# Software Aging Issues in Streaming Video Player

Jean Araujo[1*], Felipe Oliveira[1], Rubens Matos[2,3], Matheus Torquato[4], Joao Ferreira[2], Paulo Maciel[2]

[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.
[4] Campus Arapiraca, Federal Institute of Alagoas, Arapiraca, Brazil.

* Corresponding author. Email: jcta@cin.ufpe.br
Manuscript submitted February 1, 2016; accepted April 8, 2016.
doi: 10.17706/jsw.11.6.554-568

**Abstract:** Many researchers around the world are working hard to improve cloud-based services focusing on server-side and communication aspects. This is especially true for services which demand high processing power and storage space on their server's infrastructure, such as live video streaming on the web and video on demand. However, issues on client-side applications are often neglected or studied to a lower extent. In this context, this paper evaluates the occurrence of software aging in a web browser plug-in for video streaming. The case study is carried out by using an automated workload that simulates the user behavior accessing videos on YouTube. Time series were used to predict the resources utilization ahead of time, enabling the prevention of performance degradation and unexpected crashes. Finally, the prediction-based strategy is compared to the time-based strategy, showing the impact of both on system's availability.

**Key words:** Software aging, client-side applications, dependability analysis, QoS, fault monitoring.

## 1. Introduction

The development of software applications, from client-server architectures to multi-level architectures, led to the creation of services highly distributed and accessible through the web. Many providers have been using cloud computing to provide services that are accessible anywhere, anytime, through the Internet, i.e., in the clouds [1]. A lot of popular online services, such as social networks and video streaming have thousands or millions of simultaneous users. Such applications usually require large data center infrastructures, with a large amount of available resources such as CPU, memory, storage, and network bandwidth.

Many researchers around the world are working hard to improve the cloud-based services focusing on server-side and network issues. However, issues on client-side applications are often neglected or studied to a lower extent, despite being a potential cause of bad quality of service (QoS). Video streaming services, for example, offer a large variety of multimedia content, in many formats. So the user can watch what he/she wants with the proper resolution or quality, according to his/her preferences and connection speed. Even with the majority of the workload running in the cloud side, a part is still performed on the client side to decode frames and play videos.

One major threat to the availability and performance is the phenomenon of software aging, an inevitable process where system processes suffer from performance degradation over the course of time. Monitoring

such effects is an essential part of any aging characterization study [2]. In general, there is concern about the aging effects in applications that are running for a long period of time. However, these effects might also occur in applications in a short running time. Moreover, the software rejuvenation is a proactive action that aims at mitigating the effects of aging.

The aim of this work is to evaluate the effects of software aging in streaming video player. We focus on memory-related aging phenomena [3], such as memory leaking, which is proven to have serious undesirable effects on the performance and availability of many kinds of systems, e.g., operating systems [4], cloud infrastructure managers [5], and web servers [6]. In this paper, we carry out a case study by using an automated workload that simulates the user behavior accessing videos on YouTube service. Time series were used to predict the resources utilization ahead of time, what may be used to prevent performance loss and unexpected crashes. Finally, a CTMC model was adopted to verify the impact of rejuvenation strategies on systems availability.

The remaining parts of the paper are organized as follows: Section 2 discusses about the fundamental concepts of software aging and rejuvenation, the foundations of the usage of time series to forecast the behavior of a system, and basic concepts of Markov chains. Section 3 shows the proposed methodology. Section 4 describes the testbed environment and details about the experiment; Section 5 explores the experiment results and model results; and, finally, Section 6 draws conclusions and suggests possible future works.

## 2. Preliminaries

This section describes the theoretical foundations required for properly understanding this work, namely: software aging and rejuvenation; time series; and Markov models.

### 2.1. Software Aging and Rejuvenation

One of most important phases on software development is the test phase. Generally, at this phase, several tests are conducted to check system behavior under different conditions. The main goal of this phase is to find possible errors or bugs in software and correct them before the final version launch [7]. However, some bugs may not be caught at this phase, even when it is performed with diligence. There are bugs that appear after a long time of software execution, which roots are difficult to define, this types of bugs are often called Heisenbugs [8].

The aging-related faults are similar to Heisenbugs, but they appear in certain conditions (e.g. lack of OS resources) which can not be easily reproduced [9, 10]. The accumulation of aging-related faults may lead system to hangs and total failures, this phenomenon is known as software aging [11]. Therefore, software aging refers to a phenomenon which increases the failure rate or performance degradation of a system as it runs, that can be caused by accumulation of aging-related bugs and consumption of computational resources (e.g physical memory) [12]. Due to the cumulative property of the software aging phenomenon, it occurs more intensively in continuously running systems that are executed over a long time period. Problems such as data inconsistency, numerical errors, and exhaustion of operating system resources are examples of software aging consequences [12]. Since the notion of software aging was introduced [11], many researches have been conducted in order to characterize and understand this important phenomenon. Monitoring the aging effects is essential to any aging characterization study [2], [13], [14]. Many previous studies have implemented aging monitoring in different system levels, however to the best of our knowledge is that the discussion of aging effects in a streaming video player is a barely explored topic.

Once the aging effects are detected, mitigation mechanisms might be applied in order to reduce their impact on the applications or the operating system. The search for mitigation approaches resulted in the so-called software rejuvenation techniques [15]–[17]. Since the aging effects are typically caused by hard to

track software faults, rejuvenation techniques look for reducing the aging effects during the software runtime, until the aging causes (e.g., a software bug) are fixed definitively. In general, software rejuvenation mechanisms are based on system restart or OS reboot [18]. Due to system downtime caused by software rejuvenation, a proper approach is to conduct software rejuvenation with scheduling. Some papers [18], [19] show that very often migrations can reduce system availability significantly.

The software aging effects in cloud computing environments were addressed in [2] and [14]. Those papers demonstrated the occurrence of faults in an Eucalyptus-based infrastructure due to the accumulation of memory leaks. In [20] and [21], rejuvenation strategies have been proposed for mitigating the downtime caused by the aging effects in that cloud computing framework. Matos *et al*. [22] shows some software aging issues, like memory leaking and the increase of CPU utilization during consecutive attachments of remote block storage volumes by means of Eucalyptus commands. Zhao *et al*. [23] proposed a software performance evaluation model called SPE based on Back-propagation neural network [24]. In order to validate the effectiveness of the proposed method and algorithms, the performance of the software affected by the aging phenomenon was evaluated using real data sets. Some metrics like load average, free memory, CPU usage and response time were observed. Study the systems performance is a basic step to investigate software aging effects. So, some performance evaluation studies related to Android OS can be observed. In [25], a performance comparison of Android virtual machines and Java virtual machines (JVM) is presented, and their energy consumption is discussed. Araujo *et al*. [26] proposes an investigative approach to software aging in Android applications. Experimental results confirmed both the effectiveness of the procedure and the existence of software aging phenomenon in the Foursquare application running on the Android OS platform. An extensive study on software aging and rejuvenation can be found in [27].

## 2.2. Time Series

A time series is a collection of observations made sequentially in time [28]. It can also be represented by a set of observations of a random variable, arranged sequentially over time [29]. The value of the series at a given instant $t$ may be described by a stochastic process, i.e., a random variable $X(t)$, for each $t \in T$, where $T$ is an arbitrary set, and its associated probability distribution. In most situations, $t$ represents time, but can also represent another physical quantity, for example, space. The applications of time series analysis are mainly: description, explanation, process control and prediction.

This work adopts four models: linear, quadratic, exponential growth, and Pearl-Reed logistic. These models are briefly described as follows, based on $Y_t = E[X(t)]$ [30]: *Linear Trend Model (LTM)* is the default model used in the analysis of trends. Its equation is given by $Y_t = \beta_0+\beta_1 \cdot t+e_t$, where $\beta_0$ is known as the y-intercept, $\beta_1$ represents the average of the exchange of one time period to the next, and $e_t$ is the error of fit between the model and the real series; *Quadratic Trend Model (QTM)* takes into account a smooth curvature in the data. Its representation is given by $Y_t = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot t^2 + e_t$, where the coefficients have similar meanings as the previous item; *Growth Curve Model (GCM)* is the model of trend growth or fallen in exponential form. Its representation is given by $Y_t = \beta_0 \beta_1^t \cdot e_t$; Finally, *S-Curve Trend Model (SCTM)* fits the logistics of Pearl-Reed. It is usually used in time series that follows the shape of the S-curve. Its representation is given by $Y_t = 10^a/(\beta_0 + \beta_1\beta_2{}^t)$.

This paper adopts the error measures MAPE, MAD and MSD, for choosing the model that best fits the observed data. *MAPE (Mean Absolute Percentage Error)* represents the precision of the estimated values of the time series expressed in percentage. Equation 1 represents this estimator:

$$MAPE = \frac{\sum_{t=1}^{n} |(Y_t - \widehat{Y}_t)/Y_t|}{n} \times 100, \tag{1}$$

where $Y_t$ is the actual value observed at time $t$ ($Y_t \neq 0$), $\widehat{Y}_t$ is the estimated value and $n$ is the number of observations.

*MAD (Mean Absolute Deviation)* represents the accuracy of the estimated values of the time series. It is expressed in the same unit of data. MAD is an indicator of the error size and is represented by the statistics (see Equation 2):

$$MAD = \frac{\sum_{t=1}^{n} |(Y_t - \widehat{Y}_t)|}{n} \tag{2}$$

where $Y_{t,}$ $t$, $\widehat{Y}_t$ and $n$ have the same meanings index MAPE.

*MSD (Mean Squared Deviation)* is a more sensitive measure than the MAD index, especially in large forecasts. Equation 3 gives its expression:

$$MSD = \frac{\sum_{t=1}^{n} |(Y_t - \widehat{Y}_t)|^2}{n} \tag{3}$$

where $Y_t, t, \widehat{Y}_t$ and $n$ have the same meanings of the previous indexes.

### 2.3. Markov Models

Markov models are the fundamental building blocks upon which most of the quantitative analytical performance techniques are built [31]. Such models may be used to represent the interactions between various system components, for both descriptive and predictive purposes. Markov models have been in use intensively in performance and dependability modeling since around the fifties [32]. Besides computer science, the range of applications for Markov models is very extensive. Economics, meteorology, physics, chemistry and telecommunications are some examples of fields which found in this kind of stochastic[1] modeling a good approach to address various problems.

A Markov model can be described as a state-space diagram associated to a Markov process, which constitutes a subclass of stochastic processes.

Let $Pr\{k\}$ be the probability of a given event $k$ occurs. A Markov process is a stochastic process in which $Pr\{X_{tn+1} \leq s_{i+1}\}$ depends only on the last previous value $X_{tn}$, for all $t_{n+1} > t_n > t_{n-1} > ... > t_0 = 0$, and all $s_i \in S$. This is the so-called Markov property [33], which, in plain words, means that the future evolution of the Markov process is totally described by the current state, and is independent of past states [33].

In this work, there is only interest on discrete (countable) state space Markov models, also known as Markov chains, which are distinguished in two classes: Discrete-Time Markov Chains (DTMC) and Continuous-Time Markov Chains (CTMC) [34]. In DTMCs, the transitions between states can only take place at known intervals, that is, step-by-step. Systems where transitions only occur in a daily basis, or following a strict discrete clock are well represented by DTMCs. If state transitions may occur at arbitrary (continuous) instants of time, the Markov chain is a CTMC. In the case of DTMC, the geometric distribution is the only discrete time distribution that presents the memoryless property. In the case of CTMC, the exponential distribution is used.

## 3. Methodology

Fig. 1 depicts a general view of the methodology adopted to perform software aging investigation. The

[1] Stochastic refers to something which involves or contains a random variable or variables

description of each phase is presented afterwards.

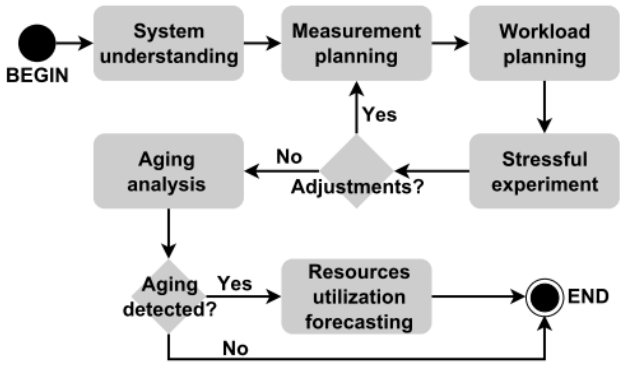



Fig. 1. Methodology.

The methodology consists of six activities: system understanding, measurement planning, workload planning, stressful experiment, aging analysis, and resources utilization forecasting. The first step is to understand the analyzed system, its components, their interactions, as well as the main involved processes.

Next, it is necessary to define a monitoring strategy that collects resources utilization, like CPU and RAM, of the entire system, and of the involved processes. The strategy proposed by [2], which uses Shell scripts to monitor Linux-based systems, was adapted to this specific environment.

The workload planning is another very important concern in a software aging investigation, because the aging effects may take days to appear, months or even years, or may never occur [2]. Therefore, it is necessary to generate a workload capable of stressing the system and help to observe the failure process.

In the next step, an experiment is performed over a long time, where the involved processes are monitored while the workload runs. If there are any problems during the experiment, then it is necessary to go back to the monitoring step and, then, follows again the same steps to fix the problem. If there is no problem, we went to the next step.

Here, in the aging analysis, the results are analyzed in order to identify any aging indicator. If there is no aging, the cycle is finished, otherwise we move on to the final step.

Finally, in this final step, data from aged resources that were collected during the experiment are submitted to a forecasting, which aims to predict the system crash.

## 4. Experimental Study

One of the principal consequences of software aging is poor device resource management, in particular memory consumption. One major memory-related phenomenon which indicates the existence of software aging is memory leak. Many memory leaking problems are difficult to detect, and there are few symptoms other than the slowing down of system performance and the steady increase in memory consumption [35]. Concisely put, memory leaking occurs when a process allocates memory during its execution, and continues to allocate more memory in each subsequent execution without freeing that previously used. Eventually, if allowed to continue, the outcome would be a complete depletion of memory resources for the machine.

In order to analyze possible aging effects during video streaming visualization, we measure the utilization of hardware and software resources in a scenario which emulates common user actions. Next, we describe the main components of our testbed, as well as the workload adopted.

### 4.1. Testbed Environment

The testbed employed a desktop machine (Intel Core 2 Duo, CPU E7500 @ 2.93GHz, 2 GB of RAM, and a 320 GB SATA hard disk) running the Ubuntu Desktop Linux 13.10 operating system (kernel 3.11.0-14), the

Mozilla Firefox web browser – version 25.0.1 – and the Adobe Flash Player 11.2.202.332.

In order to verify whether the phenomenon of software aging occurs, we decide to monitor the environment in full operation during a given time. Thus, it is necessary to generate a workload that emulates actions of a video streaming user.

The workload was automated with the set of tools called Selenium IDE [36], which is designed for performing functional tests, among other purposes. Selenium is an integrated development environment implemented as a Firefox extension that simulates the user actions, such as clicking, typing text, searching text, and searching elements on the tested web page. It generates a script in a native language, which may also be exported to other general-purpose languages such as Ruby, C#, Java, or Python.

The script generated by Selenium was exported to Python language to ease the customization of the experiment, requiring the installation of a Python library that provides support for this tool. The file that initializes the experiment and the monitoring process is called only once.

Fig. 2 illustrates the workload used in the experiment. The workload is based on accessing the YouTube [37] website to play videos, and was executed in cycles. The first action is a search for videos containing a keyword (e.g., "Cold Play"). The first result is selected. This video is played for 30 seconds, and another video is selected (click and view) until the cycle is complete, i.e., when all five visualizations are finished. At the end of the cycle, the homepage is accessed again, waiting 30 seconds to begin a new cycle. The workload runs indefinitely until stopped by an operator as soon as the desired number of cycles for the experiment is reached, or the required number of resource usage samples is collected. The monitoring process is finished at the same time.

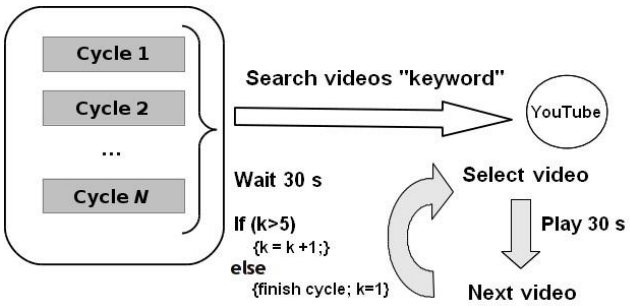



Fig. 2. Workload generated by Selenium IDE.

For this experiment, we created monitoring scripts using Linux utilities, such as: date, mpstat, free, ps and du [38]. The scripts monitor the utilization of resources such as CPU, disk and memory usage. Those resources were monitored for the entire system, as well as specifically for Mozilla Firefox and Flash Player applications. The data were collected every 10 seconds throughout the experiment. This time interval and the tools were carefully chosen to avoid interferences on the system. We also removed any background processes that were not needed for the basic system execution, and measured the system for 5 minutes before starting the workload to ensure the stability of the environment.

## 5. Result Analysis

This Section presents the results obtained by conducting an experiment study and through the CTMC model.

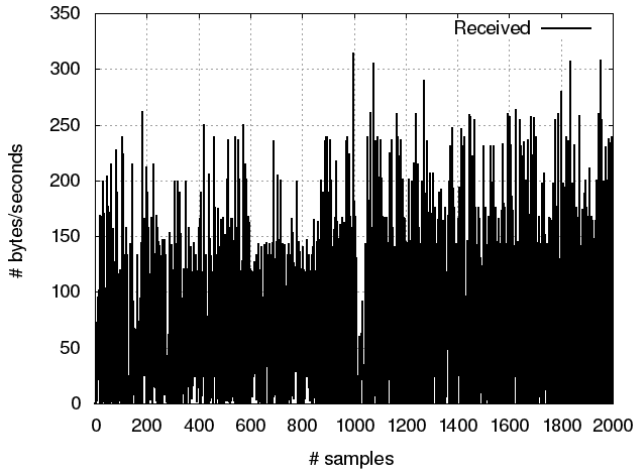
### 5.1. Experimental Results

We carried out an experimental study to investigate software aging existence in a common user environment. We focused on the Mozilla Firefox web browser and Adobe Flash Player plug-in, that are among the most common applications for web video visualization [39], [40].

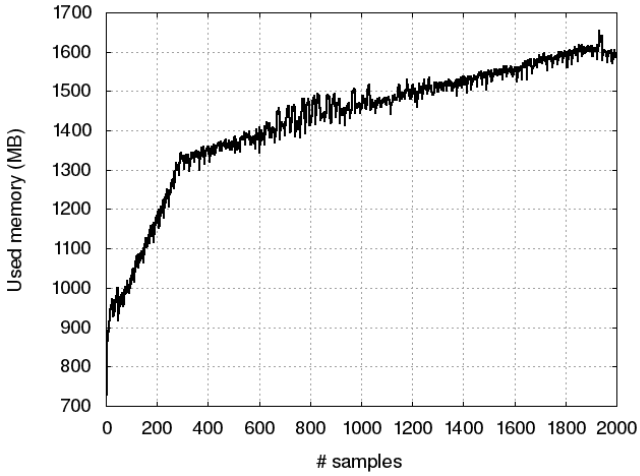
A total of 2000 resource usage samples were collected throughout the experiment. The virtual and

resident memory usage for the Flash Player process are the most representative results found in this experimental study. Other resources such as CPU and disk usage metrics, as well all data collected for the Firefox process, do not show any perceptible aging behavior, so they are not included in this paper.

Fig. 3(a) shows the network input traffic during the experiment. As we can see, the workload was fairly constant throughout the experiment, proving that no additional load was added to "force" the appearance of aging effects. Fig. 3(b) shows an increase in the used memory, that is an indication of software aging because at the begin of the experiment it was using 900 MB and at the end of the experiment had used 1600 MB. It is important to say that part of such memory usage might not be due to an aging issue, but intentionally caused by Linux memory management strategies, such as the storage of recently used data as cache. Therefore, we also present the results of specific measurements of the Flash Player process.




(a) Network traffic.

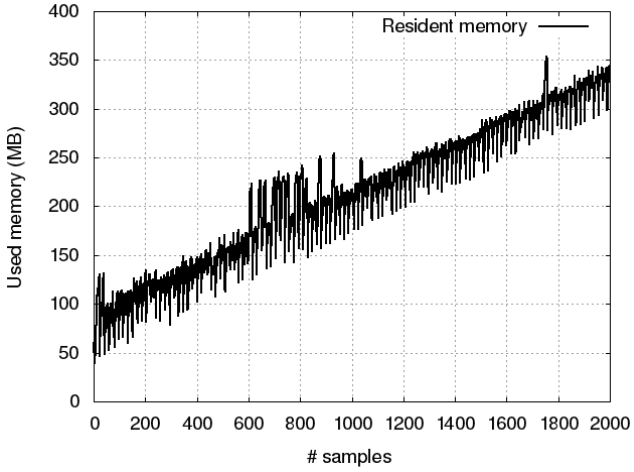



(b) Used memory.

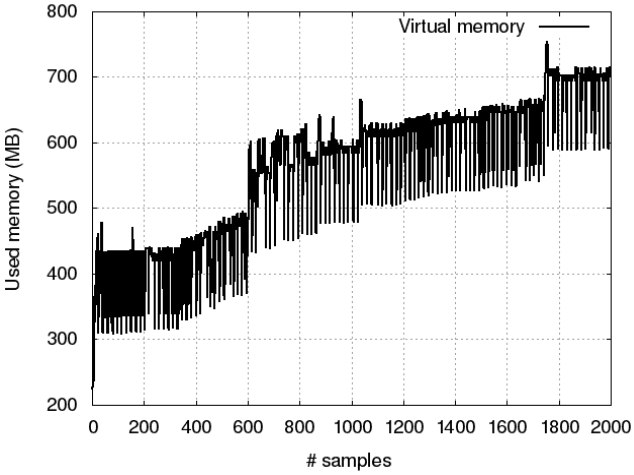
Fig. 3. System-wide resources.

Fig. 4(a) shows almost linear growth in the resident memory utilization of the Flash Player process. The process starts using 50 MB, and at the sample number 2000 (5˜ hours) reached almost 350 MB of resident memory utilization. Considering that the RAM memory is a limited resource, this is a large usage by only one process. The Fig. 4(b) shows the virtual memory that also grew during the execution of the experiment. It started using 300 MB and finished using 700 MB. One should note that this occurs despite the workload periodically interrupts the visualization of a current video to load another one. Thus we may infer that Flash player does not release all memory allocated to the previously played videos, and keep accumulating

unused resources, what may lead to performance degradation of the own video execution, as well as for the overall system.




(a) Resident memory.




(b) Virtual memory.

Fig. 4. Memory utilization of the flash player process.

Considering that the most affected resources were the resident memory and virtual memory of the Flash Player, we used the data collected in the experiments to find out which kind of time series fits best the growth of these resources.

We computed four time series models (LTM, QTM, GCM, and SCTM) for trend analysis. A summary of their information, including their error indices, is shown in Tables 1 (resident memory) and 2 (virtual memory), where $Y_{bt}$ is the predicted value of the memory consumption at time $t$.

It can be seen from Table 1 that the values of MAPE, MAD and MSD indices are smaller for the LTM, QTM, and SCTM models. So the choice must be made in relation to these three models. It is also observed that despite the MAPE values of the indices are the same for these three models, the LTM model has a smaller MAD value than QTM and SCTM models. So the LTM model was chosen as the best fit for the Flash Player resident memory utilization.

Table 2 shows the values of MAPE, MAD and MSD for virtual memory usage of Flash Player process. For this resource, the SCTM and QTM have the smallest indices, but both models could not be chosen for trend analysis because they indicate a decrease of the analyzed measure after some time whereas this is not the actual behavior of the virtual memory of Flash Player process in the experiments. Therefore, we choose the

next best option according to the error indices: the LTM.

Table 1. Summary of Information for Each Model for Resident Memory

| Model | $\widehat{Y}_t$ | MAPE (%) | MAD | MSD | Ranking |
|---|---|---|---|---|---|
| LTM | $92954 + 125 \cdot t$ | 7 | 11594 | 305374778 | 1 |
| QTM | $86471 + 144.36 \cdot t - 0.00971 \cdot t^2$ | 7 | 11688 | 296995425 | 2 |
| GCM | $107513 \cdot (1.00064^t)$ | 9 | 17635 | 567732010 | 4 |
| SCTM | $(10^7)/(27.2275 + 86.6756 \cdot (0.998414^t))$ | 7 | 12896 | 332593401 | 3 |

Table 2. Summary of Information for Each Model for Virtual Memory

| Model | $\widehat{Y}_t$ | MAPE (%) | MAD | MSD | Ranking |
|---|---|---|---|---|---|
| LTM | $408448 + 169 \cdot t$ | 7 | 36209 | 2511629057 | 3 |
| QTM | $361455 + 309.95 \cdot t - 0.07038 \cdot t^2$ | 6 | 31788 | 2071283622 | 2 |
| GCM | $413584 \cdot (1.00031^t)$ | 8 | 40907 | 2967234112 | 4 |
| SCTM | $(10^7)/(14.3884 + 15.9371 \cdot (0.997876^t))$ | 6 | 31372 | 2152789236 | 1 |

Fig. 5 shows the trend analysis for the growth of resident and virtual memory utilization, fit respectively by the linear functions 92954 + 125 · *t* (resident memory), and S-curve function 408448 + 169 · *t* ( virtual memory). These functions allows one to predict when a given critical limit for the resources utilization will be reached.

Tables 3 and 4 present the absolute percentage error between the predictions and the actual values for the resident and virtual memory utilization in this experiment. That error varies in a range of about 1% to less than 7% from one to five hours, what shows that our approach achieved an acceptable accuracy in its predictions. The best approximations are obtained at four and five hours of experiment, that are the points where the "fit" line intercepts the "actual" line in Fig. 5. The worst errors in Table 3 correspond to the regions where the lines in Fig. 5(a) have the largest distances, showing that the 6.54% error at 60 minutes is close to the higher bound of prediction errors in this experiment.

Table 3. Comparison of Resident Memory Predictions and Actual Values

| Time (min) | Predicted (MB) | Actual (MB) | Error (%) |
|---|---|---|---|
| 60 | 134.721 | 144.156 | 6.54 |
| 120 | 178.788 | 171.184 | 4.44 |
| 180 | 222.611 | 212.441 | 4.78 |
| 240 | 266.557 | 257.207 | 3.63 |
| 300 | 310.502 | 305.863 | 1.51 |

Table 4. Comparison of Virtual Memory Predictions and Actual Values

| Time (min) | Predicted (MB) | Actual (MB) | Error (%) |
|---|---|---|---|
| 60 | 458.289 | 453.074 | 1.15 |
| 120 | 517.703 | 511,473 | 1.22 |
| 180 | 577.117 | 619.469 | 6.84 |
| 240 | 636.531 | 650.801 | 2.19 |
| 300 | 695.945 | 702.234 | 0.90 |

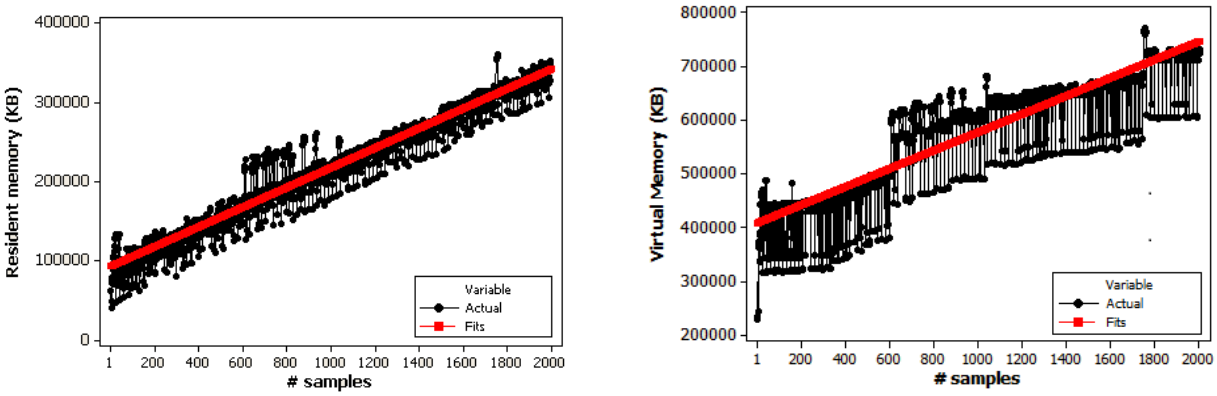
The worst error in Table 4 is 6.84% at three hours. The best approximations are obtained at one and five hours of experiment, which are respectively 1.15% and 0.90%. The errors of the predictions for both resources show that our approach can be used effectively, and enable, for example, the development of software rejuvenation approaches for such a service.

Table 5 shows the prediction of virtual and resident memory utilization of Flash Player process, for

instants on time from 6 to 10 hours. The steady growth of both measures deserves attention due to potential performance degradation resulting from this behavior. The virtual memory reaches 1 GB in about 10.52 hours, and about 553.14 MB of the resident memory. These are excessive amounts of resources for just one process, considering that users may need to run other applications simultaneously. Users might not play videos uninterruptedly for such a time lapse, but this is not even needed if the browser is not closed after the video execution, and when the operating system is put in sleep or hibernate modes (usual action nowadays), instead of shutting down, because the browsers and other open applications do not free their allocated memory. Therefore, it is actually feasible for a common user to reach the amounts of memory utilization shown in this analysis.

Table 5. Prediction for Virtual and Resident Memory

| Time (min) | Virtual memory (MB) | Resident memory (MB) |
|---|---|---|
| 360 | 755.359 | 354.447 |
| 420 | 814.773 | 398.393 |
| 480 | 874.188 | 442.338 |
| 540 | 933.602 | 486.283 |
| 600 | 993.016 | 530.229 |




(a) Resident memory. (b) Virtual memory.

Fig. 5. Trend analysis of flash memory.

The specific models shown here are directly bound to the software environment used in the experimental testbed, but it is noteworthy that the time series approach presented here is general enough to be applied on other computing environments which have similar aging characteristics.

## 5.2. Model Results

Considering that a software has a mean time to aging-related failure (MTTARF) of 24 hours, and a mean time to repair (MTTR) of 0.25 hour when the software fails. Applying the availability formula $A = \frac{uptime}{uptime+downtime}$, we have $A = \frac{24}{24+0.25} = 0.98969072164$, i.e., 98.969% of availability. However, considering that this software has a health monitor responsible to activate a rejuvenation action (RA), and it takes only 10 seconds of unavailability when is performed, the availability can be improved. The simplest strategy to be adopt would be to run the rejuvenation action periodically at a time *T*. However, the indiscriminate use of a rejuvenating action may also cause a high unavailability. In this sense, the adoption of a prediction strategy can help to reduce the impact caused by the rejuvenating action.

To identify the impact of software rejuvenation strategy adopted in system availability, we adopt the Markov model of Fig. 6, that is based on the model presented in [16].

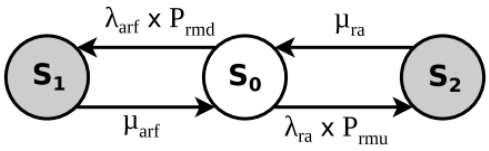



Fig. 6. CTMC model of software behavior with rejuvenation policy.

The availability of this model can be calculated from closed-form at Equation 4.

$$A_{rej} = \frac{\mu_{arf}\mu_{ra}}{P_{rmu}\lambda_{ra}\mu_{arf} + (P_{rmd}\lambda_{arf}\mu_{arf})\mu_{ra}} \tag{4}$$

The software is up in state $S_0$. The occurrence of a failure caused by software aging is represented by the transition from $S_0$ to $S_1$. This transition occurs with rate $\lambda_{arf} \times P_{md}$, where $\lambda_{arf}$ is the aging-related failure rate, and $P_{md}$ refers to the probability of unsuccessful rejuvenation, that is a failed triggering of proactive rejuvenation action. In $S_1$, the software is down, so it cannot be accessed by its users. On the other hand, a successful proactive rejuvenation action can take the model from the state $S_0$ to state $S_2$. This state represents that the software is undergoing rejuvenation. The transition from $S_0$ to $S_2$ occurs with rate $\lambda_{ra} \times P_{mu}$, where $\lambda_{ra}$ is the proactive rejuvenation action rate, and $P_{mu}$ refers to the probability of the rejuvenation mechanism being performed successfully, triggering properly a rejuvenation action. When in state $S_1$, the model may go back to state $S_0$ after a reactive recovery ($\lambda_{arf}$). The transition from the state $S_2$ to state $S_0$ represents the completion of a proactive action, and has a rate $\mu_{ra}$. The software is considered down in both states $S_1$ and $S_2$.

The mean time to activation (MTTA) of time-based rejuvenation action ($\lambda_{ra}$) are about 10% less than the value of the MTTARF, to avoid that the crash occurs before the rejuvenation action to be activated, i.e., 21.6 hours. As for the prediction-based action, we consider the percentage error of the time series. In our case, considering the MAPE metric for LTM model, the results in Tables 3 and 4 are 7%, i.e., 22.32 hours.

Table 6 shows the results of CTMC model comparing a software availability without rejuvenation strategy and with rejuvenation strategy (time-based and prediction-based rejuvenation strategies). Software behavior without rejuvenation has the worst results, with 90.30928 hours of downtime. Time-based and prediction-base rejuvenation strategies has similar results, reducing the downtime to less the one hour. Prediction-based strategy has the best results, with only 0.55905 hour of downtime (34.164 minutes), and time-based with 0.56148 hours of downtime (33.543 minutes).

It is possible to conclude that a prediction-based strategy may reach good availability results, mainly if a forecasting approach appropriate for the monitored data is adopted.

Table 6. Model Results

| | Without rej. | With rejuvenation | |
|---|---|---|---|
| **Metric** | | **Time-based** | **Prediction-based** |
| Availability (%) | 98.96907 | 99.99359 | 99.99361 |
| Uptime (hours) | 8669.69071 | 8759.43852 | 8759.44094 |
| Downtime (hours) | 90.30928 | 0.56148 | 0.55905 |

## 6. Final Remarks

Software aging is usually related to applications which execute for a long time period, but it may happen and cause a significant impact on desktop applications too. In this paper, we have investigated the evolution of client-side resources utilization when accessing a video streaming service. Data collected during experiments indicated the presence of software aging at Flash Player process, mainly related to resident

and virtual memory usage. Time series models were computed, considering the actual system behavior, to enable prediction of resources utilization ahead of time. A CTMC model was adopted to verify the impact of rejuvenation strategies on systems availability. The results shows that rejuvenation strategies may improve the system availability, reducing the total downtime of the system to less then one hour per yer.

As a future work, we intend to investigate the aging problems in other browsers and applications which enable video streaming, such as Silverlight, and HTML 5. We shall also identify or develop rejuvenation actions to mitigate the problems identified here.

## 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] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, *25(6)*, 599-616.

[2] Araujo, J., Junior, R. M., Maciel, P., & Matias, R. (2011). Software aging issues on the eucalyptus cloud computing infrastructure. *Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics*.

[3] Macedo, A., Ferreira, T. B., & Matias, R. (20100. The mechanics of memory-related software aging. *Proceedings of the 2010 IEEE Second International Workshop on Software Aging and Rejuvenation* (pp. 1 -5).

[4] Matias, R., Beicker, I., Leitao, B., & Maciel, P. (2010). Measuring software aging effects through os kernel instrumentation. *Proceedings of the 2nd Int. Workshop on Software Aging and Rejuvenation*.

[5] Araujo, J., Matos, R., Alves, V., Maciel, P., Souza, F. V. D., Jr., R. M., & Trivedi, K. S. (2014). Software aging in the eucalyptus cloud computing infrastructure: Characterization and rejuvenation. *J. Emerg. Technol. Comput. Syst.*, *10(1)*.

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

[7] Royce, W. W. (1970). *Managing the development of large software systems*. *Proceedings of IEEE WESCON*.

[8] Gray, J., & Reuter, A. (1993). *Transaction Processing*. Morgan Kaufíann Publishers.

[9] Vaidyanathan, K., & Trivedi, K. S. (2001). Extended classification of software faults based on aging. *Fast Abstract, Int. Symp. Software Reliability Eng*.

[10] 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.

[11] 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).

[12] Grottke, M., Matias, R., & Trivedi, K. (2008). The fundamentals of software aging. *Proceedings of the 1st Int. Workshop on Software Aging and Rejuvenation (WoSAR)*, *Conjunction with 19th IEEE Int. Symp. on Software Reliability Engineering*.

[13] Gross, K. C., Bhardwaj, V., & Bickford, R. (2002). Proactive detection of software aging mechanisms in performance critical computers. *Proceedings of the 27th Annual NASA Goddard Software Engineering* (pp. 17-23).

[14] 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 ACM/IFIP/USENIX International Middleware Conference*.

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

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

[17] Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2011). Software aging and rejuvenation: Where we are and where we are going. *Proceedings of the 2011 IEEE Third International Workshop on Software Aging and Rejuvenation*.

[18] 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 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[19] Melo, M., Araujo, J., Matos, R., Menezes, J., & Maciel, P. (2013). Comparative analysis of migration-based rejuvenation schedules on cloud availability. *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 4110-4115).

[20] 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 in Conjuction with the 22nd Annual International Symposium on Software Reliability Engineering*.

[21] Junior, R. M., Araujo, J., Maciel, P., Vieira, F. Matias,, R., & Trivedi, K. S. (2012). *S*oftware 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.

[22] 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 Int. Conf. on Systems, Man, and Cybernetics*.

[23] Zhao, J., Trivedi, K. S., Wang, Y., & Chen, X. (2010). Evaluation of software performance affected by aging. *Proceedings of the 2nd Int. Workshop on Software Aging and Rejuvenation, in Conjunction with 21th IEEE Int. Symp. on Software Reliability Engineering*.

[24] Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence, Prentice Hall.

[25] Kundu, P. K., & Paul, K. (2010). Android on mobile devices: An energy perspective. *Proceedings of the 2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*.

[26] 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 2013 IEEE International Conference on Systems, Man, and Cybernetics*.

[27] Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *10(1)*.

[28] Chatfield, C. (1996). *The Analysis of Time Series: An Introduction*. 5th ed. New York, USA: Chapman & Hall/CRC.

[29] Kedem, B., & Fokianos, K. (2002). *Regression Models for Time Series Analysis*. John Wiley & Sons, Inc., Publication.

[30] Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2008). *Introduction to Time Series Analysis and Forecasting*. Wiley series in probability and statistics, 2008.

[31] Trivedi, K. S. (2002). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Ltd.

[32] Maciel, P., Trivedi, K., Matias, R., & Kim, D. S. (2012). *Dependability Modeling*. In Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions, ed. Valeria Cardellini, Emiliano Casalicchio, Kalinka Regina Lucas Jaquie Castelo Branco, Julio´ Cezar Estrella and Francisco José Monaco, 53 -97.

[33] Haverkort, B. (2009). Markovian models for performance and dependability evaluation. *Lectures on Formal Methods and PerformanceAnalysis, ser. Lecture Notes in Computer Science*.

[34] Kleinrock, L. (1975). *Theory, Volume 1, Queueing Systems.* Wiley-Interscience.

[35] Xu, Z., & Zhang, J. (2008). Path and context sensitive inter-procedural memory leak detection. *Proceedings of the 8th Int. Conf. on Quality Software* (pp. 412-420).

[36] Selenium. Selenium ide. Retrieved from: http://www.seleniumhq.org/projects/ide/

[37] YouTube. Youtube. YouTube, LLC. Retrieved from: http://www.youtube.com/

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

[39] Protalinski, E. (2014). *IE11 passes IE10 in market share, Firefox slips a bit, and Chrome gains back share.* Retrieved from: http://thenextweb.com/insider/2014/02/01/ie11-passes-ie10-market-\ \share-firefox-slips-bit-chrome-gains-back-share/

[40] Casale. C. (2012). *Html5 vs. Flash - what do you need to know? part 1*. Retrieved from: http://blog.accusoft.com/2012/october/html5-vs-flash-what-do-you-need-to-know-part-1

**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.

**Felipe Oliveira** is an undergraduate student in computer science by the Academic Unity of Garanhuns from Federal Rural University of Pernambuco.

He is the author of scientific papers in some national and international conferences. His research interests include performance and dependability evaluation. He has worked on research projects encompassing software aging, performance evaluation, and cloud computing.

Mr. Oliveira is a 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 an 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.

**Matheus Torquato** received his MSc. degree in computer science from Federal University of Pernambuco (UFPE) in 2014.

He is currently teaching informatics and computer networks in Campus Arapiraca - Federal Institute of Alagoas (IFAL). His research interests include cloud computing, dependability and computer networks.

Mr. Torquato is a member of the Brazilian Computer Society.

**Joao Ferreira** received his B.Sc. degree in computer science from the Catholic University of Pernambuco, Brazil in 2005. He then worked as a network administrator and software developer in the Federal Rural University of Pernambuco before pursuing his MS degree in computer science at Federal University of Pernambuco. He received his M.Sc. degree in computer science in 2013. In 2014, Joao became a PhD student at the same institution (UFPE).

He has interest in cloud computing, sustainability; energy consumption; dependability and optimization.

Mr. Ferreira is a member of IEEE Society.

**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.