Future Generation Computer Systems 86 (2018) 134–144

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Power-aware performance analysis of self-adaptive resource management in IaaS clouds

Ehsan Ataie [a,*], Reza Entezari-Maleki [b], Sayed Ehsan Etesami [c], Bernhard Egger [d], Danilo Ardagna [e], Ali Movaghar [c]

[a] Department of Engineering and Technology, University of Mazandaran, Babolsar, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[c] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
[d] Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea
[e] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy

HIGHLIGHTS

- An analytical model is proposed for Infrastructure-as-a-Service (IaaS) clouds taking several details of such systems into consideration.
- A self-adaptive power-aware and Service Level Agreement (SLA)-aware resource management scheme is presented for cloud systems.
- The presented scheme adjusts the number of powered-on Physical Machines (PMs) according to the input workload.
- A validation of the proposed model and scheme against the CloudSim framework is presented.

ARTICLE INFO

Article history:
Received 16 November 2017
Accepted 17 February 2018
Available online 29 March 2018

Keywords:
IaaS cloud
Self-adaptive resource management
Service level agreement
Stochastic activity network

ABSTRACT

In this paper, Stochastic Activity Networks (SANs) are used to model and evaluate the performance and power consumption of an Infrastructure-as-a-Service (IaaS) cloud. The proposed SAN model is scalable and flexible, yet encompasses some details of an IaaS cloud, such as Virtual Machine (VM) provisioning, VM multiplexing, and failure/repair behavior of VMs. Using the proposed SAN, a power-aware self-adaptive resource management scheme is presented for IaaS clouds that automatically adjusts the number of powered-on Physical Machines (PMs) regarding variable workloads in different time intervals. The proposed scheme respects user-oriented metrics by avoiding Service Level Agreement (SLA) violations while taking provider-oriented metrics into consideration. The behavior of the proposed scheme is analyzed when the arriving workload changes, and then its performance is compared with two non-adaptive baselines based on diverse performance and power consumption measures defined on the system. A validation of the proposed SAN model and the resource management scheme against an adapted version of the CloudSim framework is also presented.

© 2018 Published by Elsevier B.V.

1. Introduction

Cloud computing has attracted great popularity in recent years. Infrastructure as a Service (IaaS) is one of the most popular types of services that clouds offer. In IaaS, low-level computing resources are delivered to customers in the form of Virtual Machines (VMs). IaaS cloud providers require to monitor, measure, and manage Quality of Services (QoS) delivered to their customers in order to meet Service Level Agreements (SLAs). SLAs specify QoS targets, economical revenues, and penalties associated with SLA violations [1]. Cloud providers should prevent SLA violations as much as possible if they want to maximize their profit. Amazon and Google reported a revenue loss of 1% and 20%, respectively, due to small additional response delays [2]. On the other hand, today's cloud data centers consume a huge amount of power. The power consumption of cloud infrastructures in 2020 is estimated to be 1, 963.74 TW [2]. Therefore, concerns about electricity costs and environmental impacts of cloud data centers are intensifying. Some of the big cloud providers, such as Amazon, have recently started to generate the power required to operate their data centers. At the end of 2016, more than 40% of the power consumed by Amazon's global infrastructure came from their wind and solar

* Corresponding author.
E-mail addresses: ataie@umz.ac.ir (E. Ataie), entezari@ipm.ir (R. Entezari-Maleki), eetesami@ce.sharif.edu (S.E. Etesami), bernhard@csap.snu.ac.kr (B. Egger), danilo.ardagna@polimi.it (D. Ardagna), movaghar@sharif.edu (A. Movaghar).

https://doi.org/10.1016/j.future.2018.02.042
0167-739X/© 2018 Published by Elsevier B.V.

farms, and the company set a goal to be powered by 50% renewable energy by the end of 2017 [3]. Therefore, to achieve the goal of maximizing the SLAs revenue while minimizing energy costs, simultaneous modeling and evaluation of performance and power consumption of cloud systems are of utmost importance in modern data centers.

Another critical challenge in the context of cloud computing is resiliency. Resiliency is briefly defined as the capacity of a system to remain reliable, failure tolerant, and dependable in case of any changes or failures that result in a temporal or permanent service disruption [4]. On the one hand, hardware and software resources in cloud infrastructures are subject to failure. According to the Information Week, IT outages cause more than $26.5 billion revenue loss each year [5]. On the other hand, non-periodic workload bursts, which are sudden temporary increases in usage of shared resources, are also common in large-scale systems like clouds [6]. Workload burstiness usually leads to performance degradation in terms of uncontrolled response time, server overloading, service unavailability, and SLA violation [7]. The dynamic and unpredictable nature of non-periodic workload bursts hardens performance evaluation of large-scale cloud systems [8]. Therefore, modeling and quantification of resiliency in cloud systems, including modeling of failure/repair of resources and workload bursts, are among the foremost concerns in cloud environments.

On the one hand, providing IaaS cloud services that ensure users' QoS requirements by avoiding SLA violations is a big challenge in the cloud computing paradigm [9]. The probability of a request being accepted by the system as a result of availability of resources is an example of user-oriented requirements. Cloud providers should rapidly adjust their resources with workload changes and failures of cloud components to meet QoS requirements. On the other hand, provider-oriented measures such as utilization of resources and power consumption can affect the profit margin of cloud providers [10,11]. However, most of current cloud computing systems and their management techniques are not able to react efficiently at runtime [9]. Hence, they cannot appropriately meet both user-oriented and provider-oriented requirements at the same time. To do so, cloud systems require self-management of resources without human involvement. Hence, modeling and analysis of autonomic self-adaptive resource management schemes can be useful in terms of provider achievement and user satisfaction.

In this paper, an IaaS cloud architecture is modeled in which the switching on/off of Physical Machines (PMs), VM multiplexing, provisioning and servicing steps of requested VMs, and failure/repair behaviors of VMs are taken into account. Using the model proposed for such a cloud architecture, a self-adaptive resource manager is presented wherein status data collected from the cloud environment is used through a four-step cycle of autonomic systems including monitoring, analyzing, planning, and execution [12,13]. Appropriate actions are carried out by the proposed management scheme to respond to the environmental changes. The actions are in the form of PM switch on/off commands that cause some VMs to be added to or removed from the available resources. The commands are issued so that violations of crucial metrics from both user and provider perspectives are prevented. After proposing the self-adaptive resource management scheme, the behavior of the proposed scheme is evaluated and compared with two non-adaptive baselines based on diverse performance and power consumption measures. The formalism used for modeling is Stochastic Activity Network (SAN) [14]. Generally, SANs are probabilistic extensions of activity networks that are equipped with a set of activity time distribution functions, reactivation predicates, and enabling rate functions. The nature of the extension is similar to the one that constructs Stochastic Petri Nets (SPNs) from classical Petri Nets (PNs). SANs were developed to facilitate unified performance/dependability evaluation and have features which permit the representation of parallelism, timeliness, fault tolerance, and degradable performance [15]. More detailed information about SANs can be found in [14–17]. After proposing a SAN model, the validation of the proposed model and the management scheme is performed against an extension of a well-known cloud simulator, the CloudSim framework [18]. The main contribution of this paper is to propose a scalable and flexible SAN model for IaaS clouds that encompasses details of such systems. The proposed model allows us to present our power-aware and SLA-aware self-adaptive cloud management scheme which can adjust the number of powered-on PMs with respect to the input workload to meet both user- and provider-oriented requirements.

The remainder of this paper is organized as follows. A literature review on cloud performance and power consumption analysis is given in Section 2. The system description and main assumptions about the reference architecture considered in this paper are introduced in Section 3. In Section 4, the proposed model for an IaaS cloud is presented. Section 5 introduces various measures of interest and the methods for computing them using the proposed SAN model. Details of the proposed self-adaptive resource management scheme are presented in Section 6. Numerical results obtained from the proposed SAN model and the management scheme are provided in Section 7. In Section 8, a model validation against the CloudSim framework is presented. Finally, the paper is concluded in Section 9 with some notes for future work.

2. Related work

Evaluation of performance and power consumption of cloud systems using analytical models is an emerging topic. The Markov reward approach has been employed in [19] to develop an analytical model to quantify the power-performance trade-offs in IaaS clouds. In this model, physical machines were grouped into three different pools with different VM provisioning delays and operational costs: hot (i.e., running), warm (i.e., turned on, but not ready), and cold (i.e., turned off). Interacting stochastic sub-models were used to cope with the largeness problem, and the final solution was obtained by applying the fixed-point iteration on individual sub-models. Then, the effect of pool configurations on the power consumption, job rejection probability, and mean response time was studied. It should be noted that the dependability of the cloud system was not considered in [19]. SANs have been used in [20] for modeling and evaluation of performance and power consumption of PMs in an IaaS cloud. In the proposed SAN, PMs were modeled in three different power consumption and provisioning delay modes. In order to control the supply voltage of CPUs, and consequently, manage the power consumed by PMs, a Dynamic Voltage and Frequency Scaling (DVFS) technique was investigated. Using the proposed model, several metrics, e.g., waiting time, instant service availability, blocking probability, and power consumption, were defined and evaluated. Although the work presented in [20] does not take the failure/repair behavior of VMs into account, the DVFS technique modeled in [20] can be complementary to our work. Stochastic Reward Nets (SRNs) have been employed in [21] to model and evaluate an IaaS cloud at different levels. For this purpose, an SRN was presented to model a cluster of PMs within a cloud data center that captures several aspects of cloud systems. Afterwards, the SRN model was used in a hierarchical manner to propose a monolithic model for an entire cloud system. Since the monolithic model does not scale well, two approximate models based on folding and fixed-point techniques were proposed to evaluate availability, response time, and power consumption of the IaaS cloud system. The sensitivity of output measures to the variation of input parameters was also analyzed. An analytical framework has been presented in [10] to

analyze the cost and benefit of different resource allocation policies in an IaaS cloud. Based on the separation of concerns, one sub-model for each resource management layer, physical and virtual layers, was proposed using SRNs, and the interaction between two sub-models was specified by applying guard functions. As a use case, two sample policies were defined and compared in terms of different measures, such as blocking probability, performance degradation, and power consumption. In contrast to our work, the failure/repair process of VMs was not considered in that work.

The problem of data center right-sizing to achieve energy-efficiency has been addressed in [22] while taking into consideration different factors such as PM failures and the overhead introduced by virtualization. The authors presented a stochastic model for cloud data centers based on Queuing theory to analyze and understand the dynamic nature of data centers. The model was then implemented and deployed in a real-world cloud data center. The results of the experiments were compared with state-of-the-art in terms of various metrics e.g., SLA violation and energy consumption. An analytical model has been proposed in [23], based on Continuous Time Markov Chains (CTMCs), for IaaS cloud systems. The model incorporated several aspects of cloud systems such as compound user requests, different servicing steps, and migration of PMs between different pools. The cloud manager considered in [23] consists of different modules, and each one was modeled by a separate CTMC sub-model. The interdependencies between sub-models were resolved via the fixed-point iteration method. Finally, the authors evaluated the effects of various input parameters, e.g. service time and virtualization degree, on output measures including probability of task rejection, servicing delay, and power consumption. A hierarchical correlation model has been proposed in [24] for evaluation of three correlated metrics of a large online service in an IaaS cloud named performance, reliability, and power consumption. The modeling approach integrates Queuing theory, Markov models, and a Bayesian approach. In the model proposed in [24], different characteristics of a cloud system such as multiple VMs hosted on the same server, common cause failures of co-located VMs, and logical mapping mechanisms for multi-core CPUs were investigated.

In addition to the above-mentioned research work, there are also other research papers focusing on modeling and quantifying the resiliency of cloud systems through analytical models. An SRN model has been proposed for an IaaS cloud in [25] with the aim of resiliency quantification in these environments. Physical machines in such a cloud were grouped into three different pools. Then, the effect of changing the arrival rate of jobs on the job rejection rate and the mean number of jobs in a resource provisioning decision engine were studied by transiently analyzing the proposed model. A simple but scalable analytical model based on SRNs has been presented in [26] which can model different cloud strategies and policies. Several performance measures were defined and evaluated using the proposed model including availability, utilization, service and waiting times, and responsiveness. The author has performed a resiliency analysis through a transient solution of the proposed model to consider how a workload burst affects the availability and instant service probability of the system. However, adaptation of cloud resources was not studied in [25] and [26].

There are also approaches that model autonomic and adaptive resource management in cloud systems. An SRN-based model has been proposed in [27] for an IaaS cloud data center in which the load conditions can suddenly change. In order to alleviate the effects of load bursts, the proposed reactive autonomic cloud manager adopted the PM overcommitting technique wherein different VMs were allowed to be multiplexed on the same PM. When the load burst ended, the PM overcommitting was gradually reduced and finally disabled. Several performance measures including instant service probability, blocking probability, utilization, and service time were evaluated through transient analysis of the system under-study. In contrast to our work, constant times have been considered in [27], for detecting burst arrival and expiration. Moreover, while overcommitting of physical resources has been employed in [27] to withstand the burst, we propose switching PMs on/off to react to workload changes.

An autonomic cloud manager has been proposed in [28] for supporting proactive management of applications in a geographically distributed cloud system. The goal is to redirect the load to healthy machines and cloud regions by predicting the mean time to failure of VMs. The predictions were carried out using machine learning techniques. A model based on Colored Petri Nets (CPNs) has been presented in [29] for cloud platforms in which migration of VMs between PMs can take place dynamically. Then, a method was proposed for calculating energy and transmission costs of VM migration. Using the proposed model and method, two migration policies were compared in terms of the energy and transmission costs. Adaptive energy-aware algorithms have been presented in [30] for maximizing energy efficiency while minimizing SLA violations. In order to adapt to unpredictable workloads, the proposed algorithms used an adaptive three-threshold framework for the classification of data center hosts into four different classes, i.e., less loaded, little loaded, normally loaded, and overloaded hosts. Moreover, the application types and the CPU and memory resources were taken into account during the deployment of VMs. An experimental analysis using real-world workloads was performed to demonstrate the efficacy of the proposed approaches.

3. System architecture

The general structure of the IaaS cloud system under-study is represented in Fig. 1. We consider an IaaS cloud in which user requests are received in the form of VM instantiation requests. The requests are enqueued in the *input queue*, which has a limited size. An incoming request is rejected from the system if there is not enough capacity in the input queue or no resources available in the system. Otherwise, the request is admitted and can be served by an available VM. The cloud system considered in this paper contains N_{pm} physical machines, assumed to be homogeneous for the sake of simplicity [10,11,20,22,23,26], which compose a *virtual resource pool* initially containing $N_{pm} \cdot L$ instances of VMs. The multiplexing factor, L, indicates the number of VMs that can be simultaneously provisioned on a single PM by sharing its resources.

The resource manager processes queued requests in a First In First Out (FIFO) order. To this end, a PM with enough resources should be chosen by the resource manager to host the VM request at the front of the input queue. The requested VM can be immediately provisioned if there is an available VM in the virtual resource pool, which indicates that there is at least one powered-on PM with sufficient resources. If all PMs of the data center are fully occupied by serving VMs, the request has to wait until one of the VMs finishes its associated job and is released. If there are more than one PM capable to host the requested VM, dispatching policies can be adopted to decide on which PM the request should be provisioned. Since our focus in this paper is not on request dispatching policies, we assume a random dispatching policy that randomly selects one of the candidate PMs to host the request, but generally, any dispatching policy can be considered. Once the target PM is specified by the resource manager to host the VM request, the request is sent to the hypervisor of that PM to get the resources required for the VM. Herein, we assume that each IaaS request needs only one VM, and all VMs are homogeneous. These assumptions were also made in previously presented approaches in this area [10,19,23,26,31].

The proposed resource management scheme obeys the general structure of a four-step control loop of self-adaptive systems [12].

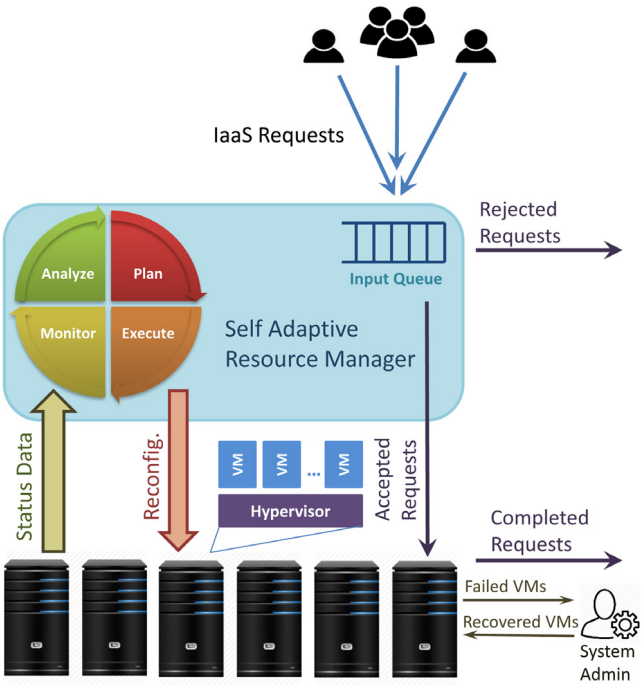

Fig. 1. Self-adaptive resource management architecture for an IaaS cloud.

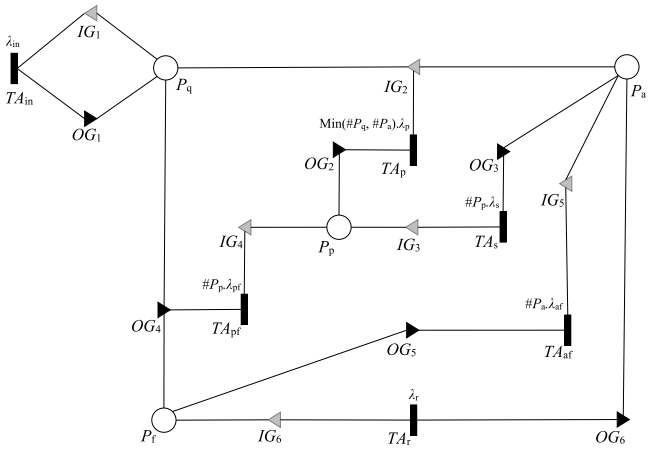

Fig. 2. SAN model proposed for a data center.

Table 1
Elements of the SAN model presented in Fig. 2.

Name	Description	Rate or initial number of tokens
P_q	Input queue of requests	0
P_a	Available VMs	$N_{pm} \cdot L$
P_p	Provisioned VMs	0
P_f	Failed VMs	0
TA_{in}	Arrival of requests	λ_{in}
TA_p	Provisioning VMs for requests	$Min(\#[P_q], \#[P_a]).\lambda_p$
TA_s	Serving requests	$\#P_p.\lambda_s$
TA_{pf}	Failure of provisioned VMs	$\#P_p.\lambda_{pf}$
TA_{af}	Failure of available VMs	$\#P_a.\lambda_{af}$
TA_r	Repair of failed VMs	λ_r

In order to prevent violations of crucial metrics, the system is continuously monitored by a *monitoring* component, which is a part of the self-adaptive resource management scheme. The monitoring component checks the state of the system by gathering important *status data*, such as the size of the input queue and the number of available and provisioned VMs. Then, these status data are used by another part of the self-adaptive scheme, named the *analyzer* component, to compute performance measures of interest. In this paper, the performance measures that should not be violated are the blocking probability and the utilization of resources. Applying these performance measures, the *planner* component of the self-adaptive scheme decides how to reconfigure the system. If the blocking probability increases to a predefined threshold T_1, the self-adaptive scheme reacts by switching a PM on, resulting in L new VMs to be added to the virtual resource pool. On the other hand, if the utilization of resources drops below a given threshold T_2, the self-adaptive scheme reacts by switching a PM off, resulting in L available VMs being eliminated from the virtual resource pool. These reactive reconfigurations are performed by another component of the self-adaptive scheme, called *executer*, which is actually a power manager responsible for switching PMs on and off. Using this control loop model of self-adaptive systems, a couple of PMs are kept in powered-off mode to save power unless they are needed to be switched on to enhance the QoS delivered to cloud users. Herein, it is assumed that power consumption of a powered-off PM is small enough to be neglected, and power consumption of a powered-on PM is a function of the number of both available and provisioned VMs running on top of that PM.

VMs are subject to failures during their continuous execution. The source of such failures could be software aging, failure in the network, failure in the shared storage system, and so forth [32–36]. When a failure occurs, the monitoring component detects the failure and possibly the failed component, and then notifies the system administrator who investigates the cause of failure and recovers the failed component. A failed VM will be available again after recovery/restart by the system administrator. Since the failure rate of a provisioned VM is larger than that of an available VM, the Mean Time To Failure (MTTF) of a provisioned VM is smaller than that of an available VM. Since we assume that VMs are stateless, the applications running on VMs are stateless, or their states are managed out of the VMs through a cloud data store [37–39], the request corresponding to a failed provisioned VM can be restarted on another available VM. Hence, such a request is sent back to the input queue to receive service later.

4. Proposed SAN model

The SAN model proposed for analyzing an IaaS cloud is presented in Fig. 2. It is assumed that the times assigned to all timed activities follow exponential distribution [10,23,40,41]. The description of all elements of this SAN is given in Table 1. Timed activity TA_{in} models the request arrival process to the cloud data center. When it completes with rate λ_{in}, one token is deposited into place P_q by output gate OG_1. Place P_q represents the input queue of the data center. A token in this place serves as a request waiting in the queue. When the number of tokens inside place P_q reaches Q_{in}, input gate IG_1 prevents activity TA_{in} from completion.

Place P_a serves as the virtual resource pool of the system. The tokens inside place P_a represent available VMs that can be provisioned and assigned to incoming requests. The initial number of tokens in place P_a is $N_{pm} \cdot L$, where N_{pm} is the initial number of powered-on PMs, and L is the maximum number of VMs that can be simultaneously provisioned on a single PM. If there is at least one token in both places P_q and P_a, timed activity TA_p is activated and can complete. Upon completion of this activity, one token is removed from both places P_q and P_a by input gate IG_2, and one token is deposited into place P_p by output gate OG_2. Since multiple VMs can be provisioned at the same time, the actual completion rate of activity TA_p is $Min([\#P_q], [\#P_a]) \cdot \lambda_p$ which expresses multiplication of the minimum number of tokens inside places P_q and P_a by λ_p, where λ_p is the provisioning rate of a single VM.

Table 2
Gate predicates/functions of the SAN model represented in Fig. 2.

Gate	Predicate	Function
IG_1	$([\#P_q] < Q_{in})$ && $([\#P_a] > 0)$	
OG_1		$[\#P_q] + +;$
IG_2	$([\#P_q] > 0)$ && $([\#P_a] > 0)$	$[\#P_q] - -;$
		$[\#P_a] - -;$
OG_2		$[\#P_p] + +;$
IG_3	$[\#P_p] > 0$	$[\#P_p] - -;$
OG_3		$[\#P_a] + +;$
IG_4	$[\#P_p] > 0$	$[\#P_p] - -;$
OG_4		$[\#P_q] + +;$
		$[\#P_f] + +;$
IG_5	$[\#P_a] > 0$	$[\#P_a] - -;$
OG_5		$[\#P_f] + +;$
IG_6	$[\#P_f] > 0$	$[\#P_f] - -;$
OG_6		$[\#P_a] + +;$

The existence of a token in place P_p indicates that a VM has been provisioned for a request. The actual servicing step is modeled by timed activity TA_s. If there is a token in place P_p, then timed activity TA_s is activated and can complete. When this activity completes, one token is moved from place P_p to place P_a through gates IG_3 and OG_3. This shows that a VM has finished serving a user and it can be allocated to serve another request later. The completion rate of activity TA_s is $[\#P_p] \cdot \lambda_s$, where $[\#P_p]$ indicates the number of tokens inside place P_p and λ_s denotes the service rate of a single VM.

Since provisioned VMs are failure-prone, timed activity TA_{pf} is used to model these failures. Upon completion of this activity, a token is removed from place P_p by input gate IG_4, and a token is added to both places P_q and P_f by output gate OG_4. Using this mechanism, the request associated to a failed VM returns to the input queue to be provisioned on another VM later. The completion rate of TA_{pf} activity is $[\#P_p] \cdot \lambda_{pf}$, where $1/\lambda_{pf}$ represents the MTTF of a provisioned VM. Tokens in place P_f represent failed VMs waiting to be repaired. In addition to provisioned VMs, available VMs may also fail. This failure event is modeled by timed activity TA_{af}. Upon completion of activity TA_{af}, one token is moved from place P_a to place P_f through gates IG_5 and OG_5. The completion rate of activity TA_{af} is $[\#P_a] \cdot \lambda_{af}$, where $[\#P_a]$ is the number of tokens inside place P_a, and $1/\lambda_{af}$ is the MTTF of an available VM.

Timed activity TA_r represents the repair process of failed VMs. This activity is activated when at least one token exists in place P_f. When TA_r completes, a token is removed from place P_f by input gate IG_6 and deposited into place P_a by output gate OG_6. The completion rate of TA_r is λ_r. The predicates and functions corresponding to all input and output gates of the proposed SAN are represented in Table 2.

5. Performance measures

In this section, several measures which can be obtained by analytically solving the SAN model presented in Section 4 are introduced. These measures can be computed using the Markov reward approach [42]. In this approach, appropriate reward rates are assigned to each feasible marking of a SAN model, and then, the expected reward rates at a given time point, t, or in the steady-state are computed.

Let S denote all possible configurations in which the system can perform its function, and $\{X(t), t \geq 0\}$ on S define a continuous time stochastic process describing the structure of the system at time t. Let $\pi_i(t)$ denote the probability that the system is in marking $i \in S$ at time t, and π_i denote the probability that the system is in marking i at the steady-state. If r denotes the reward function that associates a reward rate r_i to each feasible marking $i \in S$ of a SAN model, then the expected reward rate at time t, $E[r_{X(t)}]$, can be computed by $\sum_i r_i \cdot \pi_i(t)$. Moreover, the expected steady-state reward rate, $E[r_X]$, can be computed by $\sum_i r_i \cdot \pi_i$ [43]. We can also associate impulse rewards to the activities of SAN models in a similar way, which results in calculation of the expected throughput of activities at a given time or in the steady-state. In the following, some of the interesting measures are defined over the SAN model represented in Fig. 2.

Blocking probability at time *t* is the probability that the cloud data center is not able to accept incoming requests at time t. This probability is computed by Eq. (1).

$$BLK(t) = E[r^{blk}_{X(t)}] \tag{1}$$

where $r^{blk}_{X(t)}$ corresponds to the probability that the input queue is full or there is no available VM at time t. r^{blk} can be computed through the following reward function.

$$r^{blk} = \begin{cases} 1, & [\#P_q] >= Q_{in} \quad or \quad [\#P_a] = 0 \\ 0, & otherwise. \end{cases} \tag{2}$$

Utilization of resources at time *t* is the ratio of the number of provisioned VMs to the total number of intact VMs inside cloud data center at time t. It is computed by Eq. (3).

$$UTL(t) = \frac{E[r^{prv}_{X(t)}]}{E[r^{avl}_{X(t)}] + E[r^{prv}_{X(t)}]} \tag{3}$$

where $r^{prv}_{X(t)}$ and $r^{avl}_{X(t)}$ correspond to the number of provisioned and available VMs at time t, respectively. r^{prv} and r^{avl} can be respectively calculated by Eqs. (4) and (5).

$$r^{prv} = [\#P_p] \tag{4}$$

$$r^{avl} = [\#P_a]. \tag{5}$$

Instant service probability at time *t* denotes the probability that a request is immediately served at time t, and is computed by Eq. (6).

$$ISP(t) = E[r^{isp}_{X(t)}] \tag{6}$$

where $r^{isp}_{X(t)}$ is a reward function for computing the probability of the input queue to be empty and the data center to have at least one available VM at time t. More precisely, r^{isp} is computed using Eq. (7).

$$r^{isp} = \begin{cases} 1, & [\#P_q] = 0 \quad and \quad [\#P_a] > 0 \\ 0, & otherwise. \end{cases} \tag{7}$$

Power consumption at time *t* is the total power consumed by all PMs at time t, and can be computed by Eq. (8).

$$PWR(t) = N_{pm}(t) \cdot pwr_b + E[r^{prv}_{X(t)}] \cdot pwr_p + E[r^{avl}_{X(t)}] \cdot pwr_a \tag{8}$$

where $N_{pm}(t)$ is the number of powered-on PMs at time t, $r^{prv}_{X(t)}$ and $r^{avl}_{X(t)}$ correspond to the numbers of provisioned and available VMs at time t, as described in Eqs. (4) and (5), respectively. Furthermore, pwr_b is the base power consumed by a PM excluding the power consumed by its VMs. In Eq. (8), pwr_p and pwr_a are power consumption of a provisioned VM and an available VM, respectively. Indeed, the power consumption of a powered-on PM can be expressed as the base power consumption of that PM, plus the power consumed by a single provisioned VM times the number of VMs provisioned on that PM, plus the power consumed by a single available VM times the number of available VMs on that PM [44,45].

Waiting time at time *t* represents the expected time that requests wait in the input queue of the cloud data center which is calculated by Eq. (9).

$$WTT(t) = \frac{E[r^{que}_{X(t)}]}{Thr(TA_p)} \tag{9}$$

where $Thr(TA_p)$ represents the expected throughput of activity TA_p, and $r^{que}_{X(t)}$ corresponds to the number of enqueued requests at time t which can be computed using the following reward function.

$$r^{que} = [\#P_q]. \quad (10)$$

6. Proposed self-adaptive resource management scheme

In order to describe the operation of the proposed self-adaptive resource management scheme, an event timeline is depicted in Fig. 3 in which important time events of the general scenario considered in this paper are shown. Suppose that before time t_b, the system is in a stable condition. In this state, IaaS requests arrive to the system with a constant rate λ^1_{in} and the measures blocking probability of requests and utilization of resources, as defined in Section 5, are not violated. Now, suppose that a workload burst arrives at time t_b which increases the request arrival rate to λ^2_{in}. Increasing the arrival rate, the cloud begins to become overloaded. As a consequence, performance of the system is affected by the burst load. Specifically, by increasing the request arrival rate, the probability of either the input queue being full or there being no available VM in the virtual resource pool is increased. As a result, the blocking probability starts increasing. Suppose that at time t_{oa_1}, the blocking probability reaches a predefined threshold thr_b. The proposed self-adaptive scheme immediately reacts by switching a PM on, causing L new VMs to be added to the available resources. This adaptation results in an immediate decrease in the blocking probability. However, if adding these new VMs is not sufficient to overcome the system overload, this metric will increase again after a while. Therefore, several subsequent adaptations may take place during the burst period each one switching on one more PM to prevent a violation of the blocking probability. These overload adaptations are denoted by t_{oa_i}, $1 \leq i \leq N_{oa}$ in Fig. 3, where N_{oa} is the number of adaptations performed during the burst period when the system is overloaded. Finally, the system either reaches a stable situation before burst expiration wherein no more adaptations are required, or it experiences a transient phase raised by adaptations when burst expires.

When the burst ends at time t_e, the request arrival rate decreases to λ^1_{in} again. In the new situation, the system starts to gradually become underloaded. As a consequence, performance metrics of the system are affected. Specifically, by decreasing the request arrival rate, the ratio of the provisioned VMs to the available VMs is reduced, so the system experiences an underutilization of resources. Assume at time t_{ua_1}, the utilization of resources reaches a predefined threshold thr_u. The proposed self-adaptive scheme immediately reacts by switching a PM off, causing L available VMs to be eliminated from the virtual resource pool. This adaptation results in an immediate increase in utilization. However, during the time, more provisioned VMs finish their jobs and return to the virtual resource pool. This again causes the utilization of resources to decrease. Hence, multiple adaptations may be required to switch more PMs off and prevent the utilization from dropping under thr_u. These subsequent underload adaptations are denoted by t_{ua_i}, $1 \leq i \leq N_{ua}$ in Fig. 3, where N_{ua} is the number of adaptations performed after the burst period when the system is underloaded. Finally, the system either reaches a stable situation or experiences another burst.

Based on the timeline shown in Fig. 3, the SAN model proposed in Fig. 2 is solved in different time intervals as described in the following. It is worth mentioning that the initial number of tokens inside places of the SAN model of Fig. 2 for each step of analysis is obtained using the steady-state or transient analysis performed in its previous step. Moreover, whenever an overload adaptation is carried out, the number of tokens in place P_a of the model of Fig. 2 increases by L at the beginning of the analysis, and in the underload adaptation, the number of tokens in place P_a decreases by L when analysis starts. In the following, the action performed in each time interval is described.

- $(-\infty - t_b)$. A steady-state analysis is carried out to analyze the behavior of the system before burst starts and to compute the measures of interest.
- $[t_b - t_{oa_1})$. A transient analysis is carried out to analyze the impact of the burst on the behavior of the system and to determine when the first overload adaptation should be performed.
- $[t_{oa_i} - t_{oa_{i+1}})$, $1 \leq i \leq N_{oa} - 1$. A transient analysis is carried out to analyze the impact of the adaptation performed at time t_{oa_i} on the system behavior and to determine when the next overload adaptation should be performed.
- $[t_{N_{oa}} - t_e)$. A transient analysis is carried out to analyze the impact of the last overload adaptation on the system behavior.
- $[t_e - t_{ua_i})$. A transient analysis is carried out to analyze the impact of finishing the burst on the system behavior and to determine when the first underload adaptation should be performed.
- $[t_{ua_i} - t_{ua_{i+1}})$, $1 \leq i \leq N_{ua} - 1$. A transient analysis is carried out to analyze the impact of the adaptation performed at time t_{ua_i} on behavior of the system and to determine when the next underload adaptation should be performed.
- $[t_{N_{ua}} - +\infty)$. A steady-state analysis is carried out to analyze the behavior of the system after the last underload adaptation was made.

7. Numerical results

In this section, the autonomic resource management scheme introduced in Section 4 is analyzed and compared with two non-autonomic schemes, based on the metrics described in Section 5. To this end, the proposed analytical model is solved with the Möbius tool [46] for a wide range of input parameter values, from which we just report some interesting results here. Most of the values considered as input parameters of the cloud data center investigated in this paper are in the range of the values considered in other related work [10,19,26,32,35,38,40,41,47,48]. These parameters are set as follows.

The request arrival rate during time frames $(-\infty, t_b)$ and $[t_e, +\infty]$ is $\lambda^1_{in} = 13$ req/hr, and during time frame $[t_b, t_e)$ is $\lambda^2_{in} = 20$ req/hr indicating a workload burst. For the sake of simplicity, we assume that only one burst occurs during the time the system is analyzed. Time events t_b and t_e are assumed to be 20 and 50 h, respectively. The mean provisioning time $(1/\lambda_p)$ and the mean service time $(1/\lambda_s)$ are 5 min and 5 h, respectively. The MTTF of an available VM $(1/\lambda_{af})$ and a provisioned VM $(1/\lambda_{pf})$ are 100 and 66.7 h, respectively. The mean repair time of a failed VM $(1/\lambda_r)$ is 30 min. The basic power consumption of a PM (pow_b) is set to 134 W. The power consumed by an available VM (pow_a) and a provisioned VM (pow_p) are set to 7, and 17 W, respectively. The size of the input queue, Q_{in}, is set to 10. The maximum number of VMs that can be concurrently provisioned on a PM, L, is set to 8. The number of powered-on PMs, N_{pm}, is initially set to 10 during $(-\infty, t_b)$. Thus, at most 80 VMs can be provisioned in the system. During the burst and after that, the instantaneous value of N_{pm} changes depending on adaptations performed by the self-adaptive management scheme, as mentioned in Section 6. The values of thr_b and thr_u are set to 0.1 and 0.77, respectively.

Figs. 4–8 compare the results obtained from the transient analysis of the proposed self-adaptive resource management scheme and two non-self-adaptive schemes labeled by *baseline 1* and *baseline 2*. In *baseline 1*, 10 PMs are initially switched on, and they are

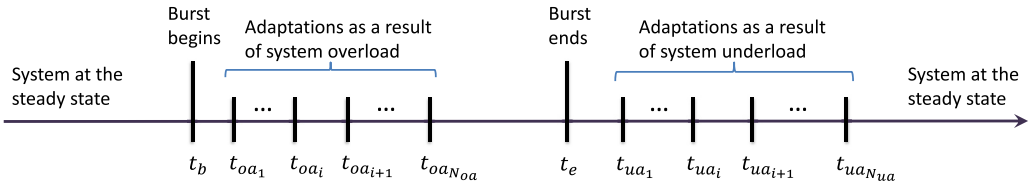

Fig. 3. Timeline of the scenario considered in this paper.

kept powered-on during entire system analysis allowing at most 80 VMs to be concurrently provisioned independent of request arrival rate. Therefore, our proposed scheme acts the same as *baseline 1* before the burst begins because it also starts with 10 PMs. On the other hand, *baseline 2* starts with 13 PMs and keeps them powered-on independent of system workload which means at most 104 VMs can be concurrently provisioned during system analysis. The number of PMs in *baseline 2* is the minimum number of required PMs in a static scenario that prevents violations of blocking probability. In the following, each of the measures introduced in Section 5 is analyzed.

7.1. Blocking probability

Fig. 4 represents the blocking probability at time t, $BLK(t)$, resulted from the proposed scheme and *baselines 1* and *2*. Through the steady-state analysis performed before t_b, it can be observed that the blocking probability is 0.011 for both the proposed self-adaptive scheme and *baseline 1*, and it is almost zero for *baseline 2*. Starting from t_b, *BLK* grows fast for the self-adaptive scheme and *baseline 1*, and it reaches the stable value 0.236 for *baseline 1*. However, the self-adaptive scheme reacts to the workload increase when *BLK* reaches thr_b at time $t_b + 2$ by adding 8 new VMs to the resource pool. These additional VMs provide enough resources for user requests for a short time which leads to a sudden drop of blocking probability for the self-adaptive scheme.

Continuing the burst, *BLK* increases again for the self-adaptive scheme since the available VMs are not still sufficient to handle the burst. Therefore, *BLK* reaches thr_b threshold at time $t_b + 4.2$. To prevent SLA violations, another overload adaptation is performed, which adds 8 new VMs to the resource pool of the system by switching on one more PM. The similar pattern repeats and another PM is switched on at time $t_b + 9.2$ when *BLK* reaches thr_b again. The blocking probability decreases suddenly and increases again just like two previous occurrences of self-adaptation, but this time, it becomes stable and reaches 0.062. Therefore, no more PMs are needed to be switched on during the burst period. This analysis shows that the total number of 13 PMs are required and sufficient for our self-adaptive scheme to prevent violations of blocking probability. This demonstrates why we choose *baseline 2* to include 13 powered-on PMs. As can be seen in Fig. 4, after the burst starts at t_b, *BLK* gradually increases for *baseline 2*, and finally, reaches 0.062 at time t_e. The blocking probability at t_e, in both the proposed scheme and *baseline 2* are the same since the number of powered-on PMs is the same for these schemes and both of them almost reach their steady-state at t_e. This is also the case in the results reported for other metrics.

At the end of burst duration, the curves of both *baseline 1* and *baseline 2* fall down and reach their corresponding steady-state values as they were before burst begins. However, *baseline 1* needs more time to become stable because the number of waiting requests for *baseline 1* is greater than that for *baseline 2* at time t_e. On the other hand, the blocking probability of the self-adaptive scheme oscillates before reaching its steady-state value. The reason can be explained by considering the behavior of the proposed scheme when utilization of resources changes according to Fig. 5.

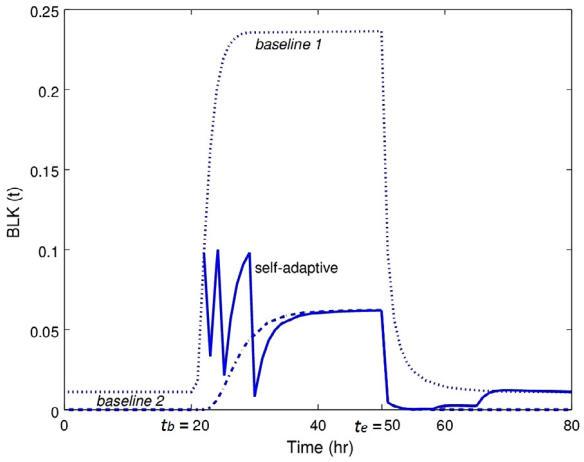

Fig. 4. Transient analysis of blocking probability.

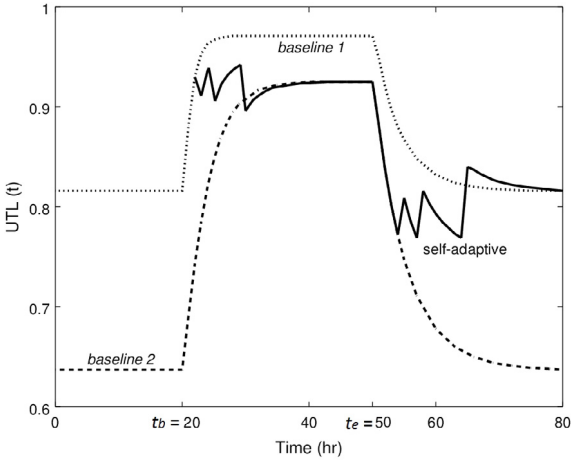

Fig. 5. Transient analysis of utilization.

7.2. Utilization of resources

The utilization of resources at time t, $UTL(t)$, for the proposed scheme and the two baselines is shown in Fig. 5. Before time point t_b, the steady-state utilization is 0.816 for both the self-adaptive scheme and *baseline 1*, and it is 0.637 for *baseline 2*. Starting from t_b, *UTL* grows fast for the self-adaptive scheme and *baseline 1*, so that it reaches the stable value 0.971 for *baseline 1*. However, since the self-adaptive scheme prevents the violation of blocking probability by adapting itself at times $t_b + 2$, $t_b + 4.2$, and $t_b + 9.2$ as mentioned before, there are three decreases and subsequent increases in the $UTL(t)$ curve of the proposed scheme at those times. At each overload adaptation, adding 8 new VMs to the resource pool causes the utilization to decrease. At the end of burst duration t_e, the utilization of resources for the self-adaptive

scheme is 0.925. On the other hand, *UTL* for *baseline 2* reacts to the burst by exponentially increasing its value to 0.925 at t_e, the same value for the self-adaptive scheme.

After the burst finishes, the utilization of resources of both *baseline 1* and *baseline 2* decreases. This is due to the reduction of the number of provisioned VMs which causes the number of available VMs to increase after the workload decreases. The utilization of resources in both *baseline 1* and *baseline 2* falls down, so that they reach their corresponding steady-state values as they were before the burst starts. Starting from time point t_e, *UTL* for the self-adaptive scheme acts the same as that for *baseline 2* before time $t_e + 4$. At time $t_e + 4$, an underload adaptation is performed to prevent the utilization from falling down the threshold thr_u by removing 8 VMs from the resource pool which is equivalent to switch an idle PM off. Decreasing the number of available VMs, a sudden increase in utilization of resources happens. As more provisioned VMs are released and returned to the resource pool over time, the utilization decreases. Therefore, two other underload adaptations are performed at time points $t_e + 7$ and $t_e + 14$, each one removing 8 more available VMs from the system. No more adaptations are needed because the utilization never reaches thr_u again, and it finally reaches the stable value 0.816 at the end of our analysis period. The oscillations of the $BLK(t)$ curve of the proposed scheme after time t_e, shown in Fig. 4, are the consequences of these underload adaptations, each of which causes a small transient increase of blocking probability.

7.3. Instant service probability

The transient analysis of the instant service probability, $ISP(t)$, for the proposed scheme and the two baselines is shown in Fig. 6. Before time t_b, the steady-state instant service probability is 0.304 for both the self-adaptive scheme and *baseline 1*, and it is 0.312 for *baseline 2*. Starting from t_b, *ISP* decreases fast for all three schemes. *ISP* decreases until it reaches 0.062 and 0.131 at t_e for *baseline 1* and *baseline 2*, respectively. The $ISP(t)$ curve for the self-adaptive scheme has the same behavior as *baseline 1*, and it reaches 0.129 at time point $t_b + 2$ when the first overload adaptation is performed by adding new VMs to the pool of available VMs. Unlike $BLK(t)$ and $UTL(t)$ that decrease when more available VMs are provided during an overload adaptation, $ISP(t)$ suddenly increases in the same situation. Continuing the burst, available VMs are gradually consumed, and thus, the instant service probability decreases until another adaptation is performed. The other two overload adaptations have similar impacts on $ISP(t)$. Immediately after time t_e, $ISP(t)$ of the proposed scheme behaves almost the same as $ISP(t)$ of *baseline 2*. However, subsequent underload adaptations cause the instant service probability to oscillate until it gradually reaches its corresponding value of *baseline 1*. It can be seen from Fig. 6 that *ISP* for our proposed scheme never becomes less than 0.115 during the analysis period, which indicates a remarkable improvement in contrast to *baseline 1*.

7.4. Power consumption

The transient analysis of the power consumption, $PWR(t)$, is shown in Fig. 7. Before time t_b, the steady-state power consumption is 2534 W for the self-adaptive scheme and *baseline 1*, and 3106 W for *baseline 2*. Starting from t_b, more VMs change their state from available to provisioned, resulting in an increase of power consumption for all three schemes. The *PWR* increases until it reaches 2654 and 3390 W at time t_e for *baseline 1* and *baseline 2*, respectively. The *PWR* curve for the self-adaptive scheme acts the same as that for *baseline 1* and reaches 2628 at time $t_b + 2$, when the first overload adaptation is performed. At this time, a stepwise increase of power consumption is made by switching a PM on. The newly added VMs, which are initially in the available state, are gradually provisioned for user requests, which further increases *PWR*. The other two subsequent adaptations during the burst period affect the $PWR(t)$ curve in the same manner. After the burst ends, each of the three underload adaptations results in a sudden decrease of power consumption by switching an idle PM off. Between every two subsequent underload adaptations, some VMs that had been provisioned during the burst are released causing *PWR* to decrease further. As shown in Fig. 7, *PWR* of our proposed scheme is much less than that of *baseline 2* in most of the analysis period which represents a significant improvement in contrast to *baseline 2*.

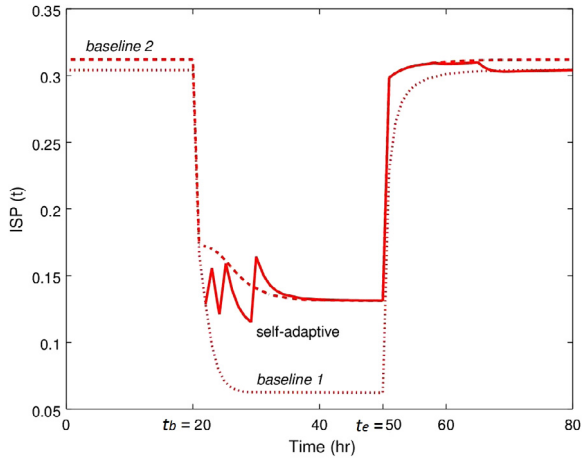

Fig. 6. Transient analysis of instant service probability.

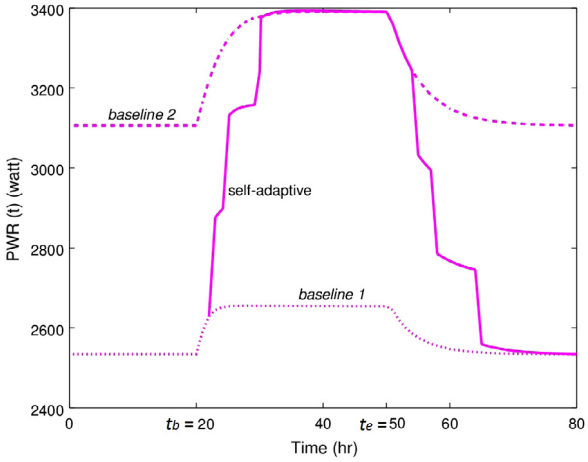

Fig. 7. Transient analysis of power consumption.

7.5. Waiting time

The transient analysis of the waiting time, $WTT(t)$, for the proposed scheme and the two baselines is represented in Fig. 8. Before time point t_b, the steady-state waiting time for both the self-adaptive scheme and *baseline 1* is 0.088 h, while it is 0.083 h for *baseline 2*. Starting from time point t_b, more requests are enqueued and less VMs are in available state, resulting in an increase of waiting time for all three schemes. The measure *WTT* increases until it reaches 0.265 and 0.126 h at time t_e for *baseline 1* and *baseline 2*, respectively. The *WTT* curve for the self-adaptive scheme acts the same as that for *baseline 1* and it reaches 0.140 h at $t_b + 2$ when the

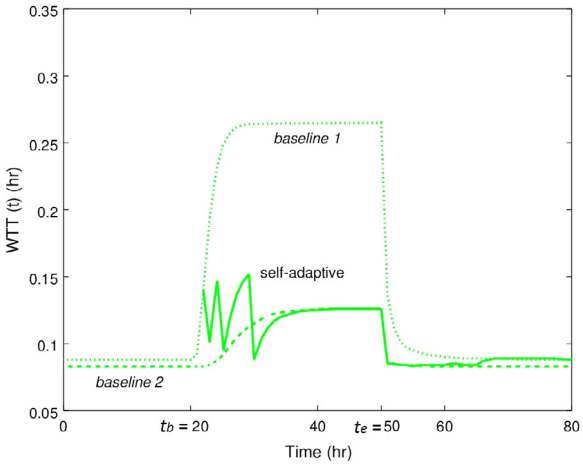

Fig. 8. Transient analysis of waiting time.

first overload adaptation is carried out. By providing new available VMs, a sudden decrease in *WTT* occurs. Continuing the burst, available VMs are gradually consumed, and thus, the waiting time increases until another adaptation is performed. The other two overload adaptations have similar impacts on *WTT*. Immediately after time t_e, *WTT* of the proposed scheme acts the same as *WTT* of *baseline 2*. However, subsequent underload adaptations cause the waiting time to gradually increase to its corresponding value for *baseline 1*. It can be observed from Fig. 8 that *WTT* for our proposed scheme never becomes greater than 0.152 h during the analysis period, which indicates a remarkable improvement in contrast to *baseline 1*.

7.6. Summary

Experiments show that the proposed scheme satisfies the crucial user- and provider-oriented measures by preventing the blocking probability from reaching threshold thr_b and the utilization of resources from falling below threshold thr_u. Meanwhile, the proposed scheme outperforms *baseline 1* in terms of the blocking probability, instant service probability, and waiting time, and *baseline 2* in terms of the utilization of resources and power consumption. An extensive analysis conducted on the proposed model and scheme indicates that the aforementioned improvements are acquired regardless of the values of input parameters. In particular, the number of PMs, N_{pm}, can be set to a larger value at the expense of some extra time for solving the model. Moreover, the blocking probability and utilization of resources can be replaced with other measures to capture other crucial metrics.

8. Model validation

To demonstrate the accuracy of our modeling technique and the presented self-adaptive resource management scheme, we compare the results obtained by analytically solving the SAN model, used in the proposed management scheme, against those obtained from the CloudSim framework. To achieve this goal, CloudSim is extended with respect to the set of functionalities already provided in the framework by defining the input queue for the data center and failure/repair events for VMs. Then, extensive simulation runs are executed and the results are gathered and analyzed from which we report two important measures, the power consumption and waiting time.

In Table 3, the $PWR(t)$ of the analytic–numeric approach is compared with that obtained from the simulation. Moreover, Table 4 compares the $WTT(t)$ acquired from the analytic–numeric approach and the corresponding values obtained from the simulation runs for different values of t. It can be observed that the maximum percent errors of the simulation results reported in Tables 3 and 4 are 4.2% and 6.8%, respectively. These results validate the proposed analytical model and management scheme against the simulation.

9. Conclusion and future work

In this paper, a scalable and flexible SAN model for an IaaS cloud, which takes details of real cloud systems into account, was proposed. The model is scalable because the number of PMs and VMs can be set to a large value as in real-world systems. It is flexible since various resource management strategies and policies can be designed and evaluated on top of this model. Applying the proposed SAN, a resource management scheme was presented for IaaS cloud systems equipped with a four-step control loop of self-adaptive systems. The proposed management scheme monitors status data collected from the cloud environment, analyzes them, plans to act, and reacts by switching PMs on/off in a way that neither user-oriented SLA metrics (e.g., blocking probability of requests) nor provider-oriented metrics (e.g., utilization of resources) are violated. Meanwhile, the power consumption of the cloud system is kept small using the proposed management scheme.

For performance evaluation, several performance measures (e.g., blocking probability, utilization of resources, instant service probability, power consumption, and waiting time) were defined. Among these measures, the blocking probability of requests and the utilization of resources are those crucial user-oriented and provider-oriented metrics, respectively, that should not be violated in our proposed scheme. Afterwards, a resiliency analysis was carried out by investigating the effect of workload changes on the behavior of the proposed self-adaptive resource management scheme and two baselines. The results indicated that, in contrast to our proposed scheme, the baselines violate crucial performance measures in many situations.

For future work, combining the proposed power-aware resource management scheme with other power saving approaches, one can reach a more comprehensive model. For example, the DVFS technique could be modeled along with our self-adaptive management scheme. In this way, VMs on top of PMs are divided into several groups in terms of processing speed and power consumption. Using colored extensions of Petri Nets, e.g., Timed Colored Petri Nets (TCPNs), tokens inside places can be differentiated by attaching type information to them. Therefore, our proposed model extended by TCPNs can support heterogeneity by modeling different classes of PMs, each one providing different processing, memory, and storage capacities. Moreover, different types of VMs with various software and hardware requirements can be modeled by colored extensions of Petri Nets. Taking network topology of data centers into account and considering performance effects due to the contention on shared resources could be other future extensions of the proposed model and management scheme.

Acknowledgments

This work has been supported in part by grant 2016R1A2B4009193 of the National Research Foundation of Korea and by the Promising-Pioneering Researcher Program of Seoul National University in 2015. ICT at Seoul National University provided research facilities for this study.

Table 3
Comparison of the power consumption resulted from the proposed approach and the simulation.

Approach	Power consumption (W)						
	20	30	40	50	60	70	80
Analytic–numeric	2534	3240	3393	3391	2767	2543	2534
Simulation	2539	3376	3411	3412	2781	2556	2550

Table 4
Comparison of the waiting time resulted from the proposed approach and the simulation.

Approach	Waiting time (h)						
	20	30	40	50	60	70	80
Analytic–numeric	0.088	0.088	0.125	0.126	0.084	0.089	0.088
Simulation	0.088	0.094	0.118	0.119	0.086	0.090	0.086

References

[1] D. Ardagna, G. Casale, M. Ciavotta, J.F. Pérez, W. Wang, Quality-of-service in cloud computing: modeling techniques and their applications, J. Internet Serv. Appl. 5 (11) (2014) 1–17.

[2] K. Bilal, S.U.R. Malik, S.U. Khan, A.Y. Zomaya, Trends and challenges in cloud datacenters, IEEE Cloud Comput. 1 (1) (2014) 10–20.

[3] AWS and sustainability, https://aws.amazon.com/about-aws/sustainability/ (Accessed: September 2017).

[4] C. Colman-Meixner, C. Develder, M. Tornatore, B. Mukherjee, A survey on resiliency techniques in cloud computing infrastructures and applications, IEEE Commun. Surv. Tutor. 18 (3) (2016) 2244–2281.

[5] Y. Sharma, B. Javadi, W. Si, On the reliability and energy efficiency in cloud computing, in: The 13th Australasian Symposium on Parallel and Distributed Computing, Sydney, Australia, 2015, pp. 111–114.

[6] M. Lassnig, T. Fahringer, V. Garonne, A. Molfetas, M. Branco, Identification, modelling and prediction of non-periodic bursts in workloads, in: The IEEE/ACM 10th International Conference on Cluster, Cloud and Grid Computing, CCGrid, Melbourne, Australia, 2010, pp. 485–494.

[7] N. Mi, G. Casale, L. Cherkasova, E. Smirni, Sizing multi-tier systems with temporal dependence: benchmarks and analytic models, J. Internet Serv. Appl. 1 (2) (2010) 117–134.

[8] K. RahimiZadeh, M. AnaLoui, P. Kabiri, B. Javadi, Performance modeling and analysis of virtualized multi-tier applications under dynamic workloads, J. Netw. Comput. Appl. 56 (2015) 166–187.

[9] S. Singh, I. Chana, R. Buyya, STAR: SLA-aware autonomic management of cloud resources, IEEE Trans. Cloud Comput. PP (99) (2017) (in press).

[10] D. Bruneo, A. Lhoas, F. Longo, A. Puliafito, Modeling and evaluation of energy policies in green clouds, IEEE Trans. Parallel Distrib. Syst. 26 (11) (2015) 3052–3065.

[11] M. Sedaghat, F. Hernandez-Rodriguez, E. Elmroth, Decentralized cloud datacenter reconsolidation through emergent and topology-aware behavior, Future Gener. Comput. Syst. 56 (2016) 51–63.

[12] B.H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al., Software engineering for self-adaptive systems: A research roadmap, in: B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems, in: Lecture Notes in Computer Science (LNCS), vol. 5525, Springer, 2009, pp. 1–26.

[13] D.G.D.L. Iglesia, D. Weyns, MAPE-K formal templates to rigorously design behaviors for self-adaptive systems, ACM Trans. Auton. Adapt. Syst. 10 (3) (2015) 1–31.

[14] A. Movaghar, J.F. Meyer, Performability modeling with stochastic activity networks, in: The 1984 Real-Time Systems Symposium, Austin, TX, 1984, pp. 215–224.

[15] J.F. Meyer, A. Movaghar, W.H. Sanders, Stochastic activity networks: Structure, behavior, and application, in: The International Workshop on Timed Petri Nets, Torino, Italy, 1985, pp. 106–115.

[16] A. Movaghar, Stochastic activity networks: A new definition and some properties, Sci. Iran. 8 (4) (2001) 303–311.

[17] W.H. Sanders, J.F. Meyer, Stochastic activity networks: Formal definitions and concepts, in: E. Brinksma, H. Hermanns, J.-P. Katoen (Eds.), Lectures on Formal Methods and Performance Analysis, in: Lecture Notes in Computer Science (LNCS), vol. 2090, Springer, 2001, pp. 315–343.

[18] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (1) (2011) 23–50.

[19] R. Ghosh, V.K. Naik, K.S. Trivedi, Power-performance trade-offs in IaaS cloud: A scalable analytic approach, in: The IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops, Hong Kong, China, 2011, pp. 152–157.

[20] R. Entezari-Maleki, L. Sousa, A. Movaghar, Performance and power modeling and evaluation of virtualized servers in IaaS clouds, Inform. Sci. 394–395 (2017) 106–122.

[21] E. Ataie, R. Entezari-Maleki, L. Rashidi, K.S. Trivedi, D. Ardagna, A. Movaghar, Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds, IEEE Trans. Cloud Comput. PP (99) (2017) (in press).

[22] D. Shen, J. Luo, F. Dong, X. Fei, W. Wang, G. Jin, W. Li, Stochastic modeling of dynamic right-sizing for energy-efficiency in cloud data centers, Future Gener. Comput. Syst. 48 (2015) 82–95.

[23] H. Khazaei, J. Misic, V.B. Misic, S. Rashwand, Analysis of a pool management scheme for cloud computing centers, IEEE Trans. Parallel Distrib. Syst. 24 (5) (2013) 849–861.

[24] X. Qiu, Y. Dai, Y. Xiang, L. Xing, A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service, IEEE Trans. Syst. Man Cybern. Syst. 46 (3) (2016) 401–412.

[25] R. Ghosh, F. Longo, V.K. Naik, K.S. Trivedi, Quantifying resiliency of IaaS cloud, in: The IEEE 29th Symposium on Reliable Distributed Systems, New Delhi, India, 2010, pp. 343–347.

[26] D. Bruneo, A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 560–569.

[27] D. Bruneo, F. Longo, R. Ghosh, M. Scarpa, A. Puliafito, K.S. Trivedi, Analytical modeling of reactive autonomic management techniques in IaaS clouds, in: The IEEE 8th International Conference on Cloud Computing, CLOUD, New York, NY, 2015, pp. 797–804.

[28] A. Pellegrini, P. Di Sanzo, D.R. Avresky, Proactive cloud management for highly heterogeneous multi-cloud infrastructures, in: The IEEE International Parallel and Distributed Processing Symposium Workshops, Chicago, IL, 2016, pp. 1311–1318.

[29] M.M. Alansari, B. Bordbar, Modelling and analysis of migration policies for autonomic management of energy consumption in cloud via Petri-nets, in: The International Conference on Cloud and Autonomic Computing, (ICCAC), IEEE, London, UK, 2014, pp. 121–130.

[30] Z. Zhou, J. Abawajy, M. Chowdhury, Z. Hu, K. Li, H. Cheng, A.A. Alelaiwi, F. Li, Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms, Future Gener. Comput. Syst. (2017) (in press).

[31] R. Ghosh, K.S. Trivedi, V.K. Naik, D.S. Kim, End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach, in: The IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan, 2010, pp. 125–132.

[32] F. Machida, D.S. Kim, K.S. Trivedi, Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration, Perform. Eval. 70 (3) (2013) 212–230.

[33] Y. Sharma, B. Javadi, W. Si, D. Sun, Reliability and energy efficiency in cloud computing systems: Survey and taxonomy, J. Netw. Comput. Appl. 74 (2016) 66–85.

[34] T.A. Nguyen, D.S. Kim, J.S. Park, Availability modeling and analysis of a data center for disaster tolerance, Future Gener. Comput. Syst. 56 (2016) 27–50.

[35] F. Machida, D.S. Kim, K.S. Trivedi, Modeling and analysis of software rejuvenation in a server virtualized system, in: The IEEE 2nd International Workshop on Software Aging and Rejuvenation, WoSAR, San Jose, CA, 2010, pp. 1–6.

[36] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, M. Scarpa, Workload-based software rejuvenation in cloud systems, IEEE Trans. Comput. 62 (6) (2013) 1072–1085.

[37] M. Nabi, M. Toeroe, F. Khendek, Availability in the cloud: State of the art, J. Netw. Comput. Appl. 60 (2016) 54–67.

[38] D.S. Kim, F. Machida, K.S. Trivedi, Availability modeling and analysis of a virtualized system, in: The IEEE 15th Pacific Rim International Symposium on Dependable Computing, Shanghai, China, 2009, pp. 365–371.

[39] S. Kaur, K. Kaur, D. Singh, A framework for hosting web services in cloud computing environment with high availability, in: The IEEE International Conference on Engineering Education: Innovative Practices and Future Trends, AICERA, Kottayam, India, 2012, pp. 1–6.
[40] R. Entezari-Maleki, K.S. Trivedi, A. Movaghar, Performability evaluation of grid environments using stochastic reward nets, IEEE Trans. Dependable Secure Comput. 12 (2) (2015) 204–216.
[41] R. Ghosh, F. Longo, F. Frattini, S. Russo, K.S. Trivedi, Scalable analytics for IaaS cloud availability, IEEE Trans. Cloud Comput. 2 (1) (2014) 57–70.
[42] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, K.S. Trivedi, Automated generation and analysis of markov reward models using stochastic reward nets, in: C.D. Meyer, R.J. Plemmons (Eds.), Linear Algebra, Markov Chains, and Queueing Models, in: The IMA Volumes in Mathematics and its Application, vol. 48, Springer, 1993, pp. 145–191.
[43] R.A. Sahner, K. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems: An Example-Based Approach using the SHARPE Software Package, first ed., Springer Science & Business Media, 1996.
[44] A.E.H. Bohra, V. Chaudhary, VMeter: Power modelling for virtualized clouds, in: The IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, IPDPSW, Atlanta, GA, 2010, pp. 1–8.
[45] Q. Chen, P. Grosso, K. van der Veldt, C. de Laat, R. Hofman, H. Bal, Profiling energy consumption of VMs for green cloud computing, in: The IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, DASC, Sydney, Australia, 2011, pp. 768–775.
[46] D. Daly, D.D. Deavours, J.M. Doyle, P.G. Webster, W.H. Sanders, Möbius: An extensible tool for performance and dependability modeling, in: The 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, 2000, pp. 332–336.
[47] E. Ataie, E. Gianniti, D. Ardagna, A. Movaghar, A combined analytical modeling machine learning approach for performance prediction of MapReduce jobs in cloud environment, in: The 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, Timisoara, Romania, 2016, pp. 431–439.
[48] D. Meisner, B.T. Gold, T.F. Wenisch, PowerNap: Eliminating server idle power, in: The 14th International Conference on Architectural Support for Programming Languages and Operating Systems, Washington, DC, 2009, pp. 205–216.

Ehsan Ataie is an Assistant Professor at the Department of Engineering and Technology, University of Mazandaran, Babolsar, Iran. He received the B.S., M.S. and Ph.D. degrees in Computer Engineering, all in Software discipline, from the Sharif University of Technology, Tehran, Iran in 2002, 2005, and 2017, respectively. He visited Politecnico di Milano in Italy in 2016. His main research interests include cloud computing, green computing, and performance and dependability modeling and evaluation.

Reza Entezari-Maleki is a Post-Doctoral Researcher in the School of Computer Science at Institute for Research in Fundamental Sciences (IPM) in Tehran, Iran. He received his Ph.D. in Computer Engineering (Software) from the Sharif University of Technology, Tehran, Iran in 2014, and M.S. and B.S. degrees in Computer Engineering (Software) from the Iran University of Science and Technology, Tehran, Iran in 2009 and 2007, respectively. He visited the Seoul National University in South Korea, twice in 2012 and 2017, Duke University in NC, USA, in 2013, and Instituto Superior Tecnico, Universidade de Lisboa in Portugal in 2015. His main research interests are performance/dependability modeling and evaluation, distributed computing systems, cloud computing, and task scheduling algorithms.

Sayed Ehsan Etesami received the M.S. in computer engineering (Information Technology) at the Department of Computer Engineering, Sharif University of Technology, Iran in 2018 and the B.S. degree in computer engineering (Information Technology) from the Department of Computer Engineering, Isfahan University of Technology, Iran in 2015. His general research interests are in the fields of performance evaluation, computer networking, internet of things, and cloud computing.

Bernhard Egger received the diploma in computer science from the Swiss Federal Institute of Technology in Zurich in 2001 and the Ph.D. degree in computer science and engineering from Seoul National University in 2008. After spending three years at SAIT, Samsung Electronics research institute, he rejoined Seoul National University in 2011 as a faculty member where he currently is an associate professor at the Department of Computer Science and Engineering. His research interests include programming language design, compilers, and operating systems for heterogeneous manycore systems. He is a member of the IEEE and ACM.

Danilo Ardagna is an Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano, Milan, Italy. He received the Ph.D. degree in Computer Engineering from Politecnico di Milano in 2004. His work focuses on the design, prototype and evaluation of optimization algorithms for resource management and planning of cloud systems.

Ali Movaghar is a Professor in the Department of Computer Engineering at Sharif University of Technology in Tehran, Iran and has been on the Sharif faculty since 1993. He received his B.S. degree in Electrical Engineering from the University of Tehran in 1977, and M.S. and Ph.D. degrees in Computer, Information, and Control Engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He visited the Institut National de Recherche en Informatique et en Automatique in Paris, France and the Department of Electrical Engineering and Computer Science at the University of California, Irvine in 1984 and 2011, respectively, worked at AT&T Information Systems in Naperville, IL in 1985–1986, and taught at the University of Michigan, Ann Arbor in 1987–1989. His research interests include performance/dependability modeling and formal verification of wireless networks and distributed real-time systems. He is a senior member of the IEEE and the ACM.