QoS and energy management with Petri nets: a self-adaptive framework

Diego Perez-Palacin[a], Raffaela Mirandola[b], José Merseguer[a]

[a]*Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain*
[b]*Dip. di Elettronica e Informazione, Politecnico di Milano, Milano, Italy*

Abstract

Energy use is becoming a key design consideration in computing infrastructures and services. In this paper we focus on service-based applications and we propose an adaptation framework that can be used to reduce power consumption according to the observed workload. The adaptation guarantees a trade-off between energy consumption and system performance. The approach is based on the principle of proportional energy consumption obtained by scaling down energy for unused resources, considering both the number of servers switched on and their operating frequencies. Stochastic Petri Nets are proposed for the modeling of the framework concerns, their analyses give results about the trade-offs. The application of the approach to a simple case study shows its usefulness and practical applicability. Finally, different types of workloads are analyzed with validation purposes.

Keywords: Energy, Performance, Stochastic Petri nets, Software architectures

1. Introduction

Problem Statement. The constant growth of energy usage in industrialized countries is creating problems to the sustainability of the Earth development. The problem of energy use concerns many fields in human activities: for this reason some new disciplines such as green computing are growing up to study how to consume less energy by providing the same quality of service [1].

As shown in [2, 1], the interest towards efficient use of technology is motivated by some alarming trends showing, for example, that computing equipment in the

Email addresses: diegop@unizar.es (Diego Perez-Palacin), mirandola@elet.polimi.it (Raffaela Mirandola), jmerse@unizar.es (José Merseguer)

U.S. alone is estimated to consume more than 20 million giga-joules of energy per year, the equivalent of four- million tons of carbon-dioxide emissions into the atmosphere [1]. IT analysis firm IDC (http://www.idc.com/) estimates the total worldwide spending on power management for enterprises was likely a staggering 40 billion dollars in 2009.

Large computing infrastructures, like data centers, web services hosting or email, in the U.S. consumed 1.5% of all electrical power in 2006 and it grows at an annual rate of 12% [3]. Nevertheless, it is possible to observe that they are so complex that some parts become inactive even during active periods. Let us consider, for example, the providers of service-based applications. Often when deciding the amount of resources -hardware and software- to be included in the platform, worst-case scenarios are considered, which can lead to over-provisioning for other scenarios of the system. The result is a static system deployment that wastes part of the available processing infrastructure and consequently causes energy waste.

Therefore, a first direction that can be followed for energy savings is the definition of adaptation plans that can be used to reduce power in time (turn off during idle times) and space (turn off inactive elements). Hence, infrastructures can be dynamically scaled to conserve power with no impact on performance while they match workload demands.

The definition of this adaptation plan is not easy, because the workload is typically variable and unpredictable and because there are also other, possibly contrasting, goals that should be satisfied. Indeed, the ultimate goal of a service provider is to maximize profits from its offered services, while for a client the main objective is to obtain a service with required QoS at the minimum cost. Therefore a suitable adaptation plan should be able to define the best trade-off between energy consumption and QoS offered. The problem is quite complex and several attempts exist in the literature proposing methods for managing power and guaranteeing the agreed quality of service (see Section 8).

In this paper we concentrate on performance quality, while the problem of maximizing providers revenues, although important, is not directly tackled. As defended in [4], quality requirements always must be met once contracted. However, the problem is indirectly addressed, since having a strategy that scales the amount of servers, while satisfying the performance requirements, reduces the expenses in the equation $profit = revenues - expenses$.

Proposed Solution. In order to reduce energy waste, the processing infrastructure of a service provider can be dynamically accommodated to the actual processing requirements for each scenario. Since the received workload varies frequently and in some cases unpredictably, human intervention to modify the amount of dedicated processing resources is not feasible. So the goal is to have the system aware of its processing resource needs, and able to self-adapt its processing infrastructure to fulfill such needs. Therefore, the objective is to build systems that can autonomously manage their processing resources in order to consume only the power necessary to satisfy their -possibly evolving- performance requirements. These new techniques actually complement the traditional and well-known off-line capacity planning [5]. To achieve the objective, we propose in this paper a reference architecture, or adaptation framework, based on the three layers architecture for self-managed systems presented in [6]. A reference architecture allows different deployments and can be used in a wide variety of situations. Specifically, in the paper, we discuss one deployment of interest for software services. Besides, the proposed framework is augmented with the definition of plans tackling the adaptation decisions that decrease as much as possible the system's energy consumption while maintaining the expected performance. The framework allows the plan regeneration when its execution context changes, which would make the current plan not suitable. The adaptation plan indeed depends on the dynamic variable workload, on the available processing resources, on the application processing demands and on the agreed QoS in terms of performance requirements.

To study the relations among these properties, we follow model-driven techniques to transform design models into different Stochastic Petri Nets (SPNs)[1] subnets. Subnets allow modeling the variable workload, the workflow, the processing resources and the logic to adapt the system energy consumption. The considered variables are not new, several works (e.g., [8, 4]) and a survey [9] exist on this topic.

As recognized in [3], queuing models, category of which SPNs are an example, are ideal to predict runtime trade-offs between performance and energy use. Moreover, queuing models have been largely validated during the last decades and we can be absolutely confident in the results they produce, which may free the modeler from the need of validating the model as long as it accurately represents the target system. This is an advantage regarding ad-hoc models, heuristics or equations when used to model complex behaviors, since they really need extensive validation to prove that the predictions they obtain actually match the real measurement. In contrast, queuing models have been accused of being difficult

[1]A formal description of the Generalized Stochastic Petri Nets [7] formalism is provided in Appendix A.

to construct. In this regard, we try to keep our models as simple, repeatable and scalable as possible and we propose tools to automatically construct them.

To generate a Petri net that represents the whole system behavior, we put together the previously mentioned subnets. Hence, this analyzable SPN includes fine-grained information regarding: mean execution times of internal activities; resource usage of activities; resource competition for passive resources (e.g., buffers) which generates "waits" and makes the system performance not scaling linearly with frequency; and resource competition for active (processors) which are the basis for power consumption.

The SPN evaluation, carried out with the GreatSPN tool [10], gives results about the suitability of the adaptation plan (in terms of whether it deteriorates performance results) and how much energy it saves. Moreover, we define a parametric Petri net that can be evaluated to discover which are the best parameters to tune the adaptation plan, in order to save as much energy as possible[2].

Motivating Example. Let us describe a kind of system for which our approach can be applied. Consider a company that develops software services which are offered in the Internet, some of them for free while others can get subscription rates. Irrespective of the implementation, the services follow the Service Oriented Architecture (SOA) paradigm. The company maintains a homogeneous computing infrastructure, around hundreds of servers, which deploys the services. These services are used all around the world and they can receive thousands of requests per minute at certain times of day, however it is also possible that the workload decreases at certain hours considerably. When the workload is at a peak the infrastructure has to be fully operative and each service will be replicated in as many servers as necessary to support the quality of service the company requires. On the other hand, when the workload is low, most of the servers will be unnecessary. Therefore, the company needs an integral *software* solution, beyond the traditional load balancer, that switches on and off the servers to adapt the infrastructure to the workload dynamically. We argue that if the software solution follows the architecture we describe in this paper, the infrastructure can achieve the advantages previously discussed, i.e., a good trade-off between QoS and energy conservation.

Paper Organization. The remainder of the paper is organized as follows. In Section 2 we present the self-adaptive framework for the management of energy and performance. The proposed SPN models for dynamic variable workload and en-

[2]A preliminary and short version of this idea has been published in [11]

ergy consumption are presented in Sections 3 and 4, respectively. The trade-off between performance goal fulfillment and energy consumption with the definition of the adaptation plan is presented in Section 5. Section 6 discusses a suitable deployment of the architecture and presents evaluation through an example, which is developed step by step to help practitioners to learn the proposal. The evaluation continues in Section 7 to experiment with variable workload. Related works are reviewed in Section 8. Section 9 draws some conclusions and provides pointers to on-going work. Two appendices are also included describing the Generalized Stochastic Petri Nets formalism (Appendix A) and the theory underlying the parameters derivation in the SPN modeling the system workload (Appendix B).

2. Self-adaptive framework

Kramer and Magee [6] proposed a three-layer reference architecture for self-adaptive systems, from now on we refer it as KM-3L. These layers (goal management, change management and component control) aim at satisfying the system goals by creating and following an adaptation plan for the system based on monitoring the platform where it executes. In [12, 13] we leveraged KM-3L to deal with the goal of enhancing the performance of a self-adaptive service integrator. Now, using this background, we work again with KM-3L to address the challenges in this work. Figure 1 describes our proposal identifying new responsibilities for each layer and the necessary software modules that can carry out them.

The *Component Control* layer accomplishes the application function of the system, in our case the workflows of the software services the infrastructure deploys. The *software services* modules represent the executable files of these software services. Note that we have replicated them, to represent several services and several running instances of each service. Each running instance, which manages requests until its maximum capacity, will execute in a server of the infrastructure. Each server can host several running instances. The *Component Control* layer also features a *HardwareController* and a *LoadMonitor* software modules, they include facilities to report the current status of the processing infrastructure and to support modifications on it. The *HardwareController* communicates with its upper layer to inform the current state of the servers (e.g., booting completion) and to receive orders to reconfigure them (increase/decrease frequency or switch on/off of servers). We think of it as a software module that manages the servers through the Wake on LAN (WoL) facility. The *LoadMonitor* monitors current system workload and informs its upper layer when the workload exceeds some thresholds, i.e., there is a problem to be solved. Thresholds of interest have been previously notified by the upper layer to this module.

The *Change Management* layer executes actions to handle the new situations reported by the lowest layer. It is made by a software module, the *EnergyManager*, and its input file, called the *Adaptation Plan*. The *EnergyManager* is informed of the system workload and the status of the processing infrastructure and it uses the energy-aware adaptation plan to decide *when* to reconfigure the infrastructure and *how* to carry it out. It orders reconfigurations when it recognizes a non optimal one: either the system load is low and the performance goal could be fulfilled using less resources, or the load is high, requiring more capacity to satisfy the goal. The energy-aware adaptation plan is received from the uppermost layer, either on demand or when the uppermost layer decides to change it (e.g., because system goal changed).

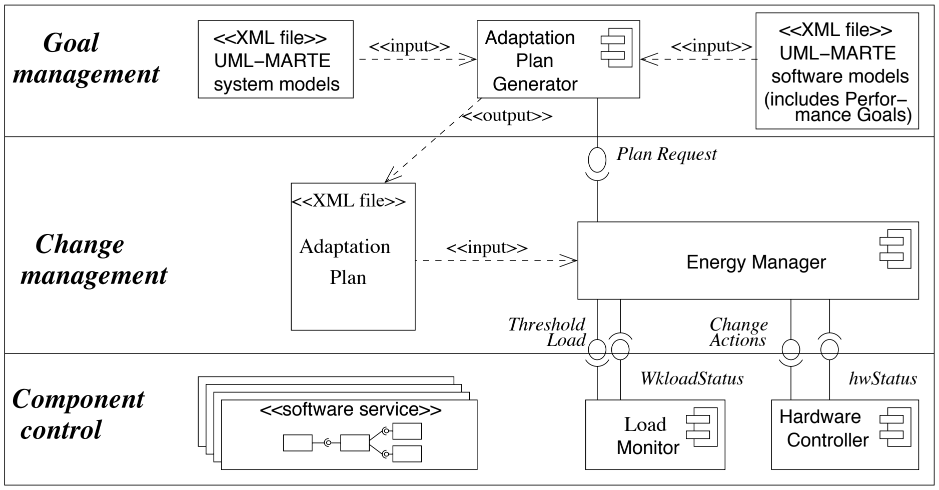

Figure 1: KM-3L adapted to energy management

The uppermost layer is the deliberation, or *Goal management* layer. It consists of time consuming computations to produce a plan to achieve the goal of the framework, in our case to allow the infrastructure to satisfy its performance goals while it spends as less energy as possible. This layer is made of a software component, called *Adaptation Plan Generator*, and a set of software and systems models, represented as UML models. These files are inputs for the *Adaptation Plan Generator*. The *Adaptation Plan Generator* creates plans following a model-driven approach where the software and system models are transformed, following the proposal in [14], into an analyzable Stochastic Petri Net (SPN) model.

The software models will represent the workflow logic of the deployed services (e.g., UML activity diagrams). These models also contain the performance characteristics of the software service (e.g., using the MARTE [15] profile to annotate the previous diagrams). The performance characteristics include: expected workload, performance goals of the service, processing demand and execution probabilities of the activities, and resource sharing. The system models represent the processing platform (e.g., UML deployment diagrams). They include: number of available servers, its processing capabilities w.r.t. power consumption and mechanisms to change power consumption.

The *Adaptation Plan Generator* carries out the SPN analysis to produce the plan, thus obtaining the maximum load the configuration can manage while the required performance goal is accomplished. Section 5 will detail how to generate a plan. Once the plan is generated the task of the *Adaptation Plan Generator* is not finished. It performs a plan evaluation to predict system behavioral characteristics using the generated plan. To execute such prediction a new SPN will be created starting from the previous one and adding information regarding: variable workload, platform energy consumption and the adaptation plan itself. New SPN sub-models will represent each one of the previous concerns, which are explained in detail in subsequent sections (Sections 4, 5 and 6 respectively).

Once the system behavioral prediction has been derived, the goal of this layer is to periodically check whether the plan is suitable, which means verifying:

- whether the system is behaving as expected.
- whether the models of workflow, workload and platform are close to the real behavior. For example, the assumed values for the workflow activities could change due to software upgrades.

If some of these issues are not adequate, this layer will update model parameters and will regenerate the plan.

3. Workload modeling

In order to carry out a proper model-based analysis of system's behavioral properties, we first need an accurate model-based representation of the workload the system is managing. This is not a trivial concern since dynamic systems should be able to cope with highly variable workloads with temporal dependencies.

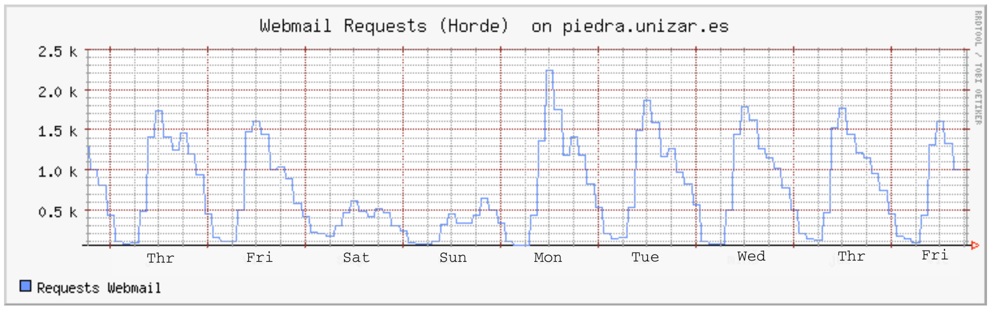

Figure 2: Webmail server, weekly supported workload

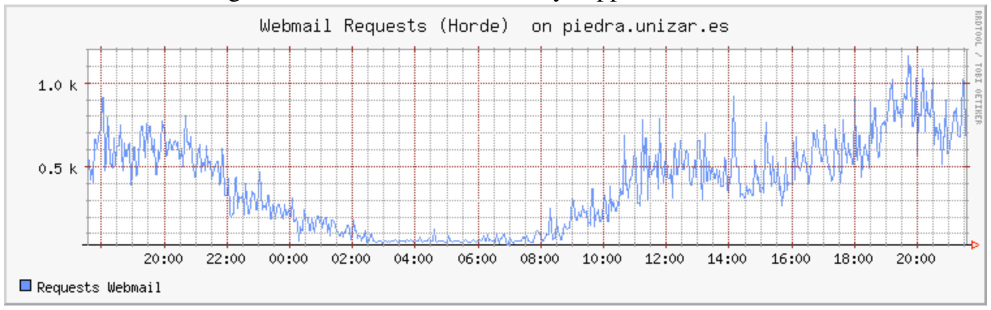

Figure 3: Webmail server, daily supported workload

Classical techniques to model workload, such as those based on phase-type ([16], [17]) or exponentially distributed inter-arrival time of requests, do not consider dependencies and correlations between inter-arrival times. However, these concepts are crucial if the system to be evaluated takes into account its incoming workload to decide its operational mode. Therefore, to evaluate this type of self-adaptive systems, our workload models have to be able to represent both the variability and the temporal dependency. To this end we adopt a SPN model since SPN have been largely used in the literature for this purpose. Indeed, since our analyzable model of the system is based on SPNs, it is an advantage to have the workload model represented in the same formalism in order to be integrated with the rest of the system model.

Let us start considering different granularities regarding workloads' time scale. In the long-term (e.g., time span of a week), it is possible to devise a pattern (or a distribution) that fits the variable workload. But in the short-term (e.g. time span of seconds), the high variability of the workload makes prediction very challenging. In our architecture, a solution could pass through waiting until the system has monitored enough data to acquire a long-term view and then proceed to adapt. However, such behavior will delay adaptation decision far away from the point in time it has been needed. Therefore, it is necessary a prediction method that, only using short-term monitored data, can quickly infer the current workload in the long-term (and then also infer the near future expected workload). Obtaining such prediction method is a real challenge since it should manage multiple long-term variables and obtain their current values managing only partial, short-term, information.

We observe that long-term arrival rates of requests for service can be clearly separated in several states. Following this assumption, our workload model will contain several states representing each one a concrete arrival rate. Figure 2[3] represents a real variable workload supported by a mail server (which receives around one million requests per day and 30,000 login operations). In the Figure, we can appreciate several states: (i) *night* with an arrival rate close to zero and duration around 8 hours; (ii) *working-hours* with an arrival rate around 1,500 requests/minute and duration of 16 hours; (iii) *peak* which is sporadic, short (it lasts for around one hour), it takes place only during working hours and can reach an arrival rate around 1,800 requests/minute; and (iv) *weekend* showing an ar-

[3]Figure taken from https://piedra.unizar.es:8080/public/monitor during the week of 4-12 November 2010.

rival rate around 500 requests/minute and lasting roughly 16 hours. However, the short-term arrival rate of requests is not so regular, making the prediction of the workload state a challenging task. To illustrate such challenge, Figure 3 shows the variability of the workload from Sat 20:00 to Sun 20:00.

To model such workload, we define the $SPN_{workload}$ with a shape like the one in Figure 4. The theory to automatically estimate the parameters of the underlying Markov model can be found in [18]. For the presented example of the webmail server, we set parameters manually to pay attention on the resulting model rather than in the modelling process. The SPN models both the long-term and the short-term workload behavior and it includes:

- as many places as workload states. A token in a place means that the system is receiving requests with the arrival rate associated with that state. Therefore only one of these state places can be marked.
- a timed transition for each state. Such transitions are bidirectionally connected to state places. These transitions inject the workload to the beginning of the workflow. Their firing rate corresponds to the expected workload in each state. In Figure 4, firing rates are denoted as λ_{state}. Since these transitions are linked to state-places, only one of them can be enabled.
- a set of timed transitions to model the state mean sojourn time[4] and probabilities of change between states. For example, transitions T_{N-Wo} and T_{N-We} model the mean sojourn time in *night* state, the sum of their rates must be $8 hours^{-1}$. Moreover, to model the change state probabilities, i.e., five changes per week from *night* to *working* and two changes per week from *night* to *weekend*, it is required that $\lambda_{T_{N-Wo}} \cdot 8 hours = \frac{5}{7}$ and $\lambda_{T_{N-We}} \cdot 8 hours = \frac{2}{7}$, which lead to $\lambda_{T_{N-Wo}} = 5.715 hours^{-1}$ and $\lambda_{T_{N-We}} = 2.285 hours^{-1}$. [5]

The derivation of the unknown parameters λ of the $SPN_{workload}$ is based on the Markov arrival processes (MAP) theory, and in particular on a type of MAP, those called Markov-modulated Poisson Process (MMPP). For the sake of readability, this theoretical part is described in Appendix B.

[4]By *mean sojourn time* we mean the average time the system spends in a given state.

[5]It has been only considered the mean amount of changes between states, so, it has not been considered that the two changes from night to weekend per week should be consecutive.

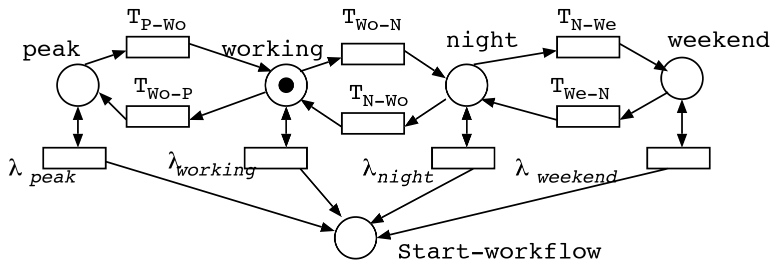

Figure 4: $SPN_{workload}$ model

We have simulated the behavior of the Petri net and compared the obtained results with the real workload in Figures 2 and 3. Figure 5 illustrates the simulation results (token arrivals to place `Start-workflow` w.r.t time): the shape and pattern match with the long-term view of the real server in Figure 2. The long-term view in Figure 5 has been achieved by counting the events generated in slots of 20 seconds. Moreover, zooming in the simulation results (right part of the figure) we can also observe the high variability in the workload that the system is receiving, which makes the workload state prediction be a challenge. See for example that the value marked with circle is higher than the value marked with the triangle, while the long-term view workload supported by the state of the circle (*working*) is lower than the long-term view workload supported in the state of triangle (*peak*). The short-term view in the figure has been achieved by counting the events generated in slots of 1 second.

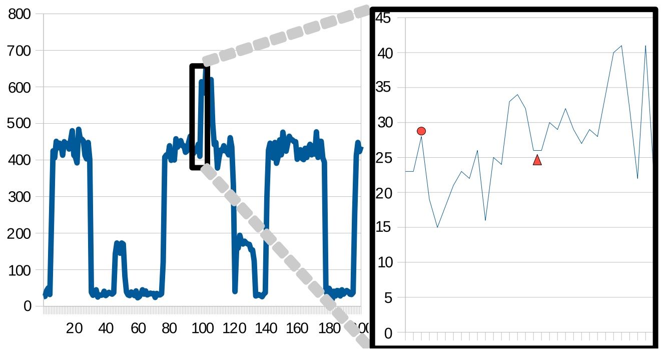

Figure 5: Simulation of the $SPN_{workload}$ model

It is worth noting that the $SPN_{workload}$ part will be "isolated" from the rest of the SPN model that represents the system. The unique element in common between them is *start-workflow* place. In this place $SPN_{workload}$ holds tokens that represent execution requests from users. In turn, the SPN that models the system will delete such tokens and will start a system execution for each of them. Therefore, the rest of the SPN cannot get information regarding which is the active state in each moment (i.e., in which place $state_i$ the token is) or regarding the firing of any transition $T_i \in SPN_{workload}$. At most, the rest of the system can monitor the token generations in *start-workflow* place during a certain period to try to predict the expected workload.

4. Energy modeling and analysis

This section proposes a SPN model that allows the evaluation of the variables related to *energy consumption* and *frequency of the servers*, both taken into account in the adaptation plan. This SPN, in Figure 6, also models the transient state of servers, from switch off to on and vice-versa, which means to embed actions defined in the adaptation plan to manage power consumption. Some places in the SPN will be shared with other subnets (e.g., the workload subnets in previous Section) to make the final SPN model. Indeed, the evaluation of the variables herein presented will be carried out in this final SPN.

From Figure 6 we see that when a switched off server receives the `SwitchOnEvent` it begins its `Booting`. That booting process lasts for $T_{startup}$ time units. When the booting is finished, the server is operative to receive requests and the completion is notified to the energy manager through a token in `BootedEvent` place. When an operative server receives a `SwitchOffEvent` it starts its shutting down process. First of all, its representative token in `OperativeServers` is deleted, meaning that it is no longer available to receive new service requests. A server changes its state from operative to a state `WaitForRequestsCompletion`, where it is finishing its ongoing requests. When all ongoing requests are finally served, it starts the `Halting` process which lasts for $T_{shutdown}$ time units. After that, it joins the pool of `SwitchedOffServers`. Tokens in `SwitchOnEvent` and `SwitchOffEvent` come from the *EnergyManager*. Tokens in `OngoingRequests` are generated by the workload balancer and deleted when a request finishes its execution, these tokens store information about the server that is executing the request. Tokens in `OperativeServers` are looked up by the workload balancer when it has to decide the target server for a request.

Regarding the *energy consumption variables*, we will evaluate in the final SPN the following:

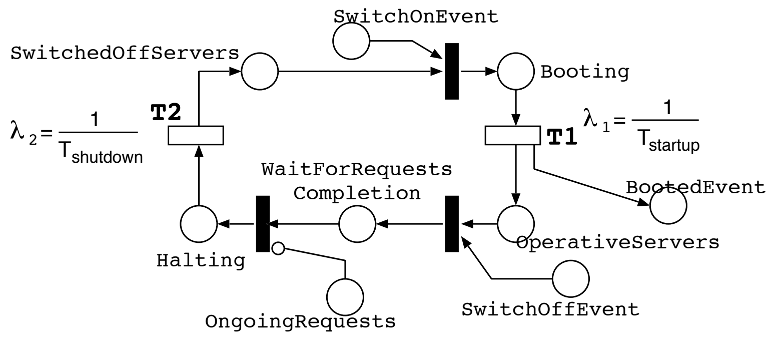

Figure 6: SPN modeling the states of servers

1. Mean power consumed by switch on and off processes,

$$W_{on-off} = C_{startup} \cdot \chi(T1) + C_{shutdown} \cdot \chi(T2)$$

where $\chi(Ty)$ is the mean throughput of transition T_y. $C_{startup}$ and $C_{shutdown}$ respectively represent the energy consumed by the server at start-up and shutdown.

2. Minimum power consumption of a server, $W_{standby}$, includes all constant consumptions that do not depend on the working frequency. Mean aggregated power consumption of servers is

$$W_{AggreagedStandby} = W_{standby} \cdot (E[\#OperativeServers] + E[\#WaitForRequestsCompletion])$$

where $E[\#Px]$ is the mean number of tokens in place P_x.

3. Maximum power consumption of a server, W_{max}, considers when a server is busy and working at its maximum frequency.

4. Since voltage supply limits the maximum operative frequency of the circuit approximately to a linear factor, then following [8, 4], we merge dynamic frequency scaling and dynamic voltage scaling, and we obtain that power consumption is proportional to the cube of the working frequency. Therefore, power consumption of a server in an operational frequency will be

$$W_{freq_i} = (W_{max} - W_{standby}) \cdot (opFreq_i)^3.$$

Finally, the total amount of power consumed by a server working at frequency $OpFreq_i$ is

$$W_{server_i} = W_{standby} + W_{freq_i}.$$

In the SPN, the mean power consumption of a single server is calculated as

$$W_{mean} = \sum_i W_{server_i} \cdot P(\#Frequency = i + 1),$$

where $P(\#p = n)$ means that the probability of the number of tokens in place p is equal to n. Meaning of place $Frequency$ is unveiled in the following.

Regarding *servers processing frequency*, dynamic frequency and voltage scaling allow varying working processors performance and reducing their power consumption. Although the working frequency could ideally range between 0 and 100% of processor capabilities, real working frequencies used must be discretized. Therefore, as in [4], we assume that the actual server frequency is restricted to a value within a set of operational frequencies $FreqSet$. We consider $FreqSet$ made of a base frequency $BaseFreq$ and increments $BaseInc$. Therefore

$$FreqSet = \{OpFreq_i\} \mid OpFreq_i = BaseFreq + i \cdot FreqIncr \\ \wedge\ (i \geq 0)\ \wedge\ (OpFreq_i \leq 100\%).$$

For example: $BaseFreq = 50\%$, $BaseInc = 10\%$ and $FreqSet = \{50\%, 60\%, 70\%, 80\%, 90\%, 100\%\}$. Anticipating the description of the Petri net model in Section 5 (Fig. 9), tokens in place `Frequency` will represent servers processing frequency. It will contain from 1 to $|FreqSet|$ tokens (from minimum to maximum frequency). A reconfiguration in the server frequency will obviously change the number of tokens in this place.

To keep the model simple, we have not modeled other variables related to power-aware adaptation such as savings in the cooling system.

5. Performance and energy trade-off

This section explains the complex process to create an energy-aware plan that cares of performance requirements. To ease the explanation, the process is decomposed in two steps: the first one, described in Subsection 5.1, illustrates the generation of a basic plan, while the second one, in Subsection 5.2, describes its optimization. Subsection 5.3 proposes a Petri net model for the plan described in Subsection 5.2. Finally, Subsection 5.4 presents a Petri net that results from merging all the PNs obtained so far. It will be useful to carry out a trade-off evaluation (performance and energy) of the system.

5.1. Generation of basic-plan

An energy-aware adaptation plan will be a set of system *configurations*, that meet the performance goals using minimum energy, and *actions* to change among configurations. A configuration defines the number of active servers as well as the frequency they are working at. Hence, actions to change a configuration will just mean to switch on/off servers and/or change their working frequency. Using the number of servers and their working frequency, the power in each configuration can be calculated.

A configuration also identifies a threshold that corresponds to the maximum system load the configuration can manage. Energy manager uses information in the plan to accommodate the system configuration to the most suitable one regarding the current number of requests (system load). System load is an information the plan receives from the lower layer, which indeed monitors the system.

In the following the process to generate the energy-saver adaptation plan is explained (Table 1 will help the process understanding).

1. Generate a SPN model of the system workflow that includes the required processing demands. Set the capacity of servers to a minimum (e.g., in Table 1, one server, k=1, at its minimum frequency, $OpFreq_0 = 50\%$).
2. Evaluate the SPN to discover the mentioned threshold, i.e., the maximum load ($Nrequest$) it can manage while the performance goals are satisfied. Compute power consumption (W_{server_i}) for this configuration. (In Table 1, 27 and 16.3 respectively for the first case.)
3. Increase the server frequency (which means modifying the SPN) and go to step 2. Repeat this step for all the frequencies the system has to manage, i.e., $|FreqSet|$.

At this point we have completed one row of the table. It is natural to assume that, if a server working at frequency $OpFreq_i$ can manage $Nrequests_i$ and spends W_{server_i} power, *k* independent and concurrent servers working at the same $OpFreq_i$ are able to manage $k \cdot Nrequests_i$ and they spend $k \cdot W_{server_i}$ power. Applying this, we compute the rest of rows in the table multiplying the first row values by the number of servers that represent each row. We will consider as many servers as available in the infrastructure. As a result, we have generated a table that contains all possible system configurations and, for each configuration, its power consumption and the load it is able to manage (the complete table is not displayed).

Using data in the generated table, Algorithm 1 can be applied to generate the basic adaptation plan. This plan contains an ordered list of a subset of possible

		Percentage of frequency, $OpFreq_i$					
		50%	60%	70%	80%	90%	100%
k=1	*Nrequests*	27	32	37	43	48	54
	W_{server}	16.3	18.4	21.3	25.1	30	36.2
k=2	*Nrequests*	54	64	74	86	96	108
	W_{server}	32.6	36.8	42.6	50.2	60	72.4

k means number of active servers

Table 1: Information required to create an adaptation plan

configurations (called *suitable* configurations) as well as threshold values indicating the moment to change from one configuration to another.

As a result we can distinguish two kinds of adaptations: those that only require to change the frequency and those that require to change the number of working servers -most probably, together with their frequency.

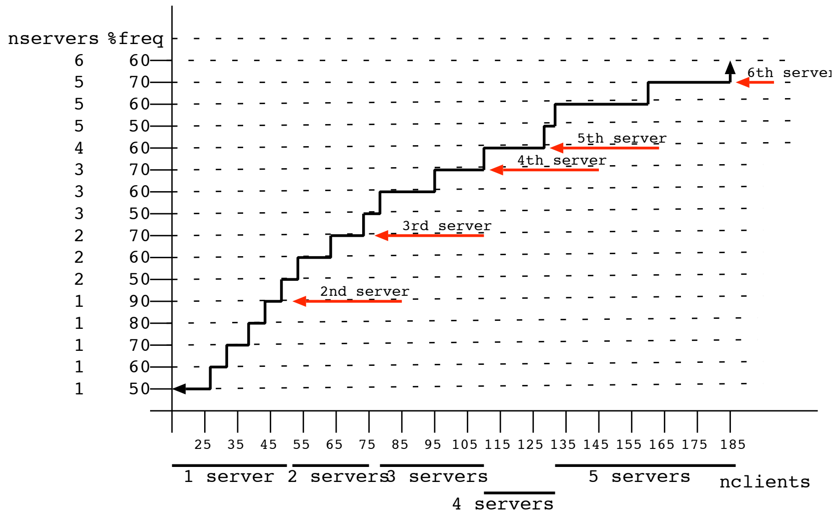

Figure 7: Graph for system reconfigurations

An example of configuration in Table 1 is the system working with only one server ($k = 1$), with frequency 60% and then with thresholds 27 and 32 requests, in this case the power consumption is 18.4. System load ranging between 0 and 48 can be managed by only one server and changing only the frequency. However,

Algorithm 1 Basic-plan generation

```
Require: Table with $Nrequests$ and $W_{server}$ (dimension KxF)
Ensure: Basic adaptation plan.
1: set $k = 1$ {considered number of servers, row index}
2: set $f = 1$ {considered frequency, col index}
3: set $Plan \leftarrow EmptyPlan()$ {create empty plan}
4: set $currentConf \leftarrow Table[k][f]$
5: $Plan \leftarrow AddToPlan(Plan, CurrentConf)$
6: set $cadidateFreq$
   {Search rest of suitable configurations until finish the table}
7: while $k < K$ do
8:    $cadidateFreq \leftarrow GetBestInRow(k + 1, currentConf)$
9:    if $IsBetterToContinueWithSame$
      $Servers(k, f, candidateFreq)$ then
10:      $f \leftarrow f + 1$ {Next configuration increases frequency}
11:      $currentConf \leftarrow Table[k][f]$
12:   else
13:      $k \leftarrow k + 1$ {Next configuration increases servers}
14:      $f \leftarrow candidateFreq$
15:      $currentConf \leftarrow Table[k][f]$
16:   end if
17:   $plan \leftarrow AddToPlan(Plan, currentConf)$
18: end while
    {Add last row of table to plan}
19: while $f < F$ do
20:   $f \leftarrow f + 1$
21:   $plan \leftarrow AddToPlan(Plan, Table[K][f])$
22: end while
23: return Plan
```

when the number of requests exceeds 48 it will be better to change to a configuration with 2 servers and frequency at 50% since power consumption is 32.6 instead of 36.2 offered by the configuration that only changes frequency. So, the configuration that uses one server at 100% will never be used. Figure 7 shows a basic adaptation plan in a chart. It depicts reconfiguration points in function of the workload, considering a six-server infrastructure.

5.2. Reconfiguration rate mitigation

The basic-plan suffers periods with high rates of switching on and off of the servers, which is a real drawback for two reasons. First, the time spent in booting and halting can be too high w.r.t. the real working time, then the energy spent in switching tasks is not spent in serving requests. Second, the more the switching rate, the more the wear and tear of servers.

To reduce the number of switch on and off of the servers we propose to use reconfiguration limits with hysteresis. In other words, the *Nrequests* threshold value indicating when the system changes between two neighboring configurations will not be unique but composed of a couple of numbers,$Nrequests^{dec}$ and $Nrequests^{inc}$, according to whether the system tendency is reducing its power (moving from the high energy consuming configuration to the lower) or increasing it (moving from the lower consuming configuration to the higher). Therefore, the association between the supported load and the system configuration will not be unique.

The meaning of these new limits are: $Nrequests_s^{inc}$ corresponds to the threshold amount of requests to change from configuration s to $s+1$. $Nrequests_s^{dec}$ corresponds to the threshold amount of requests to change from configuration $s+1$ to s. In Figure 8, bold continuous line shows the $Nrequests^{inc}$ values, which are very similar to the previous $Nrequests$ while bold dashed line depicts $Nrequests^{dec}$. In that graph, the hysteresis length is equal to 2 steps, i.e, the dashed line is moved two configurations above the continuous line. Therefore, $\forall s \in \{2..S\}$ $Nrequests_s^{dec} = Nrequests_{s-2}$. For example, supposing that configuration s is the one that uses three servers working at 60% of its frequency, and looking at the change from configuration s to $s+1$ (i.e., use three servers working at 70%), $Nrequests_s^{inc} = 94$. However, looking at the change from configuration $s+1$ to s, $Nrequests_s^{dec} = 74$; value which corresponds with the previous $Nrequests_{s-2}$.

Therefore, the higher the hysteresis length, the lower the reconfiguration rate and the less the wear and tear, but higher the mean power consumption, since the system spends more time in a configuration that consumes more energy than necessary to deal with the received workload. Section 6 gives an example of the tradeoff between these characteristics.

It is possible to observe small differences between bold continuous line in Figure 8 and black line in Figure 7: these are due to corrections made when the reconfiguration involves to turn on a new server. These corrections are intended to mitigate the non quality satisfaction during the booting time of the newly switched on server. We start to switch on a server few moments before

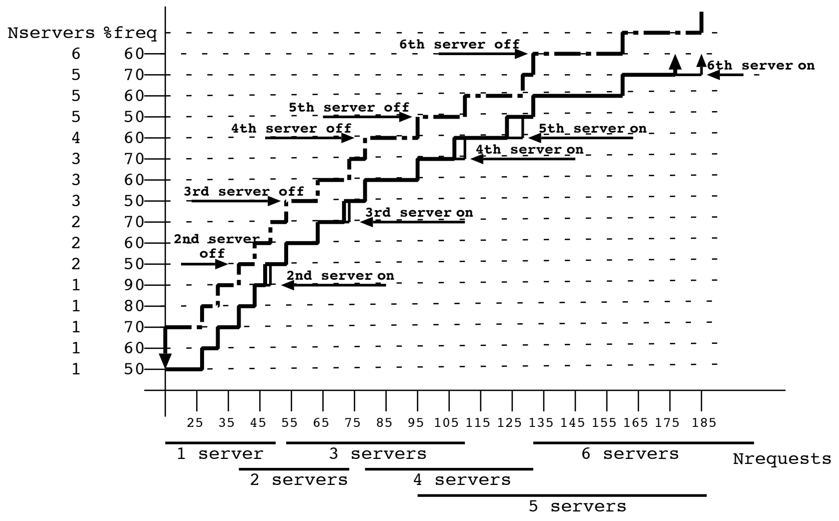

Figure 8: Graph for reconfiguration (with hysteresis)

it will compulsorily need to maintain the required quality. This helps to have it already booted and completely operative when it has to be used. Among the multiple manners to decide how much the booting moment should be brought forward, we have chosen to calculate it as a proportion of the step length called Bring Forward Proportion (BFP). Thus, when the system is in a configuration $Conf_j$ such that the immediately consecutive $Conf_{j+1}$ uses one server more, the adaptation order will take place when system load reaches $Nrequests^{inc}_{Conf_j} = Nrequests^{inc}_{Conf_{j-1}} + \lfloor (Nrequests_{Conf_j} - Nrequests^{inc}_{Conf_{j-1}}) \cdot BFP \rfloor$.

As an example, let us consider the difference between the bold continuous line in Figure 8 (in $Nrequests = 111$) and black line in Figure 7 ($Nrequests^{inc} = 96 + \lfloor (111 - 96) \cdot 0.75 \rfloor = 107$) with a BFP value equal to 0.75 and focusing on the moment to order the switching on of the 4th server.

Thus, without hysteresis and BFP, three servers were used when the load of the system ranged from 74 to 111. With the new improvement, three servers can be used to manage from 52 to 107 requests, but what happens concretely, is that two or three servers are used to manage from 52 to 74 requests, exactly 3 servers for the range 74-76, three or four servers to deal with requests from 78 to 96 and three, four or five servers manage requests from 96 to 107.

The decision to set a suitable value for the hysteresis proportion is shown in Section 6 by means of the evaluation of an example with different proportion values.

5.3. Petri net model of a Plan

The Petri net in Figure 9 models the system reconfigurations that an energy aware adaptation plan could carry out, in this example configurations $Conf_0$, $Conf_1$, $Conf_2$, $Conf_s$ and $Conf_{s+1}$ of the plan are depicted. Places representing configurations, $Conf_i$, are in mutual exclusion and can contain at most one token.

Transitions in the right hand side (t_4 and t_5) allow upgrading the power of the system changing to a configuration that increments its $Frequency$ when the system is supporting a load that exceeds the configuration threshold $Nrequest^{inc}_{Conf_i}$ (weight of the test arc linked to `systemLoad`). Transitions in the left hand side (t_1 and t_2) allow downgrading the power of the system changing to a configuration that decrements its $Frequency$ when it receives less than $Nrequest^{dec}_{Conf_i}$ requests, in this case an inhibitor arc (those having a circle at the end) prevents the firing of the transition when the number of tokens in `systemLoad` is more than $Nrequest^{dec}_{Conf_i}$. `SystemLoad` place will be filled by the workload subnet (Section 3), and its tokens removed by a timed transition with firing rate $\frac{1}{MonitoredTimeSpan}$, so it accounts for the number of requests the system has received during the lasts *monitoredTimeSpan* seconds.

Some downgrades in the system configuration imply to increment the frequency and to decrement the number of servers, transition t_3 represents them. In this case, the $Frequency$ is increased with the difference of frequency between configurations ($Freq(Conf_i) - Freq(Conf_{i+1})$). While the number of servers is decremented sending an event (token in $SwitchoffEvent$) to start the switch off process.

On the other hand, some upgrades of system configurations imply to decrease frequency and increase the number of servers, they are trickier and need of two transitions, in the example t_6 and t_7. In this case, the change of frequency and the switch on of the servers cannot be concurrently executed since switch on entails booting time. So if the frequency is changed when the new servers have not been yet added (servers are booting), the servers currently working will be the ones suffering the frequency change and they will provoke a transitory quality degradation of the system instead of its power enhancement. Then we split up the upgrade process in two steps. In the first one, t_6 switches on the server (tokens in `SwitchOnEvent` and `WaitingForBooting` places). During this booting time the system works at the frequency in the source configuration (no degradation). When the server is already booted (token in $BootedEvent$), the frequency is decreased using transition t_7, and the system reaches the new configuration.

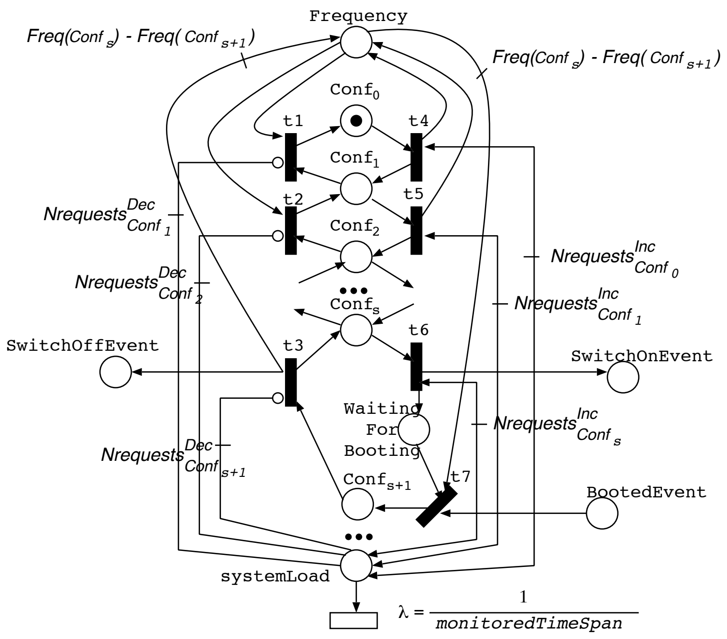

Figure 9: Petri net modeling the adaptation plan behavior

5.4. The Petri net for trade-off evaluation

The Petri nets for the adaptation plan, the workload, the state of the servers and the software service are merged to create a new one where we carry out the proposed trade-off analysis. Figure 10 depicts an abstract view of this Petri net, where the places that are interfaces clearly emphasize how the nets interact. Although we do not present in the paper a Petri net of a software service, Figure 12 illustrates the workflow of a software service, and we obtain the corresponding Petri net automatically, using ArgoSPE [19].

6. Deployment and Evaluation

The architecture proposed in Section 2 is a reference one, hence different deployments can be accomplished. Figure 11 presents a deployment in which the computing platform is made of `servers`, where the software services in the Component control layer are deployed. The bottom layer of the architecture is completed with the `LoadMonitor` and the `HwController` which are software modules that reside in the hardware that receives the requests from clients;

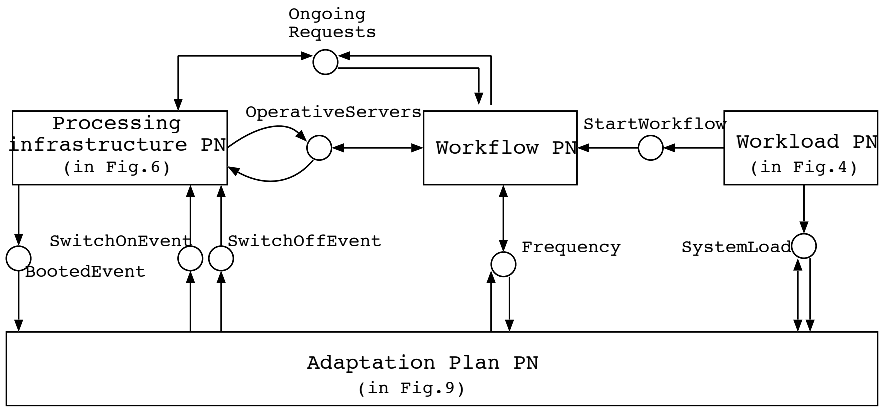

Figure 10: Abstract view of the Petri net for evaluation.

separated hardware could also be used. The software module that comprises the Change management layer, `Energy Manager`, is deployed in separate hardware that only communicates with the other two layers for sending and receiving the orders and information specified in the architecture. Finally the `Adaptation Plan Generator`, which is software that evaluates SPNs as explained in Section 5, is deployed in a high performance computing platform to create the plans on demand. This service could even be provided by a third-party in the cloud.

6.1. Evaluation framework

The SPN in Figure 10 represents all the elements in the deployment, although some implementation was required to carry out evaluation. The requests of the clients are modeled as proposed in Section 3. This fact gives us the advantages previously presented as well as the choice of performing a plethora of experiments as discussed in Section 7. The actions of the `HwController` are embedded in the SPN in Figure 6. The software services are simulated by the SPNs that represent them, note that the main interest is to simulate the time they spend, which is accurately represented by the timed transitions of the SPN. These SPNs are obtained from the UML models of the software services using the ArgoSPE tool [19]. Regarding the middle layer, Change management, it is embeded in the SPN in Figure 9 which represents the adaptation plan. We have implemented a Java program that creates the basic plan, evaluating SPNs and applying Algorithm 1. The evaluation was carried out using the GreatSPN tool [10]. The program also creates the plan with hysteresis. The resulting plan gives the param-

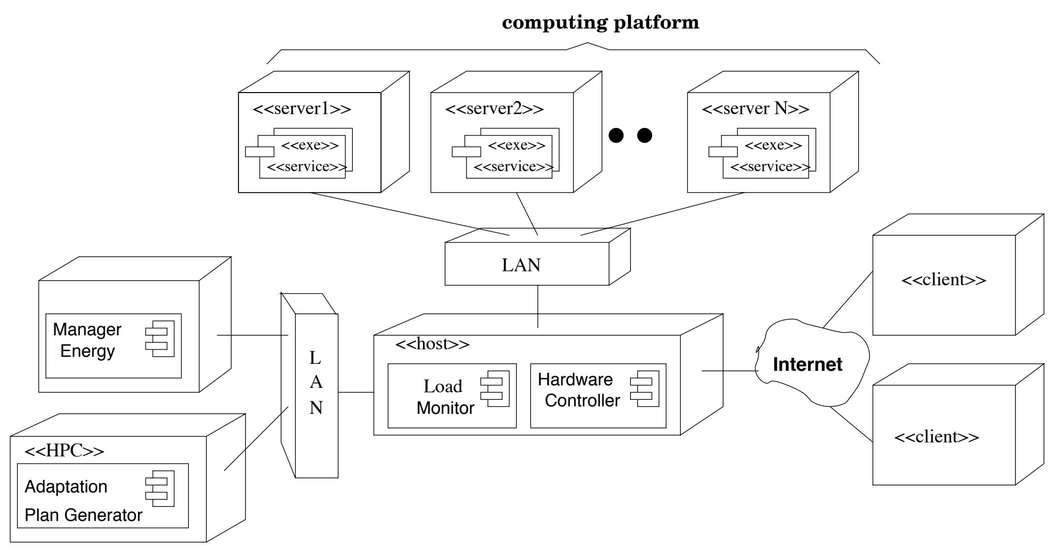

Figure 11: Deployment of the reference architecture

eters for the SPN in Figure 9. Hence, we have created a model-based framework, that being able to evaluate our approach, frees us from developing this expensive deployment, especially in regard to acquire or rent a real computing infrastructure.

Our purpose being a model-based evaluation, we consider interesting to summarize the differences between this kind of evaluation and an hypothetical evaluation carried out using the deployment in Figure 11:

- We do not have "real" clients but a model of workload. However consider that this part of the deployment does not belong to the architecture, i.e., to our contribution. Moreover, we have developed a theory in the paper to appropriately leverage the workload.

- We have not used an expensive computing platform made of hundreds of servers. However our SPNs models carefully represent the workload they support and our plan considers their consumption, frequency and booting and shutdown times.

- The `HwController` has not been implemented since we do not have the computing platform it manages. However, this task just means to program the WoL facility of the servers and the remote control of the frequency.

- The `Load Monitor` is not necessary in our evaluation since the workload is generated by our model.
- The `Adaptation Plan Generator` has been implemented for our model-based evaluation and it could be reused in the deployment in Figure 11.

6.2. Example of evaluation: relay mail server

The model-based framework above described has been used to evaluate a simplified version of a relay mail server, a kind of system very common for enterprises and institutions. Relay servers use to be replicated to cope with highly dynamic workloads usually being a few the number of replicas, except for extremely large mail providers.

The server receives requests to route mails to destinations, from both external and local users. First activity is to accept the service. For example, mails from local users are allowed to be delivered to anywhere, while external users could only be allowed to send mails to local users, then avoiding *open-relay* risky configurations. For accepted mails, the relay analyzes the content regarding security, trying to mark viruses, spam or phishing. Safe mails are delivered with a header indicating the analysis result. Mails containing viruses are rejected. The destiny of safe mails can be either an external relay server, the one of the addressee, or the own company mail inbox server. Finally, the relay server writes a log about the operations performed, time stamps and related information. Figure 12 depicts the workflow, using UML, as well as the performance information, in this case annotated with the standard MARTE [15] profile: a) mean host demand for each operation and b) system routing rates as probabilities. Host demands annotations assume the server working at its maximum frequency. The performance requirement states that mean response time for a legitimate request should be less than two seconds.

Workload model. For the sake of simplicity we adopt the monitored workload of the University web server presented in Section 3. So, the workload is the one depicted in Figure 2 and the corresponding Petri net model the one in Figure 4. Let us assume the following mean arrival request rates per minute in each state: 1800 for *peak*, 1300 for *working*, 100 for *night* and 500 for *weekend*. They are modeled by transitions of name λ_{state} in the Petri net. Rates of transitions that model state changes were explained in Section 3 and in Appendix B. Evaluating this Petri net in isolation (without considering the workflow Petri net), we obtain that its long term mean inter-arrival time is 943.4 requests per minute.

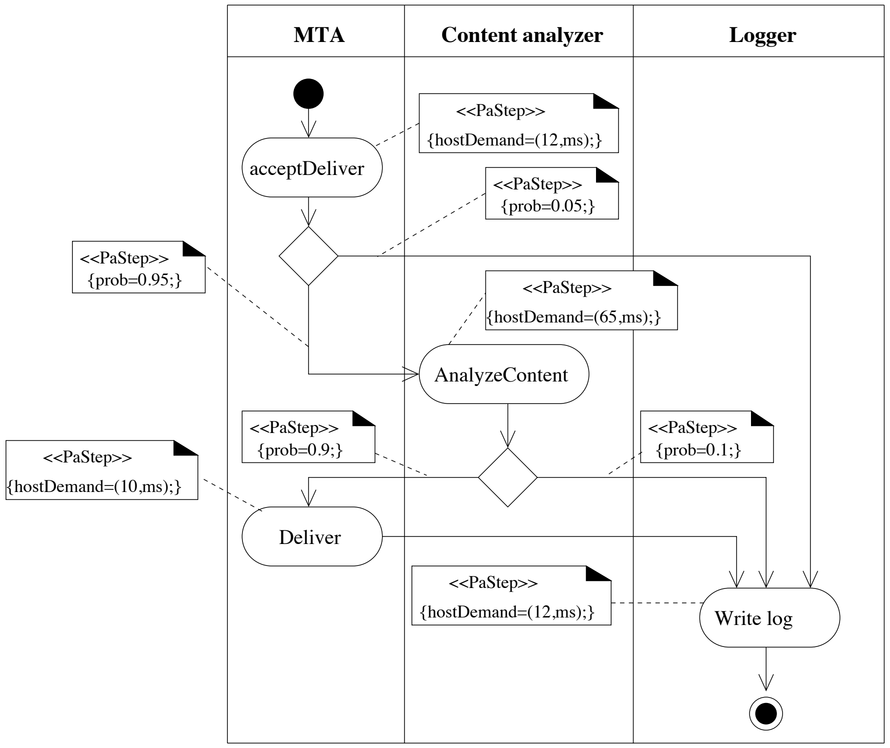

Figure 12: UML activity diagram of a relay server

Characteristics of the processing resources. We have supposed a set of identical servers and Round-robin technique to balance requests. We have followed classical techniques to create the SPN that models the load balancing technique, which is inserted in between the workload and workflow SPN submodels. The characteristics of a server are:
1. Maximum power consumption, $W_{max} = 100W$.
2. Idle power consumption, $W_{standby} = 15W$.
3. Others power consumptions, $C_{startup} = C_{shudown} = 6000\ Joules$.
4. Frequencies range from 1600MHz to 3200MHz in steps of 266.6MHz. Thus, the set of frequencies is $\{50\%, 58.33\%, 66.66\%, 75\%, 83.33\%, 91.66\%, 100\%\}$.
5. Booting and shutdown times, $T_{startup} = T_{shutdown} = 1min$.

Not energy-aware deployment. We first conduct a study intended to devise the necessary amount of servers to cope with the performance requirement. This study does not heed about energy, so the servers are working at maximum performance and power consumption. The workflow in Figure 12 is translated into a SPN following the method in [14], let us call it SPN_{wkf}. The workload model is simplified and split to only consider *working* and *peak* periods, the worst cases. These nets together with the Round robin PN are attached to the SPN_{wkf}. As a result, we get two SPN we call $SPN_{wkf}^{working}$ and SPN_{wkf}^{peak}. We evaluate these nets for a different number of servers using the GreatSPN tool [10], and applying Little's law[6] we obtain the execution mean response times. We obtain that, if the platform consists of one or two servers, the system cannot satisfy required response time in *working* or *peak* states. Actually, system is neither able to manage the workload, and mean response times tend to infinite. However, for a three server platform, system satisfies the required performance in both *working* and *peak* states, since the obtained mean response times in this case are 0.16 and 0.73 seconds respectively. So, the system would be deployed in a three server platform. Servers power consumption using this solution would be $100W \cdot 3 = 300W$.

Energy-aware adaptation. Previous not energy-aware study tells us that probably the three server solution is wasting energy since not all processing capacity is needed. Hence, we have to apply the energy-aware plan developed in Section 5 to find suitable configurations of servers for all the states in the system.

Firstly, we develop a table, as the one explained in Section 5, by evaluating the SPN that represents the workflow. This Table 2 embeds the different system configurations and shows for only one server its power consumption and the number of concurrent requests it can manage. Remember that under the assumption of "servers independence" we can obtain new rows for more servers just multiplying results in the first row.

	Operative frequencies (percentage)						
	50	58.33	66.66	75	83.33	91.66	100
W_{server}	25.6	31.8	40.1	50.8	64.1	80.4	100
Nrequests	10	12	14	15	17	19	21

Table 2: Power consumption and concurrent requests capacity

From this table the energy-aware plan is generated following indications in

[6]Little's law establishes that the average number of customers in the system is equal to the request arrival rate multiplied by the average time a customer spends in the system.

Section 5 and improved following the hysteresis in Subsection 5.2. We have generated three plans, considering different values for hysteresis step length, 1, 2 and 3. In the following, we refer to each plan as P_{H1}, P_{H2}, and P_{H3} respectively. The plans account for requests received in the last 2 seconds (*monitoredTimeSpan*). Figure 13 depicts P_{H2} as a graph. It shows that there are 11 different configurations and that the three servers deployment is used when system load exceeds 22 requests.

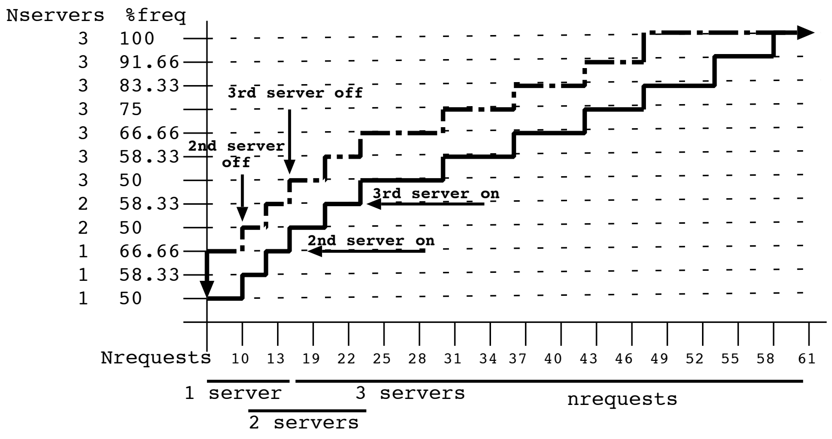

Figure 13: Graph for system reconfigurations

We evaluated the generated plans and we obtained results for the steady state system execution:

- Mean time between consecutive booting processes are 33, 38.5 and 51 minutes for P_{H1}, P_{H2}, and P_{H3} respectively. It can be seen that the more step length of hysteresis, the less frequent booting processes are. Therefore, for each plan, expected mean power consumption of a server due to booting process is 2.86W, 2.6W and 1.96W for P_{H1}, P_{H2}, and P_{H3}, respectively, calculated as $\frac{6000J}{booting(P_{Hx})min \cdot 60seg/min}$.
- Percentage of time a server is turned on for each plan should be 78.3%, 79.4% and 81.2% . This has been directly acquired from mean number of tokens in `operativeServers` and `WaitForRequestsCompletion` places
- Percentage of time that the energy manager orders the infrastructure to work in each frequency is shown in Table 3.

	Operative Frequencies (percentage)						
	50	58.33	66.66	75	83.33	91.6	100
$\%\ time\ P_{H1}$	19.8	13.6	19	18.3	17.4	8	3.7
$\%\ time\ P_{H2}$	20	9.2	11.4	18.3	22.7	12	6.4
$\%time\ P_{H3}$	20.9	13.6	0.8	9.5	27.3	18.2	9.7

Table 3: Percentage of time spent in each operative frequency

Using all this information, we calculate the average power consumption of the system. Results are: 129.8W for P_{H1}, 140.6W for P_{H2} and 150.0W for P_{H3}. Therefore, our adaptation plans should make the system save 56%, 53.1% and 49.9% respectively w.r.t the non adaptive solution (calculated as $100 \cdot (300 - Power(P_{Hx}))/300$).

7. Experimenting with variable workload

The example illustrated in the previous section showed that the application of the proposed adaptation plan could lead to more than 50% of energy saving. The results obtained so far are tied to a given workload pattern. In this section, we analyze different types of workload, trying to understand if the results present a general validity or if they are related to specific situations. Hence, we expect to answer questions such as: "does it exist a workload state making the system unstable?" or "what could happen if the mean arrival rate of a workload falls just over the value where the adaptation plan suggests to increase the executing platform by one server?". Answering these questions would increase or decrease the trust in the proposed approach and it could lead, for example, to the identification of critical workloads that require continuous system reconfigurations, thus deteriorating the energy consumption and servers wear and tear.

To this end we have performed several experiments, described below, whose goal was covering a wide range of workload rates to discover the existence of a possible critical workload state.

For the sake of system stability, while the state of the workload is not changing, the adaptations to activate servers should be close to zero since they take time (e.g., *booting* time). Indeed, in this case, high spike or deep valleys are not related to an increment or a decrement in the future workload, rather they are random events showing momentary workload variations. Therefore, a reconfiguration would entail to come back to the previous configuration in a very short time. In the worst case, it is possible to have a set of servers wasting time and energy turning on and off continuously, instead of processing requests.

To assess our approach, we have studied the behavior of the plan generated in Section 6 under several workload states, where each state has a different request arrival rate. Specifically, we have considered arrival rates from 50 to 2500 requests per minute in steps of 50 arrivals per minute. So, we have evaluated 50 kinds of workload rates. Furthermore, since the hysteresis step length of the adaptation plan was conceived to reduce these reconfiguration rates, we have included in our experimentation also different lengths of the hysteresis step. In this way, it is possible to evaluate whether the hysteresis-based adaptation plan is effective to actually mitigate the amount of unnecessary reconfigurations.

For the completeness of the study, we have performed four different studies, using hysteresis step lengths ranging from 1 to 3. The obtained results are depicted in Figure 14. The graph shows the mean rate of unnecessary adaptations that modify the amount of active servers by varying the requests arrival rates. It can be seen that for an arrival rate of less than 100 requests per minute, the system is stable because none of the plans proposes unnecessary reconfigurations. The same happens for arrival rates above 1350 requests per minute. In the former case, the system is stable using a 1-server configuration. In the latter, the system is stable using always the 3-servers configuration.

Between 950 and 1350 requests per minute, the adaptation with hysteresis step length equal to one proposes some unnecessary reconfigurations. Indeed, there are random valleys in the short term arrival rate that deceive the energy manager into changing to a 2-servers configuration, and then come back to the 3-servers one. The adaptation plans with hysteresis step length two and three, instead, are stable.

Between 250 and 500 requests per minute, it is possible to observe an increase of the rates of unnecessary reconfigurations of plans with hysteresis step of length equal to one and two.

Referring to the example of the previous section, this is the main reason of the observed difference between the value of P_{H1} and P_{H2} mean time between consecutive booting processes w.r.t. P_{H3}. Indeed, the mean arrival rate in *weekend* workload state falls in this range, which causes a certain instability in the system configuration. However, increasing the hysteresis step length to three, the reconfiguration rate decreased by a factor of 35.2% and of 24.5% respectively, mainly due to the avoidance of a number of unnecessary reconfigurations.

The adaptation plan with hysteresis step length equal to three also shows a peak in unnecessary reconfiguration rates, that is shifted to the right w.r.t. peaks of plans with hysteresis step lengths one and two. However, as expected, this peak

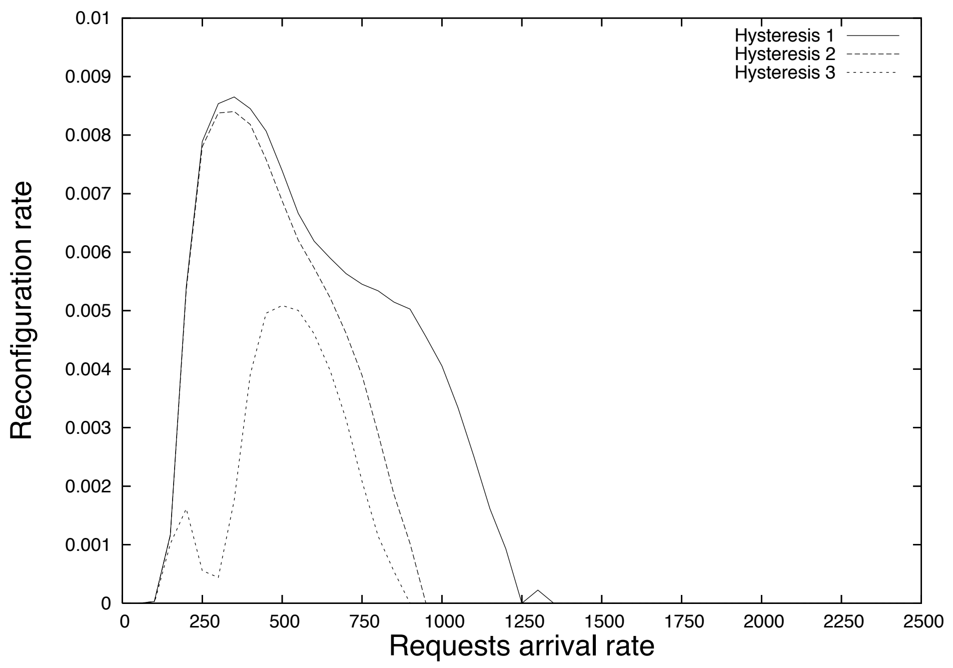

Figure 14: Reconfiguration rates for workload states with arrival rate $50r,\ r \in \{1..50\}$

is much lower than the peak of previous plans, indicating that the increase in the hysteresis step length reduces the reconfiguration rate so increasing the system stability.

Experimenting with variable workload, we can conclude that it is possible finding workload states leading to unnecessary system reconfigurations, but the introduction of hysteresis helps in reducing the rate of reconfigurations. Specifically, adding hysteresis to the approach, we have experimented that it is possible to define suitable hysteresis step length values allowing the system to execute in a stable manner. We observed that increasing the length of the hysteresis step it is possible to reduce the mean power consumption and the reconfiguration rate. Finally, note that depending on the cost of servers and its wear and tear in each switching, the optimum value of hysteresis step length varies.

8. Related work

In the last years, as outlined in [20], the topic of reconfigurable and self-adaptive computing systems has been studied in several communities and from different perspectives. The *autonomic computing* framework is a notable example of a general approach to the design of such systems [21, 22]. Our work lies in the area of models for self-adaptation of systems able to guarantee the fulfillment of performance requirements under variable workload and reducing energy consumption. Therefore, hereafter, we review works appearing in the literature dealing with (i) *dynamic variable workload*, (ii) *energy waste reduction* and (iii) *trade-off between energy consumption and performance*.

Workload. Patterns for workload recognition and characterization have been studied in [23]. Differences between systems analysis depending on whether the considered workload is open, closed or partly open are explained, and difficulties to characterize the workload and difficulties to create and set up a suitable generator are discussed. For the problem we are dealing with, the more suitable workload types are open or partly-open. Moreover, we study systems under highly variable open workloads [24] with temporal dependencies. Besides, in this work we concentrate on the modeling of workload patterns that can be separated into phases, Section 3 leverages this aspect.

Energy wastes. In [25] the authors outline a research agenda to reduce energy consumption in server clusters. The main proposal here is to improve server efficiency in terms of energy spent for each service request. The method focuses on reducing energy consumption by turning off the surplus of processing capacity when the current workload, which fluctuates, does not need it. With respect to this work, we manage the server frequency and the cubic relation with power consumption, which increases the energy saving with respect to only switching on and off servers.

Authors in [26] have considered dynamic allocation of resources and deal with multi-tier applications. Although, they do not directly address the concept of energy savings, methods given in this paper can be applied for energy consumption reduction. In [27] the management of energy consumption in data centers is studied through optimization problems taking into account: frequency scaling, servers booting times and hysteresis. We also assume these issues but we make slightly different assumptions to discover appropriate settings for each workload. For example, to accommodate an increased workload in the platform, in [27] the plan could dictate switch on or off the machines. In our case, with increasing workload if you have to select a new configuration of hardware, we decided to not reduce the number of servers. Therefore, in this case we can reduce the number of reconfigurations.

Trade-off between energy consumption and performance. This aspect has been largely investigated in hardware design, in network communities and in battery-powered devices. However, this investigation applied to hosting centers is much more recent. In [9, 1] the importance of the problem of energy wastes is recognized. They treat the problem from different points of view, such as the consumption from hardware devices, operating system or software applications. They sum up previous efforts in the field, raise current problems and devise ways to reduce the energy consumption.

[8] evaluates five strategies to save energy: two strategies manage processor frequency, another one switches on and off servers and the last two result from the combination of frequency and number of servers management. The authors study the performance degradation of applications with respect to the strategy used. In our work, we propose to generate an adaptation plan that uses the same techniques as in their latter strategy. Besides, we share the modeling of servers startup, shutdown and waiting for ongoing requests times. To predict execution demands of requests from each user our analyzable SPN models include more fine-grained information.

The goals of the work in [4] are close to ours: to reduce costs while satisfying quality contracts. We share the techniques to save energy when the system is over-dimensioned for the supported workload: switch off of the servers and modification of their frequency. Their optimization technique also considers the problem of wear and tear on servers when repeated on-off cycles are performed. They proposed methods based on queuing theory, feedback control and hybrid mechanisms, instead, we use SPN models in an architectural framework. We also differ because their proposal reconfigures the system just in predetermined time instants, while we do it as soon as a better configuration is recognized.

The authors in [28] propose a framework for hosting multi-service platforms that allows the management of reallocation of the correct amount of resources for each service while satisfying the performance requirements. The work in [29] extends the previous one by considering energy consumption constraints and situations where the system is under illegitimate users requests. Our work differs from the previous ones in the goals. While their main objective is to maximize company profits (they consider cases when providers pay penalties), our goal covers both the savings in energy consumption and continuous performance requirements satisfaction.

Mistral [30] handles multiple distributed applications and large-scale infrastructures to optimize power consumption, performance and transient costs of adaptations. As in our approach, Mistral reconfigures the system when variations in the monitored workload are appreciated, however they implement a workload predictor that estimates these workload variations, in our case the SPN model of the workload owns this knowledge. They present an algorithm, that can increase exponentially, to create a graph that represents the system configurations and adaptation actions, in our approach the reconfiguration plan is represented also by a SPN model. For the computation of applications response time, Mistral, as well as our approach, relies on formal models, in this case queuing networks instead of Petri nets.

[31] presents an approach to self-adaptive resource allocation in virtualized environments that cares for SLAs. Their adaptation algorithm differs from ours since it proceeds in two phases: a first one to allocate resources to meet SLAs and a later one to deallocate those not utilized. The approach is validated using standard benchmarks.

The approach in [32] implements and validates, using a benchmark, a dynamic resource provisioning framework for virtualized server environments. It also accounts for the switching costs of the machines. As in our approach, the excessive switching and the variations in the workload intensity are taken into account. However, the approaches differ considerably. For example, they use a Kalman filter to estimate the number of future arrivals, while our approach allows accurate modeling using SPNs of multiple kinds and combinations of variable workloads. The dynamics of the system are expressed using equations, however we use SPNs as a modeling paradigm.

Finally, [3] is an interesting work that develops and validates a measurement-based approach as alternative to queuing models, which clearly differentiates it from our work. They also create a new set of metrics to predict runtime trade-offs between performance and energy use.

9. Conclusions

In the near future the management of power consumption in open systems and computing infrastructures will necessarily become an unavoidable topic, self-adaptive frameworks have a lot to say at this regard. Starting from a framework (reference architecture) able to self-adapt a system to improve its performance, the proposal herein reported enhances the architecture to also deal with energy variables. The model-driven approach, transformation of UML models into a formal one in terms of Petri nets, bestows interesting analyses capabilities for the framework to carry out an off-line management.

Future work is a path plenty of challenges. For example, understanding how to apply virtual layers in the underlying computing infrastructure and how the architecture can manage them, servers will be able to feed different system components. Other not addressed topic refers the management of servers with heterogeneous capabilities (e.g., different frequencies). Also it is worthy to investigate automatic generation of workload models from traces. In particular, works on Markov model estimation can be found in [18, 33].

Appendix A. Generalized Stochastic Petri Nets

A PN system is a tuple $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{M_0})$, where P and T are the sets of *places* and *transitions*, $\mathbf{Pre}$ and $\mathbf{Post}$ are the $|P| \times |T|$ sized, natural valued, pre- and post- incidence matrices. For instance, $\mathbf{Post}[p,t] = w$ means that there is an *arc* from t to p with *multiplicity* w. When all weights are one, the PN is *ordinary*. $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$ is the *incidence matrix* of the net. For pre- and postsets we use the conventional dot notation, e.g., ${}^{\bullet}t = \{p \in P : \mathbf{Pre}[p,t] \geq 1\}$, that can be extended to sets of nodes. If $\mathcal{N}'$ is the subnet of $\mathcal{N}$, defined by $P' \subseteq P$ and $T' \subseteq T$, then $\mathbf{Pre}' = \mathbf{Pre}[P', T']$, $\mathbf{Post}' = \mathbf{Post}[P', T']$ and $\mathbf{M_0'} = \mathbf{M_0}[P']$. Subnets defined by a subset of places (transitions), with all their adjacent transitions (places), are called P- (T-) subnets.

A *marking* $\mathbf{M}$ is a $|P|$ sized, natural valued, vector and $\mathbf{M_0}$ is the *initial marking* vector. A transition is *enabled* in $\mathbf{M}$ iff $\mathbf{M} \geq \mathbf{Pre}[P,t]$; its *firing*, denoted by $\mathbf{M} \xrightarrow{t} \mathbf{M}'$, yields a new marking $\mathbf{M}' = \mathbf{M} + \mathbf{C}[P,t]$. The set of all reachable markings is denoted as $RS(\mathcal{N}, \mathbf{M_0})$. An *occurrence sequence* from $\mathbf{M}$ is a sequence of transitions $\sigma = t_1 \ldots t_k \ldots$ such that $\mathbf{M} \xrightarrow{t_1} \mathbf{M_1} \ldots M_{k-1} \xrightarrow{t_k} \ldots$. Given σ such that $\mathbf{M} \xrightarrow{\sigma} \mathbf{M}'$, and denoting by σ the $|T|$ sized firing count vector of σ, then $\mathbf{M}' = \mathbf{M} + \mathbf{C} \cdot \sigma$ is known as the *state equation* of $\mathcal{N}$.

A GSPN is a tuple $\mathcal{G} = (\mathcal{N}, \mathbf{\Pi}, \bar{\mathbf{S}}, \mathbf{r})$, where $\mathcal{N}$ is a PN system and the set of transitions T is partitioned in two subsets T_t and T_i of timed and immediate transitions, respectively. $\mathbf{\Pi}$ is a natural valued, $|T|$ sized, vector that specifies a priority level of each transition. Timed transitions have zero priority, immediate transitions have priority greater than zero. A transition $t \in T$, enabled in marking $\mathbf{M}$, can fire if no transition $t' \in T : \mathbf{\Pi}[t'] > \mathbf{\Pi}[t]$ is enabled in $\mathbf{M}$. Timed transition firing delays are random variables, distributed according to negative exponential probability distribution functions. Immediate transitions fire instead in zero time. $\bar{\mathbf{S}}$ is a non negative real valued, $|T_t|$ sized, vector of the mean transition firing times. The positive real valued vector $\mathbf{r}$ is $|T_i|$ sized, and specifies the weights of immediate transitions for probabilistic conflict resolution.

Appendix B. Workload characterization

We describe here the theory underlying the derivation of parameters for the $SPN_{workload}$ model described in Section 3. We first introduce the Markov arrival processes (MAP), which are processes that can model all the concepts we are interested in. After, we concentrate the workload modeling on a type of MAP, those called Markov-modulated Poisson Process (MMPP). Finally, it is explained the representation of MMPPs using SPNs, which eases the integration of the workload model in the SPN model that represents the system behavior.

Markov arrival processes

Markov arrival processes (MAP) are stochastic processes that have been extensively used to model events arrival processes and network traffic, that show high variability and temporal dependencies.

In a MAP, event arrivals are governed by an irreducible continuous time Markov chain (CTMC). Transitions between states are classified as *background* or *completion*. The former type of transitions only changes the state in the CTMC, while the latter represents the arrival of an event and can either leave the CTMC in the same state or change it. Figure B.15 shows a two-state MAP, where *completion* transitions are depicted boldface. The meaning of this example MAP is: while its CTMC is in state1, events arrive following an exponential distribution with rate λ_{11}. This state is left with rate α_{12}, which entails that the mean sojourn time in state1 is $\frac{1}{\alpha_{12}}$, and afterwards the state changes to state2. In turn, state2 is linked to state1 by both *background* and *completion* transitions, which means that state2 is left with a rate $\lambda_{21} + \alpha_{21}$. Thus, the MAP can jump from state2 to state1 having received an event arrival or not, and these probabilities are $\frac{\lambda_{21}}{\lambda_{21}+\alpha_{21}}$ and $\frac{\alpha_{21}}{\lambda_{21}+\alpha_{21}}$ respectively.

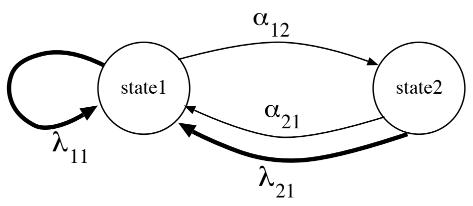

Figure B.15: Example of two-state MAP

A MAP of M states can be specified using two MxM matrices, called D_0 and D_1. Matrix D_1 stores the exponential rates of *completion* transitions, i.e., it stores non-negative real values λ_{ij}. In turn, D_0 elements out of the diagonal describe *background* transitions rates $\alpha_{ij}, i \neq j$. The infinitesimal generator of the CTMC that describes state transitions over time is given by $Q = D_0 + D_1$. Therefore, elements in the D_0 main diagonal ($D_{0;ii}$) are: $D_{0;ii} = -\sum_{j=1,j\neq i}^{M} D_{0;ij} - \sum_{j=1}^{M} D_{1;ij}$

In the previous example D_0 and D_1 are:

$$D_0 = \begin{pmatrix} -\alpha_{11} & \alpha_{12} \\ \alpha_{21} & -\alpha_{22} \end{pmatrix} D_1 = \begin{pmatrix} \lambda_{11} & 0 \\ \lambda_{21} & 0 \end{pmatrix}$$

and $\alpha_{11} = \alpha_{12} + \lambda_{11}$, $\alpha_{22} = \alpha_{21} + \lambda_{21}$

Parameter fitting of a MAP -in terms of number of states, D_0 and D_1- from a sample trace has been extensively studied, and methods to improve the parameter fitting in terms of accuracy and algorithm complexity order the are still being investigated [34, 35, 33, 36].

Markov-modulated Poisson process

MMPPs are a category of MAPs, where D_1 matrix is diagonal. It means that there cannot exist *completion* transitions that change the CTMC state but every state jump is driven by *background* transitions. When the CTMC is in state i, event arrival rate follows the exponentially distribution with parameter $D_{1;ii}$. Figure B.16 depicts an example of MMPP with four states, whose matrix definition is given by:

$$D_0 = \begin{pmatrix} -\alpha_{11} & \alpha_{12} & 0 & 0 \\ \alpha_{21} & -\alpha_{22} & \alpha_{23} & 0 \\ 0 & \alpha_{32} & -\alpha_{33} & \alpha_{34} \\ 0 & 0 & \alpha_{43} & -\alpha_{44} \end{pmatrix} D_1 = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix}$$

where $\alpha_{ii} = D_{1;ii} + \sum_{j=1,j\neq i}^{4} D_{0;ij}$

MMPPs models are simpler than generic MAPs, but they can still model the variability on the inter-arrival time of events as well as some of the important correlations between inter-arrival times [37]. The problem of parameter fitting of a MMPP from a trace has been investigated in the literature [34, 38, 37, 39, 40]. In this work, we do not propose any new method to fit a workload trace into a MMPP but we use/rely-on methods already proposed in the literature. Moreover, since MMPP behavior is more intuitive than generic MAPs, at first term a domain expert could give the initial values if he/she can intuitively separate the expected workload in different phases and can associate an arrival rate with each phase.

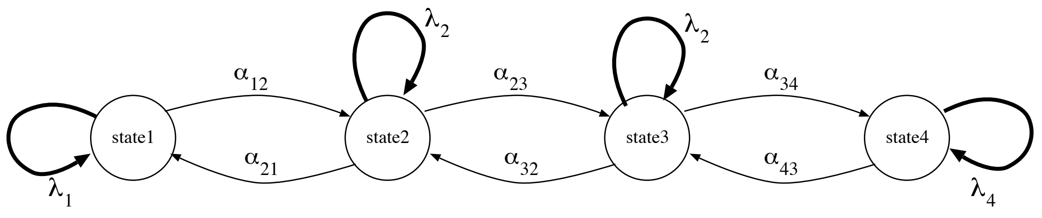

Figure B.16: Example of a four-state MMPP

SPN representation of MMPPs

Once the MMPP that models the event arrival has been created and its parameters have been set to fit the actual workload trace, we propose a SPN representation of such MMPP. The reason is that, since our analyzable model of the system is based on SPNs, it is an advantage to have the workload model represented in the same formalism in order to be integrated with the rest of the system. Hence this subsection deals with the SPN representation of the MMPP. As example, Figure B.17 corresponds to the SPN representation of the MMPP model in Figure B.16. Actually, for convenience, MMPP in Figure B.16 was chosen accordingly to derivate a SPN with the same structure as the one already shown in Section 3. Let us call $SPN_{workload}$ to the SPN model whose behavior is equal to MMPP. A $SPN_{workload}$ model includes:

- as many places as MMPP states. A token in a place means that the workload is in the phase represented by the MMPP state. Therefore only one of these state places can be marked.
- a timed transition for each $D_{1;_{ii}} \neq 0$. Such transitions are bidirectionally connected the state place i. These transitions represent the *completion* transitions in the MMPP and they inject requests to the event-arrival place. Their firing rate corresponds to the expected event arrival rate in each state. In Figure B.17 their firing rates are denoted as λ_i. Since these transitions are linked to state-places, only one of them can be enabled.
- a set of timed transitions to model the state mean sojourn time and probabilities of change between states. There is a transition with $state_i$ as input place and $state_j$ as output place if and only if $D_{0;_{ij}} > 0$. These transitions model the *background* transitions in the MMPP and their firing rates are set to $D_{0;_{ij}}$

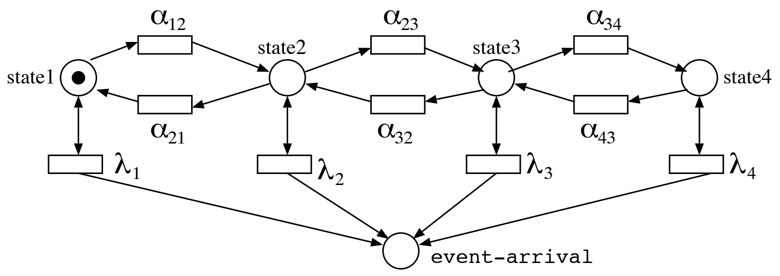

Figure B.17: SPN representation of MMPP workload model in Figure B.16

References

[1] P. Ranganathan, Recipe for efficiency: principles of power-aware computing, Commun. ACM 53 (2010) 60–67.

[2] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan, Autonomic multi-agent management of power and performance in data centers, in: AAMAS ’08, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 107–114.

[3] S. Chen, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, W. H. Sanders, Using cpu gradients for performance-aware energy conservation in multitier systems, Sustainable Computing: Informatics and Systems.

[4] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, N. Gautam, Managing server energy and operational costs in hosting centers, SIGMETRICS Perform. Eval. Rev. 33 (1) (2005) 303–314.

[5] D. Menascé, V. F. Almeida, Capacity Planning for Web Services: metrics, models, and methods., Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[6] J. Kramer, J. Magee, Self-managed systems: an architectural challenge, in: FOSE ’07: 2007 Future of Software Engineering, 2007, pp. 259–268.

[7] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with Generalized Stochastic Petri Nets, John Wiley Series in Parallel Computing - Chichester, 1995.

[8] E. N. Elnozahy, M. Kistler, R. Rajamony, Energy-efficient server clusters, PACS'02, 2003, pp. 179–197.

[9] R. Bianchini, R. Rajamony, Power and energy management for server systems, Computer 37 (11) (2004) 68 – 76.

[10] The GreatSPN tool, `http://www.di.unito.it/~greatspn`.

[11] D. Perez-Palacin, R. Mirandola, J. Merseguer, Enhancing a qos-based self-adaptive framework with energy management capabilities, in: QoSA/ISARCS, ACM, 2011, pp. 165–170.

[12] D. Perez-Palacin, J. Merseguer, S. Bernardi, Performance aware open-world software in a 3-layer architecture, in: Proceedings of WOSP/SIPEW '10, 2010, pp. 49–56.

[13] D. Perez-Palacin, J. Merseguer, Performance sensitive self-adaptive service-oriented software using hidden markov models, in: Proceedings of WOSP/SIPEW '11, 2011, pp. 201–206.

[14] J. López-Grao, J. Merseguer, J. Campos, From UML activity diagrams to stochastic Petri nets: Application to software performance engineering, in: Proceedings of WOSP'04, ACM, Redwood City, California, USA, 2004, pp. 25–36.

[15] Object Management Group, `http://www.promarte.org`, A UML Profile for MARTE. (2005).

[16] P. Buchholz, An em-algorithm for map fitting from real traffic data, in: P. Kemper, W. H. Sanders (Eds.), Computer Performance Evaluation / TOOLS, Vol. 2794 of Lecture Notes in Computer Science, Springer, 2003, pp. 218–236.

[17] A. Panchenko, A. Thümmler, Efficient phase-type fitting with aggregated traffic traces, Perform. Eval. 64 (2007) 629–645.

[18] G. Casale, N. Mi, L. Cherkasova, E. Smirni, Dealing with burstiness in multi-tier applications: Models and their parameterization, IEEE Trans. Software Eng., To appear.

[19] E. Gómez-Martínez, J. Merseguer, Argospe: Model-based software performance engineering, Vol. 4024, Springer-Verlag, Springer-Verlag, 2006, pp. 401–410.

[20] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar], Vol. 5525 of Lecture Notes in Computer Science, Springer, 2009.

[21] J. O. Kephart, D. M. Chess, The vision of autonomic computing, IEEE Computer 36 (1) (2003) 41–50.

[22] M. C. Huebscher, J. A. McCann, A survey of autonomic computing–degrees, models, and applications, ACM Comput. Surv. 40 (3).

[23] B. Schroeder, A. Wierman, M. Harchol-Balter, Open versus closed: a cautionary tale, in: NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation, USENIX Association, Berkeley, CA, USA, 2006, pp. 239–252.

[24] M. Welsh, D. Culler, Adaptive overload control for busy internet servers, in: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, 2003.

[25] J. Chase, R. Doyle, Balance of power: Energy management for server clusters, in: HotOS'01, 2001.

[26] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier internet applications, ACM Trans. Auton. Adapt. Syst. 3 (2008) 1:1–1:39.

[27] L. Bertini, J. C. Leite, D. Mossé, Power optimization for dynamic configuration in heterogeneous web server clusters, Journal of Systems and Software 83 (4) (2010) 585 – 598.

[28] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, F. Safai, Self-adaptive sla-driven capacity management for internet services, 2006, pp. 557 –568.

[29] I. Cunha, I. Viana, J. Palotti, J. Almeida, V. Almeida, Analyzing security and energy tradeoffs in autonomic capacity management, 2008, pp. 302 –309.

[30] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, C. Pu, Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures, in: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, ICDCS '10, 2010, pp. 62–73.

[31] N. Huber, F. Brosig, S. Kounev, Model-based self-adaptive resource allocation in virtualized environments, in: SEAMS, ACM, 2011, pp. 90–99.

[32] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Computing 12 (2009) 1–15.

[33] G. Casale, E. Z. Zhang, E. Smirni, KPC-Toolbox: Best recipes for automatic trace fitting using Markovian Arrival Processes, Perform. Eval. 67 (2010) 873–896.

[34] H. Okamura, T. Dohi, K. S. Trivedi, Markovian arrival process parameter estimation with group data, IEEE/ACM Trans. Netw. 17 (2009) 1326–1339.

[35] H. Okamura, T. Dohi, Faster maximum likelihood estimation algorithms for markovian arrival processes, QEST '09, IEEE Computer Society, 2009, pp. 73–82.

[36] A. Horváth, M. Telek, Markovian modeling of real data traffic: Heuristic phase type and map fitting of heavy tailed and fractal like samples, in: Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures, Springer-Verlag, 2002, pp. 405–434.

[37] W. Fischer, K. Meier-Hellstern, The Markov-modulated Poisson process (MMPP) cookbook, Perform. Eval. 18 (1993) 149–171.

[38] T. Rydén, An EM algorithm for estimation in Markov-modulated Poisson processes, Comput. Stat. Data Anal. 21 (1996) 431–447.

[39] L. Deng, J. Mark, Parameter estimation for markov modulated poisson processes via the em algorithm with time discretization, Telecommunication Systems 1 (1993) 321–338.

[40] K. S. Meier-Hellstern, A fitting algorithm for markov-modulated poisson processes having two arrival rates, European Journal of Operational Research 29 (3) (1987) 370 – 377.