Workload-Based Software Rejuvenation in Cloud Systems

Dario Bruneo, *Member*, *IEEE*, Salvatore Distefano, Francesco Longo, Antonio Puliafito, *Member*, *IEEE*, and Marco Scarpa

Abstract—Cloud computing is a promising paradigm able to rationalize the use of hardware resources by means of virtualization. Virtualization allows to instantiate one or more virtual machines (VMs) on top of a single physical machine managed by a virtual machine monitor (VMM). Similarly to any other software, a VMM experiences aging and failures. Software rejuvenation is a proactive fault management technique that involves terminating an application, cleaning up the system internal state, and restarting it to prevent the occurrence of future failures. In this work, we propose a technique to model and evaluate the VMM aging process and to investigate the optimal rejuvenation policy that maximizes the VMM availability under variable workload conditions. Starting from dynamic reliability theory and adopting symbolic algebraic techniques, we investigate and compare existing time-based VMM rejuvenation policies. We also propose a time-based policy that adapts the rejuvenation timer to the VMM workload condition improving the system availability. The effectiveness of the proposed modeling technique is demonstrated through a numerical example based on a case study taken from the literature.

Index Terms—Time-based rejuvenation, cloud computing, dynamic availability, phase type distributions, Kronecker algebra

1 Introduction

Cloud computing is the fastest growing field in ICT. It meets computing demand steadily on the rise with a supply of services modeled after public utilities, such as energy and gas traditionally available at high fees, at much lower costs, and according to a pay per use pricing schema [1].

The elastic behavior of cloud systems allows an increase of resource availability, thus fitting the requirements of users and their applications, and reducing costs. Cloud computing offers to businesses a heavy outsourcing model for computational resources, where service availability, security, and quality guarantees all become essential features [2], [3]. Understanding these problems is one of the factors that are going to steer the market in the coming years.

In particular, service availability is of primary importance being one of the main requirements and, thus, affecting user satisfaction. Research efforts have been devoted in recent years to find the optimal infrastructure size and configuration allowing to guarantee the desired availability level. Hardware redundancy, fault tolerance, and recovery mechanisms are the techniques usually used in the cloud context [4]. However, also the degradation over time of the application performance (usually referred to as *software aging*) may lead to low availability due to premature program termination caused by (*aging related*) software bugs [5].

- *D. Bruneo, F. Longo, A. Puliafito, and M. Scarpa are with the Dipartimento di Ingegneria Civile, Informatica, Edile, Ambientale e Matematica applicata, Università di Messina, Contrada di Dio, S.Agata, 98166 Messina, Italy. E-mail: {dbruneo, flongo, apuliafito, mscarpa}@unime.it.*
- *S. Distefano is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza L. Da Vinci 32, 20133 Milano, Italy. E-mail: salvatore.distefano@polimi.it.*

Manuscript received 3 May 2011; revised 14 Nov. 2012; accepted 20 Jan. 2013; published online 14 Feb. 2013.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2011-05-0298.
Digital Object Identifier no. 10.1109/TC.2013.30.

Software rejuvenation [6] is a cost effective technique for dealing with software faults. It consists of a proactive fault management technique aimed at cleaning up the system internal state to prevent the occurrence of more severe faults, such as crashes or software malfunctioning. Software rejuvenation is a mature area and several works can be found in the literature. Most of them focus on modeling aspects [7], [8] trying to provide a model of the system and to investigate the optimal time to rejuvenate. Another approach is based on real measurements [9], [10], where collected data are analyzed to estimate and characterize the time to failure of a software system.

Cloud computing is defined as an infrastructure of shared hardware and software resources, deeply rooted on the premise of *virtualization* [11]. Such technique allows to instantiate multiple *virtual machines* (VMs) on top of a physical machine managed by the *virtual machine monitor* (VMM). Since the effects of software degradation mainly impact long-term running applications and services, attention has been put in the literature to the possibility of applying rejuvenation to mitigate the effect of software aging in the VMM. A first attempt to investigate VMM failures can be found in [12]. In this context, understanding the way the VMM ages and how this can influence the service availability is extremely important. The fact that the workload impacts on the system failure distribution is a well-known result from the literature [13], [14]. With respect to the VMM, the stress condition and consequently the aging increase with the number of VMs being managed. Such aspect becomes very relevant in cloud systems, where the workload is highly variable.

Most of the techniques for modeling and assessing software aging phenomena and rejuvenation policies implement analytical approaches, specifying the problem in terms of stochastic processes. The characterization of such processes is driven by the need to incorporate the software

age into the model. This prevents the use of Markov models and similar techniques that assume a "memory-less" behavior. Nevertheless, Markov models are used to model software aging whenever software age is approximated by discretizing aging in phases or epochs. Regardless the accuracy that can be reached, Markovian models do not allow to represent some aspects and behaviors related to specific software aging and rejuvenation processes.

A contribution of this work is a non-Markovian analytical technique that allows to investigate time-based rejuvenation strategies. Such technique is able to manage the intrinsic non-Markovian nature of the software aging process as well as the influence of the workload on the software behavior. More specifically, we assume the VMM aging phenomenon is characterized by an increasing failure rate (IFR) distribution and depends on the number of VMs it is managing. Then, we characterize the VMM time to failure through continuous phase type (CPH) distributions. The system availability is, thus, modeled by an expanded process that allows to keep memory of the age reached by the VMM when the number of hosted VMs changes according to the *conservation of reliability principle* [15]. The expanded process is symbolically represented in terms of Kronecker algebra. This allows to formally represent the workload-dependent system behavior in a way that is intuitive and easily implementable in a software tool. We also address the state-space explosion problem affecting state-space models, especially when phase type (PH) expansion techniques are used.

The main goal of time-based rejuvenation models is to find an optimal rejuvenation timer that allows to minimize some objective functions. Usually, the timer is set at system start-up and it does not change with respect to the system dynamics (e.g., system workload variations). We refer to such kind of approach as *fixed timer* policy. Another contribution of the present work is the specification of a time-based policy adapting the rejuvenation timer to the VMM conditions, taking into account its workload and age (*variable timer* policy). The effectiveness of the proposed modeling technique is demonstrated through a numerical example based on a case study taken from the literature. It shows how the proposed variable timer policy outperforms the fixed one in terms of improved system availability also varying the way failure rates are affected by the workload.

The remainder of the paper is organized as follows: We first describe the cloud system under exam, as reported in Section 2. Then, in Section 3, we deal with the rejuvenation approach, characterizing related events and specifying the policies we propose. We thereafter explain how to represent the system through state-space models in Section 4 and how to evaluate such a model through PH and Kronecker algebra in Section 5. Section 6 operatively describes the variable timer policy and the modeling technique through an example taken from the literature, aiming at demonstrating the effectiveness of the proposed approach. Section 7 discusses the related work highlighting the contribution of the proposed technique to the state of the art. Concluding remarks and future work are dealt with in Section 8.

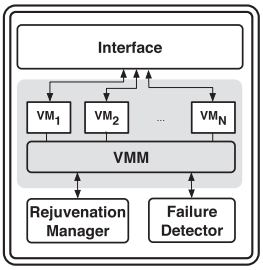

Fig. 1. A rejuvenation-enabled cloud node.

2 System Overview

A cloud infrastructure is composed of network-connected *cloud nodes* coordinated by a *cloud management system*. A cloud node can be either a workstation or a multicore system, a cluster, or also a data center, implementing a stand-alone administrative domain with its management system. To improve node availability, software rejuvenation policies can be adopted.

Fig. 1 shows a rejuvenation-enabled cloud node configuration. A cloud node offers a virtualized environment through a VMM on top of which one or more VMs can be instantiated. The *Interface* is in charge of scheduling the user requests (in terms of VMs) by loading prebuilt VM images from the repository and registering them to the VMM. The VMs can migrate between nodes to react to specific conditions, e.g., workload bursts, energy saving, and service level agreement (SLA) enforcement policies. The Interface, therefore, manages the VM migration process by monitoring the system workload, establishing connections among nodes, and transferring and/or receiving the VMs. Once instantiated, a VM starts its execution providing the user with an isolated execution environment. As soon as the service is completed, the VM instance is stopped, thus releasing the corresponding resources.

The VMM is a long-term running software module subjected to aging. For this reason, the VMM can fail giving rise to critical effects on the running VMs, such as service interruption and data loss. The VMM failure is detected by the *Failure Detector* periodically inspecting the memory usage and/or the response time. When a failure is detected, the status of all the running VMs is saved (if possible) and the VMM is repaired. As soon as the repair has been performed, the VMs can be reactivated. We assume that once repaired, the VMM age is reset.

To improve the node availability, a software rejuvenation task can be triggered on the VMM by the *Rejuvenation Manager*. In this case, the running VMs are suspended in a safe way and the VMM is rebooted, thus resetting its aging state. It has been shown that a proactive restart reduces overall downtime [16]. In fact, the repair of a system could require some manual operations with a higher complexity than a simple restart. For example, log file inspection is often required to restore the system status when it is compromised due to runtime errors. Moreover, while the failure event is unpredictable, rejuvenation is a planned task and some countermeasures to reduce the quality of service (QoS) degradation (e.g., VM live migration, data saving, checkpointing) can be taken in advance. The comparison in terms of timing between reparation and rejuvenation is

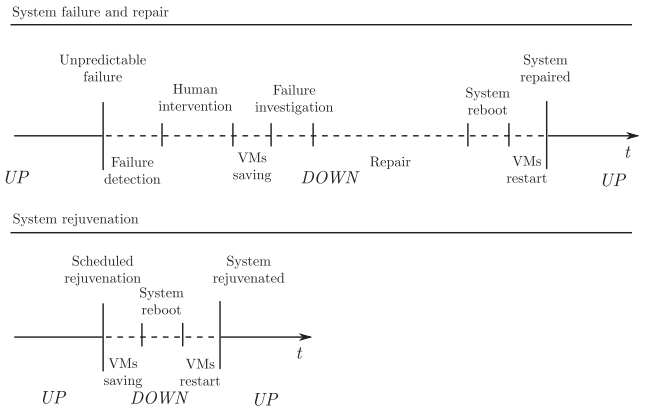

Fig. 2. Timing comparison between system repair and rejuvenation.

shown in Fig. 2, where the up and down operational states of the system are highlighted.

In the reminder of the paper, we focus on the aspects related to the aging and failure of the VMM and to its rejuvenation, thus neglecting the failure of other components. We also assume that, in the case a failure has occurred but it has not yet been detected, if a rejuvenation task is triggered, the Rejuvenation Manager is not able to save the VMs state and, as a consequence, the rejuvenation is aborted.

According to the above assumptions, the VMM can be represented, from an operating point of view, by a finite state machine (FSM) characterized by four states, as shown in Fig. 3. Once started, the VMM is in the *working* state s_{wr}, where the VMM age increases and software bugs accumulate. When a *failure* event e_{fl} occurs, the VMM enters the *nondetected failure* state s_{nd}. In this state, the VMM is not able to accomplish its work but the failure has yet to be detected. After the failure is detected by the Failure Detection module (*detection* event e_{dt}), the VMM enters the *detected failure* state s_{df} and, as soon as the repair is performed by a repairman (*repair* event e_{rp}), the VMM is rebooted thus restarting from scratch in state s_{wr}. When a rejuvenation has to be performed, the Rejuvenation Manager activates the *trigger* event e_{tr} and the VMM enters the *rejuvenation* state s_{rj}. Then, it reboots the VMM through the *resume* event e_{rs} reinitializing its age.

To analytically evaluate the behavior of the above described system, the FSM of Fig. 3 has to be translated into a stochastic process by adequately specifying the system events. This characterization can be performed associating to each event the cumulative distribution function (CDF) of its firing time. Moreover, it is necessary to formally describe the workload represented by the number of VMs running on the VMM. Indicating with $\#VM(t)$, the VMs running at time t and with N the maximum number of VMs that can concurrently be instantiated, the discrete-valued stochastic process $\{\#VM(t)|t \in \mathbb{R}^+\}$ depends on both the VM arrival process, represented by user requests, and the VM departure process, taking into account the service time of each VM and the migration tasks that the node Interface can schedule.

2.1 VMM Aging and Failure

With respect to system failures, we make the following assumptions. The cloud node workload is highly variable due to request bursts and migration tasks. To satisfy the

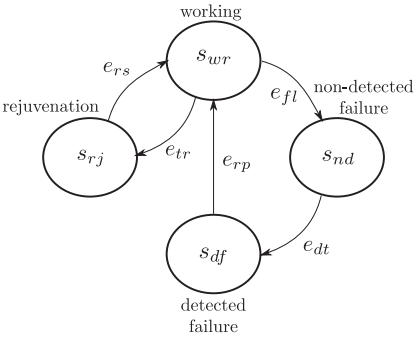

Fig. 3. Operating states of a VMM.

user requests, also in terms of QoS, the VMM has to implement different strategies to quickly instantiate, suspend, and transfer VMs. Such strategies have a critical impact on the VMM behavior, being strictly related to memory management tasks (e.g., memory allocation/deallocation). Thus, they interfere with the VMM software aging process, and therefore, we assume the VMM failure time distribution to be dependent on the VMM workload. The VMM aging can also be affected by the VM workload (e.g., due to interrupts and shared resource handling). However, in this work, we assume the VMs to run similar workloads, thus implying that the VMM aging is not dependent on the VM behavior.

More formally, being t_{fl} the random variable representing the time instant a VMM failure occurs (event e_{fl} of Fig. 3), its CDF depends on the workload stochastic process $\#VM(t)$. To model such dependency, we identify a set of failure time CDFs $\mathcal{F}_{fl} = \{F^i_{fl}(t) : 0 \leq i \leq N\}$, where

$$F^i_{fl}(t) = Pr\{t_{fl} \leq t \mid \#VM(t) = i\} \tag{1}$$

is the CDF associated with a specific workload condition *in isolation*, i.e., assuming a fixed number of running VMs.

On the contrary, the time needed by the Failure Detector for detecting the failure does not depend on the number of VMs managed by the VMM at the time the failure occurs and then it can be modeled by a single CDF $F_{dt}(t)$ associated with the detection event e_{dt}.

Finally, let t_{rp} be the random variable associated with the time to repair (event e_{rp}). We assume that a checkpoint service periodically saves the VMM status by making a dump of its memory on the disk. Since the status of all the VMs being executed on top of the VMM at the time it crashed has to be recovered from the checkpoint database, event e_{rp} can be characterized by the workload-dependent CDF set $\mathcal{F}_{rp} = \{F^i_{rp}(t) : 0 \leq i \leq N\}$ with

$$F^i_{rp}(t) = Pr\{t_{rp} \leq t \mid \#VM(t) = i\}. \tag{2}$$

3 VMM Rejuvenation

Time-based rejuvenation consists in setting a timer that schedules the time instant when the rejuvenation task has to be performed. Event e_{tr} is then characterized by the nature of such a timer and its CDF strictly depends on the adopted rejuvenation policy, as discussed in the following. On the contrary, the resume event e_{rs} represents the time needed to carry out the VMM rejuvenation process. In this case,

because the status of the running VMs has to be saved and recovered, we associate with such an event the workload-dependent CDF set $\mathcal{F}_{rs} = \{F^i_{rs}(t) : 0 \leq i \leq N\}$ with

$$F^i_{rs}(t) = Pr\{t_{rs} \leq t \mid \#VM(t) = i\}, \quad (3)$$

where t_{rs} indicates the random variable associated with the resume event.

3.1 Fixed Timer Policy

The usually adopted software rejuvenation policy relies on setting a *time interval* δ associated with a trigger event, the e_{tr} event of Fig. 3. According to this policy, when the VMM starts, the *timer firing time* T is set to δ and a *local clock*, initially set to 0, starts to increment. When the local clock reaches the δ value, the rejuvenation is performed. Neither the time interval δ nor the local clock is varied between two consecutive rejuvenation events.

In terms of CDF, event e_{tr} is then characterized by the following deterministic distribution:

$$F_{tr}(t) = \begin{cases} 0 & if \;\; t < \delta \\ 1 & if \;\; t \geq \delta. \end{cases} \quad (4)$$

3.2 Variable Timer Policy

In this paper, we present a time-based rejuvenation policy performing runtime adaptation of the rejuvenation timer according to the VMM workload condition. Thus, we propose to update the timer firing time with respect to the system workload to take into account the aging conditions. In this way, the rejuvenation task can be anticipated if the system has been overloaded or deferred if the system has not been intensively used.

Starting from a system reboot, the VMM evolution can be characterized by a sequence of different workload conditions, i.e., the number of running VMs, identified as $\{L^k, k = 1, 2, \ldots\}$, where k is the sequence index and $0 \leq L^k \leq N$. Following the sequence of workload changes, the rejuvenation timer firing time, initially set to a value T^1, is modified, thus, resulting in a sequence of different values of the firing time $\{T^k, k = 1, 2, \ldots\}$ that represents the runtime adaptation of the timer to the system workload. To take into account the different aging phenomena corresponding to the variable workload conditions, the timer is updated according to a set of workload-dependent time intervals $\{\delta_i, 0 \leq i \leq N\}$ that regulates the way the firing time is adjusted when a workload variation occurs. Such time intervals can be considered as the firing times at which the timer should be set if the workload never changes during a rejuvenation epoch, thus identifying a CDF set $\mathcal{F}_{tr} = \{F^i_{tr}(t) : 0 \leq i \leq N\}$, where

$$F^i_{tr}(t) = Pr\{t_{tr} \leq t \mid \#VM(t) = i\} == \begin{cases} 0 & if \;\; t < \delta_i \\ 1 & if \;\; t \geq \delta_i. \end{cases} \quad (5)$$

If at a given time instant, corresponding to a value of the timer local clock equal to t_x, the workload changes reaching condition $L^{k+1} = j$, then the timer firing time T^{k+1} has to be set taking into account not only the new workload condition (represented by the time interval δ_j) but also the degradation that the system has already experienced because the last reboot. An estimation of such a degradation is the fraction of the current timer firing time T^k already elapsed because the last system reboot (t_x/T^k). Then, the time interval δ_j has to be proportionally scaled of the quantity $(1 - t_x/T^k)$ that represents the fraction of the time interval that still need to be waited, under the new workload condition, before the next rejuvenation occurs. More formally, the new timer firing time T^{k+1} has to be set to

$$T^{k+1} = t_x + \left(1 - \frac{t_x}{T^k}\right) \cdot \delta_j. \quad (6)$$

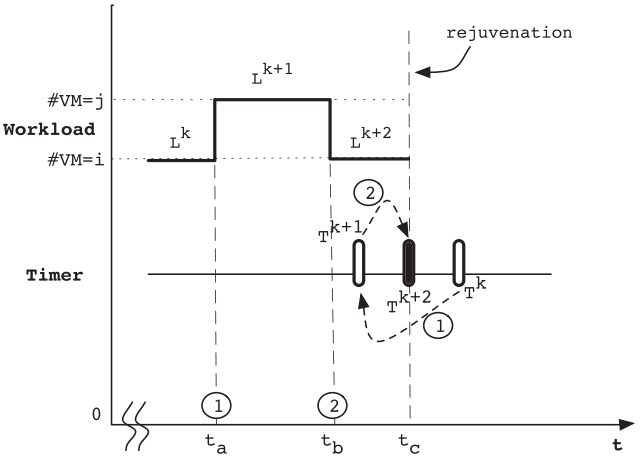

Fig. 4. Graphical representation of the variable timer policy behavior.

Summarizing, assuming we know the set of time intervals $\{\delta_i, 0 \leq i \leq N\}$, the variable timer policy acts as follows:

1. when the VMM is rebooted (in a generic workload condition $L^1 = i$) the timer firing time T^1 is set to δ_i and the local clock is set to 0;
2. each time the workload varies reaching a generic workload condition $L^k = j$ (at a generic time t_x) the timer firing time T^k is set according to (6); and
3. when the local clock reaches the latest timer firing time, the rejuvenation is performed.

Fig. 4 shows an example of how the variable timer policy works. Let us assume that the system is in the workload condition $L^k = i$ with a corresponding timer firing time T^k. At local clock t_a, the workload changes to a new condition $L^{k+1} = j$ with $j > i$. Accordingly, the timer firing time is anticipated (being the new workload condition associated with a higher degradation, i.e., $\delta_i > \delta_j$) to $T^{k+1} = t_a + (1 - t_a/T^k) \cdot \delta_j$. Similarly, at local clock t_b, the workload changes again to the old value i ($L^{k+2} = i$) and the timer firing time is deferred to $T^{k+2} = t_b + (1 - t_b/T^{k+1}) \cdot \delta_i$. Finally, at local clock t_c, the firing time is reached and the rejuvenation is performed. Notice that the value of T^{k+2} differs from that of T^k even if $L^{k+2} = L^k$. This is due to the fact that during the time interval $[t_a, t_b]$ the system experiences a higher degradation. Such result represents the key aspect of the proposed policy, providing the cloud node administrator with a more flexible mechanism to schedule the rejuvenation task.

The variable timer policy behavior can be alternatively described in an equivalent way by reversing the perspective. At a given workload changing point (from $L^k = i$ to $L^{k+1} = j$), instead of keeping the value of the timer local clock and

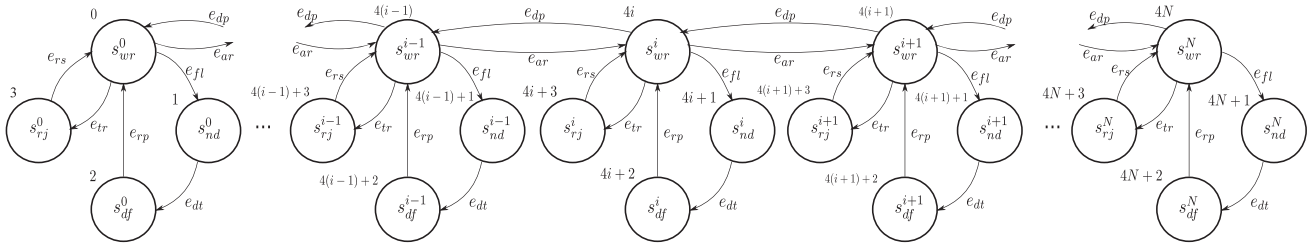

Fig. 5. State-space model of a VMM in case of exponential arrival process.

updating the timer firing time, we can equivalently fix the firing time according to the workload condition, $T^{k+1} = \delta_j$, and keep memory of the elapsed fraction of firing time by updating the timer local clock from t_x to $t_x \cdot \frac{\delta_j}{\delta_i}$.

4 The State-Space Model

Once the FSM events have been stochastically characterized, a complete state-space model able to represent the rejuvenation-enabled cloud node behavior can be derived. In such a model, the workload variations have to be explicitly taken into consideration. With respect to the workload process, we make the following assumptions:

1. The VMM cannot accept a new VM or release an already instantiated one (i.e., it cannot change its workload condition) when it is failed or under rejuvenation.
2. A maximum of N VMs can be concurrently executed on top of the VMM.
3. Only one arrival or departure event can occur within an infinitesimal time interval.
4. All the instantiated VMs have the same impact on the VMM aging. In other words, it is not possible to distinguish among the VMs.
5. VM times to arrival and departure are characterized by memory-less CDFs, i.e., exponential distributions.

According to such assumptions, the workload process can be represented as a birth-death process with $N+1$ states and the complete state-space model of the VMM is obtained by replicating the states of the FSM of Fig. 3, generating the stochastic process shown in Fig. 5.

Such a stochastic process is composed of $N+1$ sets of states each representing the evolution of the VMM life cycle loaded by a certain number of running VMs. Specifically, s^i_{wr}, s^i_{rj}, s^i_{nd}, and s^i_{df} (with $0 \leq i \leq N$) represent the working, rejuvenation, nondetected failure, and detected failure states with i VMs, respectively. Events e_{ar} and e_{dp} represent the arrival and departure of a VM. According to assumption 1, events e_{ar} and e_{dp} can be enabled only in states s^i_{wr}. Assumption 2 forces event e_{ar} to be disabled in state s^N_{wr}. Moreover, event e_{dp} is disabled in state s^0_{wr} given that no VMs are instantiated in that state. While the arrival event is independent from the number of VM currently running, the departure event depends on it. For these reasons, a single CDF $F_{ar}(t)$ is associated with event e_{ar}, while a CDF set $\mathcal{F}_{dp} = \{F^i_{dp}(t) : 0 \leq i \leq N\}$ is associated with event e_{dp} with

$$F^i_{dp}(t) = Pr\{t_{dp} \leq t \mid \#VM(t) = i\}. \quad (7)$$

According to assumption 5, such CDFs are exponential distributions characterized by rates γ_{ar} and ξ^i_{dp}, respectively.

Summarizing, the stochastic process of Fig. 5 is characterized by:

- The CDFs $F_{ar}(t)$ and $F_{dt}(t)$ associated with events e_{ar} and e_{dt}, respectively.
- The workload-dependent CDF sets $\mathcal{F}_{dp}$, $\mathcal{F}_{fl}$, $\mathcal{F}_{rp}$ whose ith components ($F^i_{dp}(t)$, $F^i_{fl}(t)$, $F^i_{rp}(t)$) are associated with events e_{dp}, e_{fl}, e_{rp} when they are enabled in states s^i_{ur} and s^i_{df}.
- The rejuvenation trigger event CDF that, depending on the rejuvenation policy, can be a single CDF $F_{tr}(t)$ associated with events e_{tr} (fixed timer policy) or a CDF set $\mathcal{F}_{tr}$ whose ith component ($F^i_{tr}(t)$) is associated with event e_{tr} when it is enabled in state s^i_{rj} (variable timer policy).

For a complete characterization of the process, it is also necessary to specify the stochastic rules (in terms of memory policies) that govern the workload changing points. As stated by assumption 3, a variation in the workload is only allowed between two neighbor working states, e.g., between states s^i_{wr} and s^{i-1}_{wr} or s^{i+1}_{wr}. During such transitions, four events are involved because they are concurrently enabled: The arrival event e_{ar}, the departure event e_{dp}, the failure event e_{fl}, and the trigger event e_{tr}.

With respect to the first two, no memory conservation policy is required thanks to assumption 5.

With respect to event e_{fl}, we have to consider the nature of the underlying physical phenomenon, i.e., the VMM aging. A reasonable assumption is to consider the reliability level $R(t) = Pr\{t_{fl} > t\}$ as an indicator of the system age. Then, at workload changing points, such a quantity needs to be preserved according to the conservation of reliability principle [15]. To explain how such a principle can be exploited, let us consider a simple example in which, during the VMM lifetime, only a single workload change occurs, e.g., at time instant $\bar{t}$ the VMM workload switches from $\#VM(\bar{t}^-) = i$ to $\#VM(\bar{t}^+) = j$, characterized by a faster aging process ($j > i$). As shown in Fig. 6, the VMM reliability function $R(t)$ follows the reliability function of the VMM in isolation $R^i(t) = 1 - F^i_{fl}(t)$ for $0 \leq t \leq \bar{t}$. At time instant $\bar{t}$, the $R(t)$ should start following the faster aging process represented by $R^j(t) = 1 - F^j_{fl}(t)$, and it is necessary to keep memory of the reached level of reliability $\bar{r} = R^i(\bar{t})$. Therefore, in order $R(\bar{t}^-) = R(\bar{t}^+)$, we have to evaluate the time instant t^* corresponding to age $\bar{r}$ according to $R^j(t)$, $t^* = R^{j^{-1}}(\bar{r})$. Then, the aging process in the new condition follows a translation in time of $R^j(t)$, i.e., $R^j(t+\tau)$, where

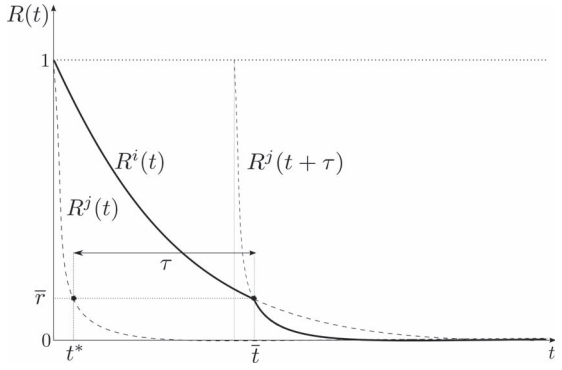

Fig. 6. A graphical representation of the conservation of reliability principle.

$$\tau = t^* - \bar{t} = R^{j^{-1}}(R^i(\bar{t})) - \bar{t}. \tag{8}$$

Similarly, the memory or age related to the trigger event e_{tr} has to be preserved according to the adopted time-based rejuvenation policy, as described in Section 3. In the first case, the memory is just the elapsed time, while in case of variable timer it is the elapsed fraction of the firing time.

The resulting stochastic process is in general non-Markovian and nonhomogeneous in time [4]. More complex and powerful techniques than Markov chains or Markov reward processes are needed to analyze this process, such as Markov additive processes [17], semi-Markov processes, and renewal theory [18]. In the following section, we propose an analytic technique for the evaluation of the model of Fig. 5 able to manage its complexity.

5 The Modeling Technique

The main characteristics of the proposed modeling technique can be summarized as follows:

- The CDFs associated with the VMM events, under different workload conditions, are represented by CPHs, moving the problem toward the solution of an expanded process [19].
- Kronecker algebra is used to implement the conservation of reliability principle and the variable timer policy [20].

5.1 The PH Expanded Process

The use of PH distributions was initiated in 1975 by Neuts [21], [22], who formally defined a PH distribution as the distribution of the time until absorption in a finite state Markov chain with a single absorbing state. *Continuous time Markov chains* (CTMCs) are identified in case of continuous time, and the corresponding PH distributions are referred to as CPHs.

More specifically, let us consider a CTMC χ with ν transient states and a single absorbing state (labeled $\nu+1$), whose infinitesimal generator matrix $\hat{\mathbf{G}}$ of dimensions $(\nu+1)\times(\nu+1)$ is in the form:

$$\hat{\mathbf{G}} = \left[\begin{array}{c|c} \mathbf{G} & \mathbf{U} \\ \hline \mathbf{0} & 0 \end{array}\right]$$

where $\mathbf{G}$ is a $\nu \times \nu$ matrix that describes the transient behavior of the CTMC, and $\mathbf{U}$ is a ν-dimension column vector grouping the transition rates to state $\nu+1$. Moreover, let us assume that the chain is started with an initial probability vector $\boldsymbol{\pi}(0) = [\boldsymbol{\alpha}, \alpha_{\nu+1}]$ such that $\alpha_{\nu+1} = 1 - \sum_{i=1}^{\nu} \alpha_i$. We say that a random variable T is distributed according to the CPH distribution with representation $(\boldsymbol{\alpha}, \mathbf{G})$ and order ν if its CDF $F_T(t)$ is the probability to reach the absorbing state of χ:

$$F_T(t) = 1 - \boldsymbol{\alpha} \cdot e^{\mathbf{G}\cdot t} \cdot \mathbf{1}_\nu, t \geq 0, \tag{9}$$

where $\mathbf{1}_\nu$ is a ν-dimensional column vector of 1s. An example of CTMC representing a CPH distribution is depicted in Fig. 7, with its matrix representation. In the following, we show how CPH distributions can be exploited to model the VMM events and specifically to represent events e_{fl} and e_{tr} at workload variations.

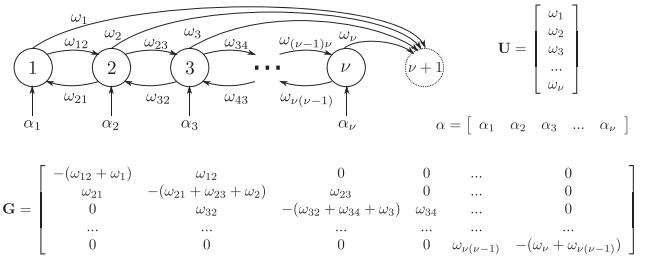

Fig. 7. Example of CPH with its matrix representation.

For the sake of simplicity, let us assume that at time $\bar{t}$ the workload varies from i to j VMs and that the failure time CDF changes from $F^i_{fl}(\cdot)$ to $F^j_{fl}(\cdot)$, accordingly. Due to the reliability conservation principle, the $F^j_{fl}(\cdot)$ has to be shifted of the equivalent time τ, as explained in Section 4. Assuming $(\boldsymbol{\alpha}_i, \mathbf{G}_i)$ and $(\boldsymbol{\alpha}_j, \mathbf{G}_j)$ be the (ν^i and ν^j order) representations of $F^i_{fl}(\cdot)$ and $F^j_{fl}(\cdot)$, at time $\bar{t}$ such a principle implies:

$$\begin{gathered} R^i(\bar{t}) = R^j(t^*) \\ \Downarrow \\ 1 - F^i_{fl}(\bar{t}) = 1 - F^j_{fl}(\bar{t}+\tau) \\ \Downarrow \\ \boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}} \cdot \mathbf{1}_i = \boldsymbol{\alpha}_j \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau)} \cdot \mathbf{1}_j, \end{gathered} \tag{10}$$

where $\mathbf{1}_i$ ($\mathbf{1}_j$) is a column vector with ν^i (ν^j) elements all equal to 1. Equation (10) translates the conservation of reliability principle in terms of CPH. Notice that, for any fixed $\bar{t}$, there exists a unique τ (dependent on $\bar{t}$) for which (10) holds. This is due to the fact that we are taking into account continuous and strictly monotone reliability functions, as also depicted in Fig. 6.

Once τ is computed by using (8), we want to express the equivalent time through a specific transformation matrix to be exploited in the context of CPH domain. Hence, the quantity $\boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}}$ can be interpreted as the vector containing the probabilities to be in the transient phases of the CPH representing $F^i_{fl}(\cdot)$ at time $\bar{t}$. Thus, (10) states that, when the VMM workload changes, the CPH representing $F^j_{fl}(\cdot)$ has to be enabled with an initial probability vector equal to $\boldsymbol{\alpha}_j \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau)}$ to preserve the reliability level in the new state. Fig. 8 shows this fact assuming that at $\bar{t}$ the CPH $(\boldsymbol{\alpha}_i, \mathbf{G}_i)$ representing $F^i_{fl}(\cdot)$ is in the phase q^i and $m^{i\to j}_{q^i,h}(\bar{t})$

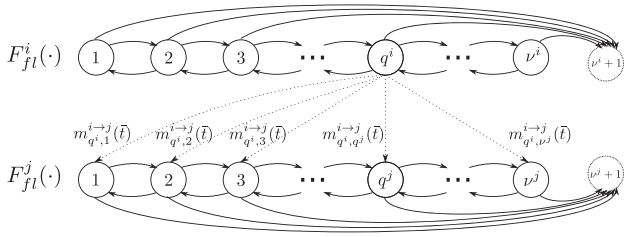

Fig. 8. Conservation of reliability principle at workload changing time.

with $h = 1, \ldots, \nu^j$ are the switching probabilities that satisfy (10). It is important to remark that $m^{i\to j}_{q^i,h}(\bar{t})$ probabilities depend on $\bar{t}$ because they set the appropriate level of reliability at that time instant. Moreover, since at $\bar{t}$ the process needs to jump in at least one of the phases of the CPH $(\boldsymbol{\alpha}_j, \mathbf{G}_j)$, the following relation holds:

$$\sum_{h=1}^{\nu^j} m^{i\to j}_{q^i,h}(\bar{t}) = 1. \tag{11}$$

The $m^{i\to j}_{q^i,h}(\bar{t})$ probabilities to switch from all the phases of $(\boldsymbol{\alpha}_i, \mathbf{G}_i)$ to the phases of $(\boldsymbol{\alpha}_j, \mathbf{G}_j)$ are collected in a $\nu^i \times \nu^j$ stochastic matrix $\mathbf{M}^{i\to j}_{fl}(\bar{t})$ that we call *memory matrix*. A row of $\mathbf{M}^{i\to j}_{fl}(\bar{t})$ stores the vector containing the probabilities to switch from the corresponding phase of the $F^i_{fl}(\cdot)$ CPH to all the phases of the $F^j_{fl}(\cdot)$ CPH keeping memory of the reliability level reached at time $\bar{t}$.

A memory matrix that satisfies (10) can be computed by solving the following equation:

$$\boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}} \cdot \mathbf{M}^{i\to j}_{fl}(\bar{t}) = \boldsymbol{\alpha}_j \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau).} \tag{12}$$

In fact, by right-multiplying both sides of (12) by $\mathbf{1}_j$ and by considering that $\mathbf{M}^{i\to j}_{fl}(\bar{t}) \cdot \mathbf{1}_j = \mathbf{1}_i$ ((11) in matrix form) we have

$$\begin{gathered} \boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}} \cdot \mathbf{M}^{i\to j}_{fl}(\bar{t}) \cdot \mathbf{1}_j = \boldsymbol{\alpha}_j \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau)} \cdot \mathbf{1}_j \\ \Downarrow \\ \boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}} \cdot \mathbf{1}_i = \boldsymbol{\alpha}_j \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau)} \cdot \mathbf{1}_j, \end{gathered} \tag{13}$$

i.e., the condition expressing the conservation of reliability principle of (10).

The computation of memory matrix by (12) is not straightforward in general, especially considering its time dependency, but in some cases additional assumptions may simplify it. Specifically, if the two CPHs are of the same order ν, they are enabled with identical initial probability vectors, and the rates in the two CPHs are proportional, i.e.,

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_j \tag{14}$$

$$\mathbf{G}_j = c \cdot \mathbf{G}_i, \tag{15}$$

the memory matrix $\mathbf{M}^{i\to j}_{fl}(\bar{t})$ can be computed from (12) as:

$$\begin{aligned} \mathbf{M}^{i\to j}_{fl}(\bar{t}) &= e^{-\mathbf{G}_i \cdot \bar{t}} \cdot e^{\mathbf{G}_j \cdot (\bar{t}+\tau)} \\ &= e^{-\mathbf{G}_i \cdot \bar{t}} \cdot e^{c \cdot \mathbf{G}_i \cdot (\bar{t}+\tau)} \\ &= e^{\mathbf{G}_i (c \cdot (\bar{t}+\tau) - \bar{t})}. \end{aligned} \tag{16}$$

Equation (10) written under the assumptions (14) and (15) becomes

$$\boldsymbol{\alpha}_i \cdot e^{\mathbf{G}_i \cdot \bar{t}} \cdot \mathbf{1}_i = \boldsymbol{\alpha}_i \cdot e^{c \cdot \mathbf{G}_i \cdot (\bar{t}+\tau)} \cdot \mathbf{1}_i, \tag{17}$$

which holds for each $\bar{t}$. From (17), we have $\bar{t} = c \cdot (\bar{t} + \tau)$, thus $c \cdot (\bar{t} + \tau) - \bar{t} = 0$ and (16) becomes

$$\mathbf{M}^{i\to j}_{fl}(\bar{t}) = e^{\mathbf{G}_i \cdot 0} = \mathbf{I}_\nu. \tag{18}$$

Equation (18) states that an identical matrix of dimension ν can be used as memory matrix under the assumptions of (14) and (15).

The assumptions just introduced are valid for the class of *accelerated life models* [17], where the relation $F^j(t) = F^i(t/q^j_i)$ holds between any generic couple of CDFs belonging to the same class. In fact, if $(\boldsymbol{\alpha}_i, \mathbf{G}_i)$ is the CPH representing $F^i(t)$, we have $F^j(t) = F^i(t/q^j_i) = 1 - \boldsymbol{\alpha}_i e^{\mathbf{G}_i \cdot t/q^j_i} \cdot \mathbf{1}$, i.e., the representation of $F^j(t)$ is $(\boldsymbol{\alpha}_i, \mathbf{G}_j/q^j_i)$, thus satisfying assumptions (14) and (15), with $c = \frac{1}{q^j_i}$.

Similarly, it is necessary to keep memory of the elapsed time when the variable timer policy introduced in Section 3 is used to manage the VMM rejuvenation timer. In particular, as discussed in Section 3.2, if at a given instant t_x of the timer local clock the workload changes from i to j VMs, the timer update policy is equivalent to fix the timer firing time to δ_j and update the timer local clock to

$$t'_x = t_x \frac{\delta_j}{\delta_i}. \tag{19}$$

We use a z-stage Erlang distribution with transition rates equal to $\frac{z}{\delta_j}$ to represent the timer CDF $F^j_{tr}(\cdot)$ through CPH distributions, because the Erlang distribution can well approximate a deterministic behavior. We indicate such CPH as $(\boldsymbol{\alpha}_j, \mathbf{E}_j)$, where $\boldsymbol{\alpha}_j = [10 \cdots 0]$ and

$$\mathbf{E}_j = \begin{bmatrix} -z/\delta_j & z/\delta_j & 0 & 0 & \ldots & 0 \\ 0 & -z/\delta_j & z/\delta_j & 0 & \ldots & 0 \\ 0 & 0 & -z/\delta_j & z/\delta_j & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & 0 & -z/\delta_j \end{bmatrix}.$$

According to such a representation, to update the timer local clock at the workload changing time, the CPH representing $F^j_{tr}(\cdot)$ has to be enabled with an initial probability vector equal to $\boldsymbol{\alpha}_j \cdot e^{\mathbf{E}_j \cdot t'_x}$. By applying (19), we have

$$\boldsymbol{\alpha}_j \cdot e^{\mathbf{E}_j \cdot t'_x} = \boldsymbol{\alpha}_j \cdot e^{\mathbf{E}_j \cdot t_x \frac{\delta_j}{\delta_i}} = \boldsymbol{\alpha}_i \cdot e^{\mathbf{E}_i \cdot t_x}, \tag{20}$$

given that $\mathbf{E}_i = \mathbf{E}_j \frac{\delta_j}{\delta_i}$ and $\boldsymbol{\alpha}_j = \boldsymbol{\alpha}_i$.

Equation (20) shows that, to represent the variable timer policy introduced in Section 3, at workload changing time t_x the probabilities to be in the phases of the CPH corresponding to $F^j_{tr}(\cdot)$ have to be equal to those of the CPH corresponding to $F^i_{tr}(\cdot)$. In the proposed CPH model, this can be expressed through an identical memory matrix of dimension z, thus $\mathbf{M}^{i\to j}_{tr}(\bar{t}) = \mathbf{I}_z$.

5.2 Symbolic Representation

Let us consider a discrete-state discrete-event model, and let S be the system state space and ε the ordered set of CPH distributed system events. A well-known result is that the stochastic process can be represented by an expanded process [22]. Such a process is composed of $\|S\|$ macrostates and it is characterized by a $\|S\| \times \|S\|$ block infinitesimal generator matrix $\mathbf{Q}$, where:

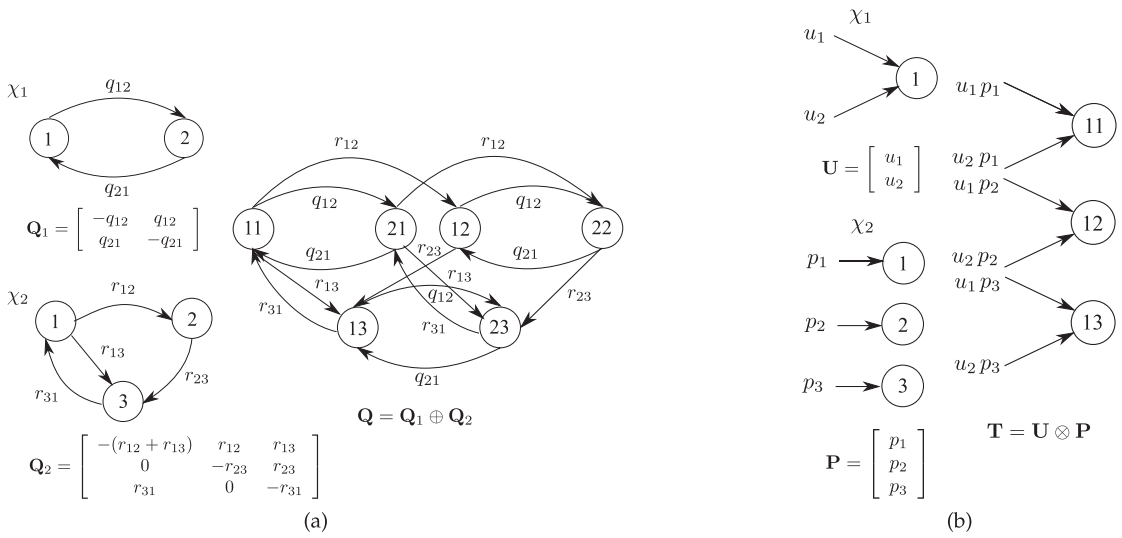

Fig. 9. Graphical representation of the semantic of operator $\oplus$ (a) and $\otimes$ (b).

- the generic diagonal block $\mathbf{Q}_{n,n}$ $(1 < n < \|S\|)$ is a square matrix that describes the evolution of the expanded process inside the macrostate related to state n, depending on the possible events enabled in such a state;
- the generic off-diagonal block $\mathbf{Q}_{n,m}$ $(1 < n < \|S\|$, $1 < m < \|S\|)$ describes the transition from the macrostate related to state n to the one related to state m, depending on the events occurred in state n and on the possible events that may still occur in state m.

The state-space expansion approach can be used to represent the VMM model, assuming the system events are associated with CPH distributions and a compact representation can be provided by Kronecker algebra [23]. Kronecker algebra is based on two main operators [24]: The product ($\otimes$) and the sum ($\oplus$). Given two rectangular matrices $\mathbf{A}$ and $\mathbf{B}$ of dimensions $m_1 \times m_2$ and $n_1 \times n_2$, respectively, their Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is a matrix of dimensions $m_1 n_1 \times m_2 n_2$. More specifically, it is a $m_1 \times m_2$ block matrix in which the generic block i, j has dimension $n_1 \times n_2$ and is given by $a_{i,j} \cdot \mathbf{B}$. On the other hand, if $\mathbf{A}$ and $\mathbf{B}$ are square matrices of dimensions $m \times m$ and $n \times n$, respectively, their Kronecker sum $\mathbf{A} \oplus \mathbf{B}$ is the matrix of dimensions $mn \times mn$ written as $\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_n + \mathbf{I}_m \otimes \mathbf{B}$.

By exploiting Kronecker algebra, the CPH generator matrix $\mathbf{Q}$ does not need to be generated and stored as a whole, but can be symbolically represented through Kronecker expressions (details on symbolic representation of PH based models can be found in [19].) In particular, the matrix $\mathbf{Q}$ blocks have the following form:

$$\mathbf{Q}_{ii} = \bigoplus_{1 \le e \le \|\varepsilon\|} \mathbf{Q}_e^d, \mathbf{Q}_{ij} = \bigotimes_{1 \le e \le \|\varepsilon\|} \mathbf{Q}_e^f. \tag{21}$$

In other words, the diagonal blocks are computed as Kronecker sums (off-diagonal blocks are computed as Kronecker products) of a series of matrices $\mathbf{Q}_e^d$ and $\mathbf{Q}_e^f$, each of which is associated with one of the system events and depends on their enabling in the states under exam.

Operators $\oplus$ and $\otimes$ assume specific meanings. Let us consider two CTMCs χ_1 and χ_2 with infinitesimal generator matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$. $\mathbf{Q}_1 \oplus \mathbf{Q}_2$ is usually interpreted as the infinitesimal generator matrix of the CTMC that models the concurrent evolution of χ_1 and χ_2. Fig. 9a shows a graphical representation of that. It is important to remark that the resulting CTMC is composed of a set of states that derives from the combination of all the states of the two initial processes.

On the other hand, let us consider a CTMC χ_1 with a single absorbing state, and let $\mathbf{U}$ be the column vector that contains the transition rates to such a state. Let us also consider a CTMC χ_2, and let $\mathbf{P}$ be a matrix containing the probabilities to enter into the χ_2 states. Thus, $\mathbf{U} \otimes \mathbf{P}$ is usually interpreted as the matrix containing the transition rates to the expanded process resulting from the combination of the absorbing state of χ_1 with all the states of χ_2. Fig. 9b depicts such situation.

According to this interpretation and due to the fact that in our model all the events are represented by CPH distributions, we use the $\oplus$ operator to describe the evolution of all the events evolving into a particular state of the VMM model ($\mathbf{Q}_{ii}$ matrices of (21)) and the $\otimes$ operator to represent the transitions from a state to another ($\mathbf{Q}_{ij}$ matrices of (21)). Thus, the $\mathbf{Q}_e^d$ matrices are the $\mathbf{G}$ of the enabled events in state i, whereas the $\mathbf{Q}_e^f$ are the $\mathbf{U}$ or the $\boldsymbol{\alpha}$ of the firing or enabling events, respectively, in the transition from state i to state j. We point out that a negative exponential distribution with rate λ can always be considered as a CPH distribution with $\mathbf{G} = [-\lambda]$, $\mathbf{U} = [\lambda]$, and $\boldsymbol{\alpha} = [1]$. As a consequence, from assumption 5 in Section 4, events e_{ar} and e_{dp} are characterized by $\mathbf{G}_{ar} = [-\gamma_{ar}]$, $\mathbf{U}_{ar} = [\gamma_{ar}]$, $\mathbf{G}_{dp}^i = [-\xi_{dp}^i]$, and $\mathbf{U}_{dp}^i = [\xi_{dp}^i]$.

Kronecker representation is suitable for applying the conservation of reliability principle and the variable timer policy when the events are implemented as CPH distributions. In fact, the memory matrix $\mathbf{M}_{fl}^{i \to j}(t)$ stores the probabilities to switch to the new CPH when a transition from state i to state j occurs keeping memory of the reached reliability. Thus, $\mathbf{M}_{fl}^{i \to j}(t)$ is used in the Kronecker product of (21) with the same meaning of the $\mathbf{P}$ of Fig. 9b. Similar considerations can be done for the trigger event for which matrix $\mathbf{M}_{tr}^{i \to j}(t)$ can be used. Regarding e_{ar} and e_{dp} events, the 1×1 matrix $[1]$ can be used during state transitions as a consequence of the memory-less property of exponential distributions.

TABLE 1
Kronecker Expressions for Matrix Blocks Related to States in the ith set of the Model of Fig. 5

Matrix block	*Kronecker expression*
$\mathbf{Q}_{4i,4i}$	$\mathbf{G}^i_{fl} \oplus [0] \oplus [0] \oplus \mathbf{G}^i_{tr} \oplus [0] \oplus [-\gamma_{ar}] \oplus [-\xi^i_{dp}]$
$\mathbf{Q}_{4i+1,4i+1}$	$[0] \oplus \mathbf{G}_{dt} \oplus [0] \oplus [0] \oplus [0] \oplus [0] \oplus [0]$
$\mathbf{Q}_{4i+2,4i+2}$	$[0] \oplus [0] \oplus \mathbf{G}^i_{rp} \oplus [0] \oplus [0] \oplus [0] \oplus [0]$
$\mathbf{Q}_{4i+3,4i+3}$	$[0] \oplus [0] \oplus [0] \oplus [0] \oplus \mathbf{G}^i_{rs} \oplus [0] \oplus [0]$
$\mathbf{Q}_{4i,4(i-1)}$	$\mathbf{M}_{fl}^{4i\to4(i-1)}(t) \otimes [1] \otimes [1] \otimes \mathbf{M}_{tr}^{4i\to4(i-1)}(t) \otimes \otimes [1] \otimes [1] \otimes [\xi^i_{dp}]$
$\mathbf{Q}_{4i,4(i+1)}$	$\mathbf{M}_{fl}^{4i\to4(i+1)}(t) \otimes [1] \otimes [1] \otimes \mathbf{M}_{tr}^{4i\to4(i+1)}(t) \otimes \otimes [1] \otimes [\gamma_{ar}] \otimes [1]$
$\mathbf{Q}_{4i,4i+1}$	$\mathbf{U}^i_{fl} \otimes \boldsymbol{\alpha}_{dt} \otimes [1] \otimes \mathbf{1}^i_{tr} \otimes [1] \otimes [1] \otimes [1]$
$\mathbf{Q}_{4i+1,4i+2}$	$[1] \otimes \mathbf{U}_{dt} \otimes \boldsymbol{\alpha}^i_{rp} \otimes [1] \otimes [1] \otimes [1] \otimes [1]$
$\mathbf{Q}_{4i+2,4i}$	$\boldsymbol{\alpha}^i_{fl} \otimes [1] \otimes \mathbf{U}^i_{rp} \otimes \boldsymbol{\alpha}^i_{tr} \otimes [1] \otimes [1] \otimes [1]$
$\mathbf{Q}_{4i,4i+3}$	$\mathbf{1}^i_{fl} \otimes [1] \otimes [1] \otimes \mathbf{U}^i_{tr} \otimes \boldsymbol{\alpha}^i_{rs} \otimes [1] \otimes [1]$
$\mathbf{Q}_{4i+3,4i}$	$\boldsymbol{\alpha}^i_{fl} \otimes [1] \otimes [1] \otimes \boldsymbol{\alpha}^i_{tr} \otimes \mathbf{U}^i_{rs} \otimes [1] \otimes [1]$

If an event is not enabled in state s_i, it does not contribute to the construction of the block $\mathbf{Q}_{ii}$, and therefore, a null 1×1 matrix $[0]$ is used in the Kronecker sum of (21). Similarly, when an event is not involved in a transition from state s_i to state s_j, the 1×1 matrix $[1]$ is used in the Kronecker product of (21).

We are now able to write the infinitesimal generator matrix $\mathbf{Q}$ of the expanded process derived from the model described in Section 4. We report in Table 1 the blocks of matrix $\mathbf{Q}$, where $(\boldsymbol{\alpha}^i_{fl},\mathbf{G}^i_{fl})$, $(\boldsymbol{\alpha}_{dt},\mathbf{G}_{dt})$, $(\boldsymbol{\alpha}^i_{rp},\mathbf{G}^i_{rp})$, $(\boldsymbol{\alpha}^i_{tr},\mathbf{G}^i_{tr})$, and $(\boldsymbol{\alpha}^i_{rs},\mathbf{G}^i_{rs})$ indicate the CPH representation of the $F^i_{fl}(t)$, $F_{dt}(t)$, $F^i_{rp}(t)$, $F^i_{tr}(t)$, and $F^i_{rs}(t)$ CDFs, respectively. Moreover, $\mathbf{U}^i_{fl}$, $\mathbf{U}_{dt}$, $\mathbf{U}^i_{rp}$, $\mathbf{U}^i_{tr}$, and $\mathbf{U}^i_{rs}$ are the corresponding firing rate column vectors. Finally, $\mathbf{1}^i_{fl}$, $\mathbf{1}_{dt}$, $\mathbf{1}^i_{rp}$, $\mathbf{1}^i_{tr}$, and $\mathbf{1}^i_{rs}$ are column vectors with all elements equal to 1 with appropriate dimensions. Details on how to construct matrix $\mathbf{Q}$ can be found in the Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2013.30.

Notice that, in general, the matrix $\mathbf{Q}$ thus obtained is time dependent. However, if the assumptions on the CPH representations of events e_{fl} and e_{tr} reported in Section 5.1 are satisfied, all the memory matrices are identical and the expanded process is a CTMC.

6 Model Evaluation

In this section, we study a cloud-enabled node as specified in Section 2, managed according to the timer policies described in Section 3. To do that it is necessary to operationally explain how to implement the variable timer policy, identifying the quantity to be optimized, and the guidelines for its application. Then, we focus on how to use the proposed modeling technique on a rejuvenation-enabled cloud node with a variable timer policy. To this specific purpose, a case study taken from the literature is investigated through the WebSPN tool [25] implementing the technique detailed in Section 5.

6.1 Optimal Rejuvenation Time

A time-based rejuvenation policy intends to identify the optimal time to rejuvenate with respect to one or more performance indices. The performance index we investigate is the VMM availability $A(t)$ that can be defined as:

$$A(t) = Pr\{\text{VMM is in state } s_{ur} \text{ at time } t\}. \tag{22}$$

More specifically, we focus on the steady-state VMM availability $A(\infty) = \lim_{t\to\infty} A(t)$.

In case of fixed timer policies, the optimal rejuvenation time can be obtained by varying the time interval $\delta \in \mathbb{R}^+$ to identify the value δ^* that maximizes the system availability:

$$\delta^* = \arg\max_{\delta} A(\infty). \tag{23}$$

With regard to the variable timer policy, the optimal rejuvenation time depends on the set $\{\delta_i, 0 \le i \le N\}$. Collecting the δ_i into the row vector $\boldsymbol{\Delta} = [\,\delta_0\ \delta_1 \cdots \delta_N\,]$, with $\delta_i \in \mathbb{R}^+$, we want to find the value $\boldsymbol{\Delta}^* = [\,\delta_0^*\ \delta_1^* \cdots \delta_N^*\,]$ that maximizes the system availability:

$$\boldsymbol{\Delta}^* = \arg\max_{\boldsymbol{\Delta}} A(\infty). \tag{24}$$

The maximization problem expressed by (24) is a $N+1$ dimension problem. In case of complex system and/or large N, such a problem becomes intractable; thus, to simplify it, we intend to exploit a relationship among the $N+1$ variables, decreasing the number of unknowns. The heuristic we propose takes inspiration from the proportionality rule among the optimal rejuvenation time of a system in isolation, i.e., a system where the workload is fixed throughout a rejuvenation epoch. In particular, fixing the system workload to i VMs, we can define the corresponding availability in isolation $A^i(t)$ as:

$$A^i(t) = Pr\{\text{VMM is in state } s_{ur} \text{ at time } t \mid \#VM(t) = i\}. \tag{25}$$

Then, it is possible to obtain $N+1$ optimal time intervals $\bar{\delta}_i$ that maximize the availability in isolation by solving the following $N+1$ problems:

$$\bar{\delta}_i = \arg\max_{\delta_i} A^i(\infty), \forall i : 0 \le i \le N. \tag{26}$$

The proportionality rule of such optimal time intervals in isolation can be obtained by choosing a value $\bar{\delta}_j$ as reference and by defining the $N+1$ vector $\mathbf{K}^j = [\,k^j_0\ k^j_1 \cdots k^j_N\,]$, where

$$k^j_i = \frac{\bar{\delta}_i}{\bar{\delta}_j}, 0 \le i \le N. \tag{27}$$

The obtained vector can then be used to compute the proportionality rule among the elements of $\boldsymbol{\Delta}$ as:

$$\boldsymbol{\Delta} = \delta_j \cdot \mathbf{K}^j, \tag{28}$$

where δ_j is the reference value.

In this way, the problem expressed by (24) can be reported to the following problem in a single variable δ_j:

$$\begin{aligned} \boldsymbol{\Delta}^* &= \arg\max_{\boldsymbol{\Delta}} A(\infty) \\ &\textit{subject to} : \boldsymbol{\Delta} = \delta_j \cdot \mathbf{K}^j. \end{aligned} \tag{29}$$

In practical terms, to adopt a variable timer rejuvenation policy three main steps have to be performed:

TABLE 2
Parameters Adopted for the Case Study Evaluation

Parameter	*Description*	*Value* [h^{-1}]	*Mean time* [h]
α	Weibull failure time scale parameter with $i = 1$	1000	923.577
β	Weibull failure time shape parameter	1.3	
μ	VMM repair rate with $i = 1$	2	0.5
η	VMM detection rate	12	0.0833
ρ	VMM resume rate with $i = 1$	30	0.0333
γ	VM arrival rate	1	1
ξ	VM departure rate with $i = 1$	0.5	2
γ_D	VM day arrival rate	1	1
γ_N	VM night arrival rate	0.2	5
σ_{DN} σ_{ND}	day-night/night-day rate	0.0833	12

1. *Evaluation setup*. Identification and characterization of all the model parameters and CDFs (failure, repair, and so on).
2. *Investigation of the system in isolation*. Computation of the optimal rejuvenation time intervals in isolation $\overline{\delta}_i$ by analyzing the model with a fixed workload. The final goal is obtaining the corresponding k_i^j by (26)-(28).
3. *Investigation of the variable workload configuration*. Analysis of the complete model for the computation of the optimal time intervals δ_i^* of (29).

6.2 Evaluation Setup

To properly set the system parameters, we refer to the work reported in [16], where a case study aiming at specifying the optimal timers for the software rejuvenation of both the VMM, and the VMs are fully implemented and investigated. Table 2 summarizes the parameters, we have adopted in our evaluation considering the maximum number of VMs is 4 ($N = 4$).

The most important difference between the parameters used in [16] and the ones reported in Table 2 is related to the VMM lifetime CDF. In [16], the aging process is modeled by splitting it into two consecutive steps. The first step represents the VMM transition from a fully operating state to a degraded one. The second step models the VMM failure from the degraded state. Both events are exponentially distributed, thus characterizing the overall aging process with a 2-stage hypoexponential CDF. Here, we can generalize such an assumption by representing the aging process through a generally distributed CDF, approximated through a multistage CPH. In particular, because in the reliability context one of the most used CDF is the Weibull distribution, we choose a Weibull CDF to characterize the failure event, fixing the parameters ($\alpha = 1{,}000$ and $\beta = 1.3$) to be in accordance with those used in [16]. In this way, the CDF $F_{fl}^1(t)$ is identified.

In the field of reliability and availability analysis, one of the most widely used approach to simply represent workload dependent conditions in analytical terms is the *proportional hazard model* (PHM) [26]. In a PHM, the effect of the workload on the reliability is expressed in terms of a multiplicative effect on the failure rate. In the specific case of the system under analysis, assuming

$$\lambda_{fl}^1(t) = -\frac{d(F_{fl}^1(t))/dt}{F_{fl}^1(t)}$$

is the failure rate of the VMM on top of which one VM is running, we can express the failure rate of the VMM with $i > 1$ VMs as $\lambda_{fl}^i(t) = -\frac{d(F_{fl}^i(t))/dt}{F_{fl}^i(t)} = c(i) \cdot \lambda_{fl}^1(t)$. Similarly, the case in which no VMs are running can be characterized by $\lambda_{fl}^0(t) = d \cdot \lambda_{fl}^1(t)$ with $0 \leq d \leq 1$. In this way, we assume that the distributions characterizing the VMM lifetime in different workload conditions differ only for a parameter $c(i)$ that depends on the workload. A relevant part of the existing literature on such a topic uses this kind of PHM [13], [14] to represent load sharing.

Therefore, being the CDFs belonging to the $\mathcal{F}_{fl}$ set Weibull distributions with rate $\beta/\alpha(\frac{t}{\alpha})^{\beta-1}$ and assuming $\beta_i = \beta$, we can easily obtain that the PHM condition can be expressed just in terms of the scale parameter as $\alpha_i = \frac{\alpha}{c(i)^{1/\beta}}$.

To investigate the impact of different PHMs, in the following, we consider three cases by varying the way the failure rates are affected by the workload, i.e., by setting function $c(i)$. In particular, we consider:

- *Sublinear*. The failure rates vary with the workload according to $c(i) = \sqrt{i}$.
- *Linear*. Failure rates linearly varying with the workload, i.e., $c(i) = i$.
- *Superlinear*. The failure rates vary with the workload according to $c(i) = i^2$.

If no VMs are instantiated ($i = 0$), we assume that the VMM does not degrade, and therefore, it is only necessary to keep memory of the age that was reached at the workload changing point. In numerical terms, this means that $\lambda_{fl}^0(t) = 0 \cdot \lambda_{fl}^1(t)$, i.e., $d = 0$.

According to [16], we consider the time to resume, repair, and detect as exponentially distributed. However, while the detect event does not depend on the workload and its rate is $\eta_{dt} = \eta$, repair and resume rates are workload dependent, i.e., $\mu_{rp}^i = \mu/i$ and $\rho_{rs}^i = \rho/i$, respectively.

Finally, let us describe the parameters related to the workload events. While the time between two departures (event e_{dp}) is characterized by an exponential distribution with workload dependent rate $\xi_{dp}^i = i \cdot \xi$, two different conditions are considered for the arrival event e_{ar}:

- *Poisson arrival process (PAP)*. We assume an exponentially distributed interarrival time with rate $\gamma_{ar} = \gamma$.
- *Markov modulated poisson arrival process (MMPP)*. We assume an MMPP that models a variation of the workload due, for example, to the day-night cycle: Two different arrival rates γ_D and γ_N are considered with $\gamma_D > \gamma_N$; the process switches between them according to day-night/night-day rates σ_{DN} and σ_{ND}.

6.3 Investigating the System in Isolation

Aim of this section is to show how to obtain the optimal rejuvenation time intervals $\overline{\delta}_i$ corresponding to the system with a fixed workload.

For each specific workload condition, the value of $\overline{\delta}_i$ can be obtained by analyzing, through the proposed technique, the submodel corresponding to the ith set of states in the model of Fig. 5 with events e_{ar} and e_{dp} disabled. In the

TABLE 3
Optimal Rejuvenation Time Intervals and Proportionality Coefficients Varying the Workload PHM

		#VM			
		1	2	3	4
$c(i)=\sqrt{i}$	$\overline{\delta}_i$ [h]	337.5	258.5	221.2	200.5
	k_i^1	1	0.765983	0.655378	0.594177
$c(i)=i$	$\overline{\delta}_i$ [h]	337.5	200.5	145.5	119
	k_i^1	1	0.594177	0.431034	0.352609
$c(i)=i^2$	$\overline{\delta}_i$ [h]	337.5	119	62.3	40
	k_i^1	1	0.352609	0.184488	0.11851

following, the workload condition $i=0$ is not taken into account because we assumed that the VMM does not age when no VMs are running and the Rejuvenation Manager is disabled.

Table 3 reports the VMM optimal rejuvenation time intervals $\overline{\delta}_i$ obtained by evaluating the steady-state availability $A^i(\infty)$ under different rejuvenation time intervals and according to the considered PHMs. Considering one VM as reference workload and consequently $\overline{\delta}_1$ as the reference time interval, it is possible to characterize the optimal time intervals as specified by (28). The values of coefficients k_i^1 (see (27)) are also reported in the table.

6.4 Workload and Policy Impact Assessment

Once the k_i^1 coefficients of (27) have been obtained, to apply the variable timer policy it is necessary to evaluate the optimal rejuvenation timer of (29), by evaluating the model of Fig. 5. More specifically, we have to evaluate the expanded process obtained by expressing the VMM lifetime distributions in terms of CPH as discussed in Section 5.

The specific PHM characterizing the VMM lifetime distributions of the example in the different workload conditions can be considered as an accelerated life model by expressing the relationship among the lifetime distributions in terms of time and verifying that a q_i^1 exists such that $F_{fl}^i(t)=F_{fl}^1(\frac{t}{q_i^1})$. Thus, in case of Weibull CDF $F_{fl}^i(t)=1-e^{-i(\frac{t}{\alpha})^\beta}$, we have that

$$1-e^{-i(\frac{t}{\alpha})^\beta}=1-e^{-(\frac{t}{\alpha q_i^1})^\beta},$$

i.e., $q_i^1=\frac{1}{i^{(1/\beta)}}$. As discussed in Section 5.1, thanks to such property, in our experiments we computed the CPH $(\boldsymbol{\alpha}_1,\mathbf{G}_1)$ representing $F_{fl}^1(t)$ by exploiting the fitting algorithm introduced in [27], and then we obtained the CPHs representing $F_{fl}^i(t)$, with $i=2,\ldots,N$, as $(\boldsymbol{\alpha}_1,\frac{1}{q_i^1}\mathbf{G}_1)$.

Due to the characteristics of such CPHs in our model, the memory matrices algebraically representing the conservation of reliability principle among the lifetime distributions, as well as those implementing the timer updating due to workload condition variations, are always identical. As a consequence the whole expanded state-space model corresponding to the VMM rejuvenation is a CTMC.

In case of MMPP arrival process, assumption 5 of Section 4 is not valid given that the alternation between day and night needs to be adequately represented even if the VMM is not in a working state. This kind of behavior can be fully obtained by replicating the states of the model in Fig. 5, thus obtaining the model described in Appendix B, available in the online supplemental material.

The steady-state availability $A(\infty)$, obtained by evaluating the model depicted in Fig. 5 varying the rejuvenation timer in both the two arrival processes (PAP and MMPP), is shown in Fig. 10. Each graph compares the results of the two rejuvenation policies taken into account. The trends, thus, obtained are quite similar. All of them initially increase by increasing the timer till a maximum value of the steady-state availability $A^{max}(\infty)$, and therefore slowly decrease to an asymptotic value. This value is the steady-state availability $A^{nr}(\infty)$ that is obtained by only considering the repair and without any rejuvenation policy. In the case of the variable timer rejuvenation policy, the value reported in the x-axis is the time interval δ_1 corresponding to the $\#VM=1$ workload condition. The optimal value δ_1^* can be used, in combination with the values of k_i^1, to compute the time intervals δ_i^* that can be exploited during the implementation of the technique.

The graphs clearly highlight that in both cases the steady-state availability with the variable timer rejuvenation policy is higher than the fixed timer one, as numerically confirmed by the values reported in Table 4. Table 4 also shows the optimal time intervals and the maximum steady-state availability in the sublinear and superlinear cases.

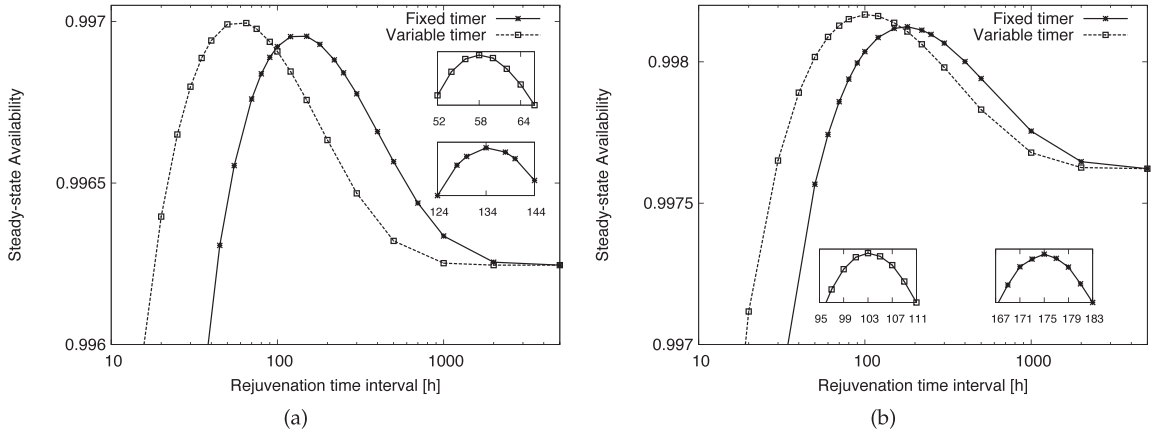

Fig. 10. Steady-state availability $A(\infty)$ of the VMM varying the rejuvenation timer, assuming a PAP (a) and an MMPP (b) VM arrival process. For the variable policy, the rejuvenation time interval reported in the x-axis is δ_1. The magnified regions show the values of the time interval corresponding to the maximum value of availability.

TABLE 4
Optimal Time Intervals and Steady-State Availability Varying the System Configuration

configuration			Optimal time int. [h]	$A^{max}(\infty)$	$A^{nr}(\infty)$
$c(i)$	arrival	timer			
$\sqrt{i}$	*PAP*	Fixed	215	0.998101	0.997661
		Variable	139	0.998113	
	MMPP	Fixed	264	0.998767	0.998748
		Variable	197	0.998784	
i	*PAP*	Fixed	134	0.996958	0.996246
		Variable	58	0.996999	
	MMPP	Fixed	175	0.998124	0.997622
		Variable	103	0.998167	
i^2	*PAP*	Fixed	51	0.991979	0.990088
		Variable	12	0.992092	
	MMPP	Fixed	74	0.995355	0.994129
		Variable	35	0.995409	

The reported time interval for the variable policy is δ_1^*.

We can argue that, as expected, the value of $A^{nr}(\infty)$ in the case of MMPP arrival process is always higher than the one corresponding to the PAP. This is due to the fact that during the night the system is underloaded, thus resulting in a slower aging phenomenon. Moreover, it is possible to quantify the effects on the availability due to the different PHMs taken into consideration. In particular, such a value decreases as the effect of the workload on the aging process becomes more evident, i.e., moving from $c(i) = \sqrt{i}$ to $c(i) = i^2$. Results in Table 4 also confirm that the variable rejuvenation timer policy always outperforms the fixed policy in all the analyzed working conditions, thus demonstrating the usefulness of the proposed approach. In fact, even very low increments in the availability reflect on remarkable reductions in the downtime per year (tens of minutes for availability increments in the order of 10^{-4}). Given that the hourly cost of service downtime for IT companies has been estimated in the order of thousand of dollars[1] and also considering its negative impact on business reputation, such gains in availability justify the increase in system management complexity.

7 Related Work

Software rejuvenation is a proactive technique that allows to prevent the occurrence of software failures [6]. We can distinguish between two main rejuvenation approaches: *model-based* and *measurement based*. The model-based approach [7], [28], [8], [29], [30] focuses on analytical models representing the system behavior to investigate the costs/benefits associated with rejuvenation. Measurement-based techniques [9], [10], [31] analyze data coming from real systems to extract information on software execution (e.g., analyzing memory or CPU usage) to characterize the software aging process. Model-based rejuvenation has the advantage to study the optimal rejuvenation parameters but it needs to know the system lifetime distribution, while the second one does not specify any assumption on the aging process even if it obtains nonoptimal solutions. A first attempt to bridge the gap between such two techniques is provided in [7].

1. International Working Group on Cloud Computing Resiliency—http://iwgcr.org.

Another classification can be made with respect to the final objective of the rejuvenation technique. One of the most analyzed parameters is the time to rejuvenation to find the optimal value that correspond to the minimum system downtime. Such techniques are referred to as *time-based* rejuvenation [7], [10]. On the other hand, techniques that analyze the system behavior to predict the failure (e.g., using alarm thresholds) are categorized as *prediction-based* rejuvenation [31], [32].

In [33], the authors demonstrate that the rate at which software ages is not constant but depends on the time-variable workload. They observe that the use of periodic rejuvenation at constant time could not be the optimal approach. Starting from such a work, we proposed the timer variable policy with the final goal to optimize the rejuvenation process with respect to the actual system workload, overcoming the limits of periodic rejuvenation.

In [7], the authors propose a comprehensive model to study the software rejuvenation of a UNIX operating system taking into account also the system workload. They combine the analytical and measurement-based techniques by first collecting and analyzing data to study the influence of the workload on the system resource depletion. Such data are then used to model the failure distribution. Finally, the workload-dependent failure distribution is introduced in an availability model to investigate the optimal rejuvenation time. With respect to this work, our approach allows to represent the workload-dependent aging process directly in the availability model, thus providing a better approximation. This choice affects the state-space size but its impact is mitigated through the symbolic Kronecker-based representation, as discussed in Section 5. Furthermore, the modeling technique we adopted does not introduce any restriction or limitation in the lifetime CDF representation.

Moreover, in [34] a variable timer approach consisting in delaying the rejuvenation if scheduled during a workload peak, with the goal of optimizing service performance, is considered. A similar approach is adopted in [30], where variable timer policies, delaying the rejuvenation at timer expiration if there are jobs awaiting to be processed in queue, are specified, evaluating the system availability through extended Petri nets. Thus, in such works, the timer is constant and the system can just delay the rejuvenation at its expiration, to serve the queued jobs with availability/performance maximization purposes. Such technique is, therefore, not able to anticipate the rejuvenation if the workload and consequently the aging speed increases. The variable time-based rejuvenation policy we propose effectively adapts the timer to workload conditions, anticipating or delaying the rejuvenation depending on the actual software degradation.

More recently, the use of rejuvenation in cloud and virtualized system has been proposed in the literature. In [35] and [36], the use of VMs and of the related management techniques (e.g., virtualization, checkpointing) is justified to reduce the system downtime during the rejuvenation. VM and VMM rejuvenations in cloud environments are investigated in [12] and [16] through the use of analytical techniques. In [12], the authors highlight the need to rejuvenate the VMs as well as the VMM, measuring the advantages obtained in terms of system availability. In [16],

three different rejuvenation schemes are proposed starting from the concept that a VMM rejuvenation affects the running VMs. According to these schemes, each time the VMM is rejuvenated the running VMs can be suspended, rebooted, or migrated.

The techniques above discussed [35], [36], [12], and [16], even if very relevant, do not take into account the VMM workload. To the best of our knowledge, this paper is the first attempt to investigate the VMM rejuvenation also considering the system workload, in terms of number of running VMs. We think that such issue is very relevant in cloud environments, where workloads are highly variable, with peaks and bursts. As a consequence, such variations assume strategic importance in designing optimal rejuvenation strategies, e.g., due to SLA and QoS constraints.

8 Conclusions and Future Work

The main contribution of this work is twofold: On the one hand an analytic technique that allows to represent any generic failure and repair distributions, adequately modeling changes in the workload through the conservation of reliability principle; on the other hand, a variable timer rejuvenation policy aiming at optimizing the software (VMM) availability in case of workload changes. The obtained results show that the proposed variable timer policy outperforms the fixed timer one, also considering different impact of the workload on the aging process.

We are working to extend the proposed policy and technique in several directions: 1) enriching the model to consider performability metrics such as throughput and service availability; 2) extending the technique also considering the VM impacts and related management and migration policies.

References

[1] L. Bittencourt, C. Senna, and E. Madeira, "Scheduling Service Workflows for Cost Optimization in Hybrid Clouds," *Proc. Int'l Conf. Network and Service Management,* pp. 394-397, 2010.

[2] S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising from Cloud Computing," *Proc. IEEE second Int'l Conf. Cloud Computing Technology and Science,* pp. 693-702, 2010.

[3] R. Ghosh, K. Trivedi, V. Naik, and D.S. Kim, "End-to-End Performability Analysis for Infrastructure-as-a-Service Cloud: An Interacting Stochastic Models Approach," *Proc. IEEE 16th Pacific Rim Int'l Symp. Dependable Computing,* pp. 125-132, 2010.

[4] S. Distefano, F. Longo, and M. Scarpa, "Availability Assessment of HA Standby Redundant Clusters," *Proc. IEEE 29th Symp. Reliable Distributed Systems,* pp. 265-274, 2010.

[5] M. Grottke and K.S. Trivedi, "Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate," *Computer,* vol. 40, no. 2, pp. 107-109, Feb. 2007.

[6] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software Rejuvenation: Analysis, Module and Applications," *Proc. 25th Int'l Symp. Fault-Tolerant Computing (FTCS),* pp. 381-390, 1995.

[7] K. Vaidyanathan and K.S. Trivedi, "A Comprehensive Model for Software Rejuvenation," *IEEE Trans. Dependable and Secure Computing,* vol. 2, no. 2, pp. 124-137, Apr.-June 2005.

[8] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Net," *Proc. Sixth Int'l Symp. Software Reliability Eng.,* pp. 180-187, 1995.

[9] K. Vaidyanathan and K. Trivedi, "A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems," *Proc. 10th Int'l Symp. Software Reliability Eng.,* pp. 84-93, 1999.

[10] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, "A Methodology for Detection and Estimation of Software Aging," *Proc. Ninth Int'l Symp. Software Reliability Eng.,* pp. 283-292, 1998.

[11] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *Nat'l Inst. of Standards and Technology,* vol. 53, no. 6, p. 50, 2011.

[12] A. Rezaei and M. Sharifi, "Rejuvenating High Available Virtualized Systems," *Proc. Int'l Conf. Availability, Reliability, and Security,* pp. 289-294, 2010.

[13] P.H. Kvam and E.A. Peña, "Estimating Load-Sharing Properties in a Dynamic Reliability System," *J. Am. Statistical Assoc.,* vol. 100, no. 469, pp. 262-272, 2005.

[14] L. Huang and Q. Xu, "Lifetime Reliability for Load-Sharing Redundant Systems with Arbitrary Failure Distributions," *IEEE Trans. Reliability,* vol. 59, no. 2, pp. 319-330, June 2010.

[15] D. Kececioglu, *Reliability Engineering Handbook (Vol. 1 and 2).* Prentice-Hall, Inc., 1991.

[16] F. Machida, D. Kim, and K. Trivedi, "Modeling and Analysis of Software Rejuvenation in a Server Virtualized System," *Proc. IEEE Second Int'l Workshop Software Aging and Rejuvenation,* pp. 1-6, 2010.

[17] M.S. Finkelstein, "Wearing-Out of Components in a Variable Environment," *Reliability Eng. and System Safety,* vol. 66, no. 3, pp. 235-242, 1999.

[18] D.R. Cox, *Renewal Theory.* Methuen Ltd, 1967.

[19] F. Longo and M. Scarpa, "Applying Symbolic Techniques to the Representation of Non-Markovian Models with Continuous ph Distributions," *Proc. Sixth European Performance Eng. Workshop Computer Performance Eng. (EPEW),* pp. 44-58, 2009.

[20] S. Distefano, F. Longo, and M. Scarpa, "Symbolic Representation Techniques in Dynamic Reliability Evaluation," *Proc. IEEE Int'l Symp. High-Assurance Systems Eng.,* pp. 45-53, 2010.

[21] M. Neuts, "Probability Distributions of Phase Type," *Liber Amicorum Professor Emeritus H. Florin,* pp. 173-206, Louvain Univ., 1975.

[22] M.F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* Johns Hopkins Univ. Press, 1981.

[23] M. Scarpa, "Non Markovian Stochastic Petri Nets with Concurrent Generally Distributed Transtions," PhD dissertation, Univ. of Turin, 1999.

[24] R. Bellman, *Introduction to Matrix Analysis,* second ed. SIAM, 1997.

[25] A. Bobbio, A. Puliafito, M. Scarpa, and M. Telek, "Webspn: A Web-Accessible Petri Net Tool," *Proc. Conf. Web-Based Modeling and Simulation,* 1998.

[26] D.R. Cox, "Regression Models and Life-Tables," *J. Royal Statistical Soc. Series B (Methodological),* vol. 34, no. 2, pp. 187-220, 1972.

[27] A. Bobbio, A. Horváth, and M. Telek, "PhFit: A General Phase-Type Fitting Tool," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* p. 543, 2002.

[28] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems," *IEEE Trans. Computers,* vol. 47, no. 1, pp. 96-107, Jan. 1998.

[29] K. Vaidyanathan, R.E. Harper, S.W. Hunter, and K.S. Trivedi, "Analysis and Implementation of Software Rejuvenation in Cluster Systems," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 29, pp. 62-71, June 2001.

[30] F. Salfner and K. Wolter, "Analysis of Service Availability for Time-Triggered Rejuvenation Policies," *J. Systems and Software,* vol. 83, no. 9, pp. 1579-1590, 2010.

[31] V. Castelli, R.E. Harper, P. Heidelberger, S.W. Hunter, K.S. Trivedi, K. Vaidyanathan, and W.P. Zeggert, "Proactive Management of Software Aging," *IBM J. R&D,* vol. 45, no. 2, pp. 311-332, 2001.

[32] L. Silva, H. Madeira, and J. Silva, "Software Aging and Rejuvenation in a Soap-Based Server," *Proc. IEEE Fifth Int'l Symp. Network Computing and Applications,* pp. 56-65, 2006.

[33] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi, "Analysis of Software Aging in a Web Server," *IEEE Trans. Reliability,* vol. 55, no. 3, pp. 411-420, Sept 2006.

[34] D. Wang, W. Xie, and K.S. Trivedi, "Performability Analysis of Clustered Systems with Rejuvenation Under Varying Workload," *Performance Evaluation,* vol. 64, pp. 247-265, 2007.

[35] L. Silva, J. Alonso, and J. Torres, "Using Virtualization to Improve Software Rejuvenation," *IEEE Trans. Computers,* vol. 58, no. 11, pp. 1525-1538, Nov. 2009.

[36] D. Simeonov and D.R. Avresky, "Proactive Software Rejuvenation Based on Machine Learning Techniques," *Proc. First Int'l Conf. Cloud Computing,* pp. 186-200, 2010.

Dario Bruneo received the PhD degree in advanced technologies for information engineering from the University of Messina, Italy, in 2005. He is an assistant professor at the Engineering Faculty of the University of Messina. His scientific activity has been focused on studying distributed systems, particularly with regard to management techniques. His primary research interests include grid and cloud computing, sensor networks, QoS management, and performance evaluation. He is a member of the IEEE.

Salvatore Distefano is an assistant professor at Politecnico di Milano. His research interests include performance evaluation, distributed computing, software engineering, and reliability techniques. He contributed to the development of several tools such as WebSPN, ArgoPerformance, and GS3. He has been involved in several national and international research projects. He serves the editorial boards and committees of several journals and conferences.

Francesco Longo received the PhD degree in advanced technologies for information engineering from the University of Messina, Italy, in 2011. He is now a postdoc researcher within the Vision Cloud European Project. His primary research interests include performance and reliability evaluation of distributed systems (in particular grid and cloud) with main attention to the use of non-Markovian stochastic Petri nets.

Antonio Puliafito is a full professor of computer engineering at the University of Messina, Italy. His interests include parallel and distributed systems, wireless technologies, and grid and cloud computing. He is also interested in the performance evaluation of such systems. He currently acts as the main investigator of the Italian PRIN2008 research project *Cloud@Home,* trying to combine cloud and volunteer computing. He is a member of the IEEE.

Marco Scarpa received degree in computer engineering from the University of Catania, Italy, in 1994, and the PhD degree in computer science in 2000, from the University of Turin, Italy. He is currently an associate professor at the University of Messina. His research interests include performance and reliability modeling of distributed systems, phase type distributions, and software performance evaluation techniques. He has been involved in several research projects.