An Integrated Platform for Distributed Resources Monitoring and Software Aging Mitigation in Private Clouds

Jean Araujo[1,2*], Céfanys Braga[1], José Belmiro Neto[1], Adriano Costa[1], Rubens Matos[3], Paulo Maciel[3]

[1] Academic Unit of Garanhuns, Federal Rural University of Pernambuco, Garanhuns, Brazil.

[2] Informatics Center, Federal University of Pernambuco, Recife, Brazil.
[3] Federal Institute of Education, Science, and Technology of Sergipe, Lagarto, Brazil

* Corresponding author. Email: jcta@cin.ufpe.br
Manuscript submitted May 15, 2016; Accepted August 11, 2016.
doi: 10.17706/jsw.11.10.976-993

Abstract: The use of distributed systems has become increasingly diffused and accessible, especially cloud computing, which is used to deliver computational services to small, medium, and large enterprises. Building and maintaining a private cloud infrastructure may not be trivial, due to requirements of being accessible from anywhere and at any time. Therefore, monitoring and management activities are essential to identify and mitigate failures that may occur, and thus keeping service availability. Several studies show that some failures are caused by a phenomenon called software aging, which refers to the system's degradation during its operational life. This paper describes the design and implementation of the DR Monitor, an integrated platform to monitor and mitigate software aging effects in private cloud computing infrastructures. Several test scenarios demonstrate the operation and effectiveness of this tool, which empowers the proactive management of distributed computing resources.

Key words: Cloud computing, software aging and rejuvenation, monitoring, distributed resources.

1. Introduction

The use of distributed systems have significantly increased over the years due their high processing power and low cost when compared to supercomputers. Much of the recent growth is due to the advent of cloud computing. Cloud computing is a computational model that allows the end user to access a pool of applications and services anywhere with the same usefulness found when these applications and services are installed in equipment with local and direct access.

This computational model relies on three groups: provider, developer, and end user. The provider is responsible for building, offering, managing, and monitoring the entire cloud infrastructure. The developer must be able to build and configure services for the end user by employing the infrastructure offered by the provider. The end user is the one who will use the resources offered in the cloud [1]. Among the provider responsibilities, the management and monitoring tasks deserve special and continuous attention because the systems may present performance degradation or instability over time, causing problems such as partial interruption or even total service unavailability. However, management and monitoring tasks are not trivial and may be very hard to perform manually.

Among the problems that might occur in computational infrastructures, we can highlight those arising from software aging, where the accumulation of errors along the software runtime may cause performance degradation, software failures, or still the system crash [2]. A counteraction might be planned and performed to mitigate the effects caused by software aging, leading to the so-called software rejuvenation. In short, software rejuvenation might be achieved by stopping the affected application, cleaning your internal state and restarting it [3].

Researchers worldwide have investigated the factors of software aging and, in some cases, proposed rejuvenation action in various environments, such as web servers [4], [5], OS kernel [6], clustered systems [7], telecommunication billeting applications and safety-critical military equipment [8], [9], cloud computing environments [10], [11], Android applications [12], and video streaming player [13]. Furthermore, different techniques have been adopted to improve aging detection and rejuvenation strategies, such as machine learning [14]–[17], time series [18], [19], neural network [20], [21], two-level rejuvenation strategy [22], and live migration mechanism [23]. Many other software aging and rejuvenation studies can be found in [24], and a collection of research artifacts is available in SAR repository [25].

In this context, systems administrators depend heavily on software tools for assisting in the management and monitoring of distributed resources, as in the case of cloud computing, in order to automate the involved activities. This paper presents DR Monitor, that is a tool designed to monitor and manage distributed computer systems, helping to identify software aging-related symptoms, and that allows the use of rejuvenating actions. All relevant details of the tool, from conception to implementation and testing, are presented in this paper. This tool combining the aging detection with rejuvenation action in a single environment. An experimental testbed in a Eucalyptus private cloud environment demonstrates the efficiency and applicability of DR Monitor.

This paper is structured as follows. Section 2 discusses about the fundamental concepts of software aging and rejuvenation. The DR Monitor is presented in detail in Section 3, comprising the architecture, features and communication details. Case studies are presented in Section 4, demonstrating results obtained using DR Monitor in distinct testbed scenarios. Section 5 introduces some related works related to tools and approaches for monitoring distributed systems. Finally, Section 6 draws conclusions and also provides some possible future works.

2. Software Aging and Rejuvenation

Software aging can be defined as a growing degradation of software's internal state during its operational life [26]. The causes of software aging have been verified as the accumulated effect of software faults activation [27] during the system runtime [28], [29]. Aging in a software system, as in human beings, is an accumulative process [28], [30]. The accumulating effects of successive error occurrences directly influence the aging-related failure manifestation. Software aging effects are the practical consequences of errors caused by aging-related fault activations [5]. These faults gradually lead the system towards an erroneous state [28], [30].

Due to the cumulative property of the software aging phenomenon, it occurs more intensively in continuously running software systems that are executed over a long period of time [31], such as cloud-computing framework software components [6]. In a long-running execution, a system suffering from software aging increases its failure rate due to the aging effect accumulation caused by successive aging-related error occurrences, which monotonically degrades the system internal state integrity [5], [31]. Problems such as data inconsistency, numerical errors, and exhaustion of operating system resources are examples of software aging consequences [2], [26].

Once the aging effects are detected, mitigation mechanisms might be applied in order to reduce the

impact of the aging effects on the applications or the operating system. The search for aging mitigation approaches resulted in the so-called software rejuvenation techniques [5], [32].

Since the aging effects are typically caused by hard to track software faults, software rejuvenation techniques look for reducing the aging effects during the software runtime, until the aging causes (e.g., a software bug) are fixed definitively [26], [28]. Examples of rejuvenation approaches may be software restart or system reboot. In the former, the aged application process is killed and then a new process is created to substitute it [5]. Replacing an aged process by a new one, we remove the aging effects accumulated during the application runtime. The same applies to the operating system in a system reboot.

A common problem during rejuvenation is the downtime caused during restart or reboot, since the application or system is unavailable during the execution of the rejuvenation action. In [5] is presented a zero-downtime rejuvenation technique for the Apache web server, which was adapted for this work. Rejuvenation action either can occur on a regularly scheduled (such as once every ten days, for example) or as needed, triggered by the monitoring of resource exhaustion or performance degradation.

3. DR Monitor: A Distributed Resources Monitor

This section is dedicated to present the details of DR Monitor, providing information about its architecture, features and some important details regarding communication modules.

The DR Monitor tool was developed to automate the monitoring activities in distributed systems, and has specific functions for private cloud environments. It is able to collect data about system-wide utilization of resources from a given machine (e.g., CPU utilization, RAM usage, disk, bandwidth, etc.). The tool also monitors the consumption of resources by specific applications. It is possible to identify some symptoms of software aging by evaluating user-informed parameters and comparing them to data collected in real-time. Additionally, the tool provides the ability to configure and perform basic rejuvenation actions. Figure 1 shows the main screen of DR Monitor tool, where the user may access the monitoring features, the chart generator features, or the tool's manual.

Fig. 1. DR Monitor main screen.

3.1. Architecture

The solution presented here is based on a client-server architecture, where a node - called Central Node (CN) - makes requests to one or more nodes in a network, these in turn are called Distributed Nodes (DN). Both components have support for TCP and UDP communication. Figure 2 shows the components and their

interactions within the adopted architecture.

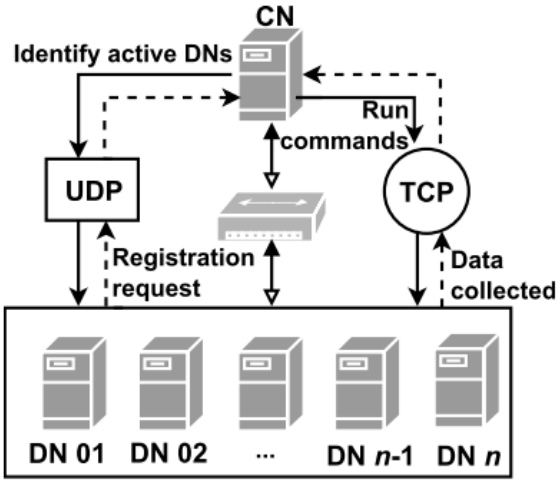


Fig. 2. DR Monitor architecture.

The CN is responsible for identifying, registering and managing the DNs. A broadcast message is sent through the network, and the active DNs respond with a registration message to the CN. Such activities are carried out through the exchange of UDP messages. Upon receiving a registration request, the CN identifies the name and IP address of the DN that requested the registration, and adds it to a list. From this moment on, the communication between CN and DNs is performed using TCP. The CN issues commands for DNs collecting resources utilization, stores such data received from DNs, process this information for user visualization, and identify actions that might be taken based on configured parameters.

Only DNs that follow the strict UDP message format of DR Monitor can be registered and send data to the the CN. So any other message with a format that is different from that expected is discarded.

3.2. Features

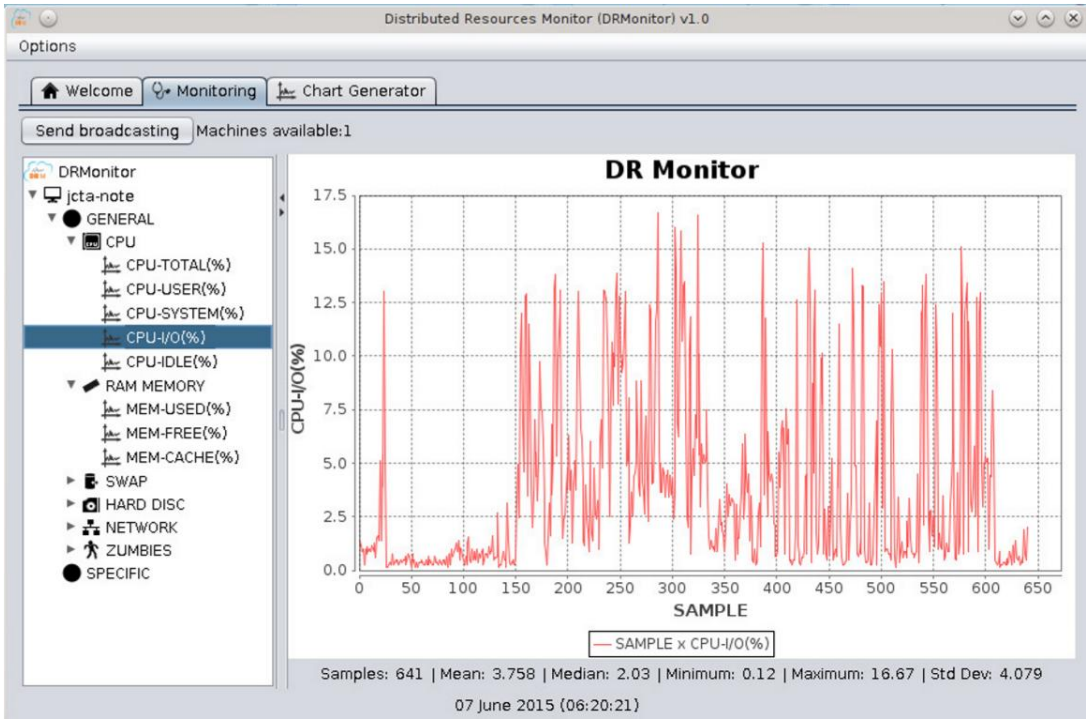


Fig. 3. Monitoring screen — Single chart.

This tool has several features that can be used in any Linux-based distributed system, and some specifics for private cloud computing environments. One is to allow real-time monitoring from machines on a network segment. Monitoring can be local or distributed. The local monitoring collects data only from the CN machine and saves to log files. The distributed monitoring collects data of active DNs over the network and saves data in the CN machine. An overview of DR Monitor monitoring screen can be seen in Figure 3, that shows a single chart for CPU utilization in user-level (CPU-USER). This metric belongs to the set of general resources, monitored by DN in the host called *jean-note*.

Fig. 4 shows multiple monitoring charts for all metrics related to the CPU utilization. This multi-chart feature allows the user analyzing many resources or metrics at a glance.

Fig. 4. Monitoring screen — Multi charts.

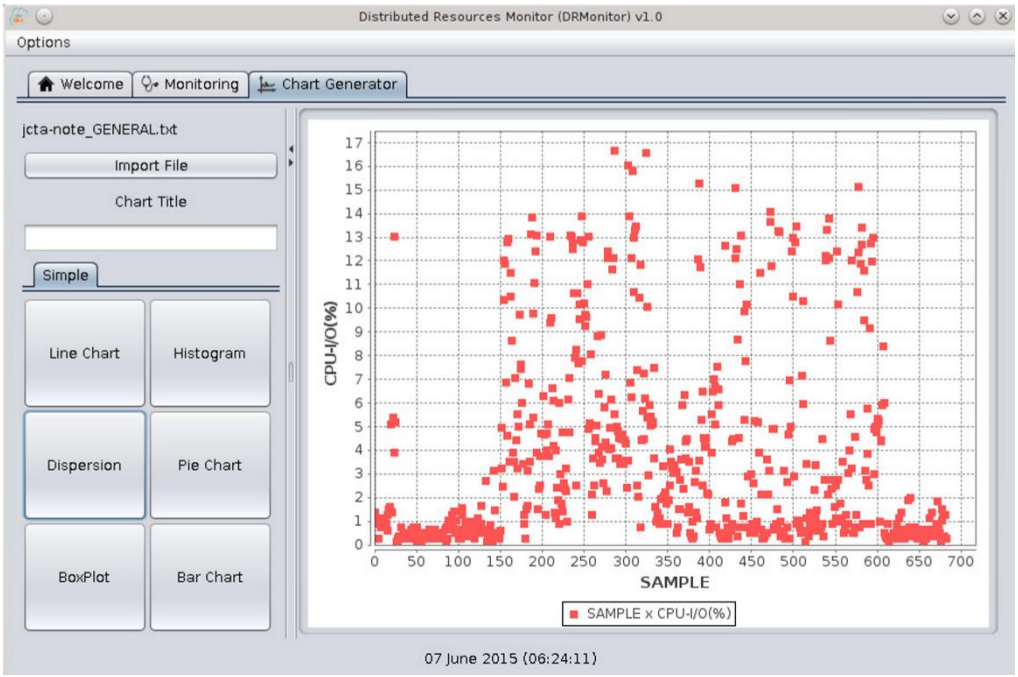


Fig. 5. Offline analysis — Dispersion chart.

On the left side of Fig. 3 and Fig. 4, it is possible to see the monitored components, organized in a tree structure. This hierarchical organization aids the association of monitored components with their respective machines. The right side shows a graph of the component that was selected in the tree. The bottom of the window has a brief statistical summary of data presented in the chart.

The DR Monitor is able to generate offline charts, as lines, histogram, dispersion, pie, box-plot, and bar charts. Fig. 5 shows an example of offline analysis using a dispersion chart.

DR Monitor classifies the monitored data into two types: general or specific. The general type refers to the system-wide utilization of resources, such as CPU utilization, memory, disk, network traffic, and number of zombie processes. When the user starts monitoring a machine, the general resources are requested by default in user-defined time intervals. The specific type refers to any running process on the monitored machine. The user chooses which processes will be monitored. The main resource consumption information of the process are collected, such as CPU utilization, virtual memory, resident memory, and total memory. After adding a machine from private network to the CN, the administrator will be able to monitor all of these features.

Data visualization occurs in real-time, as soon as the CN receives the information from the respective DN. The user can view monitored data through line graphs by simply clicking on the name of the desired resource.

The default graph contains a single data series. The tool also allows offline generation of graphs, that is a independent feature from real-time monitoring. Therefore, it is possible to analyze the data from previous monitoring logs, or any other data set structured in the format recognized by DR Monitor.

It is also possible to set a warning on resources consumption, both in general resources and in specific processes. Figure 6 illustrates a warning example.

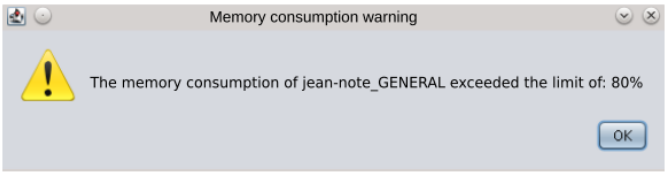


Fig. 6. Example of memory consumption warning.

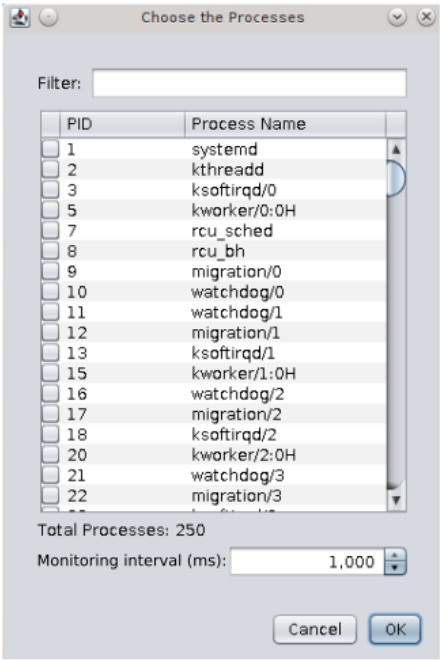


Fig. 7. Process selection.

The available processes on the machine to be monitored are listed in a table that contains the process identification number (PID) and its name. Fig. 7 shows the window with this table. It is allowed to choose one or more processes by marking the check box in the first column of the table. It is possible to set the monitoring interval and apply a filter by typing part of the name of the desired process.

Statistical information is automatically calculated when any graph is generated. The statistical measures available are: number of samples, arithmetic mean, median, the largest and the smallest value among the samples, and the standard deviation. All those measures are updated in real time.

The great advantage of this tool among the others is because it implements some software rejuvenation strategies. Our approach has characteristics of three rejuvenation categories: time-based, where the rejuvenation action is triggered periodically; threshold-based, where the rejuvenation action is triggered when a threshold has been reached; or in a hybrid way, driven by the one who comes first.

The tool has implemented two types of rejuvenation actions, a general and a specific. The general rejuvenation action is the reboot of the machine, i.e., DR Monitor sends the command "reboot -h now" to the DN target, and waits the computer restart to resume monitoring again. However, as much of the web services that manage cloud computing platforms are Apache processes, we developed a specific rejuvenation action for these processes. This action relies on the Linux signal "USR1". The CN orders the DN to run a command "kill -SIGUSR1 $pid", where the variable "$pid" is the PID of the target Apache process. This signal asks kindly that all child processes are terminated, and new processes are created [5], [33]. This action frees resources used by the child processes affected by software aging, while bringing fresh processes to service new incoming requests.

It is important to highlight that the rejuvenation action for Apache process does not causes disruption of services running on the server. Once this command arrives, the OS terminates the Apache child process, but waits while active connections are terminated. After that, creates a new child process to replace the one that was terminated, and transfer new service requests to the new process. This procedure may cause some delay in the processing of requests received during the implementation of the rejuvenation action, but does not cause service unavailability [5].

All rejuvenation actions are operations that require privileges in the operating system of the target machine. Therefore, the user must enter the password of a user with administrator privileges, if he wants to use the rejuvenation features of DR Monitor. Since this is sensitive information, and needs to travel on a computer network, the user password is encrypted. The encryption adopted in this tool uses the default implementation of the 1024-bits RSA algorithm [34], available in the Java library "*javax.crypto.Cipher*".

3.3. Communication Modules

The CN module comprises the graphical environment that allows managing a set of DNs. Its operation is based on network communication via TCP and UDP sockets. The UML activity diagram in Figure 8 shows the steps performed for data collection through the sockets.

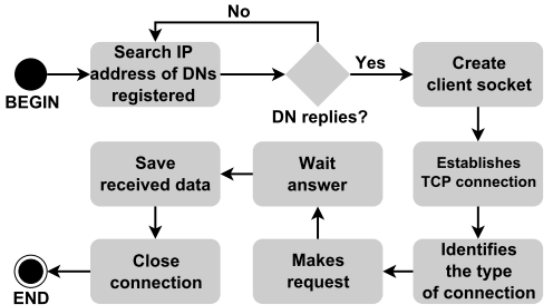


Fig. 8. Activity diagram of CN module.

The identification of available machines is required to support the monitoring features. This task is implemented in a UDP socket, which sends broadcast messages so the active DNs reply, identifying themselves and being registered in the CN. The CN searches for the DNs registered via their IPs. If any DN responds, the CN creates a client socket for communication and establishes a TCP connection. The CN identifies the type of request that is necessary in that moment, and makes that request by sending the default message. CN waits the DN response, which must contain the monitoring data. Upon response arrival, it saves the received data, and finally closes the connection.

The DN module is designed to operate using the minimum resources possible and, thus, not influence or interfere with the monitoring process results. Therefore, the implementation of this tool was guided by the requisite of a lightweight and optimized code. The first design decision for this module was to use Bash scripts [35] to collect data of resource consumption, information of running processes, and even to perform rejuvenation actions. The Bash scripts enabled achieving the intended lightweight code.

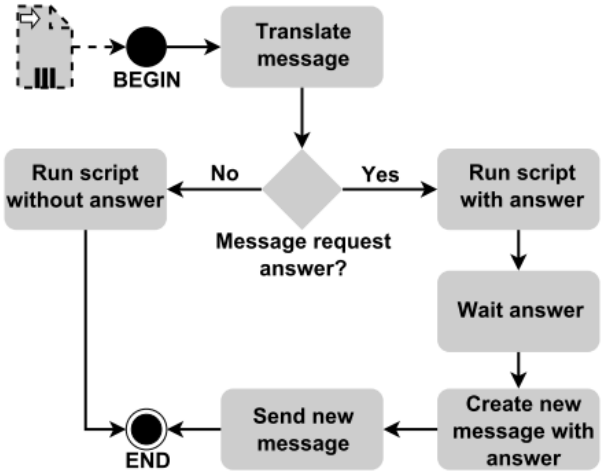


Fig. 9. Activity diagram of DN module.

The remote collector in DN module also implements UDP and TCP sockets, but with different functions. The service that implements UDP connections of this module only identifies the message sent by the CN, stating that it is present on the network, and waits for commands. So, the hostname and the IP address associated with it are formatted in a log message and sent as a response to the CN. The DN has a socket implemented to function as a server in TCP connections, which receives requests by the CN. Figure 9 shows the steps performed with the socket for execution of requests sent by the CN.

The DN module translates the messages sent by CN, and runs the script corresponding to the received request. If the request needs an answer, then the DN waits for data returned by the script, creates an answer message, and sends it to the CN. And, for the cases in which there is no need to send an answer, the DN finishes the current request as soon as the script completes its execution. The DN runs as a background process and can be configured to start during operating system boot time. The DN module does not provide a graphical user interface. This helps in keeping a small application complexity and memory footprint to achieve good performance in its activities.

4. Case Studies

This section describes some testbed scenarios designed to evaluate the main features of DR Monitor and the efficiency of the tool. Every experiment of these case studies executed for twenty-four uninterrupted hours. The testbed environment, adopted workload, and obtained results are presented as follows for each

scenario.

4.1. Scenario 01: Tool's Intrusiveness Test

This scenario was performed to verify if the influence caused by the developed tool on the resources utilization can be characterized as non-intrusive. It is divided in two sub-scenarios: 1A and 1B. In Scenario 1A, the goal is to monitor a cloud system using only a Bash script running locally. Such an strategy has been adopted in previous studies to monitor computer resources in cloud computing environments [10, 18, 33, 36]. In Scenario 1B, the same system is monitored by the DR Monitor. In order to enable a direct and fair comparison, both scenarios were carried out with same hardware and software conditions.

The testbed of this scenario employed a desktop machine (Intel Core 2 Duo, CPU 2.93GHz, 4 GB of RAM, 500 GB SATA hard disk, and a Fast Ethernet NIC) which was monitored in both sub-scenarios, using the Ubuntu Linux 14.04 LTS 64-bit, and another desktop machine (Intel Core i5, CPU 2.7GHz, 4 GB of RAM, 500 GB SATA hard disk, and a Fast Ethernet NIC) that was used as the CN for the Scenario 1B, using the Microsoft Windows 8.1.

The adopted workload comprised a video playback in high resolution, because this media type requires high amounts of CPU time and RAM, and does not need complex configurations. The VLC media player was chosen for such a task because it is open-source, easily configured and widely compatible with most file formats and codecs used for video encoding. During the experiment, the monitored computer was playing a video in Full HD resolution (1080p) in Matroska format (.MKV), located in the hard drive, with the repeat function activated in VLC.

We ensured that the experiments were performed under the same conditions in both scenarios. The computer was turned off after the first experiment, remaining so for a period of about two hours. This procedure enabled recovering all initial hardware temperature conditions and resetting the utilization of resources for usual boot time levels. The experiment was started immediately after complete OS loading.

Fig. 10 shows the CPU utilization measured with both strategies in distinct experiments. It is worth emphasizing that the machine was running the mentioned video workload. There is an overlapping of results from both scenarios, implying that the two monitoring strategies provide similar results. Although it is possible to identify that the lowest CPU utilization values measured with Bash scripts are higher than the lowest values measured with DR Monitor. This might indicate a slightly lower interference of DR Monitor.

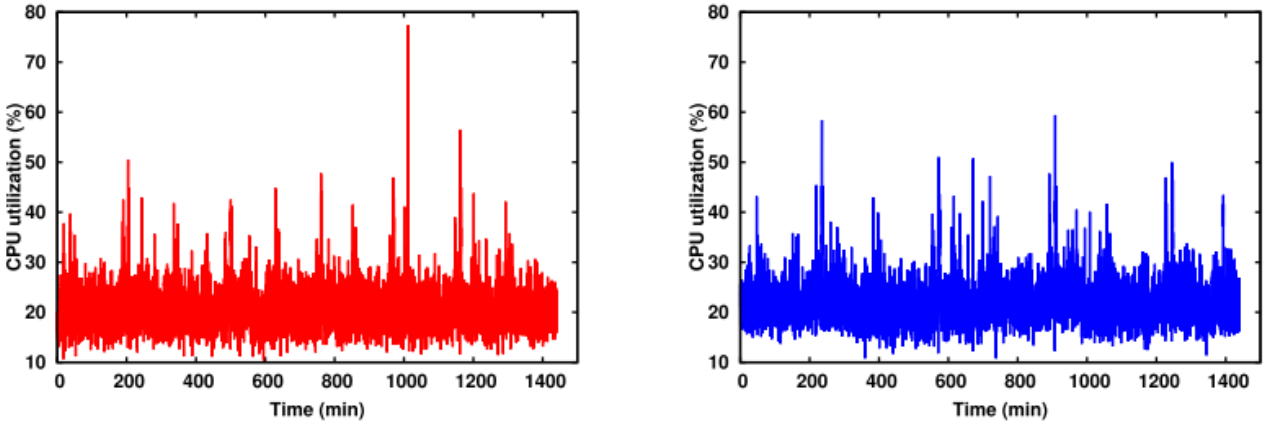
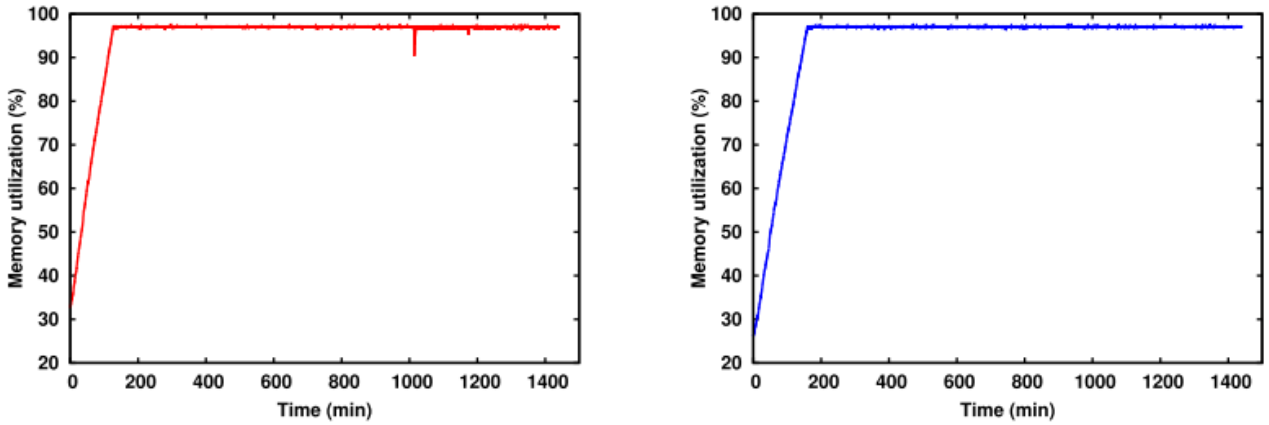


Fig. 10. Comparison of CPU utilization: DR Monitor vs Bash script.

Fig. 11 depicts the system memory usage in the same experiments considered in Fig. 10. The experiment with DR Monitor starts with slightly higher memory usage than that with the Bash scripts, but it remains

stable from around 200 minutes until the end of the experiment



(a) DR Monitor (b) Bash script

Fig. 11. Comparison of memory utilization: DR Monitor vs Bash script.

A statistical summary of these experiments is shown in Table 1. Considering the CPU utilization, the mean values measured by DR Monitor are approximately 7% lower than in those measured with Bash scripts. Moreover, the standard deviation and the standard error of the mean value in both approaches shows that they can be considered equivalent. The first quartile, third quartile, and minimum value of CPU utilization with DR Monitor are lower than the same metrics with Bash script. This is one more indicative of the slightly lower interference of DR Monitor on CPU utilization, that was previously noticed in Figure 10.

Considering the memory utilization, DR Monitor mean value is about 1.1% higher than the Bash script result. This fact might be explained by the use of JVM by DR Monitor whereas the Bash script requires only a Bash interpreter, that has a smaller memory footprint than JVM. Either way, considering the standard deviation, the difference is too small to be considered significant

4.2. Scenario 02: Time-Based Rejuvenation Test

This scenario was designed to show in a simple and practical way the functioning of the time-based rejuvenation strategy that was implemented in the tool. This feature consists in periodically triggering a rejuvenation action. For this, the most common rejuvenation strategy was adopted, that is the reboot of the computer.

Table 1. Statistical Summary

Metrics	CPU (%)		Memory (%)	
	Script	DRMonitor	Script	DRMonitor
Mean	22.198	20.666	93.217	94.288
Std error (mean)	0.0402	0.0401	0.129	0.105
Median	21.83	20.2	97.042	97.042
Standard deviation	4.016	4.014	12.948	10.493
First quartile	19.7	18.18	96.885	96.885
Third quartile	24.12	22.5	97.199	97.199
Minimum value	10.77	10.4	26.256	32.67
Maximum value	59.41	77.39	97.382	97.382

The hardware and software configuration adopted in this scenario is identical to the setup for Scenario 01. The rejuvenation action was scheduled to run every three hours. This scenario does not comprise any

workload.

Fig. 12 shows the memory usage of the monitored computer during the experiment. A visual analysis of the figure reveals the moments when the system reboots. The memory consumption decreases when the rejuvenation action is triggered, causing several breaks in the line throughout the experiment. This behavior meets the expected results of DR Monitor's rejuvenation feature, that is clearing possible effects of memory leaks or other wasted resources that aged software might leave in the system.

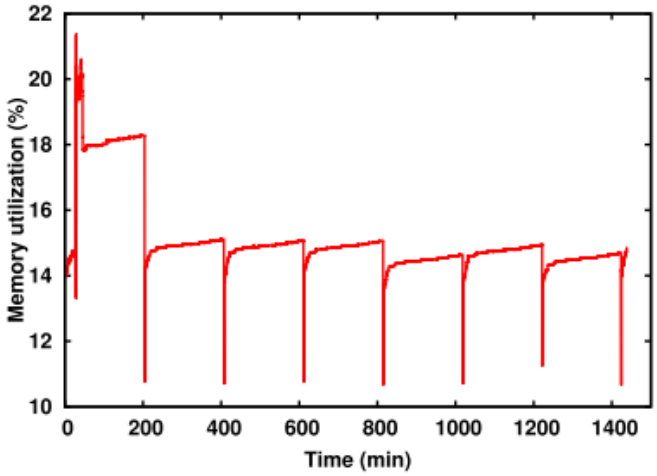


Fig. 12. Memory utilization during the time-based rejuvenation test.

It is important to highlight that the very simple design of this scenario was deliberate, since the purpose was not the identification of software aging effects but only showing that the implemented functionality works as expected. Considering this objective, there was no need for running a workload on the system.

4.3. Scenario 03: Private Cloud Rejuvenation Test

The goal of this scenario is testing the tool in a private cloud infrastructure with software aging effects. The tool monitors the distributed resources and triggers software rejuvenation actions that are especially designed to mitigate the effects of aging in Eucalyptus NC processes reported in [10], [33].

The testbed scenario comprises one computer (Intel Core i5, CPU 2.8GHz, 4 physical cores, 8 GB of RAM, 320 GB SATA hard disk, and a Gigabit Ethernet NIC) as DN, running the OS CentOS Linux 6.4 and the Eucalyptus System version 3.3.1 in cloud-in-a-box installation mode, and another desktop machine used as the CN, with the same configuration of Scenario 01.

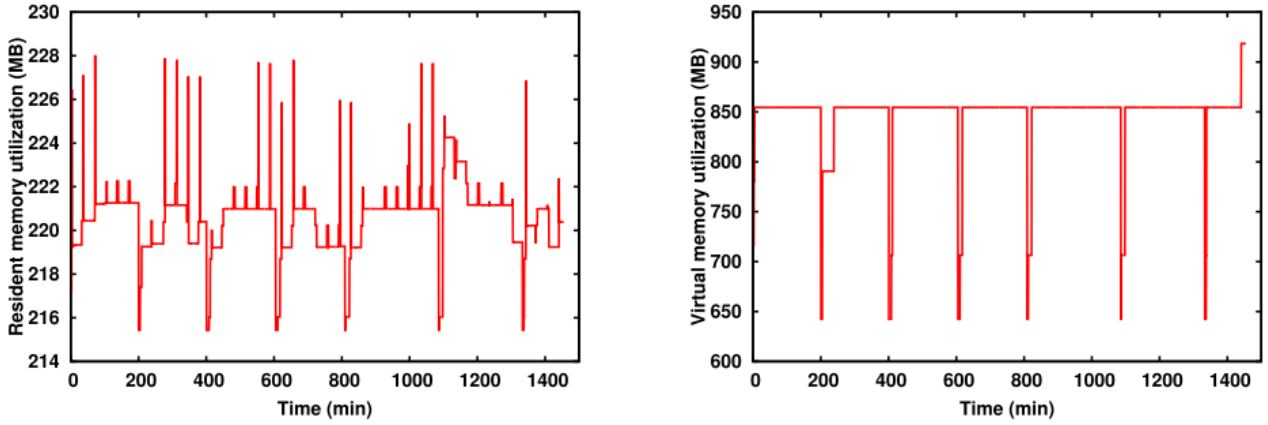
The workload used in this scenario was adapted from the VM life-cycle workload proposed in [10], consisting in start, reboot, and terminate virtual machines periodically. Previous studies [10], [19], [37] show that this kind of workload causes software aging in Eucalyptus node controller processes, characterized by increased memory usage up to the point where the system crashes in some cases.

We created a shell script that automates the workload. The script follows a cycle that instantiates two virtual machines of type "m1.xlarge", reboots these VMs every two minutes, and terminates them every two hours. Finally, the script waits five minutes to initiate a new cycle that is identical to that described.

A time-based rejuvenation action was taken every 200 minutes counted from the last run. The DR Monitor tool sends the signal "kill -SIGUSR1 $pid" to the Node Controller process, that is an Apache process responsible for the management of VMs in Eucalyptus clouds, creating a new process and terminating the old one.

Fig. 13 shows the usage of resident memory and virtual memory of the Node Controller process. Fig. 13(a) shows that there was a slight decrease in the resident memory utilization of NC process when the

rejuvenation action was triggered. There are several memory usage peaks throughout the experiment, ranging from 213 MB to 228 MB. In Figure 13(b), it is possible to identify falls in the virtual memory utilization of about 220 MB when the rejuvenation action is triggered. This behavior can be observed throughout the experiment. This shows that this functionality of DR Monitor works effectively.



(a) Resident memory (b) Virtual memory

Fig. 13. Memory utilization of NC process during rejuvenation test.

5. Related Works

The development of DR Monitor demanded an analysis of some existing monitoring tools. The analysis comprised mainly open-source applications due the good documentation, notoriety in forums and scientific articles, and possibility of testing their main features.

Google's Borg system [38] is a cluster manager that offer a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures.

Cacti [39] is a web application for networks monitoring that serves as a front-end to RRDTool [40]. Cacti stores all monitored information in a database for creation of graphs. Its interface is based on PHP. Scripts or external commands do the data collection. Once collected, such data can be used to create any graphs supported by RRDTool. Cacti also enables choosing how the graphics will be displayed. Besides, it supports data collection by Simple Network Management Protocol (SNMP). The tool also allows the creation and management of users. There are also many plug-ins for adding other features to Cacti.

Ganglia [41] is a distributed and scalable monitoring system for high-performance computing systems such as clusters and grids. It was created from a project of the University of California, Berkeley, using technologies such as XML, for data representation; XDR, to transport data in a compact and portable form; and RRDTool, for storage and data visualization [42]. According to [42], the system sends messages to a multicast address. If the node does not receive the message within a certain period of time, the node is classified as unavailable. Its developers say the system can scale up to two thousand nodes.

Apache Mesos [43] is an open-source cluster manager that provides resource isolation and sharing across distributed applications. The software enables resource sharing in a fine-grained manner, improving cluster utilization. The main design question for Mesos is how to build a scalable and efficient system that supports a wide array of both current and future frameworks. Its first version was named Nexus [44].

Munin [45] is another monitoring tool that uses the resources provided by RRDTool. It is a tool for network resources monitoring written in the Perl language. The operation is simple, and consists of a master/node architecture, where the master regularly connects to all nodes at regular intervals and asks

them the data. The default installation monitors basic information, such as CPU usage, memory, disk, and others. Munin developers emphasize that the tool focus is to be plug-and-play. There is still the possibility of installation of various monitoring plug-ins to improve its functionality.

Nagios [46] is designed to be scalable and flexible, with a powerful tool for monitoring networks. Nagios is a tool used to diagnose, prevent and deal with network problems. With a wide range of features, including a number of web interfaces, rich in monitoring tools and a large number of available plug-ins, which means it can be customized according to the user's needs [47]. All information, such as current state, history, and reports can be accessed through a web browser. It is possible to use it in Linux and UNIX systems. According to bin Mohd Shuhaimi *et al*. [47] and Issariyapat *et al*. [48], its installation and use should be only performed by experienced system administrators.

Supermon [49] is a high-speed cluster monitoring system that emphasizes low interference, high sampling rates, a data protocol and an extensible programming interface [50]. Its operation uses a client/server architecture. The authors emphasize that, although it is used in Linux-based clusters, their infrastructure is not limited only to this OS.

Zabbix [51] is an open source monitor. It is designed mainly to check availability and components performance, like user applications, network equipment, and operating systems such as Windows, Linux, and MacOS. Installation and usage require a high degree of knowledge and experience, including having a specific certification for such. Its architecture uses basically two components: Zabbix proxy, responsible for receiving, storing and managing the data of the monitored components, and is installed on a server; and Zabbix agent, that is responsible for collecting information on the target monitoring machines. The hardware and software requirements vary depending on the number of components to be monitored.

Table 2 shows a comparison of DR Monitor features and those ones found in the tools mentioned in this section. It is possible to observe that DR Monitor provides, in an integrated way, monitoring and rejuvenation features without requiring third-party plug-ins or custom implementation. In addition, it is also important to consider the effort and knowledge required for installation, configuration and use of most tools. DR Monitor does not require a long time to install and learn how to use it. This tool also exempts the user from advanced knowledge about its dependencies on specific programming libraries.

Table 2. Tool's Comparison

Feature	Borg	Cacti	Ganglia	Mesos	Munin	Nagios	Supermon	Zabbix	DRMonitor
Local monitoring	✓	✓	✓	✓	✓	✓	✓	✓	✓
Distributed monitoring	✓	✓	✓	✓	✓	✓	✓	✓	✓
Real-time charts	✓	✓	✓	✓	✓	✓	✓	✓	✓
Offline charts	✓	✓	✓	✓	✓	✓	-	✓	✓
Multi platform support	✓	✓	✓	✓	✓	✓	-	✓	✓
Warning of resources usage	✓	-	-	✓	-	✓	-	✓	✓
Native rejuvenation action	-	-	-	-	-	-	-	-	✓

On the other hand, the works presented in [52] and [53] bring a more simple and efficient vision in monitoring strategies developed especially for cloud environments. However, as these platforms are not publicly available for testing, they were not included in the features comparison.

6. Final Remarks

This paper proposed a tool to automate and support the monitoring and management of distributed resources, with special focus on software aging mitigation in private clouds. The DR Monitor is the result of integrating various technologies, programming languages, and concepts, thereby creating a tool which displays the consumption of various distributed resources in real time, thus improving the decision-making and further preventing unwanted events. The tests demonstrate the operation of the tool, indicating that it does not interfere in the monitored system or in the monitored results. In addition, it is able to apply rejuvenation actions in private clouds, improving aspects such as reliability, availability, and performance of those systems, which are directly affected by software aging phenomenon.

As future work, we intend to expand the tool's features. One important extensions is the implementation of resources utilization forecasting, through time series and similar techniques.

Acknowledgment

We would like to thank the Coordination of Improvement of Higher Education Personnel – CAPES, National Council for Scientific and Technological Development – CNPq, Foundation for Support to Science and Technology of Pernambuco – FACEPE, and MoDCS Research Group for their support.

References

[1] Marinos, A., & Briscoe, G. (2009). Community cloud computing. *Proceedings of the 1st International Conference on Cloud Computing* (pp. 472–484).

[2] Garg, S., Moorsel, A. V., Vaidyanathan, K., & Trivedi, K. S. (1998). A methodology for detection and estimation of software aging. *Proceedings of the 9th Int. Symp. on Software Reliability Engineering* (pp. 283–292).

[3] Dohi, T., Goseva-Popstojanova, K., & Trivedi, K. S. (2000). Analysis of software cost models with rejuvenation. *Proceedings of the Fifth IEEE International Symposim on High Assurance Systems Engineering* (pp. 25–34).

[4] Grottke, M., Vaidyanathan, K., & Trivedi, K. S. (2006). Analysis of software aging in a web server. *IEEE Transactions on Reliability*, *55(3)*, 411–420.

[5] Matias, R., & Freitas Filho, P. J. (2006). An experimental study on software aging and rejuvenation in web servers. *Proceedings of the 30th Annual International Computer Software and Applications Conference*.

[6] Matias, R., Beicker, I., Leitao, B., & Maciel, P. (2010). Measuring software aging effects through os kernel instrumentation. *Proceedings of the Symp. on Software Reliability Engineering, Software Aging and Rejuvenation*.

[7] Vaidyanathan, K., Harper, R. E., Hunter, S. W., & Trivedi, K. S. (2001). Analysis and implementation of software rejuvenation in cluster systems. *Proceedings of the 2001 ACM Sigmetrics International conference on Measurement and Modeling of Computer Systems* (pp. 62–71).

[8] Marshal, E. (1992). Fatal error: How patriot overlooked a scud. *Science*, *1347(255)*.

[9] Office, U. S. G. A. (1992). Patriot missile defense: Software problem led to system failure at dhahran.

[10] Araujo, J., Junior, R. M., Maciel, P., & Matias, R. (2011). Software aging issues on the eucalyptus cloud computing infrastructure. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (pp. 1411–1416).

[11] Liu, J., Zhou, J., & Buyya, R. (2015). Software rejuvenation based fault tolerance scheme for cloud applications. *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing* (pp. 1115–1118).

[12] Araujo, J., Alves, V., Oliveira, D., Dias, P., Silva, B., & Maciel, P. (2013). An investigative approach to software aging in android applications. *Proceedings of the International Conference on Systems, Man, and Cybernetics* (pp. 1229–1234).

[13] Araujo, J., Oliveira, F., Matos, R., Torquato, M., Ferreira, J., & Maciel, P. (2016). Software aging issues in streaming video player. *Journal of Software*, *11(6)*, 554–568.

[14] Andrzejak, A., & Silva, L. (2008). Using machine learning for non-intrusive modeling and prediction of software aging. *Proceedings of the IEEE Network Operations and Management Symposium*.

[15] Alonso, J., Belanche, L., & Avresky, D. R. (2011). Predicting software anomalies using machine learning techniques. *Proceedings of the 2011 10th IEEE International Symposium on Network Computing and Applications* (pp. 163–170).

[16] Avresky, D. R., Sanzo, P. D., Pellegrini, A., Ciciani, B., & Forte, L. (2015). Proactive scalability and management of resources in hybrid clouds via machine learning. *Proceedings of the 2015 IEEE 14th International Symposium on Network Computing and Applications* (pp. 114–119).

[17] Yan, Y., & Guo, P. (2016). A practice guide of software aging prediction in a web server based on machine learning. *China Communications*, *13(6)*, 225–235.

[18] Araujo, J., Junior, R. M., Maciel, P., Vieira, F., Matias, R., & Trivedi, K. S. (2011). Software rejuvenation in eucalyptus cloud computing infrastructure: a method based on time series forecasting and multiple thresholds. *Proceedings of the 3rd International Workshop on Software Aging and Rejuvenation (WoSAR' 11) in Conjuction with the 22nd Annual International Symposium on Software Reliability Engineering*.

[19] Araujo, J., Matos, R., Alves, V., Maciel, P., Souza, F. V. D., Matias Jr., R., & Trivedi, K. S. (2014). Software aging in the eucalyptus cloud computing infrastructure. *ACM Journal on Emerging Technologies in Computing Systems*, *10*, 1–22.

[20] Du, X., Xu, C., Hou, D., & Qi, Y. (2009). Software aging estimation and prediction of a real vod system based on PCA and neural networks. *Proceedings of the International Conference on Information and Automation* (pp. 111–116).

[21] Yakhchi, M., Alonso, J., Fazeli, M., Bitaraf, A. A., & Patooqhy, A. (2015). Neural network based approach for time to crash prediction to cope with software aging. *Journal of Systems Engineering and Electronics*, *26(2)*, 407–414.

[22] Guo, C., Wu, H., Hua, X., Lautner, D., & Ren, S. (2015). Use two-level rejuvenation to combat software aging and maximize average resource performance. *Proceedings of the* in *2015 IEEE 7th International Symposium on Cyberspace Safety and Security, High Performance Computing and Communications.*

[23] Melo, M., Maciel, P., Araujo, J., Matos, R., & Araujo, C. (2013). Availability study on cloud computing environments: Live migration as a rejuvenation mechanism. *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*.

[24] Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2014). A survey of software aging and rejuvenation studies. *J. Emerg. Technol. Comput. Syst.*, *10(1)*.

[25] Cotroneo, D., Iannillo, A. K., Natella, R., Pietrantuono, R., & Russo, S. (2015). The software aging and rejuvenation repository. *Proceedings of the 2015 IEEE International Symposium on Software Reliability Engineering Workshops* (pp. 108–113).

[26] Grottke, M., Matias, R., & Trivedi, K. (2008). The fundamentals of software aging. *Proceedings of the 19th IEEE Int. Symp. on Software Reliability Engineering*.

[27] Avizienis, A., Laprie, J., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 11–33.

[28] Huang, Y., Kintala, C., Kolettis, N., & Fulton, N. D. (1995). Software rejuvenation: Analysis, module and applications. *Proceedings of the 25th Symp. on Fault Tolerant Computing* (pp. 381–390).

[29] Shereshevsky, M., Crowell, J., Cukic, B., Gandikota, V., & Liu, Y. Software aging and multifractality of memory resources. *Proceedings of the Conf. on Dependable Systems and Networks* (pp. 721 – 730).

[30] Trivedi, K. S., Vaidyanathan, K., & Goseva-Popstojanova, K. (2000). Modeling and analysis of software aging and rejuvenation. *IEEE Annual Simulation Symp*.

[31] Castelli, V., Harper, R. E., Heidelberger, P., Hunter, S. W., Trivedi, K. S., Vaidyanathan, K., & Zeggert, W. P. (2001). Proactive management of software aging. *IBM Journal of Research and Development*, *45(2)*, 311–332.

[32] Vaidyanathan, K., & Trivedi, K. S. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, *2*, 124–137.

[33] Araujo, J., Junior, R. M., Maciel, P., Matias, R., & Beicker, I. (2011). Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. *Proceedings of the Middleware 2011 Industry Track Workshop*.

[34] Peng, J., & Wu, Q. (2008). Research and implementation of rsa algorithm in java. *Management of e-Commerce and e-Government, International Conference on* (pp. 359–363).

[35] Blum, R. (2008). *Linux Command Line and Shell Scripting Bible*. Wiley Publishing.

[36] Junior, R. M., Araujo, J., Alves, V., & Maciel, P. (2012). Experimental evaluation of software aging effects in the eucalyptus elastic block storage. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (pp. 1103–1108).

[37] Junior, R. M., Araujo, J., Maciel, P., Vieira, F., Matias, R., & Trivedi, K. S. (2012). Software rejuvenation in eucalyptus cloud computing infrastructure: A hybrid method based on multiple thresholds and time series prediction. *International Transactions on Systems Science and Applications*, *8*, 1–16.

[38] Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. *Proceedings of the European Conference on Computer Systems*.

[39] The Cacti Group. Cacti - the complete rrdtool-based graphing solution. Retrieved March 15, 2016, from http://www.cacti.net/

[40] RRDtool. About RRDtool. Retrieved March 15, 2016, from http://oss.oetiker.ch/rrdtool/index.en.html

[41] Ganglia. Ganglia monitoring system. Retrieved March 20, 2016, http://ganglia.info

[42] Massie, M. L., Chun, B. N., & Culler, D. E. (2003). The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, *30*.

[43] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., & Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*.

[44] Hindman, B., Konwinski, A., Zaharia, M., & Stoica, I. (2009). A common substrate for cluster computing. *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*.

[45] Munin. Munin open project. Retrieved March 25, 2016, from http://munin-monitoring.org

[46] Nagios, E. L. L. C. (2016). Nagios is the industry standard in it infrastructure monitoring. Retrieved March 25, 2016, from http:// munin-monitoring.org

[47] Shuhaimi, M. A. A. B. M., Abidin, Z. B. Z., Roslan, I. B., & Anawar, S. B. (2011). The new services in nagios: Network bandwidth utility, email notification and sms alert in improving the network performance. *Proceedings of the 7th International Conference on Information Assurance and Security (IAS)*.

[48] Issariyapat, C., Pongpaibool, P., Mongkolluksame, S., & Meesublak, K. (2012). Using nagios as a groundwork for developing a better network monitoring system. *Proceedings of the Technology Management for Emerging Technologies (PICMET'12)*

[49] Supermon. Retrieved March 25, 2016, from http://supermon.sourceforge.net

[50] Sottile, M. J., & Minnich, R. G. (2002). Supermon: A high-speed cluster monitoring system. *Proceedings*

of IEEE International Conference on Cluster Computing (pp. 39–46).

[51] Zabbix, S. (2016). The enterprise-class monitoring solution for everyone. Retrieved March 25, 2016, from http://www.zabbix.com

[52] Chaves, S. A. D., Uriarte, R. B., & Westphall, C. B. (2011). Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, *49(12)*, 130–137.

[53] Uriarte, R. B., & Westphall, C. B. (2014). Panoptes: A monitoring architecture and framework for supporting autonomic clouds. *Proceedings of the 2014 IEEE Network Operations and Management Symposium.*

Jean Araujo received his B.S. degree in information systems from the Seven of September Faculty, Brazil, in 2008. He specializes in network security computer by Gama Filho University. He earned his M.Sc. in Computer Science from the Informatics Center of the Federal University of Pernambuco in 2012, and is currently a PhD. student in Computer Science at the same university.

He is the author of scientific papers in international journals and conferences on software aging and rejuvenation, cloud computing and analytical modeling. Among the various areas, stand out performance and dependability evaluation, computational modeling and distributed systems.

Prof. Araujo is currently assistant professor by the Federal Rural University of Pernambuco, and a member of IEEE and Brazilian Computer Society.

Céfanys Braga received the B.S. degree in computer science by Academic Unity of Garanhuns from Federal Rural University of Pernambuco, Brazil, in 2015.

His research interests include programming, performance and dependability evaluation. He has worked on research projects encompassing software aging, availability evaluation, and cloud computing.

Mr. Braga is one member of the Brazilian Computer Society.

José Belmiro Neto received the B.S. degree in computer SCIENCE by Academic Unity of Garanhuns from Federal Rural University of Pernambuco, Brazil, in 2014.

He is the author of scientific papers in some national and international conferences. His research interests include business process management and programming. He has worked on research projects encompassing business process management, availability evaluation, and cloud computing.

Adriano Costa received the B.S. degree in computer science by Academic Unity of Garanhuns from Federal Rural University of Pernambuco, Brazil, in 2015.

He is the author of scientific papers in some national and international conferences. His research interests include face recognition and programming. He has worked on research projects encompassing face recognition, availability evaluation, and cloud computing.

Mr. Costa is one member of the Brazilian Computer Society.

Rubens Matos received the B.S. degree in Computer Science from Federal University of Sergipe, Brazil, in 2009, and received his M.Sc. degree in Computer Science from Federal University of Pernambuco – UFPE, Brazil in 2011, and the Ph.D. at the same university in 2016.

He is currently Assistant Professor by the Federal Institute of Education, Science, and Technology of Sergipe. His research interests include performance and dependability evaluation, Markov chains, Petri nets, and other formal models for analysis and simulation of computer and communication systems. He has worked on research projects encompassing telemedicine systems, network traffic modeling, distributed storage mechanisms, and cloud computing systems.

Dr. Matos is a member of the ACM, and an IEEE member. .

Paulo Maciel received the degree in electronic engineering in 1987 and the M.Sc. and Ph.D. degrees in electronic engineering and computer science from the Federal University of Pernambuco, Recife, Brazil, respectively.

He was a faculty member with the Department of Electrical Engineering, Pernambuco University, Recife, Brazil, from 1989 to 2003. Since 2001, he has been a member of the Informatics Center, Federal University of Pernambuco, where he is currently an Associate Professor. In 2011, during his sabbatical from the Federal University of Pernambuco, he stayed with the Department of Electrical and Computer Engineering, Edmund T. Pratt School of Engineering, Duke University, Durham, NC, USA, as a Visiting Professor. His current research interests include performance and dependability evaluation, Petri nets and formal models, encompassing manufacturing, embedded, computational, and communication systems as well as power consumption analysis.

Dr. Maciel is a research member of the Brazilian Research Council.