SWARE: An approach to support software aging and rejuvenation experiments

Matheus D'Eça Torquato de Melo
Federal Institute of Alagoas
Campus Arapiraca, Brazil
matheustor4.professor@gmail.com

I M Umesh
Bharathiar University, Coimbatore, India
umesh.mphil@rvce.edu.in

Jean Araujo
Federal Rural University of Pernambuco (UFRPE)
Campus Garanhuns, Brazil
jean.teixeira@ufrpe.br

Paulo Maciel
Center of Informatics, Federal University of Pernambuco, Brazil
prmm@cin.ufpe.br

Abstract—The need for uninterrupted computing services demands for high system availability and reliability. Techniques and methods to estimate and analyze system dependability are essential to support software deployment and maintenance. Soft- ware aging appears as a relevant issue in this context. Software aging is a cumulative process which leads long-running systems to hangs or failures. Software rejuvenation is used to counteract software aging. Software rejuvenation usually comprises system reboot or application restart to bringing software to a stable fresh state. This paper proposes an approach to investigate software aging effects and software rejuvenation effectiveness on a single experiment. The approach has three phases: (i) Stress Phase - stress environment with the accelerated workload to induce bugs activation; (ii) Wait Phase - stop workload submission to observe the system state after workload submission; (iii) Rejuvenation Phase - find the impacts caused by the software rejuvenation. We named our approach as SWARE (Stress-Wait- Rejuvenation). To validate the SWARE approach, we present a case study. This case study consists of an experiment of VM Live Migration as rejuvenation mechanism for VMM software aging. The considered testbed is a Private Cloud with OpenNebula and KVM 1.0. The obtained results show that VM live migration is useful as rejuvenation for VMM software aging.

Index Terms—Software Aging and Rejuvenation, Reliability, Dependability, Availability, Cloud Computing

I. INTRODUCTION

High availability and reliability mandatory requirements of a myriad of software systems. In Cloud Computing, for example, software reliability and availability stays as top concerns for customers [1] [2] [3]. Therefore, methods and techniques to improve system reliability are of utmost importance.

Previous works introduce software aging as a pertinent issue in the software reliability area [4] [5] [6] [7] [8]. Software aging phenomenon consists in a gradual increase in software failure rate or performance degradation during its execution [9]. These effects usually happen because of errors accumulation in software state. This accumulation can lead software to hangs and total failures [10]. Systems with long-time of execution may suffer from software aging effects [11]. Published works presented software aging indicators on software as Android OS [12] [13], Stream Video Player [14], Software-Defined Networking Controllers [15], Cloud Com- puting Open-Source Software [16], among others. Especially on Cloud Computing and Virtualized Server Environments, the VMM (Virtual Machine Monitor) is liable to suffer software aging, as presented in [6] [7].

Software rejuvenation is the countermeasure to software aging. Software rejuvenation consists of a proactive technique to clean software aging effects by rolling it back to a stable status. Software rejuvenation lies on an application restart or a system reboot [17]. Previous works propose a schedule to submit software rejuvenation actions [18] [19] to minimize system downtime caused by these operations. More details of software aging and rejuvenation are in Section II.

Usually, it is hard to define software aging causes. Errors, memory leaks and other aging-related symptoms are non-expected events [7]. A proper approach to deal with software aging and rejuvenation issues is to study them in an isolated environment running experiments and tests to understand software aging symptoms and rejuvenation effectiveness.

This paper presents an approach to investigate aging symptoms and rejuvenation effectiveness on software systems. The approach is named SWARE (Stress-**WA**it-**RE**juvenation). The SWARE approach has three phases. (I) **Stress Phase**, which aims to observe impacts of workload exposure in software internal state. (II) **Wait Phase** which observes software behavior after workload submission. (III) **Rejuvenation Phase**, with the goal of perceive consequences of software rejuvenation action [20] [7]. The phases adjustment depends on selected software for aging testing. The end of a phase triggers the start of next. SWARE approach neglects the discovery of Time to Aging Related Failure (TTARF). Section III contains more details about proposed approach.

The Section IV has an experimental setup which uses the SWARE approach. This experiment consists of investigation of software aging and rejuvenation on OpenNebula/KVM12 Private Cloud [7]. In this case study, we investigate software aging symptoms in KVM 1.0 component using an accelerated workload of successive operations of attaching and detaching 15 virtual disks of 1 GB on a VM. We also checked rejuvenation effectiveness through a VM Live Migration process. The obtained results are in the Section IV-D. From results, it is possible to understand system behavior on the three phases of the experiment. Rejuvenation Phase results show that VM Live Migration enables software rejuvenation of KVM software.

Section VI presents our conclusions and future works.

II. SOFTWARE AGING AND REJUVENATION

Software aging is the accumulation of aging-related bugs effects. Aging-related bugs often appear when the system reaches conditions (e.g. lack of computational resources) which are difficult to reproduce [21]. The consequences of bugs activation lead to software performance degradation (or its failure rate increases). Software aging can lead the system from hangs to total failures [11].

A feasible way to determine software aging existence is to observe system monitoring reports to find anomalous behavior [22] [17]. The paper [23] presents a methodology based on SNMP distributed tool for monitoring OS resources of a LAN of UNIX machines to observe software aging existence. The Figure 1 depicts the general behavior of software aging.

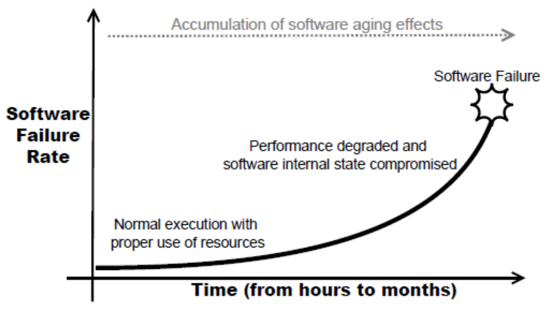


Fig. 1. Software aging general behavior

The paper [11] presents first definitions of software rejuvenation. We can define software aging as a proactive technique to avoid aging effects to reach critical levels. Software rejuvenation is also considered a cost-effective because it does not require knowledge about roots of aging effects [17]. The rejuvenation actions rely on restart an application to conduct it to a clean state, without aging effects accumulated. Some papers propose scheduling of software rejuvenation actions [18] [19] to determine when to perform rejuvenation actions to maximize overall system availability.

Experiments to measure and observe software aging symptoms may have a long duration. Some papers [6] [4] proposes an investigation with accelerated experiments. Using this type of investigation is possible to observe software internal state alterations in a shorter time.

III. SWARE APPROACH

The SWARE approach has two preliminary tasks. First, the selection of software component to test. Second, the selection of a workload to stress this component. For example, on an experiment of aging and rejuvenation in a Web Server, a workload with a high rate of requests can stress the system. The workload exposure aims to force the system to operate at different levels of usage. Therefore, this workload may trigger aging-related bugs activation. The workload selection must be careful to avoid premature failures. And, the monitoring activity should not cause high system intrusion.

As aforementioned, the SWARE approach has three phases (Stress, Wait and Rejuvenation). The details of each are in next sections.

A. *Stress Phase*

This phase aims to stress the system with selected workload. The stress workload leads the system to increase resources usage. Therefore, monitoring reports should present an increase in software failure rate or a decrease in software performance. Figure 9 depicts the expected behavior of internal system state in this phase.

The stress phase duration varies according to workload submitted to the system. Workload submission stops when resources usage or performance degradation reach a critical level. At this point, probably aging bugs already be activated as the system passes through different usage states.

B. *Wait Phase*

The major goal of Wait phase is to observe software aging symptoms existence. Software aging effects remain in the system even without incoming workload. After Stress Phase, there are two possibilities: (i) system recovers from workload overhead and returns to a stable state; (ii) or system persists degraded. Software rejuvenation cleans up software aging cumulative effects. If the system returns to a stable state without rejuvenation action, there is no evidence of software aging existence. In that conditions, the workload submitted to the system only causes overhead in resources usage.

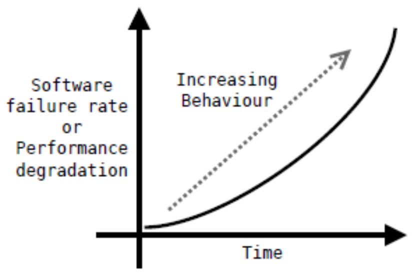


Fig. 2. Stress Phase expected behavior

The workload starts a cumulative process in the system on Stress phase. Therefore, in the absence of rejuvenation action system may continue in a degraded state. Figure 3(a) presents a possible behavior of system state which highlights software aging evidence. Figure 3(b) shows a possible behavior of system state without evidence of software aging.

Wait Phase may during sufficient time to highlight software aging cumulative process. The duration of Wait Phase should be long enough to ensure that system persists in a degraded state. Usually, to achieve this goal, Wait Phase should during the approximately same time of Stress phase.

C. *Rejuvenation Phase*

A requisite for Rejuvenation Phase start is the software rejuvenation action selection. This selection depends on the

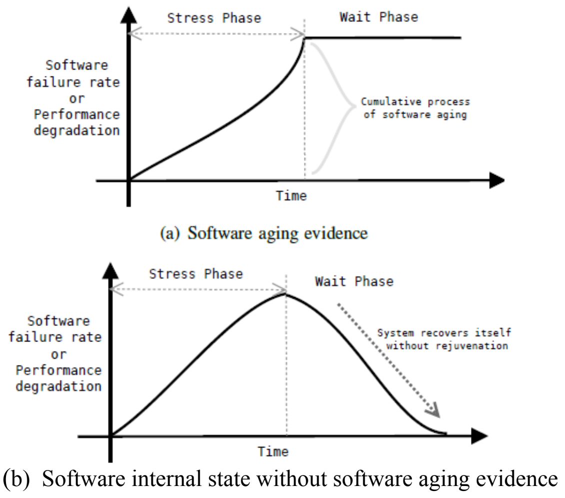
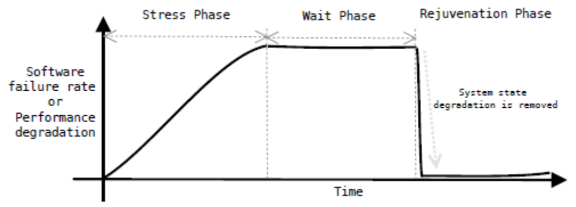
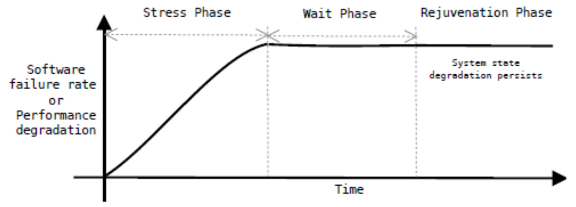


Fig. 3. Wait Phase

component stressed on the previous phases. Software rejuvenation relies on restart application or system reboot, but in some situations, other types of rejuvenation may also be valid. Rejuvenation phase starts on the software rejuvenation action submission. The objective of this phase is to observe impacts of rejuvenation action on internal system state. Figure 4 presents variations in software state when software rejuve-nation is useful or not.



(a) Software rejuvenation is effective



(b) Software rejuvenation is not effective for counteract software aging detected

Fig. 4. Rejuvenation phase

Figure 5 presents a flowchart with a summary of SWARE approach.

IV. A CASE STUDY

This section presents a case study using the SWARE approach. This case study consists of an experiment to examine software aging on VMM KVM 1.0 and to observe VM Live Migration effectiveness as software rejuvenation.

A. *Testbed*

The Cloud Computing testbed uses OpenNebula VIM 3.6 and VMM KVM 1.0. The testbed has four Physical Machines (PMs), and one Virtual Machine (VM) connected to a private local network (as presented in Figure 6). The testbed components are:

- VM → runs an Apache Web Server with a HTML page on Ubuntu Server 12.04 operating system;
- FrontEnd → responsible for managing Cloud Environment;
- Main Node → executes Virtual Machines;
- Standby Node → the target host for VM Live Migration, and;
- Stresser → in charge of sending workload and monitoring

Cloud Computing environment.
Table I presents the hardware configurations of PMs.

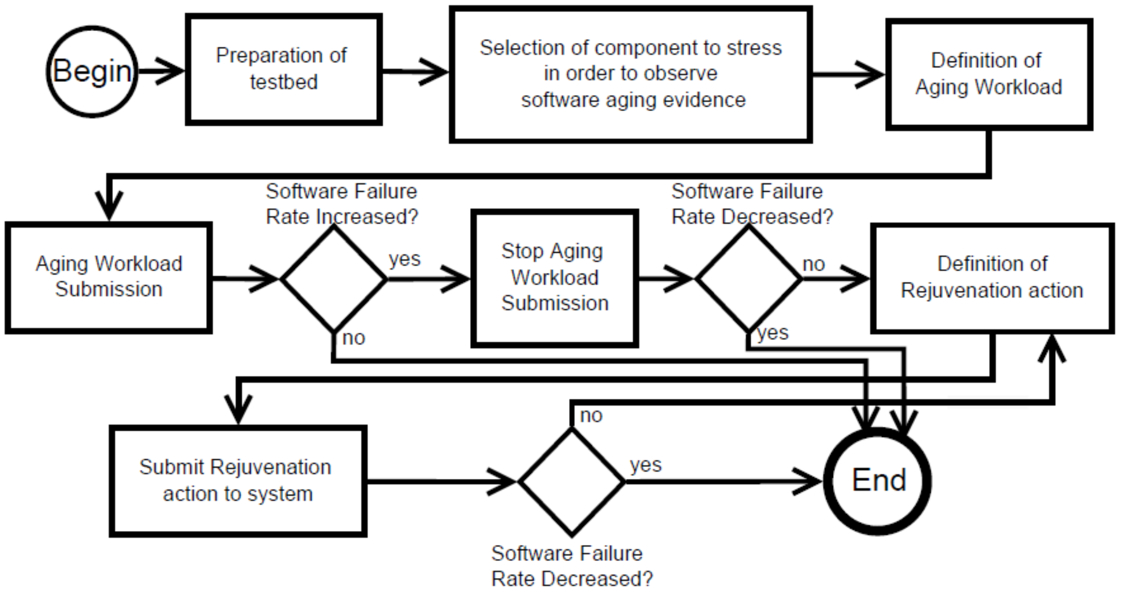
TABLE I: PMS HARDWARE CONFIGURATION

Name	Description	Processor	RAM	OS
FrontEnd	Cloud Manager	Intel Core i3 - 3.10 GHz	4 GB	Ubuntu Server 12.04 (kernel 3.2.0-23)
Main Node, Standby Node	Execution Nodes	Intel Core i3 - 3.10 GHz	4 GB	Ubuntu Server 12.04 (kernel 3.2.0-23)
Stresser	Cloud Monitor and Stresser	Intel Core 2 Quad - 2.66GHz	4 GB	Ubuntu Desktop 12.10 (kernel 3.5.0-36)

B. *Workload Selection*

The selected workload is a variation of previous experiments of software aging in Eucalyptus Cloud Computing [24]. This workload is a sequential operation of mount and unmount 15 Virtual Disks (of 1GB) on VM. The pseudo-algorithm in Algorithm 1 presents mount and unmount workload.

Algorithm 1 Software aging workload
```
loop
    while AttachedDisks < 15 do
        Mount(Disk1GB);
        Wait(15 seconds);
    end while
    while AttachedDisks >= 1 do
        Unmount(Disk1GB);
        Wait(15 seconds);
    end while
end loop
```


Begin
Preparation of testbed
Selection of component to stress in order to observe software aging evidence
Definition of Aging Workload
Aging Workload Submission
Software Failure Rate Increased?
yes
no
Stop Aging Workload Submission
Software Failure Rate Decreased?
no
yes
Definition of Rejuvenation action
Submit Rejuvenation action to system
no
yes
End
Software Failure Rate Decreased?
Software Failure Rate Decreased?
Fig. 5. SWARE Approach

Besides the mount and unmount workload, we decide to add a workload to Apache Web Server. We want to observe Web Server performance impacts during the experiment. To select the workload, we conducted a capacity test considering the same testbed used in software aging experiment. The capacity test

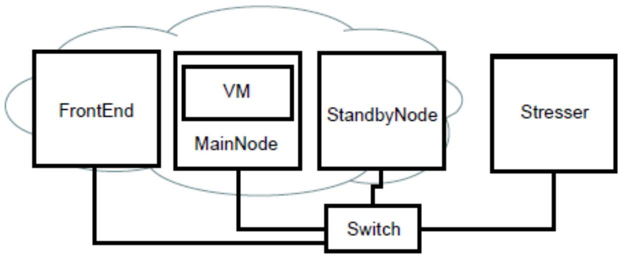


Fig. 6. Testbed architecture.

uses the httperf tool with Autobench [25] [26]. This benchmark tool sends requests to the Web Server and collects results as response time (in milliseconds) and the amount of errors observed. In httperf, the response time is the time between sending the first byte of a request and receiving the first byte of reply. And, the amount of errors take account of errors such as connection timeout, socket timeout and connection refused. The rate of requests is increasing over time. Figure 7 shows the results of the capacity test.

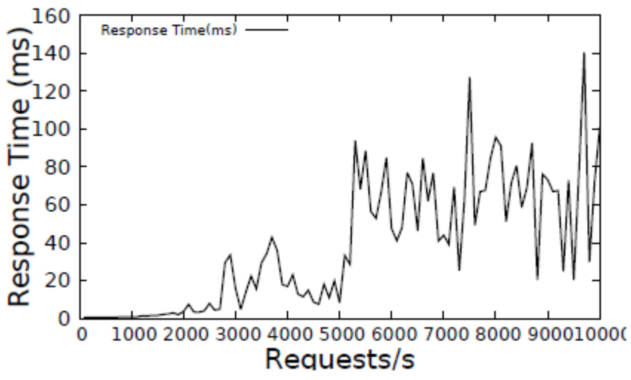
Observing these results, we selected the workload of 2000 requests per second. The results show that the server can handle this rate of requests with low response time and errors.

Figure 8 shows an overview of the selected workload to software aging and rejuvenation experiments.

C. Software Rejuvenation Strategy

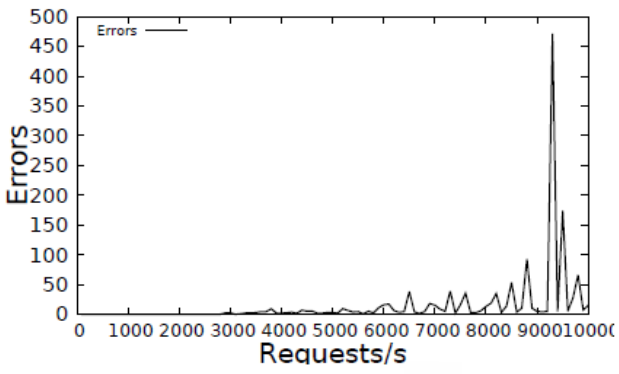
OpenNebula/KVM Clouds allow system managers to per- form VM Live Migration. VM Live Migration consists in remapping a VM from a PM to another with reduced op- eration downtime [27]. Previous studies [28] [18] shows VM Live Migration as a support mechanism to VMM software rejuvenation. Figure 9 presents software rejuvenation strategy for experiments.

On Stress Phase, the system is receiving workload to stress VMM software. In this early stage, the system does not present software aging effects yet. The Standby Node VMM is active but not receiving any system requests. Wait Phase starts when system status suffers from software aging. As Main Node VMM manages VM, software aging effects will affect VM performance too. VM Live Migration triggers the Rejuvenation Phase.



(a) Response Time (ms)

When VM arrives in the Standby Node, it can leverage a fresh state VMM. Thus, previous VMM software aging effects will not affect VM performance or failure rate. Finally, a Main Node restart removes software aging effects accumulation.



(b) Amount of errors

Fig. 7. Capacity test.

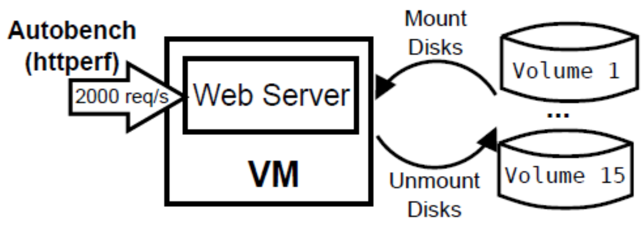


Fig. 8. Workload Selected

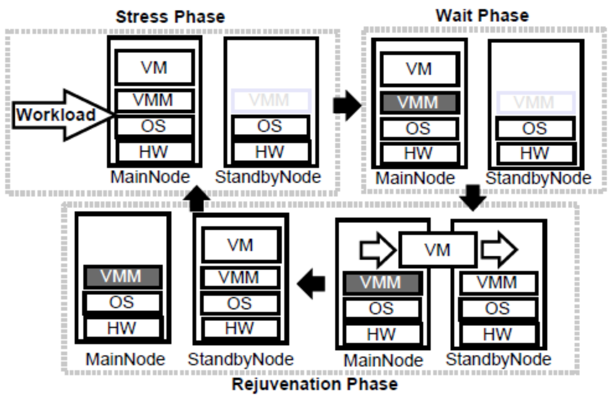


Fig. 9. Software rejuvenation strategy

When VM arrives in the Standby Node, it can leverage a fresh state VMM. Thus, previous VMM software aging effects will not affect VM performance or failure rate. Finally, a Main Node restart removes software aging effects accumulation.

System Monitoring reports support phases duration deci- sion. These reports are obtained using a script in Shell Script language [29] [30]. Based on principles presented in Section III, Table 2 shows each phase period for our experiment. The entire process during 13 consecutive days.

Phase	Span	1	2	3	4	5	6	7	8	9	10	11	12	13
Stress Phase	6 d	■	■	■	■	■	■							
Wait Phase	5 d							■	■	■	■	■		
Rejuvenation Phase	2 d												■	■
Disks Mount and Unmount	6 d	▨	▨	▨	▨	▨	▨							
Web Requests	13 d	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

D. *Results and analysis*

1) CPU and RAM monitoring: Figures 10 and 11 present results of system resources monitoring. To improve results visualization, the graphics present monitoring data from both PMs: Main Node and Standby Node. Stress Phase and Wait Phase in these plots are the results from Main Node monitoring and the Rejuvenation Phase is the result from Standby Node monitoring (which receives VM Live Migration). All plots contain limits of Stress Phase, Wait Phase and Rejuvenation Phase. The monitoring intervals are 30 seconds.

Therefore, CPU utilization results contain four metrics: **USER** - CPU utilization percentage for User-level processes; **SYS** - CPU usage for Kernel-level processes; **IO** - Waiting for In/Out operations; IDLE - CPU idle percentage, excluding otiose time waiting for In/Out.

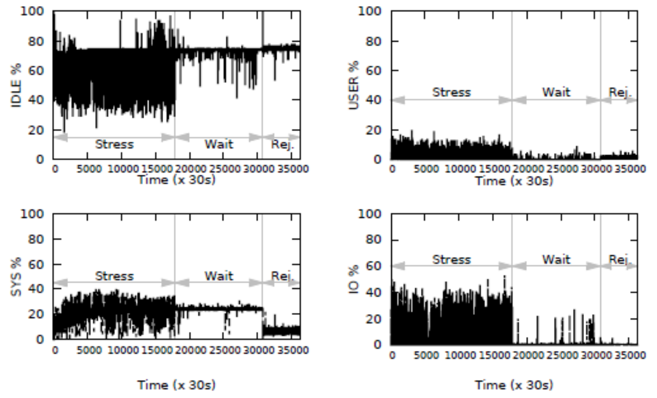


Fig. 10. CPU Utilization

Figure 10 presents results of CPU utilization. The results of this phase show significant IO requests rate for CPU. This behavior occurs because PM has to communicate with VM during mount disk workload and also has to redirect income network traffic to VM. In the Wait Phase, CPU utilization tends to return to normal levels. But, SYS requests rate remains at higher levels than normal (comparing to Rejuvenation Phase). As KVM resides on Linux Kernel (which is responsible for SYS requests), CPU SYS requests rate may return to normal state after a cleanup action. Rejuvenation Phase presents SYS requests rate returning to normal levels.

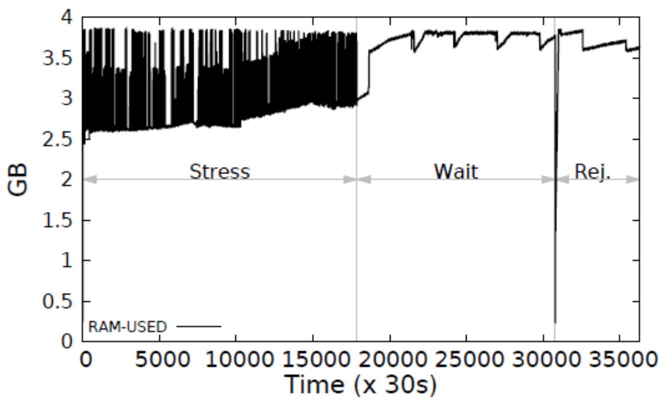


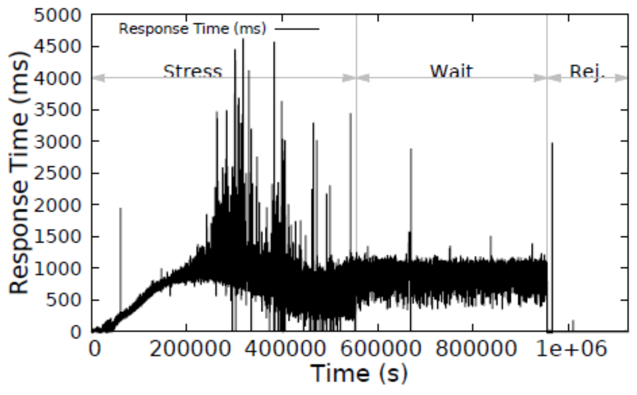
Fig. 11. RAM Usage

Figure 11 depicts results of RAM monitoring. Results for Wait Phase reveals that RAM consumption persists at high levels. In this phase, PM continues to receive web requests from the network. But, Rejuvenation Phase results presents a decreasing usage of RAM resources.

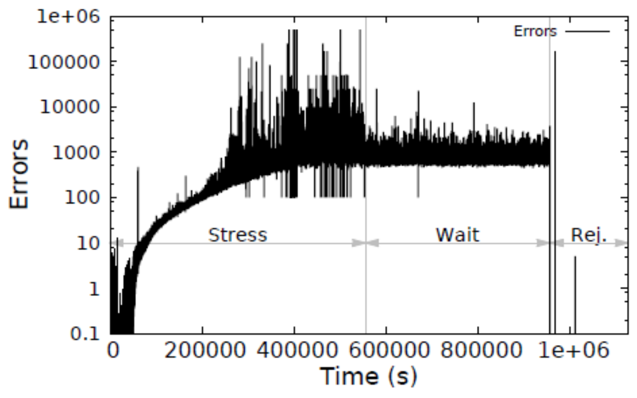
An extra consumption of resource is necessary when a PM receives a VM migration. Thus, there is a peak of RAM usage in beginning of Rejuvenation Phase. After this, RAM usage results show a decreasing behavior.

2) Web Server metrics results: VM depends on VMM to interact with PM resources. Then, VMM state affects the applications and services which run in VM. Thus, applica-tions and services monitoring may present possible effects of software aging. The results in this section show monitoring data collected from benchmark tool of Web Server.

Figure 12 presents results of Web Server monitoring. As expected, Web Server suffers effects of high workload exposure in Stress Phase. Response time and Amount of Errors were increasing during this phase. Wait Phase results show Errors and Response Time remaining at high levels. Rejuvenation phase brings the system to a stable state.



(a) Response Time (ms)



(b) Errors

3) General Discussion: Observing CPU results of Stress Phase in Figure 10 it is possible to notice a similar behavior as presented in [6]3. Still in Stress Phase, RAM results (Figure 11) and Web Server response time results (Figure 12(a)) present similar results as on [31]4.

As software aging causes internal software state degradation, its consequences may persist on software state until software rejuvenation (or failure) occurs (as explained on Section III). Results of Wait Phase shows a steadily degraded software state. For example, the Web Server used in experiments respond to requests workload specified (2000 requests per second) with minimum response time and errors. But, after Stress Phase, when possible activation of aging-related bugs occurs, Web Server metrics go to higher levels.

Results of Rejuvenation Phase emphasize software aging evidence on KVM software. After VM migration, VM arrives on a fresh state KVM software. As VM migration suffices to software internal state degradation removal, it is possible to corroborate that VM migration is a useful technique to KVM software rejuvenation.

V. RELATED WORK

The authors of [32] present an approach to apply accelerated degradation tests on software aging experiments. The paper uses a particular aging factor to control the aging effects on the experimental setup. This aging factor is obtained by sensitivity analyses based on statistical design of experiment. We also applied accelerated tests on the Stress Phase of our experiment. Different from the authors of mentioned paper, the SWARE approach also comprises software rejuvenation.

The paper [33] presents a methodology to estimate software aging effects by using time-series analyses. The authors analyzed the behavior of a Web Server under a varying workload. The main goal is to detect and estimate resource exhaustion time due to software aging effects. This paper helps us to define technologies used in our case study. As presented in the mentioned paper, we also used a Web Server and the httperf tool in our experimental setup. The paper also presents an extensive statistical analysis used in the resources estimation. Different from this paper we used a generic workload to induce software aging bugs activation. SWARE approach is simpler to apply as it does not require statistical expertise.

The papers [31] provide valuable inputs on how to perform software aging experiments in a Web Server. The main goal of the authors is to conduct a software aging analysis of a Web Server. Different from the results presented in the mentioned paper, we also include the software rejuvenation effectiveness test on SWARE approach.

The paper [34] offer a comprehensive approach to software aging and rejuvenation experiments on a Web Server. Besides the techniques to observe software aging problems, the authors implemented a rejuvenation agent to mitigate aging effects in the Web Server. The presented results show substantial reduction of software aging when the rejuvenation agent is integrated into the environment. The proposed rejuvenation agent uses a predefined interval to submit software rejuvenation in the environment. Different from this paper, the Wait Phase of SWARE approach allows the system manager to decide when to perform rejuvenation action. Therefore, it is possible to reduce overhead caused by recurrent software rejuvenation actions. Nevertheless, the mentioned paper provides helpful insights to SWARE approach.

Cotroneo et al [35] provides a broad investigation of software aging causes in Linux Operating Systems. The adopted approach aims to trace kernel activities to observe possible software aging effects. The results of the paper show that the filesystem operation causes significant contribution in software aging indicators. This result may explain the behavior of Stress Phase of our experiment when the VM filesystem is dealing with software aging workload. Matias et al [36] also present a methodology to measure software aging effects through OS kernel observation and instrumentation. Instead of showing specific causes of software aging, the SWARE approach aims to provide an overview of system status during and after software aging effects.

The paper [37] shows a study of software aging and rejuvenation in an SOAP-based Servers. The authors ran a variety of scenarios with different configurations. The adopted approach is focused on studies of software aging and rejuvenation in SOAP-based servers. SWARE approach aims to be more generic and flexible to other types of software.

The paper [38] presents an investigation of software aging on OpenStack Cloud Computing Platform. The authors used a testbed which runs OpenStack, Apache and MySQL database. The presented results show that the MySQL processes present software aging issues. The authors also used a particular workload to stress the system and present trend analysis for resources consumption. The considered workload consists of sequential operations of start and terminate VM instances. We also adopted a sequential workload in our case study. However, the SWARE approach also comprises the observation of software aging problems and software rejuvenation effectiveness. The paper [39] presents a comprehensive approach to investigating software aging and rejuvenation in a J2EE Application Server. The authors used a hierarchical approach to submit software rejuvenation in the system. The adopted methodology has two main steps: (i) software aging tests and (ii) application of the hierarchical software rejuvenation mechanism. Our experiments also comprise software aging and rejuvenation phases. But, we also include the Wait Phase to highlight effects caused by software aging bugs activation.

It is important to highlight that the present paper is an improved version of our previous publication [40].

VI. CONCLUSIONS AND FUTURE WORKS

This paper presented the SWARE approach. The SWARE is a comprehensive approach to investigating aging symptoms and rejuvenation effectiveness on software systems. SWARE approach has three phases aiming to highlight software aging effects symptoms and rejuvenation effectiveness. (I) Stress Phase, when software aging workload reaches system to stress investigated component. (II) Wait Phase, to perceive indicators of software aging. (III) Rejuvenation Phase, which aims to detect rejuvenation action effectiveness on the environment.

Section IV presents a case study to validate SWARE approach. This case study aims to investigate KVM software aging and also to study VM Live Migration effectiveness as a software rejuvenation action. The investigation shows results of software aging symptoms on KVM. Wait Phase highlights software aging effects on resources consumption and quality of service of a Web Server. Finally, after VM Live Migration it is possible to notice that degradation effects clean-up.

The major contributions of this paper is a generic approach to software aging and rejuvenation effectiveness investigation. The case study proposed was a Cloud Computing environment. But, the approach phases guidelines can be reproduced in other types of software systems. This paper also presents (as a case study of the approach) a software aging and rejuvenation study on VM Live Migration as software rejuvenation for KVM hypervisor. The results of proposed case study show practical results of VM live migration effectiveness as rejuvenation for KVM.

The approach does not quantify software aging effects. With the lack of statistical techniques on data, it is hard to define software aging failure time and a proper rejuvenation schedule. The approach may require a substantial time to produce expected results. The Stress phase may not produce expected results on software aging bugs activation.

Future research directions aim to investigate further SWARE approach application on different scenarios as Software- Defined Networking, Network Function Virtualization, Virtu- alized Containers and Fog Computing. Other research lines seek to improve approach adding multiple component software aging investigations.

REFERENCES

[1] CISCO, "Cisco global cloud networking survey summary and analysis of results worldwide results." CISCO, Tech. Rep., 2012.
[2] CDW, "Cdw's cloud 401 report," CDW, Report, 2015.

[3] W. Kim, "Cloud computing: Today and tomorrow." Journal of object technology, vol. 8, no. 1, pp. 65–72, 2009.
[4] J. Araujo, R. Matos, P. Maciel, and R. Matias, "Software aging issues on the eucalyptus cloud computing infrastructure," in Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on. IEEE, 2011, pp. 1411–1416.
[5] J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker, "Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure," in Proceedings of the Middleware 2011 Industry Track Workshop. ACM, 2011, p. 4.
[6] R. Matos, J. Araujo, V. Alves, and P. Maciel, "Characterization of software aging effects in elastic storage mechanisms for private clouds," in Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on, 2012, pp. 293–298.
[7] M. Torquato, J. Araujo, and P. Maciel, "Estudo experimental de envelhecimento de software em nuvens kvm/opennebula: Live migration como mecanismo de suporte ao rejuvenescimento de software," in XIII Workshop em Clouds e Aplicac¸o˜es in conjunction with 33rd Brazilian Symposium on Computer Networks and Distributed Systems (SBRC2015). Vito´ria, ES, Brazil: Universidade Federal do Esp´ırito Santo (UFES), may 2015, pp. 1–14.
[8] I. Umesh and G. N. Srinivasan, "Dynamic software aging detection- based fault tolerant software rejuvenation model for virtualized en- vironment," in Proceedings of the International Conference on Data Engineering and Communication Technology. Springer, 2017, pp. 779– 787.
[9] D. L. Parnas, "Software aging," in Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, 1994, pp. 279–287.
[10] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on. IEEE, 2008, pp. 1–6.
[11] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvena- tion: Analysis, module and applications," in Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Sympo- sium on. IEEE, 1995, pp. 381–390.
[12] J. Araujo, V. Alves, D. Oliveira, P. Dias, B. Silva, and P. Maciel, "An investigative approach to software aging in android applications," in Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE, 2013, pp. 1229–1234.
[13] Y. Qiao, Z. Zheng, and F. Qin, "An empirical study of software aging manifestations in android," in Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on. IEEE, 2016, pp. 84–90.
[14] J. Araujo, F. Oliveira, R. Matos, M. Torquato, J. Ferreira, and P. Maciel, "Software aging issues in streaming video player," Journal of Software, vol. 11, no. Jun 2016, pp. 554–568, 2016.
[15] F. Alencar, M. Santos, M. Santana, and S. Fernandes, "How software aging affects sdn: A view on the controllers," in Global Information Infrastructure and Networking Symposium (GIIS), 2014. IEEE, 2014, pp. 1–6.
[16] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, "Aging-related bugs in cloud computing software," in Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on. IEEE, 2012, pp. 287–292.
[17] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 10, no. 1, p. 8, 2014.
[18] M. Melo, P. Maciel, J. Araujo, R. Matos, and C. Araujo, "Availability study on cloud computing environments: Live migration as a rejuvena- tion mechanism," in Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, 2013, pp. 1–6.
[19] M. Melo, J. Araujo, R. Matos, J. Menezes, and P. Maciel, "Comparative analysis of migration-based rejuvenation schedules on cloud availabil- ity," in Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on, Oct 2013, pp. 4110–4115.
[20] M. D. E. T. Melo and P. R. M. Maciel, "Modelos de disponibilidade para nuvens privadas: Rejuvenescimento de software habilitado por agendamento de migrac¸a˜o de vms," 2014.
[21] K. Vaidyanathan and K. S. Trivedi, "Extended classification of software faults based on aging," in Fast Abstract, Int. Symp. Software Reliability Eng., Hong Kong, 2001.
[22] N. A. Valentim, A. Macedo, and R. Matias, "A systematic mapping review of the first 20 years of software aging and rejuvenation research," in Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on. IEEE, 2016, pp. 57–63.
[23] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on. IEEE, 1998, pp. 283–292.
[24] R. Matos, J. Araujo, V. Alves, and P. Maciel, "Experimental evaluation of software aging effects in the eucalyptus elastic block storage," in Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on. IEEE, 2012, pp. 1103–1108.
[25] D. Mosberger and T. Jin, "httperf—a tool for measuring web server performance," ACM SIGMETRICS Performance Evaluation Review, vol. 26, no. 3, pp. 31–37, 1998.
[26] J. T. J. Midgley. (2017) Xenoclast - autobench. [Online]. Available: http://www.xenoclast.org/autobench/
[27] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251223
[28] F. Machida, D. S. Kim, and K. S. Trivedi, "Modeling and analysis of software rejuvenation in a server virtualized system with live vm migration," Performance Evaluation, vol. 70, no. 3, pp. 212 – 230, 2013, special Issue on Software Aging and Rejuvenation. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166531612000934
[29] M. Mitchell, J. Oldham, and A. Samuel, Advanced Linux Programming, ser. Landmark Series. New Riders, 2001. [Online]. Available: http://books.google.com.br/books?id=oRqqAVyXjwAC
[30] M. Torquato, H. Mello, L. Torquato, J. Araujo, and E. Guedes, "Utilizac¸a˜o de scripts para monitoramento de sistemas linux: Abordagem para criac¸a˜o de relato´rios e gra´ficos com ferramentas open-source," in VI Workshop de Software Livre (FREEBASE), na Escola Regional de Computac¸a˜o Bahia - Alagoas - Sergipe (XV ERBASE), 2015, pp. 21– 30.
[31] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," Reliability, IEEE Transactions on, vol. 55, no. 3, pp. 411–420, 2006.
[32] R. Matias, P. A. Barbetta, K. S. Trivedi, and P. J. Freitas Filho, "Accelerated degradation tests applied to software aging experiments," IEEE Transactions on reliability, vol. 59, no. 1, pp. 102–114, 2010.
[33] L. Li, K. Vaidyanathan, and K. S. Trivedi, "An approach for estimation of software aging in a web server," in Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n. IEEE, 2002, pp. 91–100.
[34] R. Matias Jr and J. Paulo Filho, "An experimental study on software aging and rejuvenation in web servers," in Computer Software and Ap- plications Conference, 2006. COMPSAC'06. 30th Annual International, vol. 1. IEEE, 2006, pp. 189–196.
[35] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging analysis of the linux operating system," in Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on. IEEE, 2010, pp. 71–80.
[36] R. Matias, I. Beicker, B. Leit a˜o, and P. R. Maciel, "Measuring software aging effects through os kernel instrumentation," in Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop on. IEEE, 2010, pp. 1–6.
[37] L. Silva, H. Madeira, and J. G. Silva, "Software aging and rejuvenation in a soap-based server," in Network Computing and Applications, 2006. NCA 2006. Fifth IEEE

International Symposium on. IEEE, 2006, pp. 56–65.

[38] C. Melo, J. Araujo, V. Alves, and P. Maciel, "Investigation of software aging effects on the openstack cloud computing platform," Journal of Software, vol. 12, no. 2, pp. 125–138, 2017.

[39] H. Meng, X. Hei, J. Zhang, J. Liu, and L. Sui, "Software aging and rejuvenation in a j2ee application server," Quality and Reliability Engineering International, vol. 32, no. 1, pp. 89–97, 2016.

[40] M. Torquato, P. Maciel, J. Araujo, and I. Umesh, "An approach to investigate aging symptoms and rejuvenation effectiveness on software systems," in Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on. IEEE, 2017, pp. 1–6.