Chapter 10

Aging in Virtualized Environments

Roberto Pietrantuono[*], Domenico Cotroneo[†] and Stefano Russo[‡]

Università degli Studi di Napoli Federico II, Italy
[*] *roberto.pietrantuono@unina.it*
[†] *cotroneo@unina.it*
[‡] *stefano.russo@unina.it*

With virtualization technologies becoming a predominant paradigm of computing, there is a strong need to analyze long-running performance and aging issues in virtualized environments. Software aging in such environments can have a severe impact on millions of customers of business-critical applications, as well as on end-users of more critical (e.g., mission-critical) systems where virtualization is also vastly being adopted.

This chapter discusses the main existing approaches to studying the aging phenomena in virtualized environments, and the corresponding rejuvenation techniques. It targets those contexts where virtual machines are massively adopted — such as cloud computing platforms — as well as relevant virtualization technologies.

The chapter covers first model-based methods for the analysis of resource consumption and/or performance degradation, as well as for the optimal scheduling of rejuvenation in virtualized systems. Then, measurement-based approaches are discussed, where the statistical analysis of monitored health indicators is exploited to detect and forecast aging trends in the considered contexts.

10.1. Introduction

This chapter explores the main approaches to deal with software aging analysis and with rejuvenation in virtualized environments. Virtualization is a key technology for today's IT organizations. Virtualization solutions greatly ease the build and deploy of applications, increase their runtime availability, and alleviate several manual tasks — saving people time to deliver greater value. Moreover, by reducing the number of physical machines, the infrastructure cost, the maintenance and IT staffing cost as well as the backup and recovery cost are drastically lowered. For instance, a Forrester study[a] found that Red Hat Virtualization speeds up virtualization tasks and improves performance by delivering the following results: *i)* a return on investment of 103% over 3 years; *ii)* 10% to 20% of the infrastructure developer's time saved through an increased number of virtualization tasks and increased process efficiency; *iii)* a payback period of 5.6 months.

A main advantage of virtualization is its ability to make better use of existing resources, enabling multiple systems (i.e., the operating system, system-level software and application-level software) to run on the same physical machine, sharing and dynamically acquiring/releasing the same set of resources (e.g., the CPU, memory and storage) depending on the runtime need. Systems/services making massive use of virtualization in this sense are those based on cloud computing. In this case, virtualization is implemented by means of an intermediate software layer called a hypervisor or Virtual Machine Monitor (VMM), responsible for managing the execution of virtual machines and for the interaction with the hardware resources. It works by separating virtual from physical layers. Another advantage of virtualization is the much greater portability of applications (and of entire systems) across various physical environments. Besides cloud computing, a former successful example of software virtualization technology, with a strong focus on portability, is the Java Virtual Machine (JVM).

As many enterprises employ virtualization to run their services, the problem of performance degradation and, more generally of *software aging*, becomes critical. Aging has been analyzed by researchers in several modern virtualized environments, wherein long executions lead to accumulation errors and gradual performance decrease. Conventional approaches for aging detection and estimation have been used, which can be roughly divided into model-based and measurement-based approaches. At the same time, besides aging detection and estimation, which can be used to determine if and *when* an aging failure is likely to occur, the problem of *how* to rejuvenate a virtualized system is also of interest to software aging and rejuvenation research. Indeed, rejuvenation is a key high-availability technique in this context, since the advantages given by VMs as application "containers" have led to novel and lightweight rejuvenation strategies (based on restarts, migration and failover), which are capable of guaranteeing high levels of service continuity in the presence of aging problems.

[a] "The Total Economic Impact of Red Hat Virtualization", available at: *www.redhat.com/it/engage/efficiency-of-virtualization.*

The next section introduces the general characteristics of SAR research in virtualized environments. Section 10.3 contains a survey of the model-based techniques for *aging*, while Section 10.4 reviews the measurement-based techniques. Hybrid approaches combining both types of techniques are presented in Section 10.5. Section 10.6 presents the main techniques that are used to perform rejuvenation in virtualized environments. Finally, Section 10.7 contains the chapter conclusions.

10.2. Aging Analysis and Rejuvenation for Virtualized Environments

The analysis of software aging deals usually with two main tasks, the detection and the estimation of aging trends through one or more *aging indicators*. These are related to resource exhaustion (e.g., memory, storage, operating system resources, energy), and/or to user-perceived performance (e.g., response time, latency, served transactions, throughput). These *aging indicators* characterize a degradation phenomenon that is customer-affecting and that can also be detected at the system level within the *execution environment* of the application software. In virtualized environments, the *system* level — namely, the *execution environment* — includes not only the physical machine with its operating system (OS) and low-level software, but also the virtual environment, such as the Virtual Machine (VM) on which the software is running and/or the whole virtualization technology including the Virtual Machine Monitor (VMM), the most important layer responsible for creating a VM environment for an OS and its applications. Figure 10.1 sketches a generic architecture of a virtualized environment. Virtualization can be implemented: *i)* via native VMMs, also known as *bare-metal hypervisors*, such as *Xen, VMware ESX, Oracle Vm Server*, in which there is no "host" OS but the VMM runs on and controls the hardware directly, or *ii)* via"hosted" solutions, also known as *hosted hypervisors*, like, for instance, *VMware Workstation, VirtualBox and Parallels*, in which the access to hardware is managed via the host OS. On top of the VMM, there are VMs: depending on the proactive/reactive fault tolerance strategy (including rejuvenation), there can be standby VMs, which can take over the active VMs in case of failure or upon a rejuvenation request. In a more general scheme (e.g., in virtualized data centers), there may be other physical nodes and a VM can be migrated to a different node if, for instance, the hosting node or part thereof, such as the VMM, fails or needs to be rejuvenated.

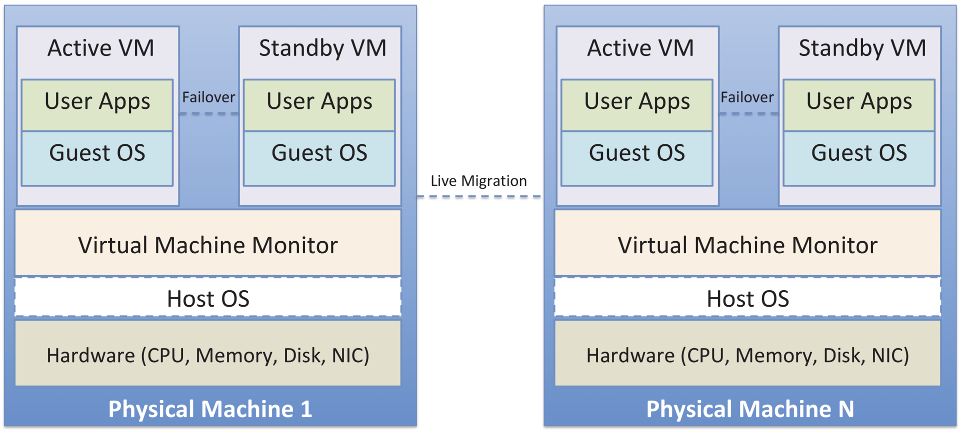

Fig. 10.1. A generic architecture of a virtualized environment.

In such a scenario, the *execution environment* is quite different from a traditional non-virtualized environment. From the SAR perspective, this affects: *i)* the choice of the most appropriate strategy for aging analysis (i.e., how to predict the most likely time of aging failure occurrence in order to figure out *when* to schedule a rejuvenation action), and *ii)* the consequent rejuvenation action to perform (i.e., *how* rejuvenation should be performed).

The ultimate goal of aging analysis — encompassing detection and estimation — is to determine the most likely time of aging failure occurrence, so as to figure out *when* to schedule a rejuvenation action. This is done either through analytical modelling — formulating the problem via stochastic models and finding the best rejuvenation time, when the system availability/survivability/performability will be maximized — or by monitoring a system's health indicators and applying statistical inference techniques on time series to forecast future trends. These approaches, also known as **model-based** and **measurement-based**, respectively, are sometimes combined into a hybrid strategy, wherein models are properly parametrized by field data. In the next sections, aging analysis and then rejuvenation techniques proposed for virtualized environments are explained.

10.3. Model-Based Analysis Techniques

Model-based aging analysis techniques for virtualized environments share the following characteristics:

- The software applications within a more or less complex virtualized environment (with one or more VMs, physical hosts, and the associated rejuvenation strategies) are most commonly modeled by Markov models, including continuous-time Markov chains (CTMCs), homogeneous and non-homogeneous, Markov regenerative processes (MRGMs), semi-Markov processes (SMPs), stochastic Petri nets (SPNs) and stochastic reward nets (SRNs), that capture the aging behaviour at different levels of abstraction (the application or virtualized environment or physical environment level) and, sometimes, the non-aging failing behavior too [1].
- Model-based aging analysis techniques do not usually refer to specific aging indicators, since the model is assumed to work regardless of which resource is being depleted or of the user-perceived metric. Even when the model refers to a specific indicator, it can easily be applied to other aging indicators. The attribute of interest is related to the final (user-perceived) metric of *availability* (or to related metrics, such as *survivability* or *performability*).
- The evaluation of alternatives is one of the objectives of model-based analyses: in most cases, model-based techniques are associated with the evaluation of alternative *rejuvenation* techniques at various levels, such as at the application level, at the VM level, at the VMM/cloud platform level, or at the node level. Indeed, exploring many such alternatives with a measurement-based approach would be too onerous. The adopted models very often encompass the rejuvenation action being performed, for which proper parameters are considered (e.g., the time/cost for rejuvenation) for the availability computation on different rejuvenation alternatives. SAR research for virtualized systems is addressed by more studies than SAR literature for generic systems, since, in virtualized environments, there are more viable options for rejuvenation.
- Model-based techniques are usually validated by means of analysis or simulation, since the collection of runtime data for all or part of the cases being modelled is expensive and may not be representative of the system being modelled.
- Models are, by their nature, applicable to many systems; their drawbacks lie in the simplifying assumptions, such as those about the underlying stochastic distributions that characterize the system. This too is not different in the case of the virtualized environment SAR research area.

An example of the model-based approach, from Chang *et al.* [2], is shown in Fig. 10.2.

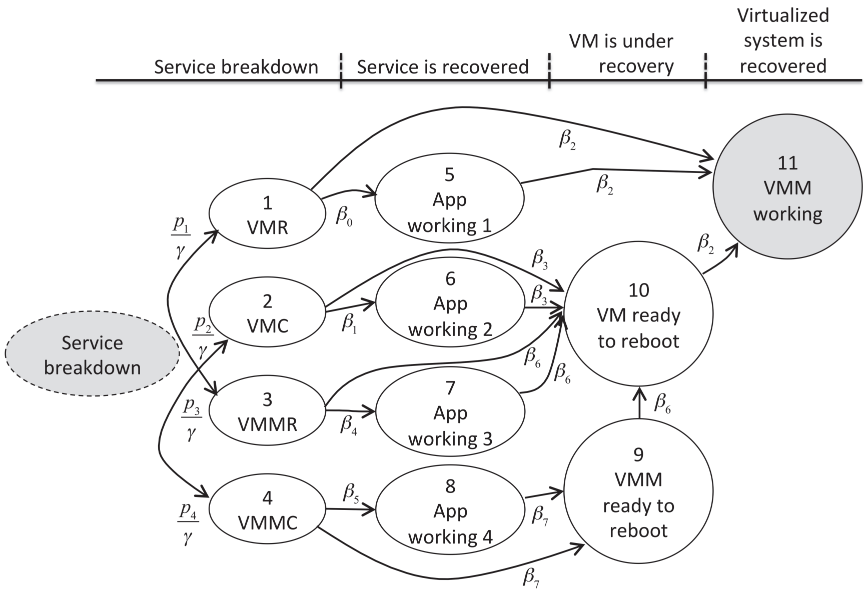

Fig. 10.2. A model of aging-related service failure/recovery in a virtualized system (from Chang *et al.* [2]).

In this CTMC model, the initial state is conditioned to be the *service breakdown*. It is assumed that no other failure occurs during the system recovery. When the service breakdown occurs due to an active VM rejuvenation or an active VM reboot caused by an aging-related bug (VMR), a standby VM on the same host is selected. The failover requires the time $1/\beta_0$; the VM restart requires $1/\beta_2$. VMC represents that the service breakdown is due to a VM crash caused by a non-aging Mandelbug. The failover takes the time $1/\beta_1$ (larger than $1/\beta_0$); the VM is repaired in the time $1/\beta_3$. After repair, the state is "VM ready to reboot", and the service is fully recovered after $1/\beta_2$. Similarly, VMMR is the service breakdown due to the VMM rejuvenation, and VMMC is the service breakdown caused by a VMM crash. Live VM migration is applied in these cases, with different time delays. In this *multi-phase CTMC model*, transient *survivability* is considered as the measure of interest, and VM/VMM rejuvenation is included as well as the bug-caused failures of these components.

Model-based approaches mostly differ from the presented example in the *type of model* adopted, the *configuration/architectural model* (e.g., single vs multiple-host), and the *attributes of interest* analyzed. Several of such works also propose rejuvenation techniques, which will be presented in Section 10.6.

In the following, a brief overview is given about the main model-based techniques proposed in the literature.

Zheng *et al.* [3] propose a *single-phase* version of the approach proposed by Chang *et al.* [2] to model again *survivability*. Silva *et al.* [4] evaluate the survivability of a cloud computing system by exploiting *Petri nets* rather than CTMCs.

Xu *et al.* [5] and Rezaei *et al.* [6] use *Stochastic Reward Nets* (SRNs) for *availability* modelling in a *single-server* virtualized system with rejuvenation applied at VMM level (by time-based policy), in combination with a measurement-based policy at VM level. SRNs for *availability* modeling are also used in the technique by Nguyen [1] which considers various failure and recovery modes of multiple VMs and VMMs, and in the one by Han and Xu [7] which considers three different rejuvenation policies (non-rejuvenation, time-based rejuvenation, and time and load-based delay rejuvenation) for single-server virtualization systems with multiple VMs on a single VMM. Similarly, Machida *et al.* [8, 9] use SRNs for cold-VM, warm-VM and VM-migration rejuvenation policies. A further example of *multiple-host multiple-VMs* availability analysis is presented by Myint and Thein [10], who designed a primary-standby server model, encompassing a load balancer VM in each node responsible for monitoring resources, and a rejuvenation agent installed on each VM. SRN are also used in the recent works by Escheikh [11, 12] where power management *performability* analysis is performed to evaluate the impact on performance and energy consumption in virtualized systems.

The work by Thein *et al.* [13] analyzes system *availability* with the time-based rejuvenation policy under different cluster configurations, 2 VMs hosted on a single physical server and 2 VMs per a physical server in dual physical servers. The same authors further present a software rejuvenation framework named VMSR to offer high *availability* for application server systems [14], proposing again a *CTMC* to model a *single-host, multiple-VM* system in the scheme with hot standby replicas.

Machida *et al.* [15, 16] present the analysis of job completion time under aging and rejuvenation of the VMM using a *semi-Markov process*. Okamura *et al.* [17] present the transient analysis of the two main rejuvenation policies, Cold- and Warm-VM rejuvenation, by means of *Markov Regenerative Stochastic Petri Nets* (MRSPNs). Zhao *et al.* [18] formulate the problem by using a game method, representing the different goals of a service provider (who wants to maximize availability) and a maintainer (who wants to minimize cost). *Markov Renewal Processes* (MRPs) are used to determine the optimal rejuvenation schedule and compute the steady-state *availability* and *maintenance cost*. Melo *et al.* [19] present an SPN model with cloud *availability* under two migration-based rejuvenation strategies (with and without a test before migration).

Finally, some researchers propose slightly different models for aging analysis: examples include the work by Melo *et al.* [20], who formulate an *availability* model considering live migration for VMM rejuvenation based on extended *Deterministic Stochastic Petri Nets* (DSPNs) *and Reliability Block Diagrams* (RBDs), and the one by Rahme *et al.* [21, 22], who use *Dynamic Fault Trees* (DFTs) to model cloud-based software rejuvenation.

10.4. Measurement-Based Analysis Techniques

On the other side of the spectrum there are measurement-based approaches, which monitor and analyze the values of aging indicators through probes at the system level related to system resources exhaustion (e.g., memory, storage), as well as at the user level (e.g., response time, latency, throughput, violations of Service Level Agreements (SLAs)). In a virtualized setting, measurement-based techniques are characterized by the following features:

- Indicators are measured at any layer of the virtualization technology stack. Specifically, monitoring can be done at application or OS level within the VM layer, at the VM layer to probe the state and resource consumption of the VMs (e.g., for load balancing, scaling, VM migration and rejuvenation decisions), as well as at the virtualization technology layer, most often referring to the VMM component. Besides system resources (CPU, memory, storage, network), other indicators of interest are related to VMs, for instance, the number of VM new allocations/releases, the time to start/stop VMs or the time to migration. These are some of the parameters of interest to the rejuvenation technique cost assessment. These metrics are collected during execution time, and can be used to forecast future trends. Other tradeoffs involved in making a rejuvenation decision are load balancing and resource allocation. These can be static or dynamic.
- Data gathered are analyzed by one or more of the following techniques:
  - *Time Series Analysis*. It is based on trend detection and an estimation of a set of aging indicators. Tests for trend detection are used to accept/reject the hypothesis of no trend in data (e.g., Mann-Kendall, t-student, Seasonal Kendall tests). Trend estimation can exploit many models, e.g., multiple linear regression, regression smoothing, Sen's slope estimate procedure, autoregressive models, non-linear models. In

the common case of the presence of correlation among multiple aging indicators, data transformation, feature selection, or dimensionality reduction techniques are used; an example is the Principal Component Analysis (PCA) followed by regression.

A time-series analysis technique in virtualized systems is the one by Araujo *et al.* [23], who use the *linear*, *quadratic*, *exponential growth*, and the *Pearl-Reed logistic* models in order to schedule software rejuvenation properly. These models have been adopted to predict memory consumption trends on the Eucalyptus cloud computing framework.

Umesh *et al.* [24] also exploit time series models to identify (and forecast) software aging patterns of the Windows active directory service.

DeCelles *et al.* [25] apply an anomaly detection technique based on *principal component analysis* (PCA) aimed at incipient faults such as software aging. Using case studies involving long-running enterprise applications, Trade6 and RuBBoS, with injected memory leaks, performance of the PCA-based detector when using just the compressed data is almost equivalent to the case in which the raw data is completely available, but with fewer samples with a compression rate over 75%.

Cotroneo *et al.* [26] define a stress test methodology applied to the HotSpot JVM, based on Design of Experiment (DoE) for a workload-based analysis, PCA to remove first-order correlation among aging indicators at JVM level, *clustering* to identify workload states, and then *multiple regression* to relate JVM aging indicators (transformed in the PC space) to the OS memory depletion and user-perceived throughput loss.

Mohan *et al.* [27] study the effect of aging on power usage by using *linear regression* to estimate the trend. Energy consumption is also considered by Villalobos [28], who presents an IDS-based self-protection mechanism at the virtual machine level inspired by software rejuvenation concepts. A correlation between IDS accuracy, attack rate, cloud system workload, energy consumption, and response time is identified.

— *Machine Learning.* Machine learning adopts algorithms from the field of artificial intelligence (e.g., classifiers and regressors) to identify trends and classify a system state as robust or failure-prone.

One of the first works in this area is by Alonso *et al.* [29]. They analyze a three-tiered J2EE system; a machine learning approach is used to automatically build regression tree models that relate several system variables (such as number of connections and throughput) to aging trends, based on the observation that such trends can be approximated using a piecewise linear model. The models are trained using data samples collected in preliminary experiments, and then used to predict the Time To Exhaustion (TTE) of system resources under conditions different than the ones observed during the training phase.

Other techniques use classifiers, like naive Bayes classifiers, decision trees, and Artificial Neural Networks (ANN). In particular, Sudhakar *et al.* [30] use ANN to capture non-linear relationships between resource usage statistics and the time to failure in cloud systems, to generate a prediction of the time to failure.

The works by Avresky *et al.* [31] and Di Sanzo *et al.* [32] present a framework using machine learning models to predict failures caused by the accumulation of anomalies and a proactive system scale up/scale down technique applied at client-server applications in the cloud. It predicts the remaining time to the occurrence of some unexpected event (system failure, service level agreement violation, etc.) of a VM hosting a server instance of the application. Machine learning is also used in the work by Simeonov *et al.* [33], which proposes a framework with three VMs, one VM master and two identical slaves, one active and the other in standby; the slaves send health data to the master, which predicts aging based on the machine learning algorithm running on such data.

— *Threshold-Based Approaches*. These approaches define thresholds for some aging indicators, so as to trigger rejuvenation when the monitored indicators exceed such thresholds. An examples of this approach is in the work by Silva *et al.* [34], which adopts thresholds on mean response time and on the quality of service indicators. The authors propose a rejuvenation approach based on self-healing techniques that exploits virtualization to optimize recovery. They propose a rejuvenation framework called *VM-Rejuv*, exploiting virtualization to optimize recovery, with an aging detector module detecting aging conditions based on the abovementioned thresholds.

The advantage of a measurement-based approach is that software aging forecasting can adapt to the current condition of the system, e.g., the current operational profile, which may not have been foreseen before operation, and which can accurately predict the occurrence of aging phenomena. However, this kind of approach may not be easily generalizable, since it exploits some peculiar aspect related to the nature of the considered system, e.g., the fact that some particular resource exhibits seasonal, fractal patterns, or the regularity of the phenomenon. Moreover, measurement-based approaches are not meant to estimate long-term dependability measures such as availability.

- Differently from model-based techniques, the analysis and validation of these techniques consider measurements from real systems, with the goal of identifying whether the system is in a failure-prone state due to software aging, in order to forecast the time-to-aging-failure and to plan software rejuvenation accordingly. The advantage of a measurement-based solution is the possibility of gathering accurate and detailed information about the aging state of the system. In addition, there is no need to make assumptions about model parameters, since real data is available. On the other hand, findings are very system-specific, and hard to generalize, even though the large number of systems being analyzed with these approaches is constantly increasing.

  In the context of virtualized environments, systems range from cloud applications (more often web/application servers running on the cloud) to entire platforms, the most studied one being Eucalyptus. Specifically, several authors [23, 35–38] analyze aging and propose rejuvenation in the Eucalyptus cloud computing infrastructure by using measurement-based approaches, e.g., time series forecasting. These are applied to common indicators about resource consumption, especially RAM memory and swap space exhaustion and CPU utilization by the VMs. Based on the time series analyses, a prediction-based rejuvenation is scheduled, so as to reduce the downtime by predicting the proper time to perform the rejuvenation.

  A quite different aging study on Eucalyptus is presented by Langner *et al.* [39] The authors propose a technique to detect aging problems shortly after their introduction by performing runtime comparisons of different development versions of the same software. In this case — as in the few other cases in the SAR literature where aging-related bugs are analyzed at development-time [40–42] — the detection aims at revealing aging problems before their long-term runtime manifestation.

- Besides studies on cloud computing, a relevant environment to which measurement-based methods have been applied is represented by the **Java Virtual Machine** (JVM) and its related applications (i.e., applications based on J2EE). In this case, the parameters of interest at the VM level are about the state of the JVM, which is captured by parameters like heap usage, instantiated classes, number of threads, object size, number of allocations, number of garbage collections, number of JIT compilations, etc. Several examples of measurement-based approaches in the JVM present techniques for **leak detection**.

Haining *et al.* [43], and then Meng *et al.* [44], studied the performance of the JVM running a J2EE Application Server, looking for memory leaks associated with the garbage collection mechanism in the JVM. They use the Virtual Machine Profiler Interface (JVMPI), to collect resource usage data from the application server, including data on the JVM heap memory usage and CPU utilization, as well as the response time and throughput at client level.

In the studies by Šor *et al.* [45, 46], memory leak detection in Java applications is performed through a statistical approach based on the "age" of objects, measured as the number of garbage collection cycles (or generations) they survive. The idea is to analyze how live instances of the class are distributed over different generations, and evaluate if objects follow the "generational hypothesis", i.e., most of them become unreachable very soon after creation (in other words, they "die" young). The growth of the number of generations where class instances are present is a symptom that the application allocates some objects without then freeing them (memory leak).

Memory leaks are also analyzed by Xu *et al.* [47], who present an assertion-based technique for leak detection within the Jikes RVM, a high-performance Java virtual machine, and then applied it to real leaks in large-scale applications like Eclipse and MySQL. Xu *et al.* [48] have proposed the Self-organizing Maps (SOMs) to capture VMM behaviors from runtime measurement data. In their work, they have used the neighborhood area density of a winning neuron as an aging quantification metric. The results of two experiments injecting different resource leaks on the Xen platform show that the algorithm has a high true positive rate and a low false positive rate. Sor *et al.* [49] use machine learning for memory leak detection implemented in a commercial tool called *Plumbr*.

An additional JVM study, Cotroneo *et al.* [26], investigates throughput loss and memory consumption in the JVM Hotspot, via a James' mail server workload. Several JVM parameters are monitored through *JVMMon*, a monitoring tool developed for the purpose of aging analysis. Correlation analysis is used to identify aging JVM components. An interesting finding is that the garbage collector, which was designed to limit aging trends in Java applications, was found to suffer from aging [26].

10.5. Hybrid Analysis Techniques

An important generalization of the previous two methods for aging analysis is what are called **hybrid** approaches, proposed by some researchers as a combination of model-based and measurement-based solutions. Hybrid solutions usually adopt a stochastic model to describe the phenomenon, and determine the model parameters through measurement, that is, via observed data. Solutions of this type are not yet widely deployed, but have the potential of taking advantage of the best features found in model-based and measurement-based approaches. A typical example is the one proposed by Trivedi and Vaidyanathan [50] in which: *i)* a measurement-based semi-Markovian model for a system workload is built; *ii)* the TTE for each considered resource and state (using reward functions) is estimated, and finally *iii)* a semi-Markov availability model is provided, based on field data rather than on assumptions about system behavior.

In the context of virtualized environments, a hybrid solution is adopted by Liu *et al.* [51], which measures the trends of various resources in a cloud-based streaming system with ATM endpoints, including the CPU, storage, and network, at several layers, and uses the measured trends to parametrize a model to schedule service rejuvenation, implemented by means of the VM failover to replicas. The end result is an improvement in terms of served transactions per second.

A further hybrid strategy is proposed by Machida *et al.* [52], who present an original countermeasure to software aging different from rejuvenation, called *software life extension* — a sort of workaround in which, upon software aging detection, additional memory is allocated to the VM executing the aging-affected software. In this case, experimental data on *memcached* are used to derive a more general model using a semi-Markov process formalism.

The solution proposed by Kadirvel *et al.* [53] is applied to a system manager for a batch-based job submission system on a virtualized platform, aimed at managing and controlling virtualized resources to support remediation approaches, such as the elastic increase and decrease of resource capacity, VM migrations, and dynamic resource configuration changes. The technique combines a Petri Net-based approach to model a system manager module, suffering from health deterioration due to resource exhaustion, with the usage of feedback control theory to control resource consumption and to delay/prevent resources. Three different rejuvenation strategies are implemented and tested — process rejuvenation, VM migration and dynamic increase of resource allocation — chosen based on the planning module.

In all the mentioned classes of studies, an additional analysis factor is workload dependency. Since aging has been shown to be correlated with workload variation [54, 55] several studies accounted for its impact; two examples are the following.

Brueno *et al.* [56] present a workload-based analysis of VMM aging and rejuvenation under different policies for availability maximization, under *variable* workload conditions. They exploit dynamic reliability theory and symbolic algebraic techniques, representing the CDFs associated with the VMM events, under different workload conditions, by continuous phase type (CPH) distributions, and using the Kronecker algebra to implement the conservation of the reliability principle and the variable timer policy.

Ficco *et al.* [57] perform a workload-dependent analysis of performance degradation and memory indicators in Apache Storm, an event stream processing (ESP) application that can deploy tasks over a cloud architecture, by means of a workload-dependent time series analysis. The Mann-Kendall test and Sen's procedure, often used for aging trend detection and estimation, are used on sliced windows where the workload and the performance/memory trends are judged to be in contrast (e.g., their trends increase despite the workload decreasing).

10.6. Rejuvenation Techniques

In virtualized environments, several further alternatives are explored besides the conventional rejuvenation techniques, e.g., application/component, OS/node reboot, cluster failover. The additional layers (i.e., the VMM and VM layers) give the opportunity to set up various scenarios (e.g., multiple hosts with multiple VMs) and this, in turn, enables the development of new rejuvenation strategies exploiting, for instance, VM restarts, migration and failover, to guarantee high levels of service continuity in the presence of aging problems. Figure 10.3 outlines the common rejuvenation techniques at VMM and at VM level. Software rejuvenation can act on a virtual machine infrastructure by rejuvenating the VMM and/or its VMs. VMM rejuvenation

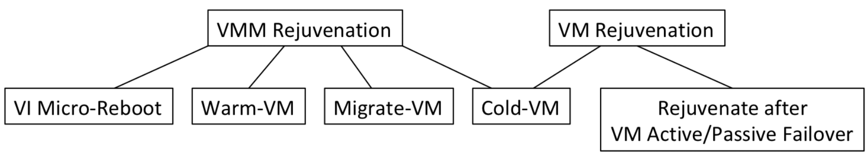

Fig. 10.3. Rejuvenation techniques.

can be performed by restarting the VMM or part thereof, and alternative strategies depend on whether rejuvenation affects only the VMM, or also the VMs running on top of the VMM [8, 9]. As for VM rejuvenation, the strategies differ in how the rejuvenation process is managed. They are explained in detail hereafter.

VMM rejuvenation can be performed by the following techniques:

- **Cold-VM rejuvenation**, in which the VMs are also restarted when the VMM is rejuvenated. Cold-VM rejuvenation simply shuts down the hosted VMs before triggering the VMM rejuvenation and restarts the VMs after the completion of VMM rejuvenation.
- **Warm-VM rejuvenation**, in which the execution state of each VM, including the OS and applications running in the VM, are stored to persistent memory, and resumed after the restart of the VMM, in order to reduce the downtime of restarting VMs and their services. In this type of rejuvenation, the software running in the VMs is not rejuvenated. This operation can be quickly performed by adopting an on-memory suspend/resume mechanism, in which the memory images of VMs are preserved in main memory during the VMM restart rather than on persistent storage, in order to avoid slow read/write operations to persistent storage. Kourai *et al.* [58, 59] propose such a variant, and show that compared with Cold-VM rejuvenation, a Warm-VM reboot improves the availability of the applications hosted on the VMs by introducing the on-memory suspend technique and the quick reload mechanism.
- **Migrate-VM rejuvenation** is a type of rejuvenation in which the downtime is further reduced by migrating a VM to another host while the VMM is being rejuvenated, in order to make it available during rejuvenation. This latter scheme does not rejuvenate VMs, and is limited by the capacity of other hosts to accept migrated VMs. Migration can further be categorized based on the type of live VM migration (*stop-and-copy* or *pre-copy*), and the type of policy with regard to returning back to the original host after VMM rejuvenation (*return-back* or *stay-on*) [9]. Specifically, the *stop-and-copy* migration stops the VM operation and copies all the memory contents to the destination server. In the *pre-copy* variant, the VM's memory is first copied to a destination server without stopping its operation, causing dirty pages in the copied memory contents; then, a stop-and-copy phase is performed, which, however, updates only the dirty pages, so considerably reducing the downtime overhead compared to that which would result from a complete stop-and-copy of the entire

VM's memory. As for the restore of the VM on the original host, a *return-back* policy foresees the migration to the original host soon after the VMM rejuvenation is completed; a *stay-on* policy allows the migrated VM to run on the other hosting server even after the completion of VMM rejuvenation on the original host.

The work by Machida *et al.* [9] uses SRNs for cold-VM rejuvenation, warm-VM rejuvenation and migration. They studied the steady-state availability of the VMs and the expected number of transactions lost, finding that Migrate-VM rejuvenation generally achieves higher steady-state availability compared to Cold- and Warm-VM rejuvenation, because of the ability to preserve the VM execution even during VMM rejuvenation.

Regarding the types of VM migration, they also found that the pre-copy approach generally achieves higher availability than the stop-and-copy approach because it minimizes the downtime caused by VM migration thanks to the pre-copy phase. For the migration back policy, the return-back policy is generally more effective than the stay-on policy because a host that just finished VMM rejuvenation is more reliable than a host that has been running for some time after the last VMM rejuvenation, i.e., the availability of the VMs running on the migrated host is affected by the uptime of the host from the last restart.

Machida *et al.* [60, 61] propose also the contemporary rejuvenation of aged VMs and the VMM in virtualized data centers; differently from the usual Cold-VM rejuvenation, this technique forces shutdown only to aged VMs which are to be rejuvenated in the near future, while the robust VMs are moved out from the host server by live VM migration before VMM rejuvenation.

Kourai *et al.* [62] present *VMBeam*, a technique that enables lightweight software rejuvenation of virtualized systems using zero-copy migration. The technique starts a new virtualized system at the same host by using nested virtualization, and then migrates all the VMs from the aged virtualized system to the clean one. VMBeam relocates the memory of the VMs on the aged virtualized system directly to the clean system, without any copy.

Melo *et al.* [19] use an SPN model under two migration-based rejuvenation strategies (with and without a test before migration).

The approach proposed by Torquato *et al.* [63] discussed in the previous section foresees a rejuvenation phase performed by means of VM Live Migration, applied on a Cloud Computing testbed which uses OpenNebula and KVM as the VMMs.

- **VI Micro reboot**. A more complex technique is to perform a very fine-grained reboot of software modules (a micro reboot), tailored for the system-level virtualization software (i.e., for the virtualization infrastructure — VI). The technique has been proposed by Le *et al.* [64], who applied a micro reboot to all the modules of the Xen virtualization software; this consists of three main components: the privileged virtual machine, the device driver virtual machine, and the virtual machine monitor. This is an instance of the more general case of micro reboot: Candea *et al.* applied a micro reboot to Java-based virtualized environments [65, 66]. The technique requires "*microrebootable*" software, in which a system can be decomposed into components that are loosely coupled, e.g., they do not share the same memory address space, and are stateless, e.g., important states are located in dedicated storage outside the application, in order to quickly restart a component without affecting the other ones — an approach suitable for modern service-based systems. The approach has been successfully implemented in a J2EE application [66] (in which individual Enterprise Java Beans can be restarted) as well as in a Java-based mission-critical system [65], consisting of a set of components running in different Java Virtual Machines. A limitation of component rejuvenation is that it can be applied only in systems made up of individually restartable components; if this is not the case, the system has to be modified based on the microreboot schema.

The best technique, or the best combination of techniques, depends on the speed of storing/migrating the state of the VMs and on the capacity of hosts, as well as on the aging rate of VMs and VMMs, therefore the rejuvenation policy should be determined according to these factors.

In order to rejuvenate VMs themselves, besides the mentioned Cold-VM rejuvenation that indirectly also rejuvenates the VM, a conventional replication approach can be applied:

- **VM Failover**: *VM Failover* techniques are based on conventional active/passive replication. The idea is to redirect, upon detection of aging in a VM, all the incoming requests to another VM, and then rejuvenate the aged VM (e.g., by restart). A VM can be rejuvenated while the other replicas are active and the workload is redirected to them, or by activating a standby (i.e., idle) replica when rejuvenation is triggered.
  For example, Silva *et al.* [34] propose a cluster failover framework for web applications based on virtualization, namely *VM-Rejuv*. The framework consists of a Load Balancer, an Active Server, and a Standby Server,

each running in a dedicated VM. The Load Balancer redirects requests to the Active Server while it is correctly working, and monitors the Active Server for aging symptoms (e.g., performance falls below a threshold). When rejuvenation is triggered, new requests are redirected to the Standby Server; the Active Server is rejuvenated only after all pending requests have been processed and session data have been migrated to the Standby Server, in order to ensure a clean restart, i.e., in order that rejuvenation does not cause the loss of session data and the failure of user requests. This framework can be implemented in a cost-effective way by using off-the-shelf application servers, monitoring, and load balancing software, at the cost of a moderate overhead.

In a virtualized environment, the management of the state transfer can also be done by exploiting the hypervisor. For instance, the work by Distler *et al.* [67] addresses the problem of stateful replica management by proposing an architecture where recovery is allowed in parallel with service execution, and which uses copy-on-write techniques for state transfer between virtual replicas of a host. Similarly, Reiser *et al.* [68] initially proposed to use hypervisor to initialize a new replica in parallel to normal system execution, and thus minimizes the time in which a proactive reboot interferes with system operation.

It is finally worth noting that the advantages of using VMs can be counterbalanced by a higher memory fragmentation induced by the virtualized environment. Alonso *et al.* [69] experimentally study the main software rejuvenation techniques, considering (i) physical node reboot, (ii) virtual machine reboot, (iii) OS reboot, (iv) fast OS reboot, (v) standalone application restart, and (vi) application rejuvenation by a hot standby server, and blame virtualization for the drastic increase in memory fragmentation, which can cause aging-related failures in long-running systems. This stresses the importance of choosing the right set of techniques to apply rejuvenation in a virtualized environment.

10.7. Conclusions

SAR techniques in virtualized environments have increasingly emerged in the last few years. More than 55% of research studies in this area were published in the last 5 years, hence it is a relatively young research area. These research studies have appeared mainly after 2007, after the widespread adoption of this kind of system. In such studies, researchers often analyze several rejuvenation policies based on virtual machine and/or virtual machine monitor reboot/rejuvenation. Such strategies are then evaluated both by model-based approaches and by measurements. The availability of cloud computing software, which can be used to experiment with such strategies without excessive costs, is favoring aging analysis in the cloud. A relevant example is represented by the studies on the Eucalyptus cloud-computing framework. From the model-based perspective, models differ from each other in the formalism adopted (e.g., CTMC, SPN, SRN, MRGP, MRP), in the configuration/architectural model (e.g., single vs multiple-host), in the attributes of interest analyzed (survivability, availability, performability), and in the rejuvenation schemes (e.g., Cold-VM, Warm-VM, Migrate-VM, VM failover). Measurement-based approaches differ from the traditional SAR approach only in the indicators being adopted: as mentioned, the *execution environment* concept is quite different from non-virtualized systems, but the techniques used (mostly, time series analysis and machine learning) are unchanged with respect to conventional SAR literature. A relevant difference is that in most of the works on SAR in the virtualized environment, researchers associate one or more rejuvenation actions to the aging evaluation task, resulting in a literature review with many more works including rejuvenation strategies with respect to the general SAR literature. Looking at the increasing trends of virtualization and cloud computing, it is easy to envision a proportionally increasing trend in the studies dealing with aging and rejuvenation in virtualized environments in the next few years. Moreover, because of the increased economical impact of availability/performability/survivability and thanks to the possibilities offered by emerging cloud-based architectures, we will likely be witness to an increase in the number of deployments of virtualized rejuvenation solutions, in each of the several layers of the virtualized environment.

References

1. T. A. Nguyen, D. S. Kim, and J. S. Park. (2014). A comprehensive availability modeling and analysis of a virtualized servers system using stochastic reward nets. *The Scientific World Journal*, **2014**, Article 165316.
2. X. Chang, Z. Zhang, X. Li, and K. S. Trivedi. (2016). Model-based survivability analysis of a virtualized system. In *Proceedings of the IEEE 41st Conference on Local Computer Networks (LCN), Dubai, UAE, November 7–10*, pp. 611–614.
3. J. Zheng, H. Okamura, and T. Dohi. (2015). Survivability analysis of VM-based intrusion tolerant systems. *IEICE Transactions on Information and Systems*, **E98.D**(12), 2082–2090.

4. B. Silva, P. R. M. Maciel, A. Zimmermann, and J. Brilhante. (2014). Survivability evaluation of disaster tolerant cloud computing systems. In *Proceedings of the Probabilistic Safety Assessment & Management Conference (PSAM 12), Honolulu, Hawaii, USA, June 22–27*, p. 453.
5. J. Xu, X. Li, Y. Zhong, and H. Zhang. (2014). Availability modeling and analysis of a single-server virtualized system with rejuvenation. *Journal of Software*, **9**(1), 129–139.
6. A. Rezaei and M. Sharifi. (2010). Rejuvenating high available virtualized systems. In *Proceedings of the 5th International Conference on Availability, Reliability, and Security (ARES), Krakow, Poland, February 15–18*, pp. 289–294.
7. L. Han and J. Xu. (2013). Availability models for virtualized systems with rejuvenation. *Journal of Computational Information Systems*, **9**(20), 8389–8396.
8. F. Machida, D. S. Kim, and K. S. Trivedi. (2010). Modeling and analysis of software rejuvenation in a server virtualized system. In *Proceedings of the Second International Workshop on Software Aging and Rejuvenation (WoSAR), San Jose, California, USA, November 2*, pp. 1–6.
9. F. Machida, D. S. Kim, and K. S. Trivedi. (2013). Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration. *Performance Evaluation*, **70**(3), 212–230.
10. M. Myint and T. Thein. (2010). Availability improvement in virtualized multiple servers with software rejuvenation and virtualization. In *Proceedings of the Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI), Singapore, June 9–11*, pp. 156–162.
11. M. Escheikh, Z. Tayachi, and K. Barkaoui. (2016). Workload-dependent software aging impact on performance and energy consumption in server virtualized systems. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Ottawa, Ontario, October 23–27*, pp. 111–118.
12. M. Escheikh, K. Barkaoui, and H. Jouini. (2017). Versatile workload-aware power management performability analysis of server virtualized systems. *Journal of Systems and Software*, **125**, 365–379.
13. T. Thein, S. Chi, and J. S. Park. (2008). Availability modeling and analysis on virtualized clustering with rejuvenation. *International Journal of Computer Science and Network Security*, **8**(9), 72–80.
14. T. Thein and J. S. Park. (2009). Availability analysis of application servers using software rejuvenation and virtualization. *Journal of Computer Science and Technology*, **24**(2), 339–346.
15. F. Machida, V. F. Nicola, and K. S. Trivedi. (2011). Job completion time on a virtualized server subject to software aging and rejuvenation. In *Proceedings of the Third International Workshop on Software Aging and Rejuvenation (WoSAR), Hiroshima, Japan, November 29–December 2*, pp. 44–49.
16. F. Machida, V. F. Nicola, and K. S. Trivedi. (2014). Job completion time on a virtualized server with software rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems*, **10**(1), 10:1–10:26.
17. H. Okamura, K. Yamamoto, and T. Dohi. (2014). Transient analysis of software rejuvenation policies in virtualized system: Phase-type expansion approach. *Quality Technology & Quantitative Management*, **11**(3), 335–351.
18. J. Zhao, Y.-B. Wang, G.-R. Ning, C.-H. Wang, K. S. Trivedi, K.-Y. Cai, and Z.-Y. Zhang. (2014). Software maintenance optimization based on Stackelberg game methods. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Naples, Italy, November 3–6*, pp. 426–430.

19. M. Melo, J. Araujo, R. Matos, J. Menezes, and P. Maciel. (2013). 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, October 13–16*, pp. 4110–4115.
20. M. Melo, P. Maciel, J. Araujo, R. Matos, and C. Araujo. (2013). Availability study on cloud computing environments: Live migration as a rejuvenation mechanism. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24–27*, pp. 1–6.
21. J. Rahme and H. Xu. (2015). A software reliability model for cloud-based software rejuvenation using dynamic fault trees. *International Journal of Software Engineering and Knowledge Engineering*, **25**(09n10), 1491–1513.
22. J. Rahme and H. Xu. (2015). Reliability-based software rejuvenation scheduling for cloud-based systems. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE), Pittsburgh, Philadelphia, USA, July 6–8*, pp. 298–303.
23. J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. Trivedi. (2011). Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In *Proceedings of the Third International Workshop on Software Aging and Rejuvenation (WoSAR), Hiroshima, Japan, November 29–December 2*, pp. 38–43.
24. I. M. Umesh and G. N. Srinivasan. (2017). Dynamic software aging detection-based fault-tolerant software rejuvenation model for virtualized environment. In Satapathy S. C., Bhateja V., and Joshi A. (eds.). *Proceedings of the International Conference on Data Engineering and Communication Technology*, Advances in Intelligent Systems and Computing, vol. 469. Singapore: Springer.
25. S. DeCelles, T. Huang, M. C. Stamm, and N. Kandasamy. (2016). Detecting incipient faults in software systems: A compressed sampling-based approach. In *Proceedings of the 9th IEEE International Conference on Cloud Computing (CLOUD), San Francisco, California, USA, June 27–July 2*, pp. 303–310.
26. D. Cotroneo, S. Orlando, R. Pietrantuono, and S. Russo. (2013). A measurement-based ageing analysis of the JVM. *Software Testing, Verification and Reliability*, **23**(3), 199–239.
27. B. R. Mohan and G. R. M. Reddy. (2015). The effect of software aging on power usage. In *Proceedings of the 9th International Conference on Intelligent Systems and Control (ISCO), Tamil Nadu, India, January 9–10*, pp. 1–3.
28. J. J. Villalobos, I. Rodero, and M. Parashar. (2014). Energy-aware autonomic framework for cloud protection and self-healing. In *Proceedings of the International Conference on Cloud and Autonomic Computing (ICCAC), London, UK, September 8–12*, pp. 3–4.
29. J. Alonso, L. Belanche, and D. Avresky. (2011). Predicting software anomalies using machine learning techniques. In *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, Massachusetts, USA, July 15–17*, pp. 163–170.
30. C. Sudhakar, I. Shah, and T. Ramesh. (2014). Software rejuvenation in cloud systems using neural networks. In *Proceedings of the International Conference on Parallel, Distributed and Grid Computing (PDGC), Himachal Pradesh, India, December 11–13*, pp. 230–233.

31. D. R. Avresky, P. D. Sanzo, A. Pellegrini, B. Ciciani, and L. Forte. (2015). Proactive scalability and management of resources in hybrid clouds via machine learning. In *Proceedings of the 14th International Symposium on Network Computing and Applications (NCA), Cambridge, Massachusetts, USA, October 31–November 2*, pp. 114–119.
32. P. D. Sanzo, A. Pellegrini, and D. R. Avresky. (2015). Machine learning for achieving self-* properties and seamless execution of applications in the cloud. In *Proceedings of the 2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA), Munich, Germany, June 11–12*, pp. 51–58.
33. D. Simeonov and D. R. Avresky. (2010). Proactive software rejuvenation based on machine learning techniques. In Avresky D. R. *et al.* (eds.). *Cloud Computing*. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, vol. 34. Berlin/Heidelberg, Germany: Springer.
34. L. Silva, J. Alonso, and J. Torres. (2009). Using virtualization to improve software rejuvenation. *IEEE Transactions on Computers*, **58**(11), 1525–1538.
35. J. Araujo, R. Matos, V. Alves, P. Maciel, F. V. de Souza, R. Matias Jr., and K. S. Trivedi. (2014). Software aging in the eucalyptus cloud computing infrastructure: Characterization and rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems*, **10**(1), 11:1–11:22.
36. R. Matos, J. Araujo, V. Alves, and P. Maciel. (2012). Experimental evaluation of software aging effects in the eucalyptus elastic block storage. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, Korea, October 14–17*, pp. 1103–1108.
37. J. Araujo, R. Matos, P. Maciel, and R. Matias. (2011). Software aging issues on the eucalyptus cloud computing infrastructure. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), Anchorage, Alaska, USA, October 9–12*, pp. 1411–1416.
38. J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker. (2011). Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In *Proceedings of the Middleware 2011 Industry Track Workshop, Lisbon, Portugal, December 12–16*.
39. F. Langner and A. Andrzejak. (2013). Detecting software aging in a cloud computing framework by comparing development versions. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Ghent, Belgium, May 27–31*, pp. 896–899.
40. D. Cotroneo, R. Natella, and R. Pietrantuono. (2012). Predicting aging-related bugs using software complexity metrics. *Performance Evaluation*, **70**(3), 163–178.
41. D. Cotroneo, R. Natella, and R. Pietrantuono. (2010). Is software aging related to software metrics? In *Proceedings of the Second International Workshop on Software Aging and Rejuvenation (WoSAR), San Jose, California, USA, November 2*, pp. 1–6.
42. D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi. (2016). How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. *Journal of Systems and Software*, **113**, 27–43.
43. M. Haining, H. Xinhong, L. Jianjun, and W. Wei. (2013). Detection and analysis of software aging in a service-oriented J2EE application server. *Information Technology Journal*, **12**(9), 1857–1862.
44. H. Meng, X. Hei, J. Zhang, J. Liu, and L. Sui. (2016). Software aging and rejuvenation in a J2EE application server. *Quality and Reliability Engineering International*, **32**(1), 89–97.

45. V. Šor, N. Salnikov-Tarnovski, and S. N. Srirama. (2011). Automated statistical approach for memory leak detection: Case studies. In Meersman R. *et al.* (eds.). *On the Move to Meaningful Internet Systems: OTM 2011*. Lecture Notes in Computer Science, vol. 7045. Berlin/Heidelberg, Germany: Springer.
46. V. Sor and S. N. Srirama. (2011). A statistical approach for identifying memory leaks in cloud applications. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2011), Noordwijkerhout, The Netherlands, May 7–9*, pp. 623–628.
47. G. Xu, M. Bond, F. Qin, and A. Rountev. (2011). Leakchaser: Helping programmers narrow down causes of memory leaks. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), San Jose, California, USA, June 4–8*, pp. 270–282.
48. J. Xu, W.-W. Wu, and C.-Y. Ma. (2014). SOM-based aging detection for Virtual Machine Monitor. In *Proceedings of the IEEE Workshop on Electronics, Computer and Applications, Ottawa, Ontario, Canada, May 8–9*, pp. 782–785.
49. V. Sor, P. Oü, T. Treier, and S. N. Srirama. (2013). Improving statistical approach for memory leak detection using machine learning. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM), Eindhoven, The Netherlands, September 22–28*, pp. 544–547.
50. K. Vaidyanathan and K. Trivedi. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, **2**(2), 124–137.
51. Y. Liu, W. Liu, J. Song, and H. He. (2015). An empirical study on implementing highly reliable stream computing systems with private cloud. *Ad Hoc Networks*, **35**(C), 37–50.
52. F. Machida, J. Xiang, K. Tadano, and Y. Maeno. (2012). Software Life-Extension: A New Countermeasure to Software Aging. In *Proceedings of the 23rd International Symposium on Software Reliability Engineering, Dallas, Texas, USA, November 27–30*, pp. 131–140.
53. S. Kadirvel and J. A. B. Fortes. (2010). Self-caring IT systems: A proof-of-concept implementation in virtualized environments. In *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), Indianapolis, Indiana, USA, November 30–December 3*, pp. 433–440.
54. A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo. (2011). Workload characterization for software aging analysis. In *Proceedings of the 22nd International Symposium on Software Reliability Engineering (ISSRE), Hiroshima, Japan, November 29–December 2*, pp. 240–249.
55. A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo. (2012). On the aging effects due to concurrency bugs: A case study on MySQL. In *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering, Dallas, Texas, USA, November 27–30*, pp. 211–220.
56. D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa. (2013). Workload-based software rejuvenation in cloud systems. *IEEE Transactions on Computers*, **62**(6), 1072–1085.
57. M. Ficco, R. Pietrantuono, and S. Russo. (2018). Aging-related performance anomalies in the Apache Storm stream processing system. *Future Generation Computer Systems*, **86**, 975–994.
58. K. Kourai and S. Chiba. (2007). A fast rejuvenation technique for server consolidation with virtual machines. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Edinburgh, UK, June 25–28*, pp. 245–255.

59. K. Kourai and S. Chiba. (2011). Fast software rejuvenation of virtual machine monitors. *IEEE Transactions on Dependable and Secure Computing*, **8**(6), 839–851.
60. F. Machida, J. Xiang, K. Tadano, and Y. Maeno. (2012). Combined server rejuvenation in a virtualized data center. In *Proceedings of the 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), Fukuoka, Japan, September 4–7*, pp. 486–493.
61. F. Machida, D. S. Kim, J. S. Park, and K. Trivedi. (2008). Toward optimal virtual machine placement and rejuvenation scheduling in a virtualized data center. In *Proceedings of the IEEE International Conference on Software Reliability Engineering Workshops, Seattle, Washington, USA, November 11–14*, pp. 1–3.
62. K. Kourai and H. Ooba. (2015). Zero-copy migration for lightweight software rejuvenation of virtualized systems. In *Proceedings of the 6th Asia-Pacific Workshop on Systems (APSys), Tokyo, Japan, July 27–28*, pp. 7:1–7:8.
63. M. Torquato, P. Maciel, J. Araujo, and I. M. Umesh. (2017). An approach to investigate aging symptoms and rejuvenation effectiveness on software systems. In *Proceedings of the 12th Iberian Conference on Information Systems and Technologies (CISTI), Lisbon, Portugal, June 14–17*, pp. 1–6.
64. M. Le and Y. Tamir. (2012). Applying microreboot to system software. In *Proceedings of the IEEE Sixth International Conference on Software Security and Reliability (SERE), Gaithersburg, Maryland, USA, June 20–22*, pp. 11–20.
65. G. Candea, J. Cutler, A. Fox, R. Doshi, P. Garg, and R. Gowda. (2002). Reducing recovery time in a small recursively restartable system. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks, Bethesda, Maryland, USA, June 23–26*, pp. 605–614. doi: 10.1109/DSN.2002.1029006.
66. G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. (2004). Microreboot — a technique for cheap recovery. In *Proceedings of the Symposium on Operating Systems Design & Implementation, San Francisco, California, USA, December 6–8*, pp. 31–44.
67. T. Distler, R. Kapitza, and H. P. Reiser. (2008). Efficient state transfer for hypervisor-based proactive recovery. In *Proceedings of the 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS), Glasgow, UK, April 1*, pp. 4:1–4:6.
68. H. P. Reiser and R. Kapitza. (2007). Hypervisor-based efficient proactive recovery. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS), Beijing, China, October 10–12*, pp. 83–92.
69. J. Alonso, R. Matias, E. Vicente, A. Maria, and K. Trivedi. (2013). A comparative experimental study of software rejuvenation overhead. *Performance Evaluation*, **70**(3), 231–250.