*algorithms*

MDPI

*Article*

# Modeling and Evaluation of Power-Aware Software Rejuvenation in Cloud Systems

**Sharifeh Fakhrolmobasheri [1,*], Ehsan Ataie [2] and Ali Movaghar [1]**

[1] Department of Computer Engineering, Sharif University of Technology, Tehran 1458889694, Iran; movaghar@sharif.edu

[2] Department of Engineering and Technology, University of Mazandaran, Babolsar 4741613534, Iran; ataie@umz.ac.ir

* Correspondence: mobasheri@ce.sharif.edu; Tel.: +98-913-326-4082

Received: 30 August 2018; Accepted: 15 October 2018; Published: 18 October 2018


**Abstract:** Long and continuous running of software can cause software aging-induced errors and failures. Cloud data centers suffer from these kinds of failures when Virtual Machine Monitors (VMMs), which control the execution of Virtual Machines (VMs), age. Software rejuvenation is a proactive fault management technique that can prevent the occurrence of future failures by terminating VMMs, cleaning up their internal states, and restarting them. However, the appropriate time and type of VMM rejuvenation can affect performance, availability, and power consumption of a system. In this paper, an analytical model is proposed based on Stochastic Activity Networks for performance evaluation of Infrastructure-as-a-Service cloud systems. Using the proposed model, a two-threshold power-aware software rejuvenation scheme is presented. Many details of real cloud systems, such as VM multiplexing, migration of VMs between VMMs, VM heterogeneity, failure of VMMs, failure of VM migration, and different probabilities for arrival of different VM request types are investigated using the proposed model. The performance of the proposed rejuvenation scheme is compared with two baselines based on diverse performance, availability, and power consumption measures defined on the system.

**Keywords:** cloud computing; software rejuvenation; aged software; performance evaluation

## 1. Introduction

Cloud computing is the practice of storing, managing, and processing data using a network of remote servers accessed via the Internet [1]. As such, cloud computing is a model for easy and ubiquitous access to a set of servers, storage devices, networks, applications, and services with minimal interaction with the service provider and minimal management cost [2]. Virtualization techniques are used in cloud systems to appropriately handle user requests. By applying virtualization, multiple Virtual Machines (VMs) are simultaneously run on a single Physical Machine (PM) controlled by software usually known as Hypervisor or Virtual Machine Monitor (VMM) [3–5]. In Infrastructure-as-a-Service (IaaS) clouds, each user request is in the form of one or more VM requests.

The tendency of software to fail or cause a system failure after running continuously for a specific time period is referred to as software aging [6,7]. Software aging is a phenomenon in long-run software systems that causes an increased failure rate and/or degraded performance due to accumulation of aging errors [8,9]. Since VM requests in IaaS clouds are usually different in terms of software and tools for which they are initiated, their corresponding VMs exhibit complex behaviors and sophisticated interactions throughout their lifetime that enable VMMs to manage a wide variety of VM behaviors. Consequently, after running for a long time or managing heavy workloads, VMMs, like any other software, age and slow due to a multitude of internal errors and diverse behaviors of VMs [7,10–13].

If these errors are not prevented or addressed properly, the performance of the system degrades and the cloud provider may fail to meet Quality of Service (QoS) requirements [14].

Software rejuvenation is an effective method used to cope with software aging and performance degradation [11,15,16]. In this method, software is restored to its original state before causing aging-induced crashes and system failures. In cloud systems, providers can also benefit from software rejuvenation by rejuvenating their VMMs before they fail. However, the main problem in this regard is how to choose the right VMM to be rejuvenated and the correct time to do so. In addition to the selection of the appropriate VMM and the proper time for rejuvenation, the selection of the right method amongst existing options is of utmost importance. The majority of models proposed in the literature use a series of side programs to measure the efficiency and performance of VMMs. A study showed that the majority of previous studies in the field of software rejuvenation adopted time-based or prediction-based strategies [17], both of them show low performance.

The rapid worldwide growth of the cloud computing paradigm has led to the emergence of massive data centers with high power consumption across the world [18]. In 2016, the total energy consumption of the world's data centers was 416 terawatt hours (TWh), which is 38% higher than the 300 TWh of energy that the entire U.K. consumed in the same year [19]. As a result, the power consumption of cloud computing servers and the field of green computing are worthy of more attention [20]. Active PMs, i.e., switched-on PMs, consume a major portion of a cloud system's power. Therefore, a cloud provider with many active but lightly loaded PMs/VMMs has high power waste, resulting in low energy efficiency. The technique proposed to increase the energy efficiency in this paper involves migration of VMs from lightly loaded VMMs, and then switching free PMs off, leading to a smaller number of active VMMs and PMs. In real-world cloud systems, requested VMs are heterogeneous and they differ in terms of processing speed, memory, and storage capacities. Therefore, in order to match real-world systems as much as possible, we consider two categories of VMs in this paper: (1) small VMs, which are VMs that need to be served for a short while and therefore utilize a small share of resources, and (2) big VMs, which are the VMs that need to be executed for a longer period of time and thus use a significant share of system resources. Over the course of their life, larger VMs have a greater impact on software aging of their host VMMs in contrast to the smaller VMs.

In this paper, we present a Stochastic Activity Networks (SAN) model for cloud data centers and provide a strategy for applying rejuvenation to candidate VMMs at the appropriate time. Model-based studies are amongst the most prevalent analyses that researches completed to study software aging and software rejuvenation [21]. Since one of the main causes of failures of VMMs is the continuous execution of VMs running on VMMs [17], the type and the total number of VMs executed by a VMM serve as the measure for determining the appropriate time of rejuvenation. Therefore, we define an upper bound for the maximum number of VMs that a VMM can run, starting from its last boot time, and force a rejuvenation whenever the age of a VMM reaches this limit. The age of a VMM can be expressed as the cumulative workload that the VMM has served since its last failure or rejuvenation. By analyzing the proposed model with the rejuvenation scheme, the benefits of applying the rejuvenation to VMMs of an IaaS cloud provider in reducing the number of VM failures and improving the performance and availability in the steady-state are observed. In addition, the proposed model enables power-awareness in the cloud management system by migrating active VMs from lightly loaded VMMs to more heavily loaded ones and switching off free PMs. Evaluation of the proposed scheme demonstrates that this power saving mechanism improves the system's steady-state performance and availability. The formalism used in this paper to model the power-aware VMM rejuvenation scheme is SANs [22–24]. SANs are probabilistic extensions of activity networks that are equipped with a set of activity time distribution functions, reactivation predicates, and enabling rate functions. They were developed to facilitate performance and dependability evaluation and have features that permit the representation of parallelism, timeliness, fault tolerance, and degradable performance [25]. More detailed information about SANs can be found in the previous literature [26–28].

The main contributions of this paper are summarized as follows. (1) A SAN model is presented for performance evaluation of IaaS clouds that considers many details of real systems, including VM multiplexing, migration of VMs between VMMs, VM heterogeneity, failure of VMMs, failure of VM migration, and different probabilities for arrival of different VM request types. (2) Using the presented SAN, a two-threshold power-aware software rejuvenation scheme is proposed and compared with two baselines based on diverse performance, availability, and power consumption measures defined on the system.

The rest of this paper is organized as follows. Section 2 reviews the previous work in the field of software rejuvenation in cloud systems. Section 3 provides a brief introduction to the SAN formalism. In Section 4, the general structure of our system is explained. In Section 5, the proposed SAN model and the rejuvenation scheme presented for the SAN are introduced in detail. Section 6 presents the output measures of interest and the results of the performance evaluation, and finally, Section 7 concludes the paper and provides some directions for future work.

## 2. Related Work

Evaluation of performance, availability, and power consumption of rejuvenation techniques in cloud systems using analytical models is an emerging topic. Machida et al. studied different methods of rejuvenation and suggested three main methods for software rejuvenation in a server virtualized system: cold, warm, and migration rejuvenation [11]. In the cold rejuvenation method, active VMs on a VMM that want to be rejuvenated are switched off. After completion of the VMM rejuvenation, the switched-off VMs are restarted on the rejuvenated VMM. In the warm method, active VMs are suspended, and after completion of the VMM rejuvenation, the suspended VMs resume their running on the rejuvenated VMM. In the migration method, active VMs on the intended VMM are moved to another host VMM before rejuvenation of the intended VMM occurs. When rejuvenation is completed, they move back to their original PM. Therefore, the downtime caused by the migration method is shorter than that of other methods. Our model is also based on the migration rejuvenation method. In the model presented in Machida et al. [11], the rejuvenation is periodically performed without considering the system's workload. In contrast, the rejuvenation in our model is performed according to the VMM's age.

Bruneo et al. analyzed the best period for rejuvenation to achieve high availability considering the VMM's workload [15]. They suggested a specific period of rejuvenation as the best one for each specified input workload. By changing the rate of input workload, the rejuvenation period also changes. More simply, the heavier the amount of workload, the longer the rejuvenation period; and the lighter the input workload, the shorter the rejuvenation period. Thus, rejuvenation time is adopted according to the VMM's workload. In contrast to Bruneo et al. [15], in our proposed model, the rejuvenation time is determined based on the age of a VMM instead of using a constant period, which provides a flexible model and an accurate analysis.

Melo et al. presented a rejuvenation model for cloud data centers to evaluate the availability in the steady-state [10]. In their model, only one active VMM and one standby VMM were considered. Our proposed model is similar to the model presented in Melo et al. [10]. In both models, the age of a VMM increases when a new workload is admitted. However, in Melo et al. [10], rejuvenation is periodically performed, and the status of a VMM is checked at the end of each period to ensure that performance degradation is due to the age of the VMM and not the overload of the data center itself. If rejuvenation is required, all active VMs on the target VMM are moved to the standby VMM to continue their execution. Notably, the model with only one active VMM is not realistic for cloud data centers. In contrast to Melo et al. [10], we consider several active VMMs in our model.

Melo et al. studied cloud data centers in different scenarios and analyzed different periods for setting rejuvenation timer in each scenario [29]. The authors specified the best rejuvenation time for achieving high availability in the steady-state. Liu et al. studied rejuvenation from different aspects [16]. Amongst the presented ideas in Liu et al. [16], saving a system's checkpoints periodically can also

be used in case of failure to recover the state of the system. The authors applied warm rejuvenation, in which paused VMs are maintained on external memory. Compared to the approach presented in Liu et al. [16], both migration rejuvenation and power-awareness are simultaneously applied in our presented model.

Roohitavaf et al. represented a SAN model using the Möbius tool to evaluate the availability and power consumption of cloud data centers [22]. Based on their evaluation, by switching off low load VMMs, providers not only decrease the power consumption of a cloud data center but also reach higher availability if the migration time is short enough. Roohitavaf et al. [22] only considered the energy saving aspect of cloud data centers; in contrast, we studied the simultaneous effect of power saving and rejuvenation on cloud data centers.

There are also other related studies in the field of VMM rejuvenation that presented models and schemes using techniques other than analytical modeling. Kourai and Ooba proposed VMBeam, which enables lightweight software rejuvenation of virtualized systems using zero-copy migration [30]. The proposed VMBean has been implemented in Xen, an open source hypervisor. Sudhakar et al., proposed a neural network-based approach to approximate the non-linear relationship between resource usage statistics and the time to failure of VMMs caused by aging-related errors [31]. Araujo et al., investigated the software aging effects in the Eucalyptus framework, considering workloads composed of intensive requests for remote storage attachment and VM instantiations [32]. They also presented an approach that applies time series analysis to schedule rejuvenation to reduce the downtime by predicting the proper moment to perform the rejuvenation.

In summary, the approaches that analyze the performance, availability, and power consumption of VMM rejuvenation tasks in cloud systems using analytical models are rare. In contrast to existing work, our presented scheme allows power-aware rejuvenation of VMMs using a two-threshold mechanism. The proposed model supports PM heterogeneity, which has not been considered in previously presented models. Moreover, the model considers several other aspects of real cloud systems that have not usually been considered in previously published reports. In addition, the number of PMs/VMMs and the number of VMs that can simultaneously be serviced on a PM/VMM are larger than those modeled in other approaches.

## 3. Overview of SANs

SANs are stochastic generalizations of Petri Nets (PNs), defined for modeling and analysis of distributed real-time systems [27,28]. SANs, which are more powerful and flexible than other extensions of PNs, can be formally defined as a 7-tuple *(P, IA, T A, IG, OG, IR, OR)* [24], where:

- $P$ is a finite set of places. Places are similar to places in PNs and are represented by circles.
- $IA$ is a finite set of instantaneous activities that represent system activities that are completed almost instantly compared to the corresponding performance variables. Each instantaneous activity is assigned to a case probability function, which returns a value between zero and one, that allows us to express uncertainty. Instantaneous activities are represented by thin lines.
- $TA$ is a finite set of timed activities that represent system activities whose running time affects the system's functionality and evaluation results. A timed activity is exhibited with $X$ inputs and $Y$ outputs, where $X + Y > 0$. Each input can be a place or an input gate, and each output can be a place or an output gate. Each timed activity is assigned to an activity time distribution function and an enabling function. Timed activities are represented by thick lines.
- $IG$ is a finite set of input gates. Each input gate has a finite number of inputs. Each $G \in IG$ with $m$ inputs is associated with a function $f_G : N^m \rightarrow N^m$, called the function of G; and a predicate $g_G : N^m \rightarrow \{true, false\}$, called the enabling predicate of $G$. Input gates provide more flexibility in defining activity enabling and completion rules. Input gates are represented by triangles.

- $OG$ is a finite set of output gates. Each output gate has a finite number of outputs. Each $G \in OG$ with $m$ outputs is associated with a function $f_G : N^m \rightarrow N^m$, called the function of $G$. Like input gates, output gates are represented by triangles.
- $IR \subseteq P \times \{1, ..., |P|\} \times IG \times (IA \cup TA)$ is the input relation. $IR$ satisfies the following conditions:
  - For any $(P_1, i, G, a) \in IR$ such that $G$ has $m$ inputs, $i \leq m$,
  - For any $G \in IG$ with $m$ inputs and $i \in N$, $i \leq m$, there exists $a \in (IA \cup T\ A)$ and $P_1 \in P$, such that $(P_1, i, G, a) \in IR$,
  - For any $(P_1, i, G_1, a), (P_1, j, G_2, a) \in IR$, $i = j$ and $G_1 = G_2$,
- $OR \subseteq (IA \cup TA) \times OG \times \{1, ..., |P|\} \times P$ is the output relation. $OR$ satisfies the following conditions:
  - For any $(a, G, i, P_1) \in OR$, such that $G$ has $m$ outputs, $i \leq m$,
  - For any $G \in OG$ with $m$ outputs and $i \in N$, $i \leq m$, there exists $a \in (IA \cup T\ A)$ and $P_1 \in P$. such that $(a, G, i, P_1) \in OR$,
  - For any $(a, G_1, i, P_1), (a, G_2, j, P_1) \in OR$, $i = j$ and $G_1 = G_2$.

Several tools have been developed, such as Möbius [33], UltraSAN [34], and METASAN [35], to provide support for SANs as a modeling formalism. All these software packages are intended for automatic evaluation of performance, dependability, and performability of systems. Among them, we chose Möbius as the modeling tool in this paper.

## 4. System Description

Figure 1 shows the structure of an IaaS cloud system where user requests are expressed as requests for VMs. This system contains $N$ physical machines, each one with an active VMM. User requests are placed in the input queue upon arrival. The migration queue is used when VMs need to be migrated between VMMs.

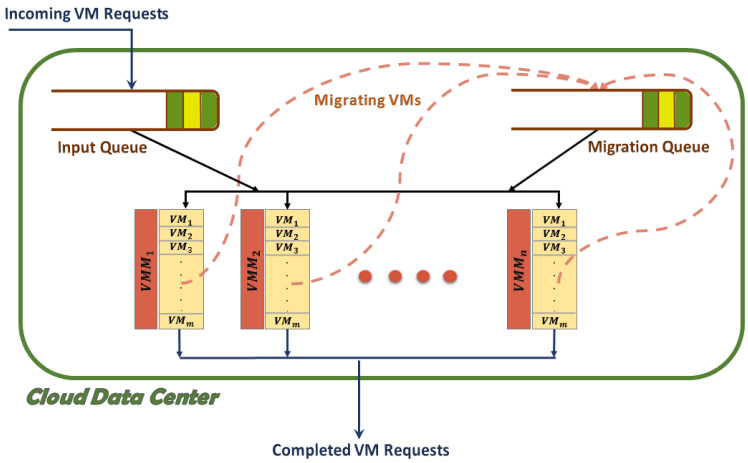


**Figure 1.** The structure of the Infrastructure-as-a-Service (IaaS) cloud system under study.

Right after system startup, all PMs of the cloud data center are assumed to be switched-off and they are switched on only on demand after receiving requests for VMs. Since active PMs consume more power, as long as there is free capacity on active VMMs, no PM is switched on. In other words, a new PM is switched on only if none of the active VMMs have sufficient capacity to accept migrating or requested VMs. When there is a VM request in the input queue, one of the active VMMs with free capacity is randomly selected to service the request. Similarly, when there is a VM in the migration queue, one of the active VMMs with free capacity is randomly chosen to service the migrating VM. We supposed that a PM can host up to $M$ VMs concurrently. After finishing the tasks defined for a VM,

the resources assigned to that VM are returned to their corresponding VMM, thereby providing the VMM some capacity to accept another VM. In order to make the model more realistic, we assumed that the system is able to handle two different types of VMs: big and small VMs. Big VMs are VMs that have a greater impact on VMM aging due to having a longer runtime or equivalently, using a more significant share of system resources. Here, we assumed that a big VM has a runtime twice as large as the runtime of a small VM.

A VMM may fail due to different reasons. One of the notable causes of VMM failure is aging-related errors. If an aged VMM is not rejuvenated at in time, the VMM fails. Thus, the failure of all VMs running on top of that VMM is possible. In serving a VM, the age of its hosting VMM increases, but completion of that VM does not decrease the VMM's age. This is because running a VM has some adverse effects on the corresponding VMM that are not removed once VM's tasks are completed. Therefore, both the number and the type of VMs served by a VMM can be considered indicators of its age. By VMM age, we mean the cumulative workloads that are accepted and served by that VMM after its last reboot. In our considered system, once the age of a VMM reaches the maximum threshold $F$, regardless of there being free capacity on other VMMs, the active VMs of that VMM are put in the migration queue, and the VMM is then rejuvenated. In order to improve system reliability, when the age of a VMM reaches the threshold $F'$ ($F' < F$), and if other VMMs have enough free capacity to accept the VMs running on that VMM, these VMs are put in the migration queue, and the aged VMM is rejuvenated before reaching the threshold $F$. With the aim of power saving, when a VMM is relatively idle and other VMMs can serve its active VMs, these VMs are put in the migration queue to be migrated to those VMMs, and the emptied PM is switched off.

## 5. The Proposed Model

The proposed model is composed of five sub-models shown in Figure 2. The insertion sub-model (Figure 2a) models the arrival of user requests to the system, the migration sub-model (Figure 2b) addresses the migration of VMs between VMMs, the service and green computing sub-model (Figure 2c) models serving VMs and termination of lightly loaded VMMs with the aim of green computing goals, the rejuvenation sub-model (Figure 2d) addresses rejuvenation of the aged VMMs, and the failure sub-model (Figure 2e) models the failure of VMMs. We assumed that the times assigned to all timed activities follow an exponential distribution [36–39]. In Figure 2a, the arrival of new requests to the system and selection of appropriate VMM to handle them are modeled. Timed activity $TA_{Arr}$ models the arrival of VM requests. Upon completion of $TA_{Arr}$, a token is inserted into place $P_{Ini}$. Place $P_{Ini}$ represents the input queue, and a token in this place serves as a request waiting in the queue. Once the number of tokens in place $P_{Ini}$ reaches the threshold $Q_{in}$, representing the capacity of the input queue, the input gate $IG_{Arr}$ prevents timed activity $TA_{Arr}$ from completion. The completion rate of activity $TA_{Arr}$ is $\lambda_{va}$. The timed activity $TA_{Ini}$ randomly selects one of the active VMMs to serve a requested VM. Whenever at least one token is in place $P_{Ini}$, representing a VM request, and there is enough capacity on one of the VMMs to accept a new VM, timed activity $TA_{Ini}$ is enabled and can complete the activity. Upon completion of this activity, one token is removed from place $P_{Ini}$ by input gate $IG_{Ini}$. If more than one VMM can accept an incoming request, $TA_{Ini}$ randomly chooses the destination VMM. When $VMM_i$ is chosen by this activity, one of the output gates $OG_{Ini_{S_i}}$ or $OG_{Ini_{B_i}}$ is selected. If the requested VM is small, the output gate $OG_{Ini_{S_i}}$ is enabled; otherwise, the output gate $OG_{Ini_{B_i}}$ is enabled. Here, we assumed that the chance of arrival of a small VM is twice as large as that of a big VM. In the output gate $OG_{Ini_{S_i}}$, the number of active VMs running on VMM$_i$ and the age of that VMM are both increased by one. However, in the output gate $OG_{Ini_{B_i}}$, the number of active VMs running on VMM$_i$ increases by one and the related age increases by two. The completion rate of activity $TA_{Ini}$ depends on the number of VMs that can be assigned to all available VMMs and the number of VM requests waiting in place $P_{Ini}$whichever is smaller—and the initiation rate of a VM is $\lambda_{vi}$. The rate of activity $TA_{Ini}$ is demonstrated in Table 1. The predicates and functions corresponding to the input gates of the insertion sub-model are presented in rows 1 and 2 of Table 2,

and the functions corresponding to the output gates of the insertion sub-model are presented in rows 1 and 2 of Table 3. In these tables, the $P_i \rightarrow Mark()$ function returns the number of tokens inside place $P_i$. Notably, though places, activities, and gates of the proposed sub-models are not explicitly connected to each other, these sub-models are implicitly interconnected. The actual connections can be considered by reviewing the predicates and functions of the gates of these sub-models presented in Tables 2 and 3. As an example, the predicate of $IG_{Arr}$ shown in Table 2 indicates that the activation of this input gate of insertion sub-model is dependent on the markings of places $P_{CurrentVMs_i}$ and $P_{Mig}$, which are elements of service and green computing sub-model and migration sub-model, respectively.

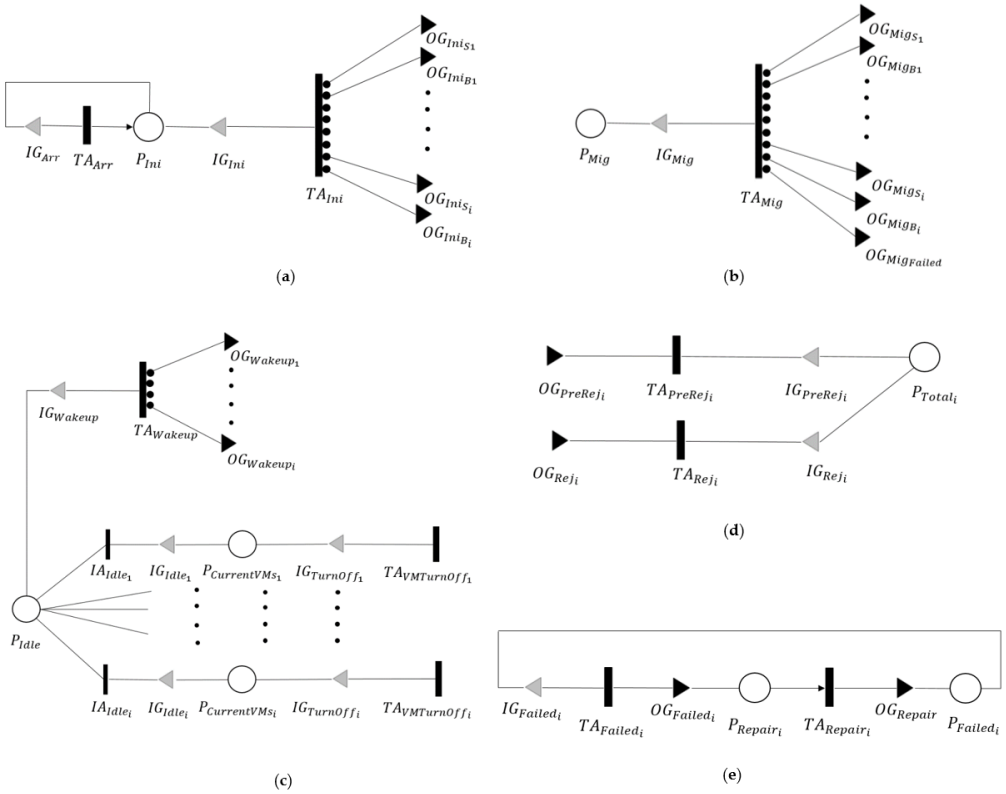


**Figure 2.** The proposed Stochastic Activity Network (SAN) for (**a**) insertion sub-model, (**b**) migration sub-model, (**c**) service and green computing sub-model, (**d**) rejuvenation sub-model, and (**e**) failure sub-model.

**Table 1.** Rate of timed activities of the proposed Stochastic Activity Network (SAN) model.

| Timed Activity | Rate |
|---|---|
| $TA_{Ini}$ | $possibleIni = \sum_{i=1}^{N} (P\_total_i \rightarrow Mark() == T_2)$ && $(P\_Failed_i \rightarrow Mark() == 1)$?<br>M: (M – $P_{CurrentVMs_i} \rightarrow Mark()$);<br>$possibleIni$ - = $P_{Mig} \rightarrow Mark()$;<br>$minimum$ = ($possibleIni$ <= $P_{Ini} \rightarrow Mark()$)? $possibleIni$: $P_{Ini} \rightarrow Mark()$;<br>return $minimum * \lambda_{vi}$; |
| $TA_{Mig}$ | $possibleIni = \sum_{i=1}^{N} (P\_total_i \rightarrow Mark() == T_2)$ && $(P\_Failed_i \rightarrow Mark() == 1)$?<br>M: (M – $P_{CurrentVMs_i} \rightarrow Mark()$);<br>$possibleIni$ - = $P_{Mig} \rightarrow Mark()$;<br>$minimum$ = ($possibleIni$ <= $P_{Mig} \rightarrow Mark()$)? $possibleIni$: $P_{Mig} \rightarrow Mark()$;<br>return $minimum * \lambda_{vmg}$; |

**Table 2.** Predicates and functions of input gates of the proposed SAN mod.

| Gate | Predicate | Function |
|---|---|---|
| $IG_{Arr}$ | $systemLoad = \sum_{i=1}^{N} P_{CurrentVMs_i} \rightarrow Mark() + P_{Ini} \rightarrow Mark() + P_{Mig} \rightarrow Mark();$<br>$(Q_{in} > systemLoad)?$ return *1: 0;* | ; |
| $IG_{Ini}$ | $availCapacity = \sum_{i=1}^{N} (P\_total_i \rightarrow Mark() == T_2)$<br>&& $(P\_Failed_i \rightarrow Mark() == 1)$ ?<br>M : (M – $P_{CurrentVMs_i} \rightarrow Mark()$);<br>$availCapacity$ - = $P_{Mig} \rightarrow Mark()$;<br>($P_{Ini} \rightarrow Mark() > 0$) && ($availCapacity > 0$)? return 1: 0; | $\boldsymbol{P_{Ini} \rightarrow Mark() - -;}$ |
| $IG_{Mig}$ | $availCapacity = \sum_{i=1}^{N} (P_{Total_i} \rightarrow Mark() == T_2)$ && $(P_{Failed_i} \rightarrow Mark() == 1)$ ?<br>M : (M - $P_{CurrentVMs_i} \rightarrow Mark()$);<br>($P_{Mig} \rightarrow Mark() > 0$) && ($availCapacity > 0$)? return 1 : 0; | $\boldsymbol{P_{Mig} \rightarrow Mark() - -;}$ |
| $IG_{Wakeup}$ | $availCapacity = (N - \sum_{i=1}^{N} P_{Failed\,i} \rightarrow Mark())$ * M;<br>$systemLoad = \sum_{i=1}^{N} P_{CurrentVMs_i} \rightarrow Mark();$<br>($availCapacity < systemLoad + P_{Ini} \rightarrow Mark() + P_{Mig} \rightarrow Mark()$) &&<br>$(P_{Idle} \rightarrow Mark() > 0)?$ return *1: 0;* | $\boldsymbol{P_{Idle} \rightarrow Mark() - -;}$ |
| $IG_{Idle_i}$ | $availCapacity = \sum_{j=1\ .i \neq j}^{N} (P_{Total_j} \rightarrow Mark() == T_2)$<br>&& $\left(P_{Failed_j} \rightarrow Mark() == 1\right)$ ?<br>0 : (M - $P_{CurrentVMs_i} \rightarrow Mark()$);<br>($P_{Failed_i} \rightarrow Mark() < 1$) &&<br>($availCapacity > P_{CurrentVMs_i} \rightarrow Mark() + P_{Ini} \rightarrow Mark() + P_{Mig} \rightarrow Mark()$)?<br>return 1: 0; | $\boldsymbol{P_{Mig} \rightarrow Mark() +=}$<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark()}$;<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark() = 0}$;<br>$\boldsymbol{P_{Total_i} \rightarrow Mark() = 0}$; |
| $IG_{TurnOff_i}$ | $P_{CurrentVMs_i} \rightarrow Mark() > 0$ | $\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark() - -;}$ |
| $IG_{Rej_i}$ | ($P_{Total_i} \rightarrow Mark() == T_2$) && ($P_{Failed_i} \rightarrow Mark() < 1$) | $\boldsymbol{P_{Mig} \rightarrow Mark() +=}$<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark()}$;<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark() = 0}$; |
| $IG_{PreRej_i}$ | $availCapacity = \sum_{j=1\ .i \neq j}^{N} (P_{Total_j} \rightarrow Mark() == T_2)$<br>&& $\left(P_{Failed_j} \rightarrow Mark() == 1\right)$ ?<br>0: (M – $P_{CurrentVMs_i} \rightarrow Mark()$);<br>($availCapacity > P_{CurrentVMs_i} \rightarrow Mark()$ + $P_{Ini} \rightarrow Mark() + P_{Mig} \rightarrow Mark()$) &&<br>($P_{Failed_i} \rightarrow Mark() < 1$) && ($P_{Total_i} \rightarrow Mark() == T_1$)? return 1: 0; | $\boldsymbol{P_{Total_i} \rightarrow Mark() = T_2}$;<br>$\boldsymbol{P_{Mig} \rightarrow Mark() +=}$<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark()}$;<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark() = 0}$ |
| $IG_{Failed_i}$ | $VMM\_load_i = (P_{Failed_i} \rightarrow Mark() < 1)?$ $P_{CurrentVMs_i} \rightarrow Mark()$: M+1;<br>($(P_{Failed_i} \rightarrow Mark() == 0)$ && ($VMM\_load_i$ < M + 1))? return 1: 0; | $\boldsymbol{P_{Failed_i} \rightarrow Mark() + +;}$<br>$\boldsymbol{P_{CurrentVMs_i} \rightarrow Mark() = 0}$; |

**Table 3.** Functions of outputs gates of the proposed SAN model.

| Gate | Function |
|---|---|
| $OG_{Ini_{S_i}}$ | $VMM\_Load_i = ((P_{Failed_i} \rightarrow Mark() < 1)$ && $(P_{Total_i} \rightarrow Mark() < T_2))?$ $P_{CurrentVMs_i} \rightarrow Mark()$ : M;<br>$(VMM\_Load_i < M)?$ {$P_{currentVMs_i} \rightarrow Mark()$ ++; $P_{Total_i} \rightarrow Mark()$ ++;}: $P_{Ini} \rightarrow Mark()$ ++; |
| $OG_{Ini_{B_i}}$ | $VMM\_Load_i = ((P_{Failed_i} \rightarrow Mark() < 1)$ && $(P_{Total_i} \rightarrow Mark() < T_2 - 2))?$ $P_{CurrentVMs_i} \rightarrow Mark()$ : M;<br>$(VMM\_Load_i < M)?$ {$P_{CurrentVMs_i} \rightarrow Mark()$ ++; $P_{Total_i} \rightarrow Mark()$ += 2;}: $P_{Ini} \rightarrow Mark()$ ++; |
| $OG_{Mig_{S_i}}$ | $VMM\_Load_i = ((P_{Failed_i} \rightarrow Mark() < 1)$ && $(P_{Total_i} \rightarrow Mark() < T_2))$ ? $P_{CurrentVMs_i} \rightarrow Mark()$ : M;<br>$(VMM\_Load_i < M)?$ {$P_{CurrentVMs_i} \rightarrow Mark()$ ++; $P_{Total_i} \rightarrow Mark()$ ++;}: $P_{Mig} \rightarrow Mark()$ ++; |
| $OG_{Mig_{B_i}}$ | $VMM\_Load_i = ((P_{Failed_i} \rightarrow Mark() < 1)$ && $(P_{Total_i} \rightarrow Mark() < T_2$ - 2))? $P_{CurrentVMs_i} \rightarrow Mark()$ : M;<br>$(VMM\_Load_i < M)?$ {$P_{CurrentVMs_i} \rightarrow Mark()$ ++; $P_{Total_i} \rightarrow Mark()$ += 2;}: $P_{Mig} \rightarrow Mark()$ ++; |
| $OG_{Mig_{Failed}}$ | $P_{Mig} \rightarrow Mark()$ ++; |
| $OG_{Wakup_i}$ | $P_{CurrentVMs_i} \rightarrow Mark() = 0;$ |
| $OG_{Rej_i}$<br>$OG_{PreRej_i}$ | $P_{Total_i} \rightarrow Mark() = 0;$<br>$P_{Total_i} \rightarrow Mark() = 0;$ |
| $OG_{Failed_i}$ | $P_{Total_i} \rightarrow Mark() = 0;$<br>$P_{Repair_i} \rightarrow Mark()$ ++; |
| $OG_{Repairi}$ | $P_{Failed_i} \rightarrow Mark()$ - -;<br>$P_{CurrentVMs_i} \rightarrow Mark() = 0;$ |

In Figure 2b, the migration of VMs between VMMs is modeled. When the rejuvenation mechanism or the power saving scheme requires a VM to migrate, a token is put into place $P_{Mig}$, representing the migration queue. Then, timed activity $TA_{Mig}$ randomly selects one of the active VMMs that have enough capacity to accept the migrating VM. When there is no VMM with free capacity to serve a migrating VM, input gate $IG_{Mig}$ prevents timed activity $TA_{Mig}$ from completion. Assuming that $VMM_i$ is selected to serve the migrating VM, if the VM is small, output gate $OG_{Mig_{S_i}}$ is enabled, and the number of active VMs and the age of that VMM increases by one. If the migrating VM is large, output gate $OG_{Mig_{B_i}}$ is enabled, the number of active VMs running on $VMM_{\mathrm{i}}$ increases by one, and the age of the corresponding VMM increases by two. The completion rate of activity $TA_{Mig}$ depends on the number of VMs that can be assigned to all available VMMs and the number of VMs waiting for migration in place $P_{Mig}$, whichever is smaller, and the migration rate of a VM is called $\lambda_{vmg}$ [40]. The rate of activity $TA_{Mig}$ is demonstrated in Table 1. We also assumed a constant number for probability of failure of a VM migration task. Therefore, we assumed that output gate $OG_{MigFailed}$ is enabled with probability $p_{vmf}$. In this output gate, the removed token from place $P_{Mig}$ turns back to place $P_{Mig}$. Using this mechanism, the migration of a VM fails and should be retried. The predicate and function corresponding to the input gate of the migration sub-model are presented in row 3 of Table 2, and the output functions of the output gates of this sub-model are presented in rows 3–5 of Table 3.

In Figure 2c, timed activity $TA_{VMTurnOff_i}$ models the serving process of a VM on the $\mathrm{VMM_i}$. Upon completion of activity $TA_{VMTurnOff_i}$, the number of tokens in place $P_{CurrentVM_{S_i}}$ decreases by one. The tokens in place $P_{CurrentVM_{S_i}}$ represent the number of active VMs on the $\mathrm{VMM_i}$. When there is no token in place $P_{CurrentVM_{S_i}}$, input gate $IG_{TurnOff_i}$ disables time activity $TA_{VMTurnOff_i}$. Serving individual VMs occurs at rate $\lambda_{vt}$, so the actual service rate of all VMs running on a VMM during the execution of timed activity $TA_{VMTurnOff_i}$ is equal to $\lambda_{vt} * Mark\left(P_{CurrentVM_{S_i}}\right)$, where $Mark(P_{CurrentVM_{S_i}})$ indicates the number of tokens inside place $P_{CurrentVM_{S_i}}$. In other words, the actual service rate of VMs running on a VMM is the product of the $\lambda_{vt}$ and the number of active VMs running on that VMM.

Timed activity $TA_{WakeUp}$, with completion rate $\lambda_{pw}$, models the PM wakeup process. The tokens inside place $P_{Idle}$ represent the number of inactive or switched-off PMs in the system. Right after system start up, all PMs are assumed to be switched off, so place $P_{Idle}$ holds as many tokens as there are PMs in the cloud data center. Upon completion of activity $TA_{WakeUp}$, a token is removed from place $P_{Idle}$ by input gate $IG_{WakeUp}$. If there is no token in places $P_{Ini}$ or $P_{Mig}$, input gate $IG_{WakeUp}$ prevents the completion of activity $TA_{WakeUp}$. If more than one token exists in place $P_{Idle}$, representing several switched-off PMs in the system, one token is randomly chosen by timed activity $TA_{WakeUp}$. In this case, $\mathrm{PM_i}$ is activated by enabling output gate $OG_{WakeUpi}$ and becomes ready for accepting VMs.

Instantaneous activity $IA_{Idlei}$ models the shutdown process of the active $VMM_i$ that is lightly loaded or has no workload. When other active VMMs have enough free capacity to accept all VMs running on top of $VMM_i$, the VMs are placed into the migration queue, and then $VMM_i$ and its hosting PM are switched off. Upon completion of instantaneous activity $IA_{Idlei}$, the number of tokens in place $P_{Idle}$ increases by one, and the number of tokens in place $P_{CurrentVM_{S_i}}$ is set to zero. If the tokens in place $P_{CurrentVM_{S_i}}$ cannot be moved to places $P_{CurrentVM_{S_j}}$ ($i \neq$ j), input gate $IG_{Idlei}$ prevents instantaneous activity $IA_{Idlei}$ from completion because other VMMs are unable to accept VMs running on $VMM_i$. The predicates and functions corresponding to the input gates of the service and green computing sub-model are presented in rows 4–6 of Table 2, and the function corresponding to the output gate of this sub-model is presented in row 6 of Table 3.

In Figure 2d, the number of tokens in place $P_{Total_i}$ represents the age of the $\mathrm{VMM_i}$. Therefore, the number of tokens in place $P_{Total_i}$ is set to zero just after rejuvenation or reboot of $\mathrm{VMM_i}$. When the number of tokens in place $P_{Total_i}$ reaches threshold $T_2$, the input gate $IG_{Rej}$ is enabled and moves tokens from place $P_{CurrentVM_{S_i}}$ to place $P_{Mig}$. By applying this mechanism, the migration of VMs from $VMM_i$ to other VMMs of the data center is modeled when a VMM has to be rejuvenated. Timed activity

$TA_{Rej_i}$ models the rejuvenation process of a VMM with rate $\lambda_{hrj}$. Upon completion of this activity, the number of tokens in place $P_{Total\,i}$ is set to zero by output gate $OG_{Rej_i}$, which represents the age of the newly rejuvenated $VMM_i$. A rejuvenated VMM is active and can accept and serve VMs. However, if there is no VM to be hosted by the rejuvenated VMM, the VMM is switched off by input gate $IG_{Idlei}$ modeled in the service and green computing sub-model.

The model also uses timed activity $TA_{PreRej_i}$ for early rejuvenation of aged VMM$_i$. When the number of tokens in place $P_{Total\,i}$ reaches the threshold $T_1$, where $T_1 < T_2$, and there are less tokens in place $P_{CurrentVM_{S_i}}$ than free capacity on other active VMMs, the input gate $IG_{PreRej_i}$ is enabled. In this situation, input gate $IG_{PreRej_i}$ moves all tokens of place $P_{CurrentVM_{S_i}}$ to place $P_{Mig}$. The completion rate of timed activity $TA_{PreRej_i}$ is $\lambda_{hrj}$. The output gate $OG_{PreRej_i}$ sets the number of tokens in place $P_{Total\,i}$ to zero right after the rejuvenation process. The predicates and functions corresponding to the input gates of the rejuvenation sub-model are presented in rows 7 and 8 of Table 2, and the functions corresponding to the output gates of this sub-model are presented in rows 7 and 8 of Table 3.

In Figure 2e, the failure of active VMMs in the system is modeled. Whenever $VMM_i$ fails, a token is put in place $P_{Failed\,i}$. After failure detection and during the repair of VMM$_i$, a token is put into place $P_{Repair\,i}$. Timed activity $TA_{Failed\,i}$ models the failure event of VMM$_i$. If there is a token in place $P_{Failed\,i}$ or there are as many tokens in place $P_{Total\,i}$ as in $T_2$, the input gate $IG_{Failed_i}$ disables activity $TA_{Failed\,i}$. Upon completion of activity $TA_{Failed\,i}$, the number of tokens in place $P_{Total\,i}$ is set to zero by output gate $OG_{Failed\,i}$, and the number of tokens in place $P_{Repair\,i}$ increases by one. Once VMM$_i$ is repaired, the numbers of tokens in places $P_{Repair\,i}$ and $P_{Failed\,i}$ become zero, and the VMM is reactivated. The completion rate of timed activity $TA_{Failed\,i}$ is the product of the number of tokens in place $P_{Total\,i}$ and the failure rate of an active VMM called $\lambda_{hf}$. The greater the age of a VMM, the greater the failure probability of that VMM.

Timed activity $TA_{Repair\,i}$ represents the repair process of VMM$_i$. When there is a token in place $P_{Repair\,i}$, timed activity $TA_{Repair\,i}$ is enabled and can complete the activity. Then, the number of tokens in places $P_{Total\,i}$ and $P_{Failed\,i}$ is set to zero by the output gate $OG_{Repair\,i}$. The completion rate of timed activity $TA_{Repair\,i}$ is $\lambda_{hrp}$. The predicate and function corresponding to the input gate of failure sub-model are presented in row 9 of Table 2, and the functions corresponding to the output gates of this sub-model are presented in rows 9 and 10 of Table 3.

## 6. Performance Evaluation

In this section, the desired output measures that can be obtained using the proposed SAN model are defined, and then the proposed model is evaluated. The output measures are computed applying the Markov reward approach [41]. In this approach, appropriate reward rates are assigned to each feasible marking of a SAN model, and then the expected accumulated reward is computed in the steady state. To this end, the proposed analytical model is solved with the Möbius tool [33]. In the following formulas, $Mark(P_i)$ returns the number of tokens inside place $P_i$.

The ratio of serving VMs to accepted requests is defined as the ratio of the steady-state number of VMs that are being served to the steady-state number of accepted VM requests in the cloud system. This measure is computed using Equation (1):

$$RSA = \sum_{i=1}^{N} E[Mark(P_{CurrentVM_{S_i}})]/(E[Mark(P_{Ini})]+\mathrm{E}[Mark(P_{Mig})] + \sum_{i=1}^{N} E[Mark(P_{CurrentVM_{S_i}})]) \quad (1)$$

The other measure that is obtained by applying the proposed SAN model is the probability of VMM failure, which can be computed by the following equation:

$$PR_{VMMF} = \sum_{i=1}^{N} E[Mark(P_{Failed\,i})]/N \quad (2)$$

One of the main goals of introducing rejuvenation in cloud systems is to reduce the mean number of failed VMs, which can be computed by the product of the mean number of VMs that are being

served on each VMM and the probability of VMM failure in the steady state. The mean number of failed VMs can be calculated as follows:

$$NFV = (\sum_{i=1}^{N} E\left[Mark\left(P_{CurrentVM_{S_i}}\right)\right]/N) * (\sum_{i=1}^{N} E[Mark(P_{Failedi})]/N) \quad (3)$$

The *total power consumption* of cloud servers is another measure which can be computed by the proposed model and is the mean value of power consumed by PMs, VMMs, and VMs of the data center. It can be computed by Equation (4).

$$TPC = \rho_b * (N - E[Mark(P_{Idle})]) + \rho_v * \sum_{i=1}^{N} E[Mark(P_{CurrentVM_{S_i}})] \quad (4)$$

where $\rho_b$ is the average power consumed by a switched-on PM running a VMM, $\rho_v$ is the average power consumed by a VM, and $N$ denotes the number of PMs in the system.

In order to evaluate the proposed model, we assumed a system with 3 PMs and the maximum of 3 VMs per each VMM. In this setting, the input parameter $T_1$ is 8, which means a VMM of age 8 could be rejuvenated if other VMMs have enough capacity to accept its VMs. The parameter $T_2$ is set to 10, which means a VMM of age 10 has to be rejuvenated instantaneously. In other words, regardless of the capacity of other VMMs, the active VMs of a VMM aged 10 should be placed on the migration queue, and the VMM should be rejuvenated.

The other values set for the input parameters of the proposed model are presented in Table 4. Most of the values used herein as input parameters of the proposed SAN model are in the range of the values considered in other related work [10,11,15,22,29,36,37,39,42]. In order to compare different situations, the proposed model was evaluated in three modes: (1) pure power saving, (2) pure rejuvenation, and (3) with simultaneous application of rejuvenation and power saving, called power-aware rejuvenation. Modes 1 and 2 are the baselines for our main proposed scheme, which is mode *3*. In mode 3, all five sub-models of Figure 2 are used. In mode 1, the rejuvenation sub-model shown in Figure 2d is omitted. In mode 2, the power saving sub-model represented in Figure 2c is removed, and the model is adjusted so that no token would be placed in place $P_{Idle}$. The results obtained from solving the proposed model in the three modes are presented in Figure 3.

**Table 4.** Configuration of the cloud system under study.

| Parameter | Description | Value/Range | Unit |
|---|---|---|---|
| $\lambda_{hf}$ | VMM failure rate | 0.00139 | VMM/h |
| $\lambda_{hrj}$ | VMM rejuvenation rate | 30 | VMM/h |
| $\lambda_{hrp}$ | VMM repair rate | 2 | VMM/h |
| $\lambda_{pw}$ | PM wakeup rate | 1440 | PM/h |
| $\lambda_{va}$ | VM requests arrival rate | [5 . . . 20] | VM/h |
| $\lambda_{vt}$ | VM switch off rate | 0.5 | VM/h |
| $\lambda_{vmg}$ | VM migration rate | 1200 | VM/h |
| $\lambda_{vi}$ | VM initiation rate | 120 | VM/h |
| $p_{vmf}$ | Probability of VM migration failure | 0.01 | - |
| $N$ | Number of PMs | 3 | - |
| $M$ | Number of VMs per PM | 3 | - |
| $Q_c$ | Size of input queue | 5 | - |
| $Q_m$ | Size of migration queue | 6 | - |
| $\rho_b$ | Power consumption of a PM running a VMM | 100 | W |
| $\rho_v$ | Power consumption of a VM | 10 | W |

Figure 3a shows the average probability of VMM failure in the proposed model in the three evaluation modes. One of the main causes of VMM failure is the increasing number of served VMs and the errors introduced due to VMM aging. With the pure rejuvenation scheme, aged VMMs are

identified and rejuvenated before age-induced failures occur. Thus, the forced rejuvenation of mode 2 decreases the probability of VMM failure compared to mode 1. The power saving scheme switches off idle VMMs, which in turn restarts the age of those VMMs, and result in even lower probability of VMM failure when combined with pure rejuvenation in mode 3.

As shown in Figure 3b, simultaneous modeling of power saving and rejuvenation schemes in mode 3 improves the availability-related measure RSA of the cloud system. Such implementation of power-aware rejuvenation involves the migration of VMs between VMMs, so the number of tokens in place $P_{Mig}$ increases and the denominator of the fraction in Equation (1) increases. However, as shown in Figure 3a, pure rejuvenation decreases the average probability of VMM failure, and thereby increases the mean number of active VMMs, increasing the average number of VMs that can be served at a given time. Since $P_{CurrentVMs_i}$ increases more than $P_{Mig}$, power-aware rejuvenation causes a significant improvement in the ratio of serving VMs to accepted requests.

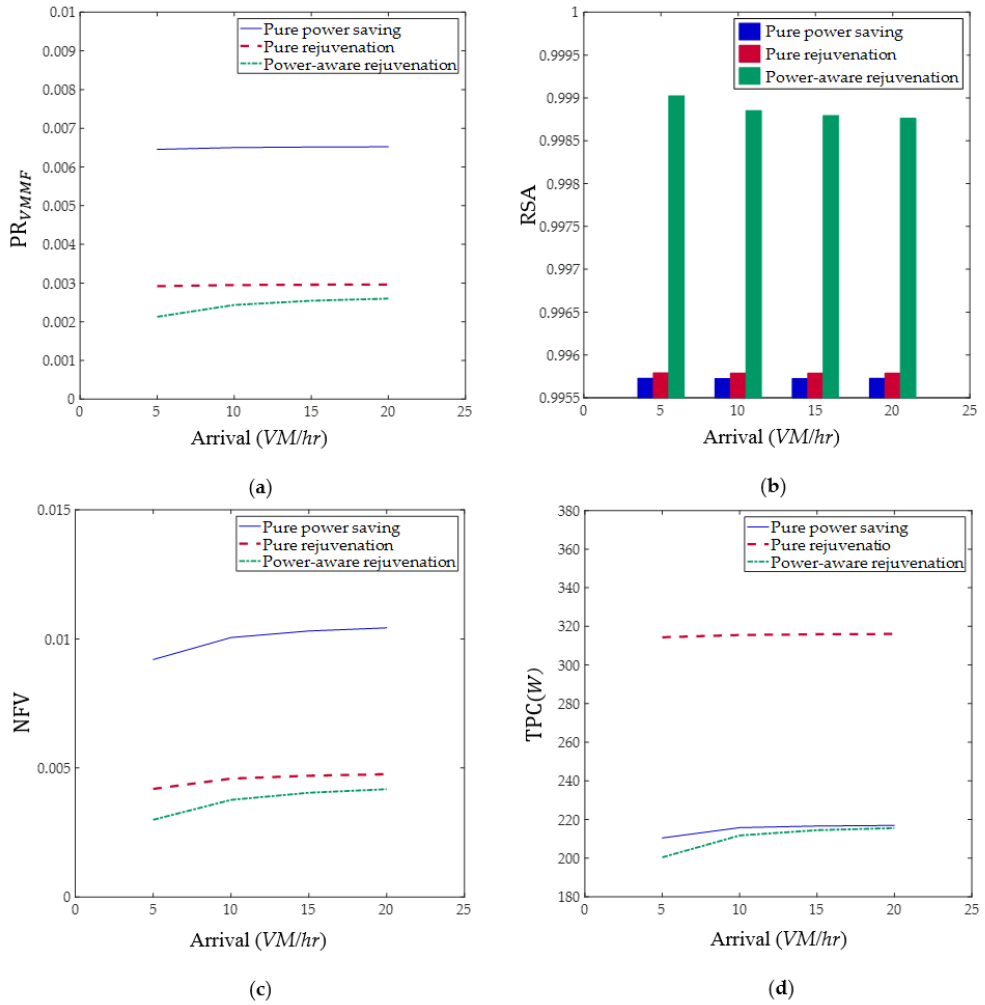


**Figure 3.** Results obtained by solving the proposed SAN model representing (**a**) probability of VMM failure, (**b**) ratio of serving VMs to accepted VMs, (**c**) mean number of failed VMs and (**d**) total power consumption.

Figure 3c represents the effect of applying rejuvenation and power-awareness schemes to cloud systems on the mean number of failed VMs. The greater the average probability of VMM failure, the greater the number of failed VMs. Consequently, for the same VMM failure probability results shown in Figure 3a, mode 3 performed the best and mode 1 the worst.

In Figure 3d, the impact of software rejuvenation and power-awareness on power consumption of cloud system is considered. As indicated in Equation (4), the power consumption is directly related to the number of switched-on PMs and active VMs in a data center. In the pure rejuvenation mode, mode 2, all PMs are always on and therefore the power consumption in this mode is the greatest compared to the power consumed in modes 1 and 3, which are power-aware. In mode 3, newly rejuvenated VMMs are free of running VMs. Thus, they are more likely to be switched off by the power saving mode. Therefore, the total power consumption of mode 3 is less than that of mode 1. Figure 3d acknowledges the fact that by applying rejuvenation and power saving simultaneously, the number of active PMs decreases compared to the situation where rejuvenation is not applied.

## 7. Conclusions and Future Work

In this paper, a power-aware VMM rejuvenation scheme was presented for IaaS cloud systems based on a SAN model proposed for IaaS clouds. Several real aspects of cloud systems that were not considered in previous related work are encompassed in the proposed model, including failure/repair behavior of VMMs, VM heterogeneity, VM migration, VM multiplexing, and different PM power consumption states. The proposed scheme uses two different thresholds for optional and mandatory rejuvenation of aged VMMs, in which the age of a VMM is computed as the cumulative workload that is accepted and served by that VMM after its last reboot. Möbius software was used to solve the proposed SAN. Then, the performance, availability, and power consumption of the presented scheme were compared with two different baselines: a pure power saving scheme and a pure rejuvenation scheme. The results showed that the proposed scheme improves all output measures of interest including the ratio of serving VMs to accepted VM requests, the probability of VMM failure, the mean number of failed VMs, and the total power consumption of cloud servers.

Though the proposed scheme is more scalable than the models and schemes presented in the literature, scalability is still an issue that should be investigated in the future. This limitation can be addressed by applying approximation techniques to presented analytical model, such as folding and fixed-point methods. We plan to complete real-world experiments to validate the analytical results and to address the scalability limitation. Another interesting extension to the model proposed in this work could be to support heterogeneous PMs in addition to heterogeneous VMs. Colored extensions of PNs and SANs could be used for this extension. Considering cloud data center topologies and network architectures and considering availability and power consumption of network equipment are other interesting extensions of current work. Investigating the overhead of VM migration on the performance and availability of the cloud system under study is another possible future line of this research work.

**Author Contributions:** All the authors contributed equally to the work. All authors read and approved the final manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** The authors thank the support of their respective universities.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.; Katz, R. A View of cloud computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]
2. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2011.
3. Han, Y.; Chan, J.; Alpcan, T.; Leckie, C. Using virtual machine allocation policies to defend against co-resident attacks in cloud computing. *IEEE Trans. Dependable Secur.* **2017**, *14*, 95–108. [CrossRef]
4. Zhu, W.; Zhuang, Y.; Zhang, L. A three-dimensional virtual resource scheduling method for energy saving in cloud computing. *Future Gener. Comput. Syst.* **2017**, *69*, 66–74. [CrossRef]

5. Abbasi, A.A.; Jin, H. v-Mapper: An Application-Aware Resource Consolidation Scheme for Cloud Data Centers. *Future Internet* **2018**, *10*, 90. [CrossRef]
6. Ever, E. Performability analysis of cloud computing centers with large numbers of servers. *J. Supercomput.* **2016**, *73*, 2130–2156. [CrossRef]
7. Hasan, M.; Goraya, M.S. Fault tolerance in cloud computing environment: A systematic survey. *Comput. Ind.* **2018**, *99*, 156–172. [CrossRef]
8. Nguyen, T.A.; Kim, D.S.; Park, J.S. A comprehensive availability modeling and analysis of a virtualized servers system using stochastic reward nets. *Sci. World J.* **2014**, *2014*, 165316. [CrossRef] [PubMed]
9. Colman-Meixner, C.; Develder, C.; Tornatore, M.; Mukherjee, B. A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2244–2281. [CrossRef]
10. Melo, M.; Araujo, J.; Matos, R.; Menezes, J.; Maciel, P. Comparative analysis of migration based rejuvenation schedules on cloud availability. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), Manchester, UK, 13–16 October 2013.
11. Machida, F.; Seong Kim, D.; Trividi, K.S. Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration. *Perform. Eval.* **2013**, *70*, 212–230. [CrossRef]
12. Xia, Y.; Han, Y.; Zhou, M.; Li, J. A stochastic model for performance and energy consumption analysis of rejuvenation and migration-enabled cloud. In Proceedings of the International Conference on Advanced Mechatronic Systems, Kumamoto, Japan, 10–12 August 2014.
13. Zheng, J.; Okamura, H.; Li, L.; Dohi, T. A Comprehensive Evaluation of Software Rejuvenation Policies for Transaction Systems with Markovian Arrivals. *IEEE Trans. Reliab.* **2017**, *66*, 1157–1177. [CrossRef]
14. Sharma, S.; Singh, S.; Singh, A.; Kaur, R. Virtualization in Cloud Computing. *Int. J. Sci. Res. Sci. Eng. Technol.* **2016**, *2*, 181–186.
15. Bruneo, D.; Pulionio, A. Workload based software rejuvenation in cloud systems. *IEEE Trans. Comput.* **2013**, *62*, 1072–1085. [CrossRef]
16. Liu, J.; Zhou, J.; Buyya, R. Software rejuvenation based fault tolerance scheme for cloud application. In Proceedings of the IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015.
17. Valentim, N.A.; Macedo, A.; Matias, R. A systematic mapping review of the first 20 years of software aging and rejuvenation research. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Ottawa, ON, Canada, 23–27 October 2016; pp. 57–63.
18. Alshathri, S.; Ghita, B.; Clarke, N. Sharing with Live Migration Energy Optimization Scheduler for Cloud Computing Data Centers. *Future Internet* **2018**, *10*, 86. [CrossRef]
19. Bawden, T. Global Warming: Data Centres to Consume Three Times as Much Energy in Next Decade, Experts Warn, Independent. Available online: http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html (accessed on 10 July 2018).
20. Uchechukwu, A.; Li, K.; Shen, Y. Energy Consumption in Cloud Computing Data Centers. *Int. J. Cloud Comput. Serv. Sci.* **2014**, *3*, 31–48.
21. Cotroneo, D.; Natella, R.; Pietrantuono, R.; Russo, S. A survey of software aging and rejuvenation studies. *ACM J. Emerg. Technol. Comput.* **2014**, *10*, 8. [CrossRef]
22. Roohitavaf, M.; Entezari-maleki, R.; Movaghar, A. Availability modeling and evaluation of cloud virtual data centers. In Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS), Seoul, Korea, 15–18 December 2013.
23. SAN Atomic Formalism, Möbius Wiki. Available online: https://www.mobius.illinois.edu/wiki/index.php/SAN_Atomic_Formalism (accessed on 10 July 2018).
24. Sanders, W.H. *Mobius Manual*; University of Illinois: Chicago, IL, USA, 2012.
25. Sanders, W.H.; Meyer, J.F. Stochastic activity networks: Formal definitions and concepts. In *Lectures on Formal Methods and Performance Analysis*; Brinksma, E., Hermanns, H., Katoen, J.-P., Eds.; Springer: Berlin, Germany, 2001; pp. 315–343.
26. Movaghar, A. Stochastic activity networks: A new definition and some properties. *Sci. Iran.* **2001**, *8*, 303–311.
27. Movaghar, A.; Meyer, J.F. Performability modeling with stochastic activity networks. In Proceedings of the 1984 Real-Time Systems Symposium, Austin, TX, USA, 6–8 August 1984.

28. Meyer, J.F.; Movaghar, A.; Sanders, W.H. Stochastic activity networks: Structure, behavior, and application. In Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, 1–3 July 1985.
29. Melo, M.; Maciel, P.; Araujo, J.; Matos, R.; Araujo, C. Availability Study on Cloud Computing Environments: Live Migration as a Rejuvenation Mechanism. In Proceedings of the 43rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, 24–27 June 2013.
30. Kourai, K.; Ooba, H. Zero-copy migration for lightweight software rejuvenation of virtualized systems. In Proceedings of the ACM 6th Asia-Pacific Workshop on Systems, Tokyo, Japan, 27–28 July 2015.
31. Sudhakar, C.; Shah, I.; Ramesh, T. Software rejuvenation in cloud systems using neural networks. In Proceedings of the IEEE International Conference on Parallel, Distributed and Grid Computing (PDGC), Solan, India, 11–13 December 2014; pp. 230–233.
32. Araujo, J.; Matos, R.; Alves, V.; Maciel, P.; Souza, F.; Trivedi, K.S. Software aging in the eucalyptus cloud computing infrastructure: Characterization and rejuvenation. *ACM J. Emerg. Technol. Commun.* **2014**, *10*, 11. [CrossRef]
33. Daly, D.; Deavours, D.D.; Doyle, J.M.; Webster, P.G.; Sanders, W.H. Mobius: An extensible tool for performance and dependability modeling. In Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, Schaumburg, IL, USA, 25–31 March 2000; pp. 332–336.
34. Sanders, W.H.; Oballl, W.D.; Qureshi, M.A.; Widjanarko, F.K. The UltraSAN modeling environment. *Perform. Eval.* **1995**, *24*, 89–115.
35. Sanders, W.H.; Meyer, J.F. METASAN: A performability evaluation tool based on stochastic activity networks. In Proceedings of the 1986 ACM Fall Joint Computer Conference, Dallas, TX, USA, 2–6 November 1986.
36. Ataie, E.; Entezari-Maleki, R.; Rashidi, L.; Trivedi, K.; Ardagna, D.; Movaghar, A. Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds. *IEEE Trans. Cloud Comput.* **2017**, in press. [CrossRef]
37. Bruneo, D.; Lhoas, A.; Longo, F.; Puliafito, A. Modeling and evaluation of energy policies in green clouds. *IEEE Trans. Parall. Distrib.* **2015**, *26*, 3052–3065. [CrossRef]
38. Entezari-Maleki, R.; Sousa, L.; Movaghar, A. Performance and power modeling and evaluation of virtualized servers in IaaS clouds. *Inf. Sci.* **2017**, *394–395*, 106–122. [CrossRef]
39. Ataie, E.; Entezari-Maleki, R.; Etesami, E.; Egger, B.; Ardagna, D.; Movaghar, A. Power-aware Performance Analysis of Self-adaptive Resource Management in IaaS Clouds. *Future Gener. Comput. Syst.* **2018**, *86*, 134–144. [CrossRef]
40. Hlaing, P.P.; Thein, T. Availability Enhancement for Cloud Services by Migration based Rejuvenation: Analytical Modeling. In Proceedings of the 3rd International Conference on Computational Techniques and Artificial Intelligence, Singapore, 11–12 February 2014.
41. Ciardo, G.; Blakemore, A.; Chimento, P.F.; Muppala, J.K.; Trivedi, K.S. Automated generation and analysis of markov reward models using stochastic reward nets. In *Linear Algebra, Markov Chains, and Queueing Models*; The IMA Volumes in Mathematics and Its Application; Meyer, C.D., Plemmons, R.J., Eds.; Springer: New York, NY, USA, 1993; pp. 145–191.
42. Ataie, E.; Gianniti, E.; Ardagna, D.; Movaghar, A. A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Cloud Environment. In Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 24–27 September 2016.