Performance Analysis of Computing Servers — A Case Study Exploiting a New GSPN Semantics

Joost-Pieter Katoen[1], Thomas Noll[1], Thomas Santen[2], Dirk Seifert[2], and Hao Wu[1,⋆]

[1] Software Modeling and Verification Group
RWTH Aachen University, Germany
{katoen,noll,hao.wu}@cs.rwth-aachen.de
[2] Advanced Technology Labs (ATL) Europe
Microsoft Research, Aachen, Germany
{thomas.santen,dirk.seifert}@microsoft.com

Abstract. Generalised Stochastic Petri Nets (GSPNs) are a widely used modeling formalism in the field of performance and dependability analysis. Their semantics and analysis is restricted to "well-defined", i.e., confusion-free, nets. Recently, a new GSPN semantics has been defined that covers confused nets and for confusion-free nets is equivalent to the existing GSPN semantics. The key is the usage of a non-deterministic extension of CTMCs. A simple GSPN semantics results, but the question remains what kind of quantitative properties can be obtained from such expressive models. To that end, this paper studies several performance aspects of a GSPN that models a server system providing computing services so as to host the applications of diverse customers ("infrastructure as a service"). Employing this model with different parameter settings, we perform various analyses using the MaMa tool chain that supports the new GSPN semantics. We analyse the sensitivity of the GSPN model w.r.t. its major parameters –processing failure and machine suspension probabilities– by exploiting the native support of non-determinism. The case study shows that a wide range of performance metrics can still be obtained using the new semantics, albeit at the price of requiring more resources (in particular, computation time).

Keywords: Computing Services, Model-Based Analysis, Generalized Stochastic Petri Nets, Markov Automata.

1 Introduction

Goal of the paper. The goal of this paper is to introduce an industrial case study that demonstrates the application of a newly developed semantics [1,2] of Generalised Stochastic Petri Nets (GSPN). This semantics covers basically every GSPN, in particular those which exhibit non-determinism that, e.g., is due to the presence of confusion. This paper presents a simple, abstract GSPN model of a

⋆ Supported by ATL, the RWTH Aachen University Seed Fund, and the EU FP7 SENSATION project.

K. Fischbach and U.R. Krieger (Eds.): MMB & DFT 2014, LNCS 8376, pp. 57–72, 2014.

computing service, and analyses its performance properties from both a customer and a provider point of view. We do so by exploiting the MAMA tool chain [3] which supports the new GSPN semantics and relies on various algorithms from Markov decision theory.

Computing services. The growing cost of installing and managing computer systems leads to outsourcing of computing services to providers. We use the term computing services to refer to any kind of server system providing computing resources such as physical or virtual machines in order to host the applications of diverse customers. Examples of such systems are hosting centers that provide out-sourced computing capabilities to customers. If these resources are accessible via Internet, one speaks about cloud computing. In particular, the most basic form of cloud service, the so-called infrastructure-as-a-service model, conforms to this setting, providing physical or virtual machines to the customer. Here, the cloud OS can support large numbers of machines and has the ability to scale services up and down according to customers' varying requirements. Quality of service and cost efficiency are important factors from both the customer's and the provider's perspective.

Focus of this study. This paper investigates non-functional properties of a simple computing service. We are in particular interested in the performance under several users' requirements. Questions of interest are, e.g., how to scale the server system, i.e., how many machines are needed to achieve a certain service level? When can certain bottleneck situations such as long waiting times occur? An important focus is on the sensitivity of these performance aspects when varying the two main system parameters – the rate of processing failures and the rate of putting machines into suspended mode. Apart from these performance issues, we are also interested in power consumption by computing services.

Approach. The computing server is succinctly modelled as a GSPN [4]. Machines can be either operational ("ready") or stand-by ("suspended"), and randomly switch between these modes. Job loads are generated by a Poisson process, and job processing may randomly fail. It is a closed model, i.e., jobs that have been processed return to a pool from which new job requests can be generated. Two user profiles are considered: scientific computing tasks with high computational demands and small (or large) client populations, and web service tasks with low computational demands and large client populations. The GSPN is analysed using the recently developed MAMA tool chain[1]. Put in a nutshell, the GSPN is mapped onto a Markov automaton (MA; [5]), an extension of a continuous-time Markov chain with non-determinism, and analysed using several new algorithms [3]. The capability to deal with non-determinism allows for analysing non well-defined GSPNs; in fact, the new GSPN-semantics is (weak) bisimulation equivalent to the classical GSPN semantics [1]. We exploit non-determinism for analysing the sensitivity of various performance metrics w.r.t. variations of the main system parameters. The response-time distribution in steady state is obtained by a combination of the PASTA theorem and the tagged token technique.

[1] Publicly available at: `http://wwwhome.cs.utwente.nl/~timmer/mama/`.

Main results and findings. GSPNs turn out to be a convenient modelling formalism in our setting; the resulting model is succinct and easily extendible. Our analysis gives useful insight into the initial number of ready machines so as to keep the time until jobs cannot directly be processed below a certain threshold, and to keep the long-run average number of queued jobs at a minimum. This is valuable information to adequately dimension the computing server. The response-time distribution shows that being in equilibrium, the system can guarantee a reasonably low response time in the majority of cases when job requests are organised in a FIFO queue. Taking a random service policy however gives rise to a substantial increase in response time. The sensitivity analyses reveal that the failure rate has not a significant impact on the expected time until no ready machine is available but has a large influence on reaching 50% and 90% job queue capacity.

Main contributions. To summarize, our main scientific contributions are:

1. a simple GSPN model for computing servers in which machines can be operational ("ready") or stand-by ("suspended"), and job processing may randomly fail;
2. an extensive set of evaluation results focusing on expected durations until given system occupancies, long-run objectives, response-time distribution and energy consumption;
3. a sensitivity analysis of the two major model parameters –processing failures and machine down rates– by exploiting non-determinism;
4. a first industrial case study using the recent GSPN-to-Markov automata semantics [1] and accompanying analysis algorithms [3].

Organization of this paper. Sect. 2 briefly introduces GSPNs, their mapping onto MA, and the analysis algorithms. Sect. 3 presents the GSPN model of the computing server. Sect. 4 is the main section of the paper and presents our evaluation results. Sect. 5 discusses related work, and Sect. 6 concludes.

2 Modeling Formalism: GSPNs

The scenario. In this paper we analyse an exemplary service offering computing resources. We explore two scenarios: first, an application that is supposed to process a number of jobs as they occur, for example, in scientific calculations for weather forecasts, or biometric and medical simulations. Typically, such jobs are long running and processing is requested with a proportionally low frequency. Often they are executed in batches. The second scenario reflects a web service. Here jobs are running much shorter but they occur at a considerably higher frequency. In both scenarios we analyse the influence of various parameters on the overall performance of the service. For example, an insufficient number of available machines impacts the response time of the service, as jobs have to be queued until they can be processed. If new instances are requested, the time to start these instances delays their availability. Furthermore, instances can be detracted from the user due to application failures or for maintenance reasons. On the other hand, an over-dimensioned number of available (and mostly idling) machines unnecessarily increases the costs to run the service as well as the energy consumption of the data center. In the following we introduce our formal model that enables such analyses.

GSPNs. Generalised Stochastic Petri Nets (GSPNs) [4] are a modelling formalism for concurrent systems involving randomly timed behaviour. GSPNs inherit from Petri nets the underlying bipartite graph structure, partitioned into *places* and *transitions* and extend this by distinguishing between *timed* and *immediate* transitions. The latter can fire instantaneously and in zero time upon activation. The firing time of a timed transition is governed by a rate which uniquely defines a negative exponential distribution. A special form of timed transitions is of type *n-Server*, meaning that the given rate is multiplied by the number of predecessor tokens to yield the actual transition rate. Timed transitions are depicted as non-solid bars labelled by "n-Server", if applicable, while immediate transitions are depicted as solid bars. An example of a GSPN is shown in Fig. 2.

GSPN semantics. The semantics of a GSPN may conceptually be considered as consisting of two stages [4]. The first (abstract) stage describes when and which transitions may fire, and with what likelihood. This basically conforms to playing the token game of a net. The second stage defines the underlying stochastic process –typically a continuous-time Markov chain (CTMC)– which represents the intended stochastic behaviour captured in the first stage. This CTMC is obtained by amalgamating sequences of immediate firings. Performance analysis of the GSPN then amounts to analysing the transient or steady-state behaviour of its underlying CTMC. This trajectory works fine for well-defined GSPNs, i.e., nets that are confusion-free. Recently, a new GSPN semantics has been proposed [1] that covers all definable GSPNs, in particular nets that contain confusion. The semantics is defined in terms of Markov automata (MA), which are basically transition systems in which transitions are either labelled with the action τ (representing the firing of an immediate transition), or with the rate of an exponential distribution. The target of an action transition is a discrete probability distribution over the states while for a rate-labelled transition it is simply a state. For confusion-free GSPNs, this semantics is (weakly) bisimilar to the two-phase GSPN semantics. States in MA correspond to markings of the net. It falls outside the scope of this paper to give a full-fledged treatment of this semantics; we rather present a simple example, see Fig. 1. The example net is confused, as transitions t_1 and t_2 are not in conflict, but firing transition t_1 leads to a conflict between t_2 and t_3, which does not occur if t_2 fires before t_1. Transitions t_2 and t_3 are weighted so that in a marking $\{p_2, p_3\}$ in which both transitions are enabled, t_2 fires with probability $\frac{w_2}{w_2+w_3}$ and t_3 with its complement probability. Fig. 1 depicts the MA semantics of this net. Here, states correspond to sets of net places that contain a token. In the initial state, there is a non-deterministic choice between the transitions t_1 and t_2. In this paper, we

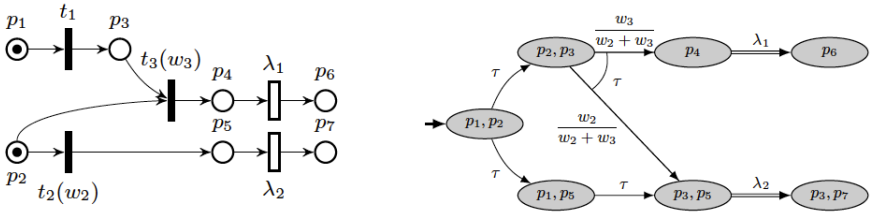

Fig. 1. A simple confused GSPN (left) and its MA semantics (right)

will exploit non-determinism for analysing the sensitivity of various performance metrics w.r.t. some rates in the net. The MAMA tool chain [3] realizes the new semantics via a translation to process algebra. This tool is used in our analysis of the computing server.

Performance metrics. We consider three basic measures on (possibly confused) GSPNs. *Long-run* average measures are the pendant to steady-state probabilities in CTMCs. Given a state m in an MA, i.e., a marking in the net, and a set T of target states, this measure is the minimal (or maximal) fraction of time spent in some state in T in the long run, when starting in m. In absence of non-determinism, the minimal and maximal long-run average coincide. The computation of long-run averages on MA can be reduced to a combination of several standard algorithms on Markov decision processes (MDPs); for details we refer to [3]. The (minimal or maximal) *expected time* of reaching a set T of target states from a given state m can be obtained by a reduction to a stochastic shortest-path problem. Such problems can efficiently be solved by linear programming. The third measure is determining *timed reachability probabilities*. They are the pendant to transient probabilities in CTMCs. Given a set T of target states, a given state m and a deadline d, the central question here is to determine the minimal (or maximal) probability of reaching some state in T within time d when starting in m. Such problems are a bit more involved and can be tackled using discretisation techniques. Further details are outside the scope of this paper and are provided in [3].

3 A GSPN Model of the Computing Server

Figure 2 shows our GSPN model of a computing server. The places, transitions, and parameters have the following interpretation:

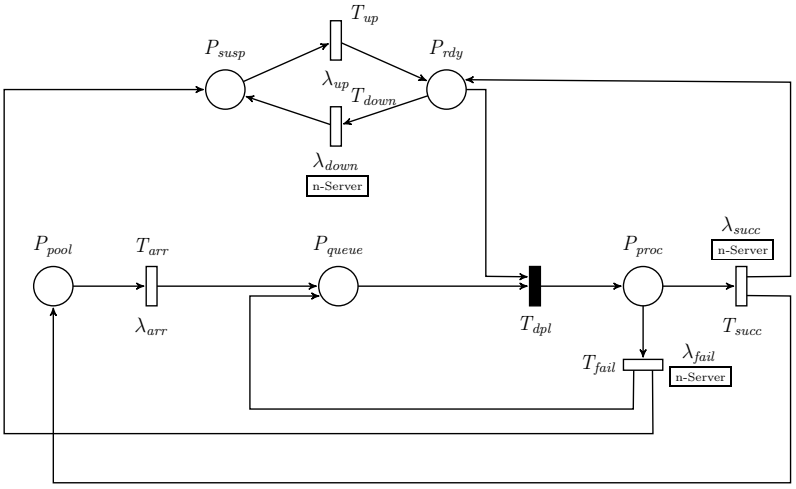

Fig. 2. GSPN model of computing server

P_{pool} represents the pool of job requests. After successful processing, they re-enter this place.

P_{queue} models the waiting queue of the system. Job requests enter this place via T_{arr} with rate λ_{arr}, and re-enter the queue if their processing fails.

T_{dpl} immediately deploys a waiting request on a (ready) machine if available.

P_{proc} represents the actively executing requests. From here, the jobs are either returned via T_{succ} to P_{pool} (with rate λ_{succ}) after successful completion, or the machine fails (or is preempted by the provider). In the former case, $1/\lambda_{succ}$ gives the expected execution time. In the latter case, $1/\lambda_{fail}$ is the mean time between failures, and requests are enqueued again via T_{fail}.

P_{rdy}/P_{susp} divide the available machines into two categories. The first ("ready") can directly process service requests, while the second ("suspended") first need to be set up to become ready. Two associated transitions, T_{up} and T_{down} with respective rates λ_{up} and λ_{down}, represent the corresponding startup and shutdown operations that are taken when additional machines are required or when they are idle, respectively. In particular, λ_{up} models the booting time of a machine by an exponential distribution. Initially we assume that all machines in the server are ready. Note that T_{dpl} requires P_{rdy} to be non-empty in order to process requests, and that a machine re-enters P_{rdy} after successful processing of a request. If processing is aborted via T_{fail}, the machine becomes suspended as it has to be restarted (in case of failure) or reallocated (in case of preemption).

Note that transitions T_{succ}, T_{fail} and T_{down} are marked as "n-Server" to reflect the fact that the respective rates refer to single requests and machines. Since we assume the failed machines can only be rebooted one by one, T_{up} is not marked as "n-Server". Moreover, the computing server with different types of services and machines can be easily modeled from the "simple" one by using *colored* GSPN or assigning immediate transitions with weights (when the probabilistic distribution of such types of services (machines) is known), and these extended semantics can again be translated into MA semantics without extra effort.

This GSPN is specified using the Petri Net Markup Language (PNML), which is a standardized XML-based interchange format for Petri nets [6]. Many tools can be used to generate Petri Nets in PNML notation, such as the Platform-Independent Petri Net Editor (PIPE). Using the MAMA tool chain, the GSPN is automatically mapped onto a Markov automaton for further analysis.

4 Quantitative Evaluation of the GSPN Model

This section presents the results of evaluating our GSPN model. They are obtained by using the recently developed MAMA tool chain that supports the mapping of GSPNs onto Markov automata and their quantitative assessment. We first present some statistics about the underlying state space size, detailing the user profiles –scientific computing tasks and web service tasks– that are used, and the set of properties considered. Due to space limitations, we focus on presenting the main outcomes. Sect. 4.2 presents the results for expected-time and long-run metrics for scientific computing and web-service tasks. These figures give insight into the quantitative behaviour of the computing system, and

provide useful information concerning dimensioning the system in terms of the (initial) number of ready machines. Sect. 4.3 investigates the sensitivity of our evaluation results on varying the parameters λ_{fail} and λ_{down}. Sect. 4.4 considers the long-run energy consumption, whereas Sect. 4.5 focuses on the response-time distribution of user requests. The latter results are of interest to both service providers and users. All experiments are obtained on a AMD 48-core CPU @ 2.2 GHz, 192 GB RAM and Linux kernel 2.6.32.

4.1 Experimental Setup

State space. The state space of the GSPN model is determined by two parameters: the size of the client population and the initial number of ready machines. They are respectively represented by $\mathcal{I}(P_{pool})$ and $\mathcal{I}(P_{rdy})$ where $\mathcal{I}(P)$ denotes the initial number of tokens in place P. The state space sizes are summarized in Table 1 where the last column indicates the state space generation time (in seconds).

User profiles. We consider two application scenarios: scientific computing and web-services. In the scientific computing setting, tasks arrive at a relatively low rate and have a substantial processing time (usually ranging from minutes to hours). Web-service tasks such as bing search queries arrive much more frequently and have a short processing time, typically in the range of seconds.

Table 1. GSPN state space statistics

$\mathcal{I}(P_{rdy})$	$\mathcal{I}(P_{pool})$	# states	# trans.	time
100	100	20,201	55,250	23s
100	500	100,601	255,650	79s
100	1000	201,101	506,150	153s
100	2000	402,101	1,007,150	314s
100	2500	502,601	1,257,650	387s
200	1000	401,201	1,021,300	321s
250	1000	501,251	1,282,625	394s

Table 2. Parameter settings for the application scenarios

	λ_{arr}	λ_{succ}	λ_{fail}	λ_{up}	λ_{down}
Scientific computing	1.667	0.1	0.00208	0.05	0.005
Web service	180	20	0.006	0.05	0.005

Our parameters settings are listed in Table 2, where the time unit is one minute. Scientific computing tasks arrive at a rate of 100 requests per hour ($\lambda_{arr} = 1.667$), and require a processing time of 10 min ($\lambda_{succ} = 0.1$), failures occur once per eight hours, booting a machine requires 20 min, and a ready machine suspends once every 200 min (please bear in mind that transition T_{down} is of type n-Server). Three web service requests are issued per second, and each requires an average execution time of three seconds. The scientific computing case is considered for a small client population, i.e., $\mathcal{I}(P_{pool}) = 100$, and a large one, i.e., $\mathcal{I}(P_{pool}) = 500$. The web service setting is of interest only for a large client population.

Properties. For each scenario, we consider the following six properties:

p1. The expected time until P_{queue} exceeds 90% of its capacity
p2. The expected time until P_{queue} exceeds 50% of its capacity
p3. The expected time[2] until no ready machine is available for requests in P_{queue}
p4. The average long-run occupancy of P_{queue}
p5. The average long-run occupancy of P_{proc}

[2] In some dedicated cases, we also study the probability until this phenomenon happens within a given deadline. This *timed reachability* property is however more complex to evaluate.

The verification times are listed in Table 3. It clearly shows that expected-time properties are simpler to analyse than long-run properties.

This is due to the fact that the former involve a single LP problem to be solved, whereas the latter require a (non-trivial) graph decomposition as well as solving of several LP problems. Moreover, evaluating **p4** requires $\mathcal{I}(P_{pool}) \cdot \mathcal{I}(P_{rdy})$ iterations when using value iteration, whereas the other long-run properties require $\mathcal{I}(P_{rdy})$ iterations. This explains the difference in runtimes between the long-run properties. We stress that these runtimes should not be compared to evaluating similar properties on CTMCs; as MA include non-determinism, the algorithms are intrinsically more complex and have a higher time complexity.

Table 3. Property evaluation times per scenario

	p1	**p2**	**p3**	**p4**	**p5**
$\mathbf{s}_{small}$	12m2s	5m52s	1m13s	18h44m	6h30m
$\mathbf{s}_{large}$	6h6m	1h54m	10m13s	495h28m	109h16m
$\mathbf{w}_{large}$	5h6m	1h39m	9m52s	402h38m	50h35m

4.2 Expected-Time and Long-Run Properties

Expected-time properties. First consider the properties **p1** and **p2**, i.e., the expected time until P_{queue} reaches 90% and 50% of its capacity, respectively. For a small population, the capacity of P_{queue}, denoted C_{queue}, equals $\mathcal{I}(P_{pool}) = 100$. Analysing these properties boils down to computing the expected time from the initial marking to the set of markings satisfying $\mathcal{M}(P_{queue})/C_{queue} \geq p\%$ where $\mathcal{M}(P)$ refers to the current marking of place P. The evaluation results are shown in Fig. 3 (left) for $p = 90\%$ and Fig. 3 (right) for $p = 50\%$, where the number of initial ready machines (x-axis) is varying. They suggest that about 25 initial ready machines is a rather good choice.

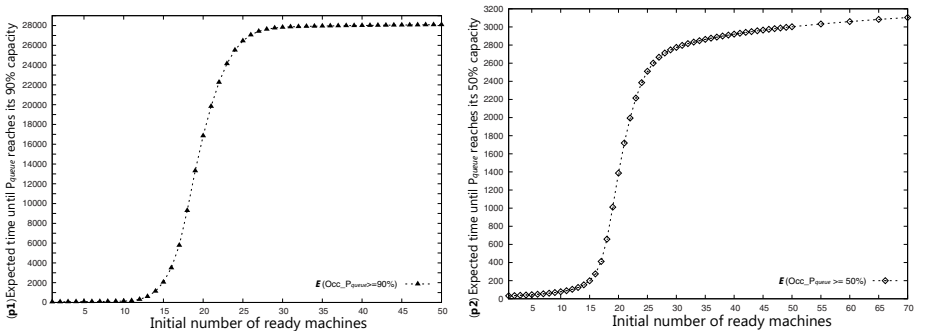

Fig. 3. Expected time until exceeding 90% (left) and 50% (right) of client queue capacity for scenario $\mathbf{s}_{small}$

Property **p3** refers to the expected time until there is no more ready machine available for waiting requests in P_{queue}. We check this by determining the expected time from the initial marking to the set of markings where $\mathcal{M}(P_{queue}) > 0$ and $\mathcal{M}(P_{rdy}) = 0$. The results are plotted in Fig. 4 (left). To get more insight into the likelihood of encountering this situation, we also evaluate the probability to reach this state within a given time frame d. As the computation of

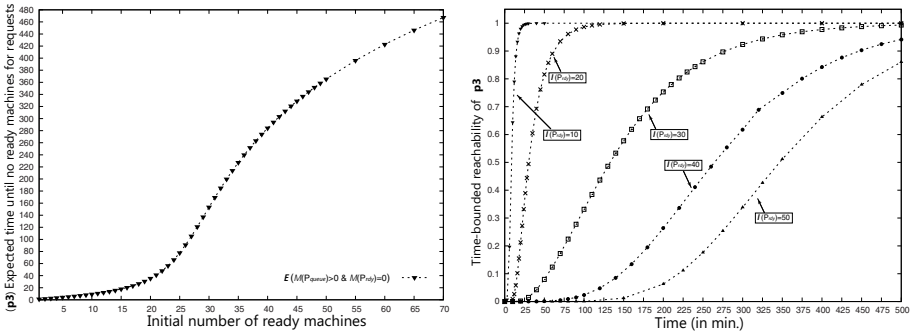

Fig. 4. Expected-time and timed reachability probabilities until client requests wait

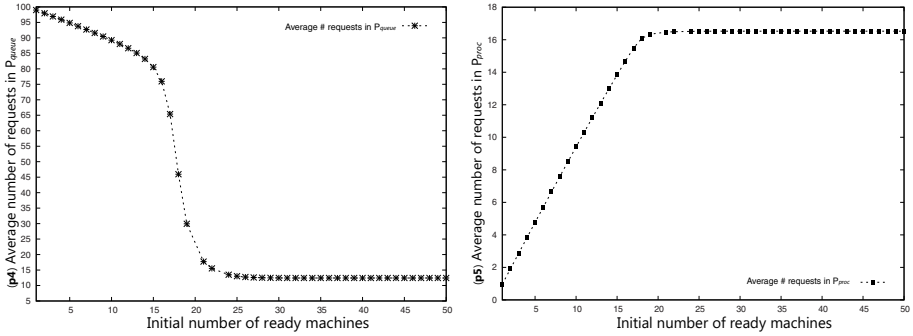

Fig. 5. Long-run average number of tasks in P_{queue} (left) and P_{proc} (right) for $\mathbf{s}_{small}$

these probabilities is very time-consuming, we let the initial number of ready machines vary from 10 to 50. The results for varying d (along the x-axis, in minutes) and different values of $\mathcal{I}(P_{rdy})$ are plotted in Fig. 4 (right). Clearly, the timed reachability probabilities rapidly approach one for a relatively small number of machines (e.g., 10 or 20), but this effect substantially diminishes on increasing this number. The evaluation times for obtaining the latter results ranges from around 28 s for 10 machines up to nearly 60 h for 50 machines where we have set an accuracy of 10^{-1}.

Long-run properties. For a set T of target states, let $\mathcal{L}(T)$ denote the long-run average fraction of time of residing in T. The set of target states where exactly $i \in \mathbb{N}$ tokens are in place P is denoted as $T_i(P) = \{\mathcal{M} \mid \mathcal{M}(P) = i\}$. The long-run average number of tokens in place P, denoted $\mathcal{L}(\#\,P)$, in our GSPN model is then given by $\mathcal{L}(\#\,P) = \sum_{i \in \mathbb{N}} \mathcal{L}(T_i(P)) \cdot i$. The average number of waiting (in P_{queue}) and processing tasks (in P_{proc}) are shown in Fig. 5 (left) and Fig. 5 (right), respectively. Our results indicate that an equilibrium is reached for about 25 initial ready machines. To show the impact of the size of the client population, we check both properties for $\mathcal{I}(P_{rdy}) = 500$, see Fig. 6. The results indicate an (expected) increase of waiting requests in P_{queue}, whereas the average number of processing requests is only negligibly affected.

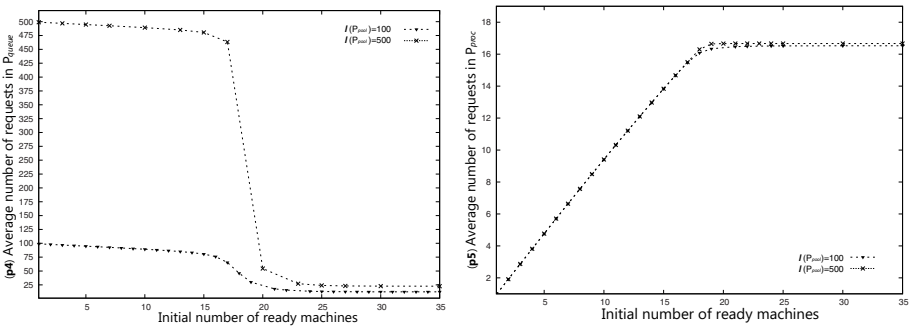

Fig. 6. Long-run average number of tasks in P_{queue} (left) and P_{proc} (right) for $\mathbf{s}_{large}$

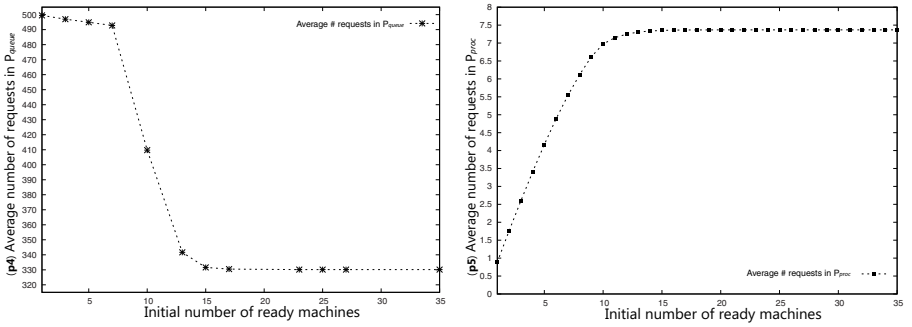

Fig. 7. Long-run average number of tasks in P_{queue} (left) and P_{proc} (right) for $\mathbf{w}_{large}$

Web-service tasks. We evaluated all properties also for the web-service setting. In contrast to the scientific computing case, jobs have a short processing time but arrive at a substantially higher rate (cf. the parameter settings in Table 2). Due to space reasons, we only present the results of the long-run properties **p4** and **p5** (see Fig. 7). Whereas for the scientific computing application scenario, a stable situation is reached for about 25 ready machines, this is now the case for 15 machines. In that case, the average number of concurrently processing tasks in the server will be around 7.5, and there are about 330 tasks on average waiting in the queue.

4.3 Sensitivity Analysis

In order to a get a better insight into the influence of two important modelling parameters –the failure rate λ_{fail} of job processing and the rate λ_{down} of ready machines becoming suspended– we carry out a sensitivity analysis by exploiting the native support of non-determinism in our setting. (Note that such an analysis is not possible using the classical GSPN semantics for "deterministic" nets.)

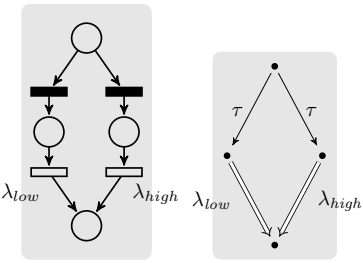

Fig. 8. Non-determinism in GSPN and MA

This is done as follows. We first only vary λ_{fail}. In the underlying MA, this is accomplished by adding two internal (immediate) transitions which are directly followed by two timed transitions with rates λ_{low} and λ_{high}, respectively (see Fig. 8). The benefit of this approach is that when analysing our properties, we obtain *bounds*. For instance, for the expected-time properties, we would obtain minimal and maximal expected-time values. This means in particular, that for any failure rate between λ_{low} and λ_{high}, the expected time lies between the obtained bounds. The analysis algorithms in the MAMA tool chain obtain these bounds from the adapted GSPN (and thus MA) by running a single algorithm. We first vary the failure rate λ_{fail} from 0.00208 ($\approx$ 480 min) to 0.00312 ($\approx$ 320 min) and 0.00416 ($\approx$ 240 min) respectively, then reduce the failure rate λ_{fail} to 0.00156 ($\approx$ 640 min) and 0.00104 ($\approx$ 960 min) respectively. The obtained results for the expected-time properties are shown in Fig. 9 and 10.

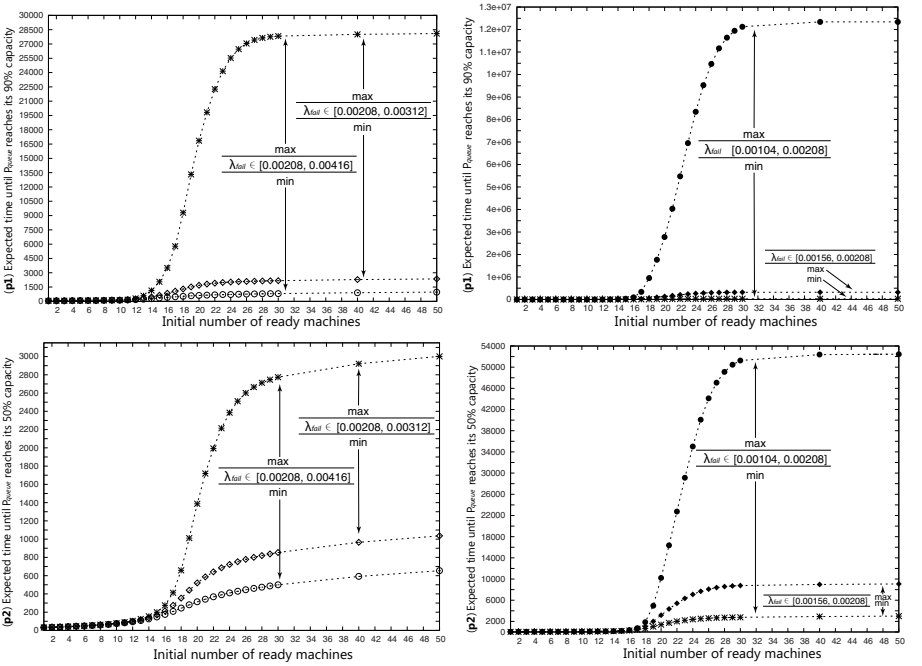

Fig. 9. Evaluation results for properties **p1** and **p2** by varying λ_{fail}

From these results, we make three observations. First, we consider the number of clients for which the lower and upper expected times start to become distinct. On increasing λ_{fail}, i.e., with shorter average failure intervals, the minimal and maximal expected times start to differ for $\mathcal{I}(P_{pool}) = 14$, 15 and 22 in **p1**, **p2**, and **p3**, respectively. On decreasing λ_{fail}, i.e. with longer average failure intervals, these points shift to 17, 18, and 24, respectively. Secondly, we observe (e.g., from both the right curves) a significant impact on the difference between minimal and maximal expected times when increasing the minimal failure rate. Reducing $\lambda_{fail} = 0.00208$ to just one half (0.00104) results in an increase of about 400 times of the expected time (after reaching an equilibrium), whereas a reduction to 0.00156 (= three quarters) results in just an increase by a factor

of 11 (cf. Fig. 9 (top right)). This is because transition T_{fail} (with rate λ_{fail}) has a two-fold effect on the number of tasks in P_{queue} in our GSPN model. First, T_{fail} has a more direct effect on tasks in P_{queue} than T_{succ} (λ_{succ}). If a task is successfully processed, it arrives in P_{pool} and waits for turning to be a new request. Here, T_{arr} is not an *n-Server* type time transition. However, if the processing of a task has failed, it will directly arrive in P_{queue} (and thus increases the number of waiting tasks). Second, T_{fail} also affects the availability of ready machines in P_{rdy}. If the task is successfully finished, the ready machine goes back to P_{rdy} and does not need to re-initialize. But if it has failed, the machine serving the task needs to be restarted. Note that the startup of a suspended machine (transition T_{up}) is not a *n-Server* typed transition. As a result, these twofold effects will cause a superposed influence on the growth of the number of tasks in P_{queue} and hence have a strong impact on the expected-time property. The third observation we make is that λ_{fail} does not have significant influence on **p3** in comparison with **p1** and **p2** (cf. Fig. 10 (left)). The results of other evaluations (cf. Fig. 10 (right)) also confirm that rather than λ_{fail}, λ_{down} has a stronger impact on property **p3**.

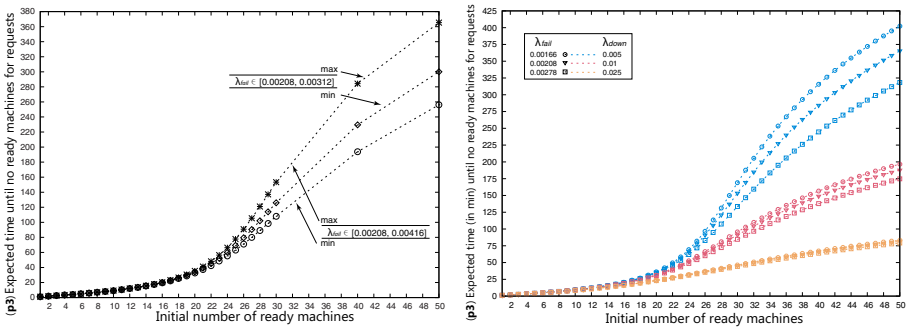

Fig. 10. Checking **p3** by varying λ_{fail}

4.4 Long-Run Energy Consumption

In the following section, we are going to answer two questions which service providers may especially be interested in: what is the operating cost and the quality of service of their computing server? To answer these questions, we try to estimate the long-run energy consumption of the server and the response-time distribution of requests as respective metrics for these two parameters.

We assume that active machines (i.e., those in P_{proc}) have a power consumption of 40 W [7], ready machines (P_{rdy}) have a power consumption of 20 W (50% of the active value), and suspended machines (P_{susp}) have a power consumption of 8 W (20% of active). Furthermore, if we initialize m ready machines in P_{rdy}, in the long run these will be distributed to three possible places P_{susp}, P_{rdy} and P_{proc}. In other words, at any moment the overall number of machines in these three places will be m. Thus we can compute the power consumption of the server in the long run based on the steady-state probabilities. After we have determined the average number of machines in these places in the long run,

which can be analogously derived as **p4**, the overall power consumption in the long run is computed as

$$P = 40\ W \cdot \mathcal{L}(\#P_{susp}) + 20\ W \cdot \mathcal{L}(\#P_{rdy}) + 8\ W \cdot \mathcal{L}(\#P_{proc}).$$

The results are shown in Fig. 11 (left) for failure rate $\lambda_{fail} = 0.00208$. We observe that λ_{down} and $\mathcal{I}(P_{rdy})$ do not significantly influence the average number of active machines in P_{proc} in the long run; they are always around 16 (cf. Fig. 5 (right)). Although the more ready machines we initialize in P_{rdy} (i.e. $\mathcal{I}(P_{rdy})$), the more redundant machines will be in P_{rdy} and P_{susp}. Their distribution is controlled by λ_{down}. If there are few redundant machines, e.g., $\mathcal{I}(P_{rdy}) = 20$ (i.e., only about 4 machines left for λ_{down} to distribute these to P_{rdy} and P_{susp}), then the energy consumption only varies in a small range. In contrast, if $\mathcal{I}(P_{rdy}) = 40$, we notice that when λ_{down} is 0.0025, about 23.10 machines are suspended, 0.65 are ready, and 16.25 are active on average, whereas when λ_{down} is 0.001, about 9.61 machines are suspended, 13.73 are idle, and 16.66 are active on average. This wide range distribution of suspended and ready machines caused by λ_{down} yields a drastic increase of energy consumption as shown in Fig. 11 when $\mathcal{I}(P_{rdy})$ is large. On the one hand, keeping a certain number of ready machines in P_{rdy} guarantees a better response time of requests, on the other hand, too many redundant ready machines lead to a higher energy consumption.

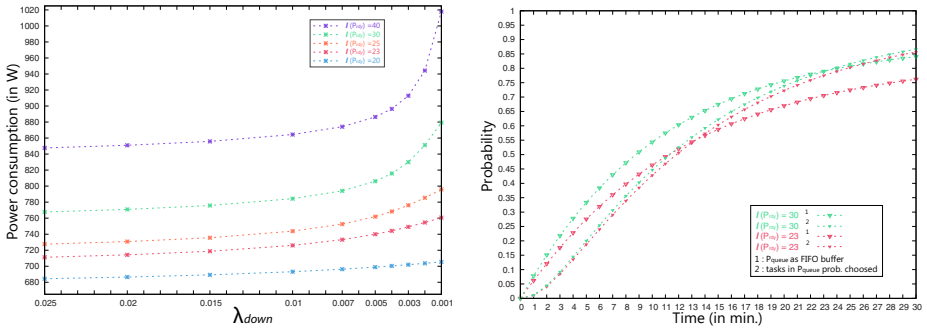

Fig. 11. Power consumption and response-time distribution of computing server

4.5 Response-Time Distribution in Steady State

Now we compute the response-time distribution in steady state approximately [3]based on a combination of the PASTA theorem [8] and tagged-token techniques [9] for GSPNs. From the former we know that an arrival of a Poisson process sees the system as if it was arriving at a random instance of time (i.e. the system in steady state). Since the steady-state probabilities of the computing server can easily be computed from the resulting MA by using the MaMa tool, the response-time distribution in steady state can be computed by tagging a customer's job and following the tagged request until it has been successfully processed in the server which is in steady state. The right side of Fig. 12 illustrates the tagged task (represented by $*$ at P_{queue}) in the computing server (with

[3] Since our model is closed.

$\lambda_{fail} = 0.00208$, $\lambda_{down} = 0.001$, $\mathcal{I}(P_{rdy}) = 30$) which is just in a steady state with probability 0.00192925986. The number attached to each place represents the current number of tokens in that place. By adding a boolean variable to each place P_{queue}, P_{proc} and P_{pool} indicating the position of the tagged request, the response-time distribution is then obtained via a time-bounded reachability computation using the MAMA tool. Note that adding a tagged token increases the state space. The left side of Fig. 12 shows the advantage of using probabilistic transitions in MAs during our computation. They are used for setting up the initial probabilistic distribution for different steady states of the server. After these steps, we can again generate the MA and compute the response-time distribution, all with the MAMA tool. The result is shown in Fig. 11 on the right.

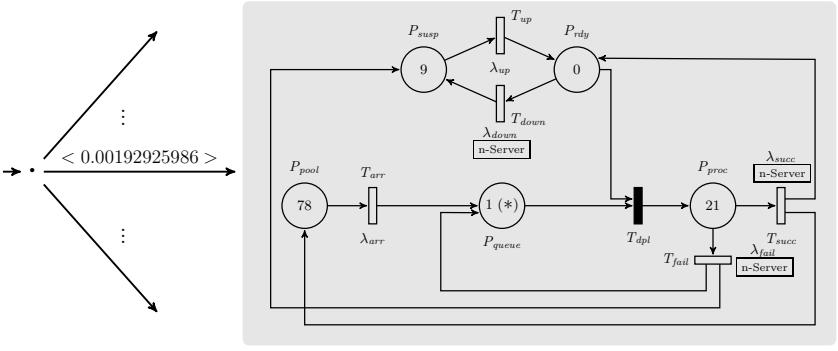

Fig. 12. Computing server in steady state with tagged token (∗) placed in P_{queue}

5 Related Work

The closest work on performance evaluation that is similar to our work was carried out in the field of cloud computing. Appropriate stochastic models have recently been proposed in [10,11,12,7]. [10] describes a prototype tool for translating expressions of probabilistic resource usage patterns into Markov decision processes with rewards. This allows to check costing and usage queries expressed by reward-augmented probabilistic temporal logic (PCTL). [13] models cloud computing systems with migration by CTMCs using PRISM, verifying some quantitative properties such as the time-bounded probability of migration operation in the cloud computing system. [12] introduces a stochastic reward net model for the IaaS cloud, based on which resiliency metrics are computed by changing the job arrival rate and the number of available PMs. [11,7] models a cloud computing server as different kinds of scalable interacting stochastic (sub)models. In [11], interaction happens when results from sub-models are used as input for other sub-models. In the end, two quality-of-service metrics, the effective job rejection probability and the effective mean response delay, are obtained. [7] represents the IaaS cloud with tiered service offerings by three pools for hot, warm and cold virtual machines, which have different response time and power consumption characteristics, respectively. These pools are

represented by interacting stochastic sub-models (basically CTMCs), from which the power consumption and performance metrics are computed.

Another strand of research is based on queueing theory. [14,15,16] analytically evaluates the performance of cloud computing services. [14] proposes an approximation method to compute the probability and cumulative distribution of the response time by applying the inverse Laplace-Stieltjes transformation (LTS) on the LTS of total response time obtained analytically. [15] models the cloud computing server by a M/G/m/m+r queueing system. Here, quantitative properties such as the distributions of the number of tasks and of waiting and response times etc., are analytically computed. Besides cloud computing, [9] presents a general distributed computing technique for response-time analysis with GSPN models based on Laplace transformation and its inversion [17,18].

6 Concluding Remarks

This paper has presented a simple GSPN for computing servers in which virtual machines can be stand-by or operational, and the processing of requests may randomly fail. Our analysis has focused on both long-run and expected-time properties that give insight into the number of required virtual machines so as to yield a given service level (i.e., low response time, and low probability of lack of processing power). A sensitivity analysis by exploiting non-determinism in the GSPN model shows the influence of the failure probability. Finally, long run energy consumption was analysed. The exploited analysis algorithms are based on analysing Markov automata, and are more intricate –hence more time-consuming– than those for classical (confusion-free) GSPN which are based on CTMCs. Although our original net does not exhibit confusion, the one used for the sensitivity analysis does. Future work will focus on improving the performance of the analysis algorithms.

References

1. Eisentraut, C., Hermanns, H., Katoen, J.-P., Zhang, L.: A semantics for every GSPN. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 90–109. Springer, Heidelberg (2013)
2. Katoen, J.P.: GSPNs revisited: Simple semantics and new analysis algorithms. In: Application of Concurrency to System Design (ACSD), pp. 6–11. IEEE (2012)
3. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.-P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 55–71. Springer, Heidelberg (2013)
4. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons (1995)
5. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS, pp. 342–351. IEEE Computer Society (2010)
6. Billington, J., Christensen, S., van Hee, K.M., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)

7. Ghosh, R., Naik, V.K., Trivedi, K.S.: Power-performance trade-offs in Iaas cloud: A scalable analytic approach. In: Dependable Systems and Networks Workshops, pp. 152–157 (2011)
8. Wolff, R.W.: Poisson arrivals see time averages. Operations Research 30, 223–231 (1982)
9. Dingle, N.J., Knottenbelt, W.J.: Automated customer-centric performance analysis of Generalised Stochastic Petri Nets using tagged tokens. In: PASM 2008. ENTCS, vol. 232, pp. 75–88 (2009)
10. Johnson, K., Reed, S., Calinescu, R.: Specification and quantitative analysis of probabilistic cloud deployment patterns. In: Eder, K., Lourenço, J., Shehory, O. (eds.) HVC 2011. LNCS, vol. 7261, pp. 145–159. Springer, Heidelberg (2012)
11. Ghosh, R., Trivedi, K.S., Naik, V.K., Kim, D.S.: End-to-end performability analysis for Infrastructure-as-a-Service cloud: An interacting stochastic models approach. In: IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 125–132. IEEE CS (2010)
12. Ghosh, R., Longo, F., Naik, V.K., Trivedi, K.S.: Quantifying resiliency of Iaas cloud. In: IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 343–347. IEEE (2010)
13. Kikuchi, S., Matsumoto, Y.: Performance modeling of concurrent live migration operations in cloud computing systems using PRISM probabilistic model checker. In: IEEE Int. Conf. on Cloud Computing (IEEE CLOUD), pp. 49–56. IEEE (2011)
14. Xiong, K., Perros, H.G.: Service performance and analysis in cloud computing. In: IEEE Congress on Services, Part I (SERVICES I), pp. 693–700. IEEE Computer Society (2009)
15. Khazaei, H., Misic, J.V., Misic, V.B.: Performance analysis of cloud computing centers using M/G/m/m+r queuing systems. IEEE Trans. Parallel Distrib. Syst. 23, 936–943 (2012)
16. Yang, B., Tan, F., Dai, Y.S., Guo, S.: Performance evaluation of cloud service considering fault recovery. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) Cloud Computing. LNCS, vol. 5931, pp. 571–576. Springer, Heidelberg (2009)
17. Abate, J., Whitt, W.: The Fourier-series method for inverting transforms of probability distributions. Queueing Syst. 10, 5–87 (1992)
18. Harrison, P.G., Knottenbelt, W.J.: Passage time distributions in large Markov chains. In: Int. Conf. on Measurements and Modeling of Computer Systems (SIGMETRICS), pp. 77–85. ACM (2002)