JOURNAL OF INFORMATION SYSTEMS ENGINEERING & MANAGEMENT

Journal of Information Systems Engineering & Management, 2018, 3(2), 15
ISSN: 2468-4376

lectito

SWARE: A Methodology for Software Aging and Rejuvenation Experiments

Matheus Torquato [1]*, Jean Araujo [2], I. M. Umesh [3], Paulo Maciel [4]

[1] *Federal Institute of Alagoas (IFAL), Campus Arapiraca,* Arapiraca, AL, BRAZIL
[2] *Federal Rural University of Pernambuco (UFRPE), Campus Garanhuns,* Garanhuns, PE, BRAZIL
[3] *Bharathiar University, Coimbatore,* INDIA
[4] *Federal University of Pernambuco (UFPE), Center of Informatics (CIn),* Recife, PE, BRAZIL

***Corresponding Author:** matheustor4.professor@gmail.com, matheus.torquato@ifal.edu.br

Citation: Torquato, M., Araujo, J., Umesh, I. M. and Maciel, P. (2018). SWARE: A Methodology for Software Aging and Rejuvenation Experiments. *Journal of Information Systems Engineering & Management*, 3(2), 15. https://doi.org/10.20897/jisem.201815

Published: April 07, 2018

ABSTRACT

Reliability and availability are mandatory requirements for numerous applications. Technical apparatus to study system dependability is essential to support software deployment and maintenance. Software aging is a related issue in this context. Software aging is a cumulative process which leads systems with long-running execution to hangs or failures. Software rejuvenation is used to prevent software aging problems. Software rejuvenation actions comprise system reboot or application restart to bringing software to a stable fresh state. This paper proposes a methodology to conduct software aging and software rejuvenation experiments. 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 methodology as SWARE (Stress-Wait-Rejuvenation). To validate the SWARE methodology, 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.

Keywords: software aging and rejuvenation, reliability, dependability, availability, cloud computing

INTRODUCTION

High availability and reliability are crucial for different software systems. In Cloud Computing, for example, software reliability and availability stay as concerns for customers (CISCO, 2012; CDW, 2015; Kim, 2009). Previous works introduce software aging as a pertinent issue in this area (Araujo et al., 2011a; Araujo et al., 2011b; Matos et al., 2012; Torquato et al., 2015; Umesh and Srinivasan, 2017).

The software aging phenomenon consists in a gradual increase in software failure rate or performance degradation during its execution (Parnas, 1994). These effects usually happen because of errors accumulation in software state. This accumulation can lead software to hangs and total failures (Grottke et al., 2008; Huang et al., 1995). Systems with long-time of execution may suffer from software aging effects (Huang et al., 1995). On Cloud Computing systems, the VMM (Virtual Machine Monitor) is liable to suffer software aging, as presented in (Matos et al., 2012; Torquato et al., 2015).

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 usually lies on an application restart or a system reboot (Cotroneo et al., 2014). Previous works propose a schedule to submit software rejuvenation actions (Melo et al., 2013a; Melo et al., 2013b) to minimize system downtime caused by these operations. Other techniques use genetic algorithms and time series forecasting to predict software aging behavior and support software rejuvenation actions (Umesh et al., 2017a; Umesh et al., 2017b). More details of software aging and rejuvenation are in *Software Aging and Rejuvenation* section.

Due to characteristics of software aging, it is hard to determine its roots. Errors, memory leaks and other aging-related symptoms are non-expected events (Torquato et al., 2015). A proper approach to deal with software aging and rejuvenation issues is to investigate and analyze them in an isolated environment. By running experiments and tests to understand software aging symptoms and rejuvenation effectiveness.

To support this type of research, this paper presents a methodology to support software aging and rejuvenation experiments. The approach is named SWARE (Stress-WAit-REjuvenation). The SWARE approach has three phases. (A) Stress Phase, which aims to observe workload exposure impacts in the internal software state. (B) Wait Phase which observes software behavior after the stress workload submission. This phase seeks to highlight software aging effects in the system. As software aging is a cumulative process, its effects may remain even after workload exposure. If the software returns to a stable state without software rejuvenation action, there is no evidence of software aging symptoms. (C) Rejuvenation Phase, with the goal to perceive consequences of software rejuvenation action (Melo, 2014; Torquato et al., 2015). This phase highlights the effectiveness of software rejuvenation action to mitigate software aging effects observed in the Wait Phase.

The phases adjustment depends on selected software for aging testing. The phases are sequential. The end of a phase triggers the start of next. The proposed approach does not comprise the detection of Time to Aging Related Failure (TTARF). *SWARE Methodology* section contains details about proposed approach.

Case Study section has an experimental setup which uses the proposed approach. This experiment consists of an investigation of software aging and rejuvenation on OpenNebula/KVM 12 Private Cloud (Torquato et al., 2015). In this case study, we investigate software aging symptoms in KVM 1.0 component using an accelerated workload. We also checked rejuvenation effectiveness through a VM Live Migration process. The obtained results are in the *Results and Analysis* section. From results, it is possible to understand system behavior during the three phases of the experiment. Rejuvenation Phase results show that VM Live Migration enables software rejuvenation of KVM software. We also present a numerical analysis of the results obtained from the Wait and Rejuvenation phases. These results show that the system may require more than 12 days to recover without software rejuvenation.

Related Works section has the related works comparison, and *Conclusions and Future Work* section presents our conclusions and future works. This paper is an improvement and extension of our previous articles (Torquato et al., 2017) and (De Melo et al., 2017).

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 (Vaidyanathan and Trivedi, 2001). The consequences of bugs activation lead to software performance degradation (or its failure rate increases). Software aging can influence the system from hangs to total failures (Huang et al., 1995).

A feasible way to determine software aging existence is to observe system monitoring reports to find anomalous behavior (Valentim et al., 2016; Cotroneo et al., 2014). The paper (Garg et al., 1998) presents a methodology based on SNMP distributed tool for monitoring OS resources on a LAN of UNIX machines to observe software aging existence.

The paper (Huang et al., 1995) 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 (Cotroneo et al., 2014). 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 (Melo et al., 2013a; Melo et al., 2013b) 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 (Matos et al., 2012; Araujo et al., 2011a) proposes an investigation with accelerated experiments. Thus, it is possible to observe software internal state alterations in a shorter time.

SWARE METHODOLOGY

The experiment methodology is obtained from the papers (Torquato et al., 2017) and (De Melo et al., 2017). This methodology has two preliminary steps. First, the selection of software component to test. Second, the choice

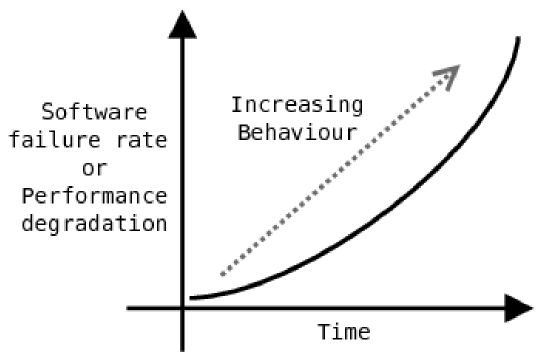

Figure 1. Stress phase expected behavior

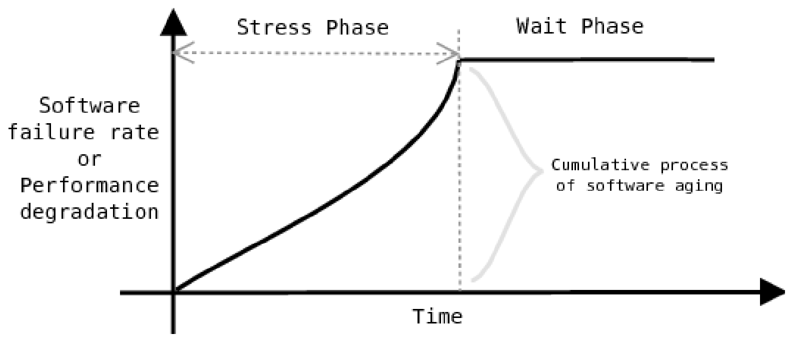

Figure 2. Software aging evidence

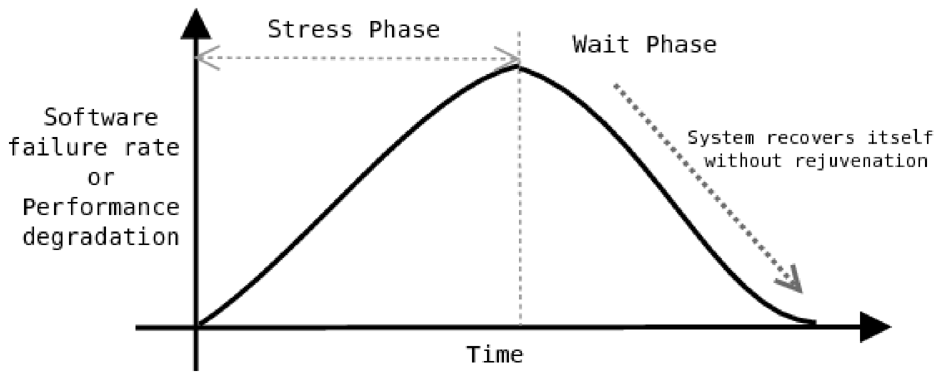

Figure 3. Absence of software aging evidence

of a specific workload to stress this component. For example, a specific workload for Web Server is a workload of connections high rate of requests. The workload exposure aims to induce the system to operate in different levels of usage, seeking to trigger aging-related bugs. It is essential to ensure that workload does not cause premature failures. And, the monitoring activity should not cause high system intrusion.

After these preliminary steps, we can apply the proposed approach. As aforementioned, the SWARE methodology has three phases (Stress, Wait and Rejuvenation). The details of each are in next sections.

Stress Phase

This phase aims to stress the system with the selected workload. The stress workload leads to system internal state degradation. Monitoring reports of this phase should present an increase in software failure rate or a decrease in software performance. Figure 1 depicts the expected behavior of internal system state.

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.

Wait Phase

The primary 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. If the software 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.

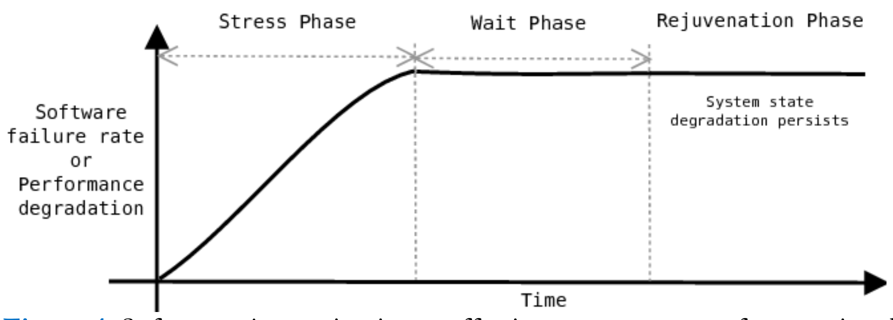

Figure 4. Software rejuvenation is not effective to counteract software aging detected

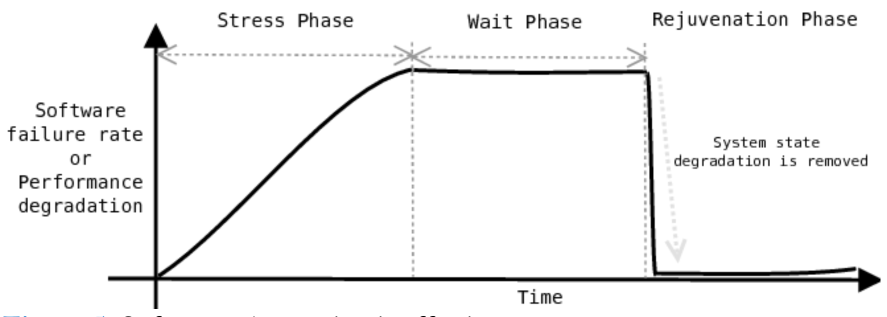

Figure 5. Software rejuvenation is effective

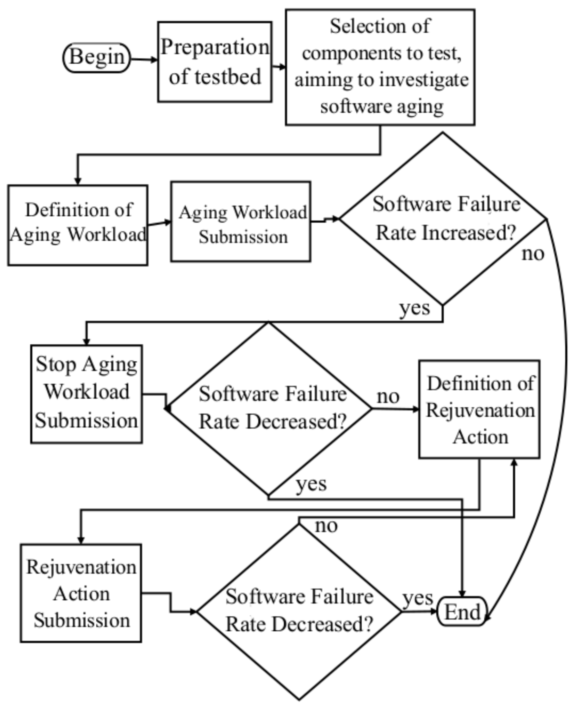
Begin
Preparation of testbed
Selection of components to test, aiming to investigate software aging
Definition of Aging Workload
Aging Workload Submission
Software Failure Rate Increased?
no
yes
Stop Aging Workload Submission
Software Failure Rate Decreased?
no
yes
Definition of Rejuvenation Action
no
Rejuvenation Action Submission
Software Failure Rate Decreased?
yes
End

Figure 6. Proposed Approach

Figure 2 presents a possible behavior of system state which highlights software aging evidence. **Figure 3** shows a likely behavior of the system state which does not highlight evidence of software aging.

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 approximately the same time of Stress phase.

Rejuvenation Phase

A requisite for Rejuvenation Phase start is the software rejuvenation action selection. This selection depends on the component stressed in the previous phase. Software rejuvenation usually relies on restart application or system reboot, but in some situations, other types of rejuvenation may also be valid.

Rejuvenation phase starts with the software rejuvenation action submission. The primary goal of this phase is to observe impacts of rejuvenation action on internal system state. **Figures 4** and **5** present variations in software state when software rejuvenation is useful or not.

Figure 6 presents a flowchart with a summary of proposed approach.

Table 1. Physical Machine Configurations

Name	Description	Processor	RAM	Software
FrontEnd	Cloud Manager	Intel Core i3 – 3.10 GHz	4GB	Ubuntu Server 12.04 (kernel 3.2.0-23), OpenNebula 3.6
Main Node, Standby Node	Execution Nodes	Intel Core i3 – 3.10 GHz	4GB	Ubuntu Server 12.04 (kernel 3.2.0-23), KVM 1.0
Stresser	Cloud Monitor and Stresser	Intel Core 2 Quad – 2.66 GHz	4GB	Ubuntu Desktop 12.10 (kernel 3.5.0-36), Autobench and monitoring tools

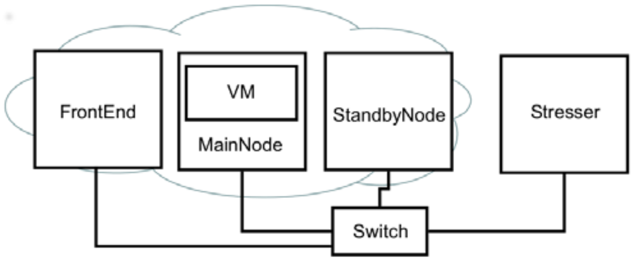

Figure 7. Testbed architecture

CASE STUDY

System Architecture

The considered Cloud Computing testbed uses OpenNebula VIM 3.6 and VMM KVM 1.0. The environment has four Physical Machines (PMs) and one Virtual Machine (VM). The VM runs an Apache Web Server with a simple HTML page on Ubuntu Server 12.04 operating system. *FrontEnd* Machine is responsible for managing Cloud Environment. *Main Node* is the PM which executes VM. *Standby Node* is a spare host to receive VM Live Migration purposes. *Stresser* is an external machine which is responsible for sending workload and monitor Cloud system. All components are in a Private Network. The **Table 1** has the Physical Machine configurations. **Figure 7** depicts the testbed architecture.

Workload Selection

The selected workload to stress KVM is a sequential operation of mounting and unmounting 15 Virtual Disks (of 1GB) on VM (Matos et al., 2012). The pseudo-algorithm of this workload is in Algorithm 1.

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
```

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 uses the httperf tool with Autobench (Mosberger and Jin, 1998). This benchmark tool sends requests to the Web Server and collects results as response time (in milliseconds) and the number 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 takes account of errors such as connection timeout, socket timeout and connection refused. The rate of requests is

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 10** shows a summary of the selected workload to software aging and rejuvenation experiments.

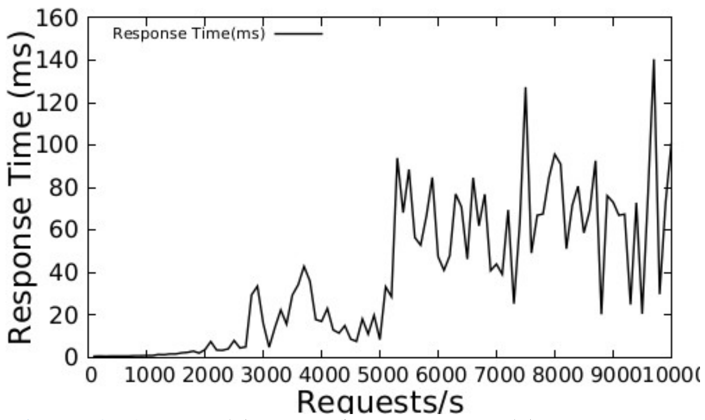

Figure 8. Capacity Test Results – Response Time (ms)

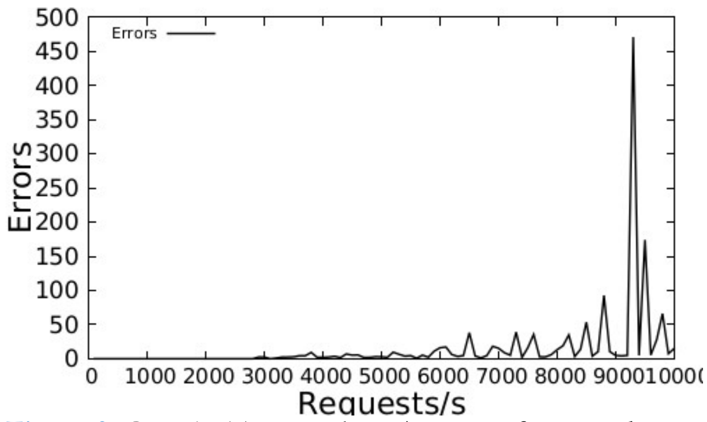

Figure 9. Capacity Test Results – Amount of Errors observed

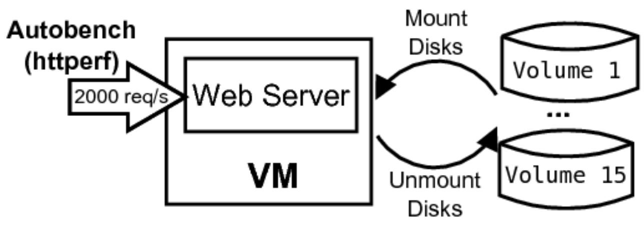

Figure 10. Selected Workload

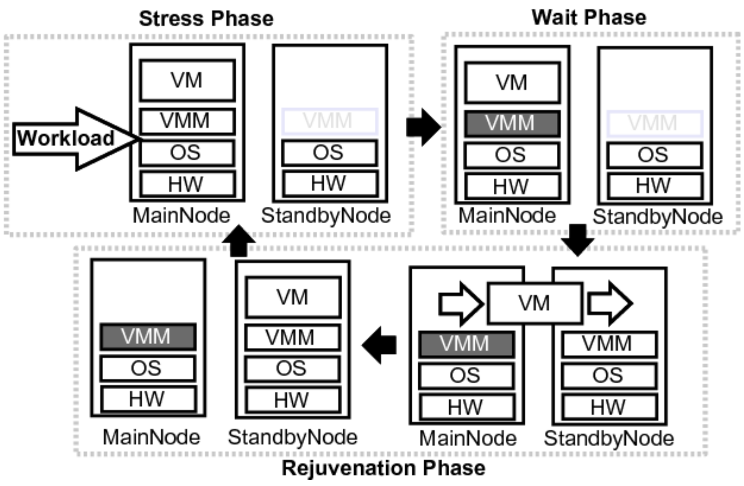

Figure 11. Software rejuvenation strategy

Software Rejuvenation Strategy

OpenNebula/KVM Clouds allow system managers to perform VM Live Migration. VM Live Migration consists in remapping a VM from a PM to another with reduced operation downtime (Clark et al., 2005). Previous studies (Machida et al., 2013; Melo et al., 2013a) shows VM Live Migration as a support mechanism to VMM software rejuvenation. Figure 11 presents software rejuvenation strategy used in the experiments.

Table 2. Experiment Phases Duration

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

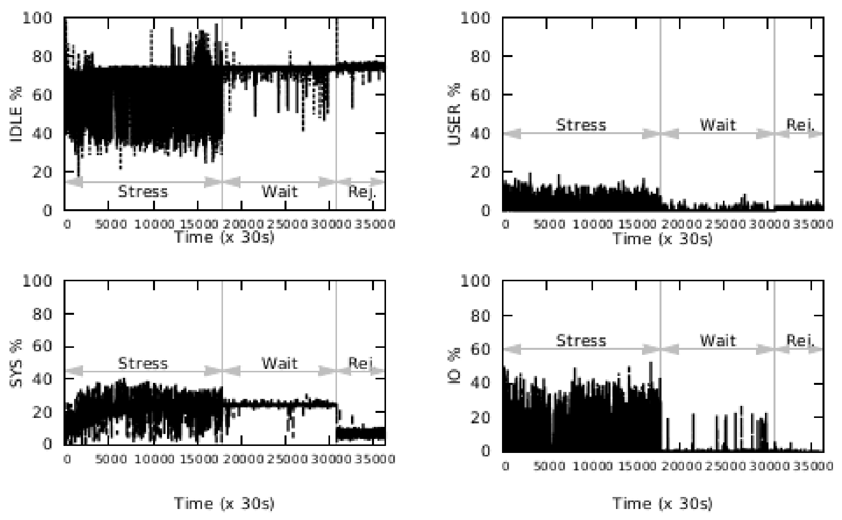

Figure 12. Results of CPU utilization results

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.

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, the Main Node restart removes software aging status.

System Monitoring reports support phases duration decision. Based on principles presented in *SWARE Methodology* section, Table 2 shows each phase period for our experiment. The entire process during 13 consecutive days.

RESULTS AND ANALYSIS

CPU and RAM Monitoring

The 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.

The 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 time waiting for In/Out.

Figure 12 presents results of CPU utilization. In the Stress Phase is possible to observe 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 average (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.

Figure 13 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.

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

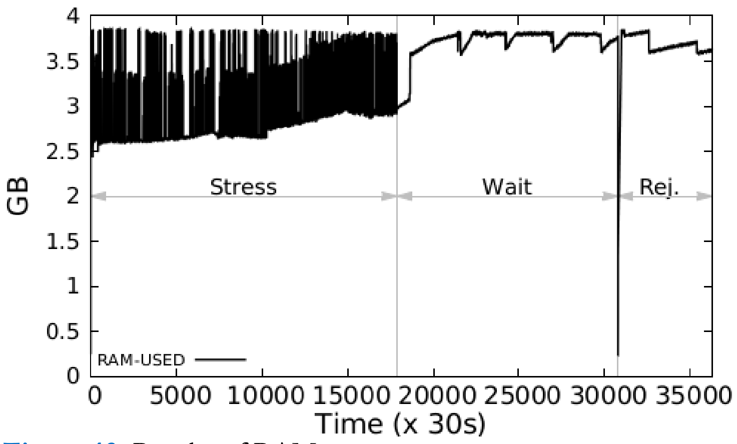

Figure 13. Results of RAM usage

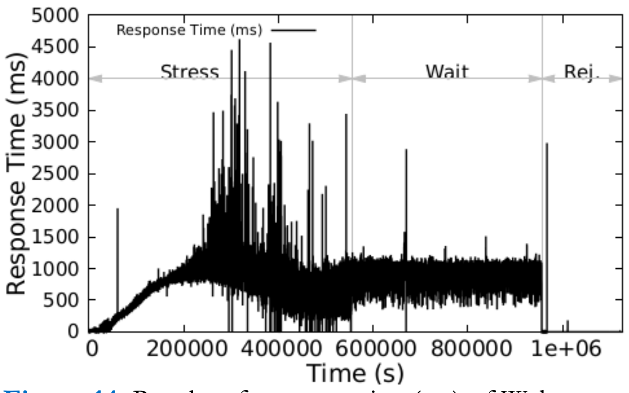

Figure 14. Results of response time(ms) of Web server

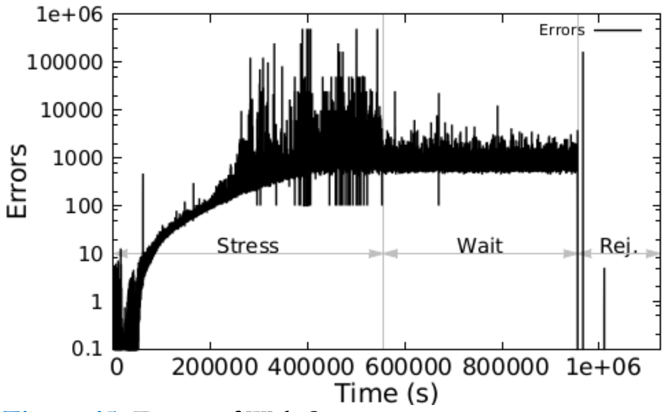

Figure 15. Errors of Web Server

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, applications 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.

Figures 14 and 15 present 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.

Numerical Analysis

In this section, we show the results of a brief numerical analysis of obtained results. We selected the Response Time results of Web Server (Figure 14) for this analysis. First, we compute the details of the response time of the Web Server with software aging effects (Wait Phase results) and without software aging (Rejuvenation Phase). The summary of these results is in Table 3. This analysis aims to analyze the effects of software aging and rejuvenation in the response time of the Web Server. Therefore, we ignored the results of Stress Phase. It is important to

Table 3. Web Server Response Time Results (ms) Summary

Phase	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Wait Phase	0.0	848.5	934.9	**917.9**	1004.8	2888.6
Rejuvenation Phase	0.0	0.5	0.5	**1.063**	0.600	2982.9

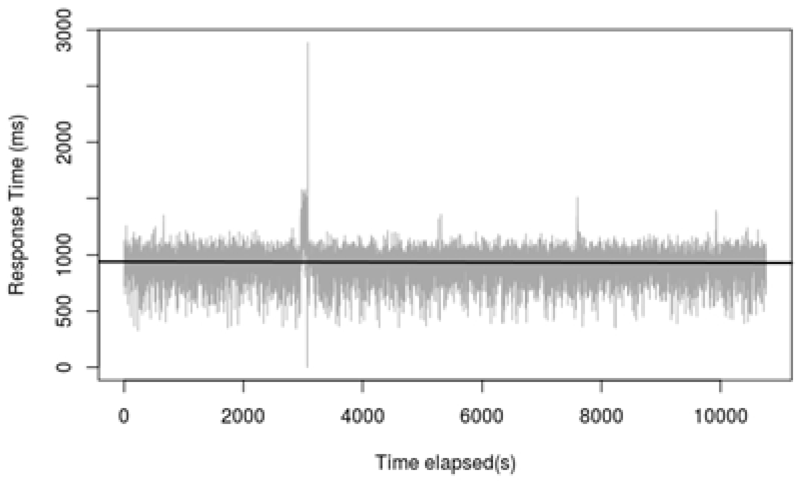

Figure 16. Web Server Response Time of Wait Phase with trend analysis

Table 4. Web Server Response Time Trend Analysis

Parameter	Value
Mann-Kendall Z-Value	-2.3
Estimated slope	-0.0009 ms/s
95% confidence interval for the slope	(-0.0001 ms/s, -0.0016 ms/s)

highlight that the Stress Phase goal is only to accelerate software aging bugs activation. The obtained results reveal that the response time in a Web Server with software aging effects surpass the response time in a Web Server without software aging effects in more than 860%.

We also perform a trend analysis in the results of Wait Phase. The goal of the trend analysis is to verify the existence of a trend in the Web Server response time. We used the Mann-Kendall Test. The Mann-Kendall test is usually applied in software aging tests (Grottke et al., 2006; Garg et al., 1998; Machida et al., 2013). The Mann-Kendall analysis (Mann, 1945) checks the null hypothesis, H0, which shows that there is no trend in the data during the time, against the alternative hypothesis, H1, which indicates an upward or a monotonic downward trend in the data. As software aging is a cumulative process, the Mann-Kendall test can be used to reveal patterns of software internal state degradation. Among Mann-Kendall tests results, we have the Z-value which is used to accept or reject the null hypothesis. Z-value close to zero suggests no trend in the data; a high absolute value indicates the existence of a trend. Figure 16 shows the results and trend of Web Server response time during the Wait Phase.

Table 4 presents the results of statistical analysis made in the data. The results show Mann-Kendall Z-value is lower than zero. Therefore, it is possible to reject the null hypothesis (no trend in the data). The negative value indicates a monotonic downward trend in the data. To calculate the slope of the monotonic trend, we used the Sen method (Sen, 1968). Table 4 also presents the estimated slope and the 95% confidence interval for this slope.

These results show that the downward trend obeys the following function: $y = -0.0009x + 939.6568$. We compute the time to system returns to normal operation. To conduct this calculation, we consider the Web Server Response Time in the Rejuvenation Phase, which is of 1.063 milliseconds. This calculation shows that the time to Web Server return to normal operation (with Response Time equals to 1.063 ms) is about 12 days. The Figure 17 below shows the results comparing the Wait Phase and Rejuvenation Phase.

General Discussion

Observing CPU results of Stress Phase in Figure 10, it is possible to notice a similar behavior as presented in (Matos et al., 2012). Still, in Stress Phase, RAM results (Figure 11) and Web Server response time results (Figure 12) show similar results as on (Grottke et al., 2006).

As software aging causes internal software state degradation, its consequences may persist on software state until software rejuvenation (or failure) occurs (as explained in *SWARE Methodology* section). Results of Wait Phase shows a degraded software state. The numerical analysis reveals that, without rejuvenation actions, the system only achieves a normal operation state (with Response Time in 1.063 ms) after 12 days. It is important to highlight that this estimative considers that the workload is constant and of 2000 requests per second.

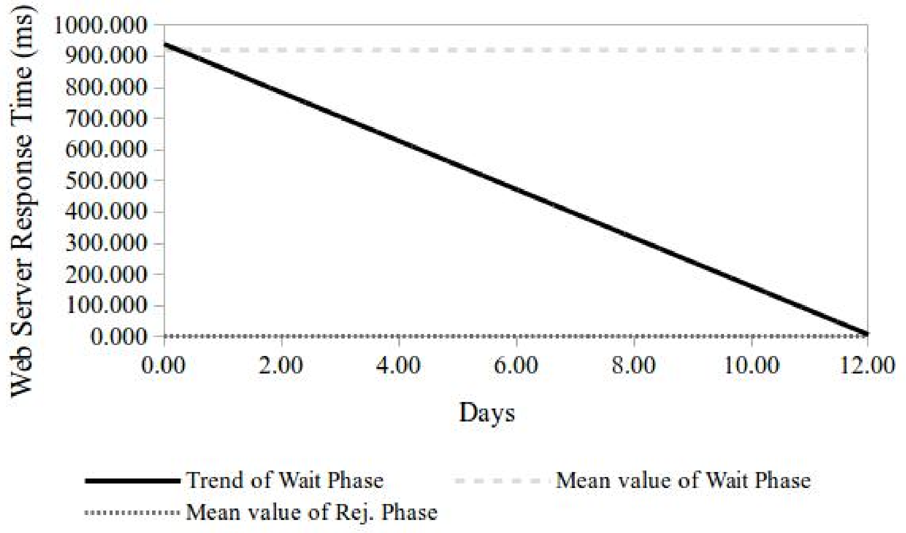

Figure 17. Comparison of trend analysis and Wait and Rejuvenation Phases results

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.

RELATED WORK

The authors of (Matias et al., 2010) 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 a statistical design of experiment. We also applied accelerated tests on the Stress Phase of our experiment. Different from the authors of the mentioned paper, the SWARE approach also comprises software rejuvenation.

The paper (Li et al., 2002) 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 primary 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. The SWARE methodology is simpler to apply as it does not require statistical expertise.

The paper (Grottke et al., 2006) provide valuable inputs on how to perform software aging experiments on 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 (Matias et al., 2006) offers 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 into the SWARE methodology.

Cotroneo et al. (Cotroneo et al., 2010) provide 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. (Matias et al., 2010) 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 (Silva et al., 2006) shows a study of software aging and rejuvenation in a 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. The SWARE approach aims to be more generic and flexible to other types of software.

The paper (Melo et al., 2017) 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 terminates 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 (Meng et al., 2016) 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.

CONCLUSIONS AND FUTURE WORK

This paper presented the SWARE methodology. The SWARE is a comprehensive approach to investigating aging symptoms and rejuvenation effectiveness on software systems. The 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.

Case Study section 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 numerical analysis shows the relevance of the VM Live Migration as a rejuvenation mechanism. Using mathematical approximations, we conclude that the system may take more than 12 days to recover from aging effects. The VM Live Migration usually takes less than 10 seconds. Therefore, the system can improve service quality by applying this technique.

The major contribution of this paper is a generic methodology to conduct software aging and rejuvenation effectiveness investigation. The case study proposed consists of a Cloud Computing environment. But, the methodology phases guidelines can be reproduced in other types of software systems. This paper also presents (as a case study) 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, Virtualized Containers and Fog Computing. Other research lines seek to improve approach adding multiple component software aging investigations.

ACKNOWLEDGEMENTS

This paper is an extended and improved version of our previous works (Torquato et al., 2017) and (De Melo et al., 2017).

REFERENCES

Araujo, J., Matos, R., Maciel, P. and Matias, R. (2011). Software aging issues on the eucalyptus cloud computing infrastructure. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on* (pp. 1411-1416). IEEE. https://doi.org/10.1109/ICSMC.2011.6083867

Araujo, J., Matos, R., Maciel, P., Matias, R. and Beicker, I. (2011). Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In *Proceedings of the Middleware 2011 Industry Track Workshop* (pp. 4). ACM. https://doi.org/10.1145/2090181.2090185

CDW. (2015). Cdw's cloud 401 report. *CDW, Report.*

CISCO. (2012). Cisco global cloud networking survey summary and analysis of results worldwide results. *CISCO, Tech. Rep.*

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., ... and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2* (pp. 273-286). USENIX Association.

Cotroneo, D., Natella, R., Pietrantuono, R. and Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1), 8. https://doi.org/10.1145/2539117

Cotroneo, D., Natella, R., Pietrantuono, R. and Russo, S. (2010). Software aging analysis of the linux operating system. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on* (pp. 71-80). IEEE. https://doi.org/10.1109/ISSRE.2010.24

de Melo, M. D. E. T., Araujo, J., Umesh, I. M. and Maciel, P. R. M. (2017). SWARE: An approach to support software aging and rejuvenation experiments. *Journal on Advances in Theoretical and Applied Informatics*, 3(1), 31-38. https://doi.org/10.26729/jadi.v3i1.2441

Garg, S., Van Moorsel, A., Vaidyanathan, K. and Trivedi, K. S. (1998). A methodology for detection and estimation of software aging. In *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on* (pp. 283-292). IEEE. https://doi.org/10.1109/ISSRE.1998.730892

Grottke, M., Matias, R. and Trivedi, K. S. (2008). The fundamentals of software aging. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on* (pp. 1-6). IEEE. https://doi.org/10.1109/ISSREW.2008.5355512

Grottke, M., Li, L., Vaidyanathan, K. and Trivedi, K. S. (2006). Analysis of software aging in a web server. *IEEE Transactions on Reliability*, 55(3), 411–420. https://doi.org/10.1109/TR.2006.879609

Huang, Y., Kintala, C., Kolettis, N. and Fulton, N. D. (1995). Software rejuvenation: Analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers, Twenty-Fifth International Symposium on* (pp. 381-390). IEEE.

Kim, W. (2009). Cloud computing: Today and tomorrow. *Journal of object technology*, 8(1), 65-72. https://doi.org/10.5381/jot.2009.8.1.c4

Li, L., Vaidyanathan, K. and Trivedi, K. S. (2002). An approach for estimation of software aging in a web server. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n* (pp. 91-100). IEEE. https://doi.org/10.1109/ISESE.2002.1166929

Machida, F., Kim, D. S. and Trivedi, K. S. (2013). Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration. *Performance Evaluation*, 70(3), 212-230. https://doi.org/10.1016/j.peva.2012.09.003

Machida, F., Andrzejak, A., Matias, R. and Vicente, E. (2013). On the effectiveness of Mann-Kendall test for detection of software aging. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on* (pp. 269-274). IEEE. https://doi.org/10.1109/ISSREW.2013.6688905

Mann, H. B. (1945). Nonparametric tests against trend. *Econometrica: Journal of the Econometric Society*, 245-259. https://doi.org/10.2307/1907187

Matias, R., Barbetta, P. A., Trivedi, K. S. and Freitas Filho, P. J. (2010). Accelerated degradation tests applied to software aging experiments. *IEEE Transactions on reliability*, 59(1), 102-114. https://doi.org/10.1109/TR.2009.2034292

Matias, R. and Paulo Filho, J. F. (2006). An experimental study on software aging and rejuvenation in web servers. In *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International* (vol. 1, pp. 189-196). IEEE. https://doi.org/10.1109/COMPSAC.2006.25

Matias, R., Beicker, I., Leitão, B. and Maciel, P. R. M. (2010). Measuring software aging effects through OS kernel instrumentation. In *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop on* (pp. 1-6). IEEE. https://doi.org/10.1109/WOSAR.2010.5722094

Matos, R., Araujo, J., Alves, V. and Maciel, P. (2012). Characterization of software aging effects in elastic storage mechanisms for private clouds. In *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on* (pp. 293-298). IEEE. https://doi.org/10.1109/ISSREW.2012.82

Melo, M., Maciel, P., Araujo, J., Matos, R. and Araujo, C. (2013). Availability study on cloud computing environments: Live migration as a rejuvenation mechanism. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on* (pp. 1-6). IEEE.

Melo, M., Araujo, J., Matos, R., Menezes, J. and Maciel, P. (2013). Comparative analysis of migration-based rejuvenation schedules on cloud availability. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on* (pp. 4110-4115). IEEE. https://doi.org/10.1109/SMC.2013.701

Melo, M. D. (2014). Modelos de disponibilidade para nuvens privadas: Rejuvenescimento de software habilitado por agendamento de migracao de vms.

Melo, C., Araujo, J., Alves, V. and Maciel, P. R. M. (2017). Investigation of Software Aging Effects on the OpenStack Cloud Computing Platform. *JSW*, 12(2), 125-137.

Meng, H., Hei, X., Zhang, J., Liu, J. and Sui, L. (2016). Software aging and rejuvenation in a j2ee application server. *Quality and Reliability Engineering International*, 32(1), 89-97. https://doi.org/10.1002/qre.1729

Mosberger, D. and Jin, T. (1998). httperf—a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3), 31-37. https://doi.org/10.1145/306225.306235

Parnas, D. L. (1994). Software aging. In *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on* (pp. 279-287). IEEE. https://doi.org/10.1109/ICSE.1994.296790

Sen, P. K. (1968). Estimates of the regression coefficient based on Kendall's tau. *Journal of the American statistical association*, 63(324), 1379-1389. https://doi.org/10.1080/01621459.1968.10480934

Silva, L., Madeira, H. and Silva, J. G. (2006). Software Aging and Rejuvenation in a SOAP-based Server. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on* (pp. 56-65). IEEE.

Torquato, M., Araujo, J. and Maciel, P. (2015). 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 Aplicacoes in conjunction with 33rd Brazilian Symposium on Computer Networks and Distributed Systems (SBRC2015). Vitoria, ES, Brazil: Universidade Federal do Esprito Santo (UFES)* (pp. 1-14). https://doi.org/10.23919/CISTI.2017.7975806

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

Umesh, I. and Srinivasan, G. N. (2017). Dynamic software aging detection-based fault tolerant software rejuvenation model for virtualized environment. In *Proceedings of the International Conference on Data Engineering and Communication Technology*. Springer, pp. 779–787. https://doi.org/10.1007/978-981-10-1678-3_75

Umesh, I. M., Srinivasan, G. N. and Torquato, M. (2017). Software Aging Forecasting Using Time Series Model. *Indonesian Journal of Electrical Engineering and Computer Science*, 7(3), 839-845.

Umesh, I. M., Srinivasan, G. N. and Torquato, M. (2017). Software Rejuvenation Model for Cloud Computing Platform. *International Journal of Applied Engineering Research*, 12(19), 8332-8337.

Vaidyanathan, K. and Trivedi, K. S. (2001). Extended classification of software faults based on aging. In *Fast Abstract, Int. Symp. Software Reliability Eng., Hong Kong.*

Valentim, N. A., Macedo, A. and Matias, R. (2016). 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* (pp. 57-63). IEEE. https://doi.org/10.1109/ISSREW.2016.42