planning for Web Services: metrics, models, and methods, Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[3] D. A. Menascé, L. W. Dowdy, V. A. F. Almeida, Performance by design: computer capacity planning by example, Prentice Hall, Upper Saddle River, NJ, USA, 2004.

[4] A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE transactions on dependable and secure computing 1 (1) (2004) 11–33.

[5] G. J. Myers, C. Sandler, The Art of Software Testing, John Wiley & Sons, 2004.

[6] J. D. Musa, A. Iannino, K. Okumoto, Software reliability: measurement, prediction, application, McGraw-Hill, Inc., New York, NY, USA, 1987.

[7] M. P. Papazoglou, Service-oriented computing: concepts, characteristics and directions, in: Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 2003), 2003, pp. 3–12.

[8] M. P. Papazoglou, D. Georgakopoulos, Service-oriented computing, Communications of the ACM 46 (10) (2003) 25–28.

[9] M. N. Huhns, M. P. Singh, Service-oriented computing: key concepts and principles, IEEE Internet Computing 9 (1) (2005) 75–81.

[10] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: State of the art and research challenges, Computer 40 (11) (2007) 38–45.

[11] A. Argent-Katwala, J. Bradley, A. Clark, S. Gilmore, Location-aware quality of service measurements for service-level agreements, in: Proceedings of the 3rd Conference on Trustworthy Global Computing (TGC'07), Springer-Verlag, Berlin, Heidelberg, 2008, pp. 222–239.

[12] Z. Liu, M. Squillante, J. L. Wolf, On maximizing service-level-agreement profits, in: Proceedings of the 3rd ACM Conference on Electronic Commerce (EC'01), ACM, New York, NY, USA, 2001, pp. 213–223.

[13] D. Ardagna, M. Trubian, L. Zhang, SLA based resource allocation policies in autonomic environments, Journal of Parallel and Distributed Computing 67 (3) (2007) 259–270.

[14] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, F. Safai, Self-Adaptive SLA-Driven Capacity Management for Internet Services, in: 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), 2006, pp. 557–568.

[15] D. A. Menascé, V. A. F. Almeida, R. Fonseca, M. A. Mendes, Business-oriented resource management policies for e-commerce servers, Performance Evaluation 42 (2-3) (2000) 223–239.

[16] G. Ferrari, P. Ezhilchelvan, I. Mitrani, Performance Modeling and Evaluation of E-Business Systems, in: Proceedings of the 39th Annual Simulation Symposium (ANSS'06), IEEE Computer Society, Washington, DC, USA, 2006, pp. 135–142.

[17] A. Clark, S. Gilmore, M. Tribastone, Service-Level Agreements for Service-Oriented Computing, in: A. Corradini, U. Montanari (Eds.), Proceedings of the 19th International Workshop (WADT 2008), Vol. 5486 of Lecture Notes in Computer Science, Springer-Verlag, Pisa, Italy, 2009, pp. 21–36.

[18] A. Clark, S. Gilmore, Evaluating quality of service for service level agreements, in: Proceedings of the 11th international workshop, FMICS 2006 and 5th international workshop, PDMC conference on Formal methods: Applications and technology, FMICS'06/PDMC'06, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 181–194.

[19] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, M. Surendra, A. Tantawi, Modeling Differentiated Services of Multi-Tier Web Applications, in: 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2006), 2006, pp. 314–326.

[20] J. Sauvé, F. Marques, A. Moura, M. Sampaio, J. Jornada, E. Radziuk, SLA Design from a Business Perspective, in: J. Schonwalder, J. Serrat (Eds.), Ambient Networks, Vol. 3775 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 72–83.

[21] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Quantitative system performance: computer system analysis using queueing network models, Prentice Hall, Upper Saddle River, NJ, USA, 1984.

[22] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, John Wiley & Sons, Inc., New York, NY, USA, 1998.

[23] A. Avritzer, A. Lima, An Empirical Approach for the Assessment of Scheduling Risk in A Large Globally Distributed Industrial Software Project, in: Proceedings of the 4th International Conference on Global Software Engineering (ICGSE'09), IEEE Computer Society, Washington, DC, USA, 2009, pp. 341–346.

[24] J. R. Norris, Markov Chains, Cambridge University Press, New York, USA, 1998.

[25] W. J. Stewart, Introduction to the numerical solution of Markov chains, Princeton University Press, 1994.

[26] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, ACM SIGMETRICS Performance Evaluation Review 13 (2) (1985) 147–154.

[27] P. Fernandes, B. Plateau, W. J. Stewart, Efficient descriptor-vector multiplication in Stochastic Automata Networks, Journal of the ACM 45 (3) (1998) 381–414.

[28] L. Brenner, P. Fernandes, A. Sales, The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations, International Journal of Simulation: Systems, Science & Technology 6 (3-4) (2005) 52–60.

[29] R. M. Czekster, P. Fernandes, A. Sales, T. Webber, Analytical Modeling of Software Development Teams in Globally Distributed Projects, in: Proceedings of the International Conference on Global Software Engineering (ICGSE'10), IEEE Computer Society, 2010, pp. 287–296.

[30] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer, A. F. Zorzo, Analytical Modeling for Operating System Schedulers on NUMA Systems, Electronic Notes in Theoretical Computer Science (ENTCS) 151 (3) (2006) 131–149.

[31] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, A. Sales, Performance Models for Master/Slave Parallel Programs, Electronic Notes In Theoretical Computer Science (ENTCS) 128 (4) (2005) 101–121.

[32] F. L. Dotti, P. Fernandes, C. M. Nunes, Structured Markovian models for discrete spatial mobile node distribution, Journal of the Brazilian Computer Society 17 (2011) 31–52.

[33] R. M. Czekster, P. Fernandes, T. Webber, GTA express - A Software Package to Handle Kronecker Descriptors, in: Proceedings of the 6th International Conference on Quantitative Evaluation of SysTems (QEST'09), IEEE Computer Society, Budapest, Hungary, 2009, pp. 281–282.

[34] R. M. Czekster, P. Fernandes, J.-M. Vincent, T. Webber, Split: a flexible and efficient algorithm to vector-descriptor product, in: International Conference on Performance Evaluation Methodologies and tools (ValueTools'07), ACM International Conferences Proceedings Series, ACM Press, Nantes, France, 2007, p. 83.

[35] A. Sales, B. Plateau, Reachable state space generation for structured models which use functional transitions, in: Proceedings of the 6th International Conference on Quantitative Evaluation of SysTems (QEST'09), IEEE Computer Society, Budapest, Hungary, 2009, pp. 269–278.

[36] P. Fernandes, J.-M. Vincent, T. Webber, Perfect Simulation of Stochastic Automata Networks, in: K. Al-Begain, A. Heindl, M. Telek (Eds.), Proceedings of 15th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'08), Vol. 5055 of Lecture Notes in Computer Science, Springer-Verlag, Nicosia, Cyprus, 2008, pp. 249–263.

[37] L. Lu, L. Cherkasova, V. de Nitto, N. Mi, E. Smirni, AWAIT: Efficient Overload Management for Busy Multi-tier Web Services under Bursty Workloads, in: B. Benatallah, F. Casati, G. Kappel, G. Rossi (Eds.), 10th International Conference on Web Engineering (ICWE'10), Vol. 6189 of Lecture Notes in Computer Science, Vienna, Austria, 2010, pp. 81–97.

[38] I. Sbeity, L. Brenner, B. Plateau, W. J. Stewart, Phase-type distributions in stochastic automata networks, European Journal of Operational Research 186 (3) (2008) 1008–1028.