SIWN
siwn.org.uk

International Transactions on Systems Science and Applications
Volume 7 Number 3/4 December 2011 pp. 278-294

iTSSA

Software Rejuvenation in Eucalyptus Cloud Computing Infrastructure: A Hybrid Method Based on Multiple Thresholds and Time Series Prediction

Rubens MATOS [a], Jean ARAUJO [a], Paulo MACIEL [a], F. Vieira DE SOUZA [b], Rivalino MATIAS [c] and Kishor TRIVEDI [d]

[a] *Informatics Center, Federal University of Pernambuco, Recife, Brazil*
[b] *Statistics and Informatics Department, Federal University of Piauí, Teresina, Brazil*
[c] *School of Computer Science, Federal University of Uberlandia, Uberlândia, Brazil*
[d] *Department of Electrical and Computer Engineering, Duke University, Durham, USA*

Abstract: The need for reliability and availability has increased in modern applications, in order to handle rapidly growing demands while providing uninterrupted service. Cloud computing systems fundamentally provide access to large pools of data and computational resources through a variety of interfaces, similarly to existing grid and HPC resource management and programming systems. This work presents an approach that uses time series to schedule rejuvenation, so as to reduce the downtime by predicting the proper moment to perform the rejuvenation. We show the results of our approach through experiments using the Eucalyptus cloud computing framework.

Keywords: software aging and rejuvenation; cloud computing; dependability analysis, time series.

1. Introduction

Deployment of cloud-based architectures has grown over recent years, mainly because they constitute a scalable, cost-effective and robust service platform. Such features are made possible due to the integration of various software components that enable reservation and access to computational resources, by means of standard interfaces and protocols, mainly based on web services. Virtualization is an essential requirement to build a typical cloud-computing infrastructure [1].

Cloud-oriented data centers allow the success of massive user-centric applications, such as social networks, even for start-up companies that would not be able to provide, by themselves, the performance and availability guarantee needed for those systems. Although availability and reliability are major requirements for cloud-oriented infrastructures, an aspect usually neglected by many service providers is the effect of software aging phenomenon [2], which has been verified (e.g., [2], [3], [4]) to play an important role to the reliability and performance degradation of many computing systems.

While flexible and essential to the concept of "elastic computing", the usage of virtual machines and remote storage volumes requires memory and disk intensive operations during virtual machines allocation, reconfiguration or destruction. Such operations may speed up the exhaustion of hardware and operating system resources in the presence of software aging due to software faults or poor system design [2].

sai: itssa.2011.12.049 ISSN 1751-1461 (print) ISSN 1751-147X (cd-rom)

Software faults should ideally have been removed during the debugging phase. Even if software may have been thoroughly tested, it still may have some design faults that are yet to be revealed [5]. Another type of fault observed in software systems is due to the phenomenon of resource exhaustion. Operating system resources, e.g. swap space and free memory, are progressively depleted due to defects in software such as memory leaks and incomplete cleanup of resources after use. These faults may exist in operating systems, middleware and application software [6].

In this paper, we present a rejuvenation policy using multiple thresholds for forecasting based on time series. The trend analysis of aging-related resources consumption, matched against multiple time series models, enables an accurate forecast of critical points of resource exhaustion. The proposed strategy also relies on multiple thresholds, to guarantee a safe scheduling of rejuvenation actions, aiming at minimizing the impact on system performance due to the prediction functionalities. The proposed approach is applied to the Eucalyptus cloud-computing framework. The remaining parts of the paper are organized as follows. In Section 2 we present fundamental concepts about software aging and rejuvenation. Section 3 describes some aspects related to dependability in cloud computing, and in Section 4 we present the foundations of the usage of time series to forecast the behavior of a system. Section 5 describes our cloud computing environment based on the Eucalyptus framework. Section 6 shows the experimental study which demonstrates the software aging symptoms in the Eucalyptus framework, and describes our proposed approach as well as its applicability to Eucalyptus systems. Section 7 draws some conclusions and shows 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 [7], [2]. The causes of software aging have been verified as the accumulated effect of software faults activation [8] during the system runtime [9], [10]. Aging in a software system, as in human beings, is an accumulative process. The accumulating effects of successive error occurrences directly influence the aging-related failure manifestation. Software aging effects are the consequences of errors caused by aging-related fault activations. These faults gradually lead the system towards an erroneous state [9]. This gradual shifting is a consequence of aging effects accumulation, being the fundamental nature of the software aging phenomenon. It is important to highlight that a system fails due to the consequences of aging effects accumulated over time. For example, considering a specific load demand, an application server system may fail due to unavailable physical memory, which may be caused by the accumulation of memory leaks related to the lack of some variables deallocation, or similar software faults. In this case, the aging-related fault is a defect in the code that causes memory leaks; the memory leak is the observed effect of an aging-related fault.

The aging factors [2] are those input patterns that exercise the code region where the aging-related faults may be activated. The occurrence of such faults may lead the system to erroneous states. Considering such memory leakage per aging-related error occurrence, the server system may fail due to the main memory unavailability. Nevertheless, the related effect may be observable only after long run of the system. The time to aging-related failure (TTARF) is an important metric for reliability and availability studies of systems suffering from software aging [4]. Previous studies (e.g., [3], [4], [11]) on the aging-related failure phenomenon show that the TTARF probability distribution is strongly influenced by the intensity with which the system gets exposed to aging factors such as system workload.

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, such as cloud-computing framework software components. 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. Problems such as data inconsistency, numerical errors, and exhaustion of operating system resources are examples of software aging consequences [2].

Since the notion of software aging was introduced [7], [9], many theoretical and experimental researches have been conducted in order to characterize and understand this important phenomenon. Monitoring the aging effects is an essential part of any aging characterization study. Many past-published studies have implemented aging monitoring in different system levels, however to the best of our knowledge this is the first work discussing the aging effects in a cloud-computing environment.

A proactive fault management method to deal with the software aging phenomenon is software rejuvenation [12]. [9] proposed the technique of software rejuvenation which involves occasionally stopping the software application, removing the error conditions and then restarting the application in a clean environment. This process removes the accumulated errors and frees up or defragments operating system resources, thus postponing or preventing, in a proactive manner, unplanned and potentially expensive future system outages.

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 [3], [13]. 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.

Examples of rejuvenation techniques may be software restart or system reboot. In the former, the aged application process is killed and then a new process is created as a substitute. 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 [3] is presented a zero-downtime rejuvenation technique for the apache web server, which was adapted for this work.

3. Dependability in Cloud Computing

Cloud computing is the access to computers and their functionality via the Internet or a local area network [14]. The US National Institute of Standards and Technology - NIST [15], defines cloud computing as follows: “Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.

Cloud types (including public, private, and hybrid) refer to the nature of access and control with respect to the usage and provisioning of virtual and physical resources. The most common service styles are referred to by the acronyms IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service) [16].

Numerous advances in application architecture have helped to promote the adoption of cloud computing. These advances help to support the goal of efficient application development while helping applications to be elastic and scale gracefully and automatically [16]. Cloud computing is seen by some as an important forward-looking model for the distribution and access of computing resources because it offers these potential advantages:

-Scalability: Applications designed for cloud computing need to scale with workload demands so that performance and compliance with service levels remain on target [14] [16].

-Security: Applications need to provide access only to authorized, authenticated users, and those users need to be able to trust that their data is secure [16].

-Availability: Regardless of the application being provided, users of Internet applications expect them to be up and running every minute of every day [16].

-Reliability and fault-tolerance: Reliability means that applications do not fail and most important they do not lose data [16] i.e. it is the ability to perform and maintain its functions in unexpected

circumstances. depend Many of the wished features of a cloud system are related to the concept of dependability. There is no unique definition of dependability. By one definition, it is the ability of a system to deliver the required specific services that can justifiably be trusted [17]. It is also defined as the system property that prevents a system from failing in an unexpected or catastrophic way [8].

Indeed, dependability is also related to disciplines such as availability and reliability. Availability is the ability of the system to perform its slated function at a specific instant of time or over a stated period of time [18] [19] [20].

Dependability is a very important property for a cloud system as it should provide services with high availability, high stability, high fault tolerance and dynamical extensibility. Because cloud computing is a large-scale distributed computing paradigm and its applications are accessible anywhere, anytime, and in anyway, dependability in cloud systems becomes more important and more difficult to achieve [21]. Software aging effects in cloud systems may affect the performance of communication among their components, what in a critical level would also have an impact on the dependability of that system. Therefore, the presence of many software layers in cloud systems raise the need of monitoring aging effects and proposing proper rejuvenation mechanisms in order to assure the dependability aspects cited here.

4. Time Series

A time series can be represented by a set of observations of a random variable, arranged sequentially over time [22]. The value of the series at a given instant t can be described by a stochastic process, i.e. a random variable $X(t)$, for each $t \in T$, and 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.

Time series enables one to build models that explain the behavior of the observed variable and the types of time series analyses may be divided into frequency-domain [23] [22] and time-domain methods [24] [25] [26].

The modeler must decide how to use the chosen model according to her goals. Many forecasting models are based on the "least squares" method. The most common time series models are based on errors (or regression), autoregressive moving average (ARMA) models, the autoregressive integrated moving average (ARIMA) models, the long memory ARIMA (also called ARFIMA models), the structural models and the nonlinear models [22].

This work adopts five models, namely: the linear model, the quadratic model, the exponential growth model, the model of the Pearl-Reed logistic, and the ARIMA model. These models are briefly described as follows, based on $Y_t = E[X(t)]$:

- *Linear Trend Model (LTM)*: this 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 β_0 is known as the y-intercept, β_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 [27].
- *Quadratic Trend Model (QTM)*: this model 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 [27].
- *Growth Curve Model (GCM)*: this is the model of trend growth or fallen in exponential form. Its representation is given by $Y_t = \beta_0 \cdot \beta_1^t \cdot e_t$.
- *S-Curve Trend Model (SCTM)*: this model fits the logistics of Pearl-Reed. It is usually used in time series that follow the shape of the curve S. Its representation is given by $Y_t = 10^a/(\beta_0 + \beta_1\beta_2^t)$.
- *ARIMA:* in general, an ARIMA model is characterized by the notation ARIMA(p,d,q) where p,d and q denote orders of auto-regression, integration (differencing) and moving average,

respectively. Such parameters may be estimated by means of autocorrelation analysis and verification of differencing steps needed to transform the series in a stationary series [25].

Error measures [28] are adopted for choosing the model that best fits the observed data. MAPE, MAD and MSD were the error measures adopted in this work:

- *MAPE (Mean Absolute Percentage Error)* represents the accuracy of the estimated values of the time series expressed in percentage. This estimator is represented by:

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

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:

$$MAD = \frac{\sum_{t=1}^{n} |(Y_t - \widehat{Y_t})|}{n},$$

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. Its expression is given by

$$MSD = \frac{\sum_{t=1}^{n} |(Y_t - \widehat{Y_t})|^2}{n},$$

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

5. Eucalyptus Environment Description

In order to analyze possible aging effects in the Eucalyptus cloud-computing framework, we use a testbed composed of six Core 2 Quad machines (2.66 GHz processors, 4 GB RAM) running the Ubuntu Server Linux 10.04 (kernel 2.6.35-24) and the Eucalyptus System version 1.6.1. The operating system running in the virtual machines is a customized Ubuntu Linux 9.04 that runs a simple FTP server. The cloud environment under test is fully based on the Eucalyptus framework and the KVM hypervisor. We analyze the utilization of hardware and software resources in a scenario in which some cloud operations, related to instantiation of virtual machines, are continuously performed. Next, we describe the main components of our testbed, as well as the workload adopted.

5.1. Eucalyptus Framework

Eucalyptus [14] implements scalable IaaS-style private and hybrid clouds [29]. It was created with the purpose of cloud computing research and it is interface-compatible with the commercial service Amazon EC2 [14], [30]. This API compatibility enables one to run an application on Amazon and on Eucalyptus without modification. In general, the Eucalyptus cloud-computing platform uses the virtualization capabilities (hypervisor) of the underlying computer system to enable flexible allocation of resources decoupled from specific hardware. There are five high-level components in the Eucalyptus architecture, each one with its own web service interface: *Cloud Controller, Node Controller, Cluster Controller, Storage Controller*, and *Walrus* [29]. A brief description of these components follows:

- *Cloud Controller (CLC)* is the front-end to the entire cloud infrastructure. The CLC is responsible for exposing and managing the underlying virtualized resources (servers, network, and storage) via Amazon EC2 API [16]. This component uses web services interfaces to receive the requests of client tools on one side and to interact with the rest of Eucalyptus components on the other side.
- *Cluster Controller (CC)* usually executes on a cluster front-end machine [31] [29], or on any machine that has network connectivity to both the nodes running *Node Controllers (NC)* and to the machine running the *Cloud Controller*. CCs gather information about a set of VMs and schedule VM execution on specific NCs. The *Cluster Controller* has three primary functions: schedule incoming requests to run VM instances on specific NCs, control the instance virtual network overlay, and gather/report information about a set of NCs [29].
- *Node Controller (NC)* runs on each node and controls the life cycle of instances running on the node. The NC interacts with the operating system and with the hypervisor running on the node. The actions of a NC are managed by a *Cluster Controller (CC)*.
  NCs control the execution, inspection, and termination of VM instances on the host where it runs, fetches and cleans up local copies of instance images. It queries and controls the system software on its node in response to queries and control requests from the *Cluster Controller* [31]. A NC makes queries to discover the node's physical resources - number of CPU cores, size of memory, available disk space - as well as to learn about the state of VM instances on that node [29], [32].
- *Storage controller (SC)* provides persistent block storage for use by the virtual machine instances. It implements block-accessed network storage, similar to that provided by Amazon Elastic Block Storage – EBS [33], and it is capable of interfacing with various storage systems (NFS, iSCSI, etc.). An elastic block storage is a Linux block device that can be attached to a virtual machine but sends disk traffic across the locally attached network to a remote storage location. An EBS volume can not be shared across instances [32].
- *Walrus* is a file-based data storage service, that is interface compatible with Amazon's Simple Storage Service (S3) [29]. Walrus implements a REST interface (through HTTP), sometimes termed the "Query" interface, as well as SOAP interfaces that are compatible with S3 [29], [32]. Users that have access to Eucalyptus can use Walrus to stream data into/out of the cloud as well as from instances that they have started on nodes. In addition, Walrus acts as a storage service for VM images. Root filesystem as well as kernel and ramdisk images used to instantiate VMs on nodes can be uploaded to Walrus and accessed from nodes.

5.2. Testbed Environment

The Figure 1 shows the components of our experimental environment. The *Cloud Controller, Cluster Controller, Storage Controller* and *Walrus* have been installed on the same machine, and the VMs were instantiated on four different physical machines, so that each of them ran a *Node Controller*. Three of those machines have a 32-bit (i386) Linux OS, whereas one has the 64-bit (amd64) Linux OS, which allow us to capture possible different aging effects related to the system architecture. A single host is used to monitor the environment and also to perform requests to the Cloud Controller, acting as a client for the infrastructure in our testbed.

The environment was monitored during 72 hours, for each experiment. The definition of duration for the experiment was based on empirical observation of the elapsed time until the manifestation of some aging symptoms, considering the workload that was adopted to stress the system.

The workload uses some of the Eucalyptus features to accelerate the life cycle of the VMs, which is composed by four states: *Pending, Running, Shutting down* and *Terminated*, as shown in Figure 2.

Scripts are used in order to start, reboot and kill the VMs in a short time period. Such operations are essential to this kind of environment, because they enable quick scaling of capacity, both up and down, as the computing requirements change (the so-called elastic computing) [30]. Cloud-based

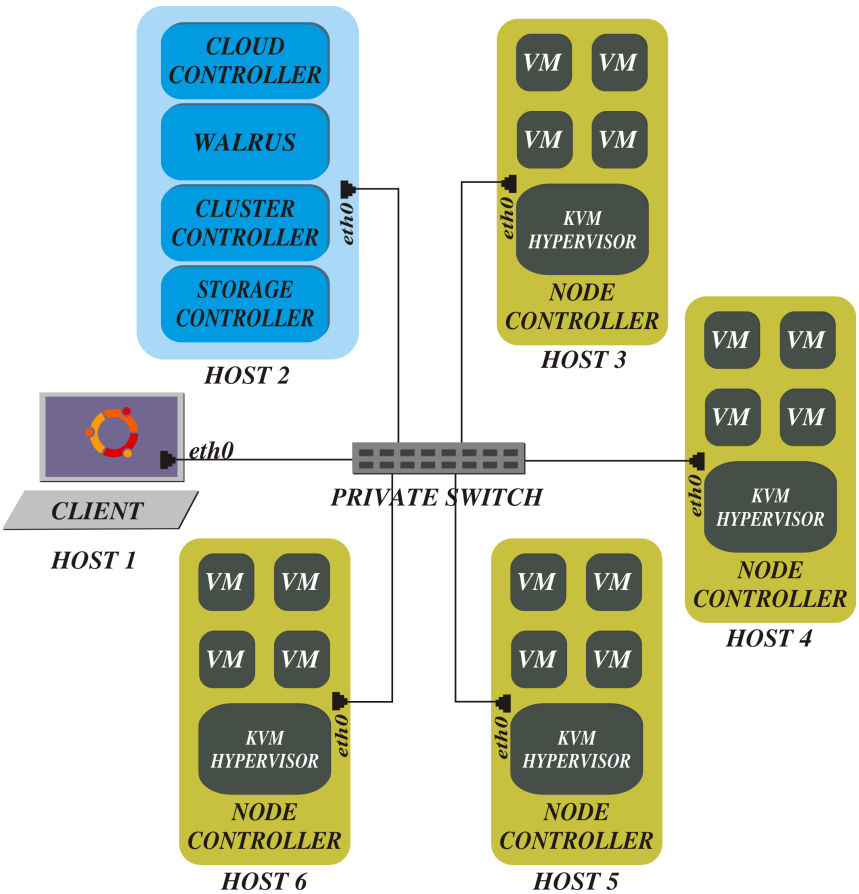

Figure 1. Components of the testbed environment

applications adapt themselves to increases in the demand by instantiating new virtual machines, and save resources by terminating underused VMs when the load of requests is low. VM reboots are also essential to high availability mechanisms which automatically restart VMs on other physical servers when a server failure is detected. Our workload was implemented by means of shell script functions that perform the operations we have just mentioned, as it may be seen below:

- *Instantiate VMs Function*: This function instantiates 8 VMs in a cluster. Those VMs are instances of an Ubuntu Server Linux running a FTP server.
- *Kill VMs Function*: This function finds out which "instances" are running in the cloud and kills them all.
- *Reboot VMs Function*: Much like the previous function, it also finds all existing instances, but instead of killing, it requests their reboot.

Every two minutes the script checks whether more than two hours have passed from the beginning of the last initialization. If so, all VMs are killed, otherwise all VMs are rebooted. Empirically, we decided to use the time from two minutes to reboot eucalyptus service and two hours for carrying out the kill command because of the amount of executions that we had in the monitoring period, which would generate a workload that can stress constantly the Eucalyptus's infrastructure.

The environment is monitored for about two hours without workload. After that time, the script instantiates all virtual machines and follows the workload cycle previously described. The monitoring period without running any workload was chosen to enable stating a relationship between the data obtained with and without stress of the environment.

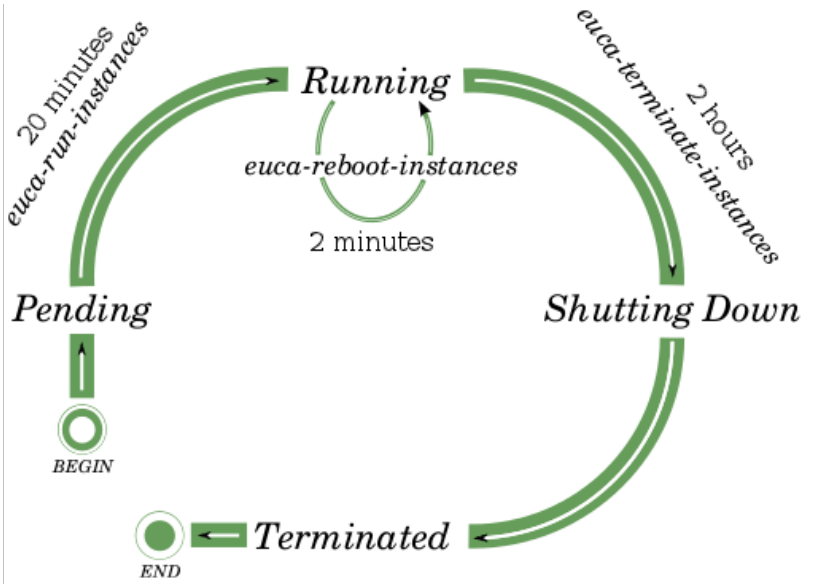

Figure 2. VM instance lifecycle (Adapted from [32])

6. Experimental Study

We have used the testbed environment just described to perform experiments looking for aging symptoms, characterized by the consumption of the following resources: CPU time, memory space, hard disk space, and process IDs. These experiments provided the basis for our proposal of a rejuvenation method that is presented further in this section.

6.1. Results of the Software Aging Experiments

The virtual and resident memory usage, for the Eucalyptus node controller process, are the most representative results found in this experimental study. Other important results are found by monitoring the number of zombie processes in the cloud controller host. CPU and disk usage metrics do not show any important aging behavior, so they are not included in this paper.

Figure 3 shows the usage of virtual memory by the node controller process at machine 3. There was increasing memory usage during the reboots, shutdowns and instantiations of VMs. Such behavior indicates the aging of Eucalyptus software, that continued until the process reaching about 3055 MB of virtual memory, at which time the process has stopped growing. At that point, the node controller could not instantiate VMs anymore, probably due to not being possible for that process handling more virtual memory because of the limitations of the 32-bits operating system. A manual restart of the single process that is responsible for the Eucalyptus node controller made the memory usage fall to less than 110 MB.

After the process restarts, the same pattern is repeated. The virtual memory of the process has grown until about 3064 MB and again the node controller is not able to attend the requests of virtual machines instantiation. Nodes 5 and 6 (the other ones with 32-bit OS) showed the same behavior as node 3. The graphics for those machines are visually similar, so they will not be shown here. The manual restart of Eucalyptus process when it reaches the critical point of virtual memory usage is an eventual rejuvenation action. This continuous growth of memory consumption indicates a memory leak, that should by fixing the problem in Eucalyptus source code, what is not the current focus of our research.

In the single machine using 64-bit architecture, we have also observed a monotonic increasing use of virtual memory, as shown in Figure 4. In this case, there is no “drops", because the node

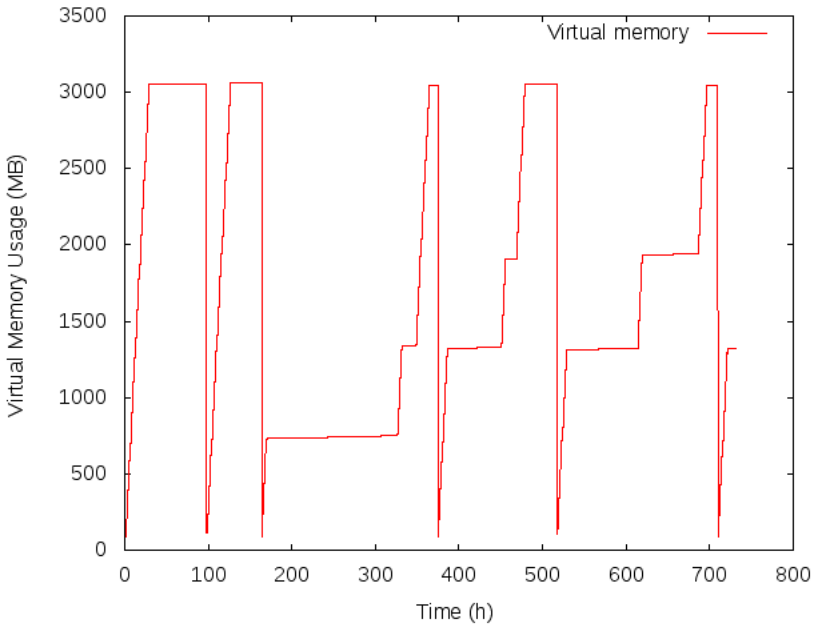

Figure 3. Virtual memory used in the NC process at *Host3*

controller process in that machine never failed in processing the requests of VM instantiation, so the process was not restarted. Such fact confirms that process problems noticed in three machines are caused by limitations of 32-bit architecture. There is no evidence that demonstrates when that system would reach a critical level, i.e., the point where it will hang. It is known that the 64-bit architecture allows up to 256 TB of virtual memory addressing, however if the virtual memory usage had continued growing throughout a longer period, the hard disk space used by Linux memory management system would be exhausted.

Another monitored resource is the use of resident memory in node controllers. Also using the host 3 as the standard for analysis of 32-bit machines, there are many similar variations of the use of virtual memory, as shown in Figure 5. The growth of resident memory usage is also evident and the "falls" of each peak are related to the moment when the node controller process is restarted. However, such growth is not enough to take a larger proportion of crashes because the hosts reach the marks of 3055 MB and 3064 MB of virtual memory usage, as previously mentioned.

The resident memory usage in the 64-bit machine, depicted in Figure 6, has a nearly uniform growth. We can't see any "drop" because, as mentioned above, this machine did not lock in any time. If the resident memory usage had continued growing throughout a longer period, probably the RAM used by Linux memory management system would be exhausted.

Due to the behavior seen in Figure 6, we have used a linear trend model (LTM) to analyze the resident memory usage of NC process in the 64-bit machine. We have found the following relation: $rm = 44,157 + 2.850 \cdot t$, where rm is the amount of resident memory (in kilobytes) used by the node controller process, and t is the time of execution for that process. That equation enables one to estimate the amount of memory used by that process at a given time, and the other way round. For instance, if we would want to know when node controller process will reach 1 GB of resident memory, using the regression equation, we will find a value of about 352,427 minutes, or, 244 days. Table 1 shows the estimates for distinct time periods, expressed in months. Those estimates confirm that the increasing memory consumption deserves attention, in order to avoid that only one process uses almost all memory available in the system.

Figure 7 shows the usage of resident memory by the cloud controller process. In this case, there is also no "drops", because the cloud controller process never hanged and, therefore, there was need

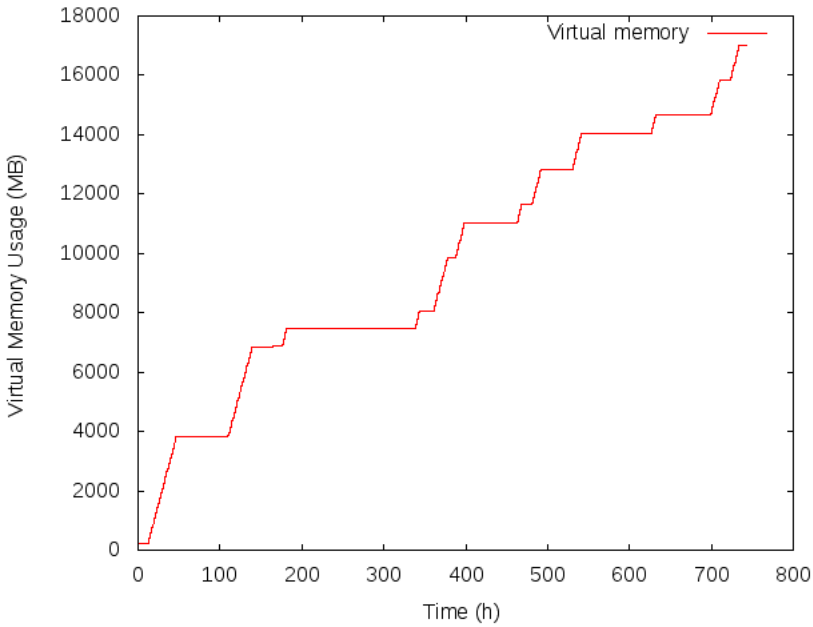

Figure 4. Virtual memory used in the NC process in a 64-bits machine

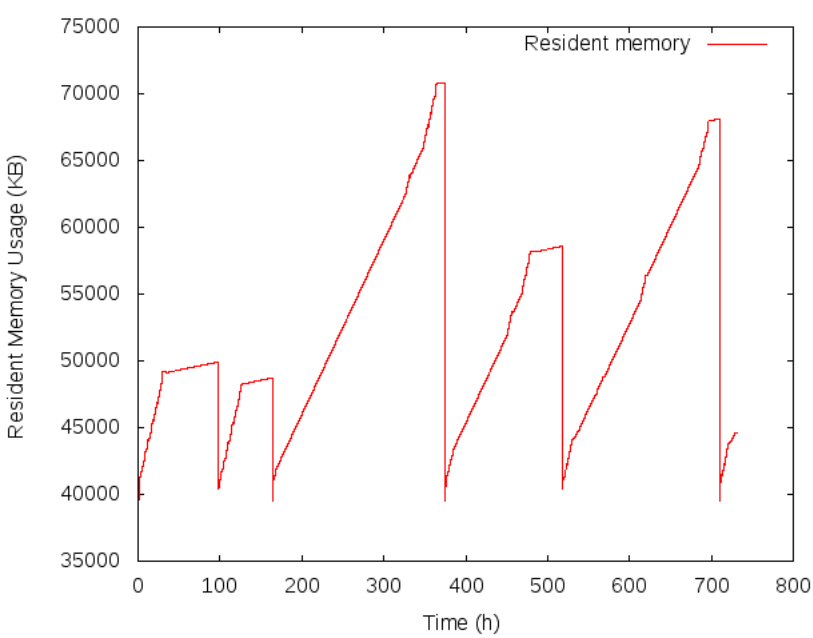

Figure 5. Resident memory used in the NC process at *Host3*

to restart it. We can see some peaks of growth, probably related to the fact that some node controller stopped working at a given instant, and the cloud controller was trying to instantiate new VMs, but could not. The amount of virtual memory used by the cloud controller process did not present a significant increase, so it is not shown here.

The number of zombie processes is other measure that highlights an aging effect in the machine

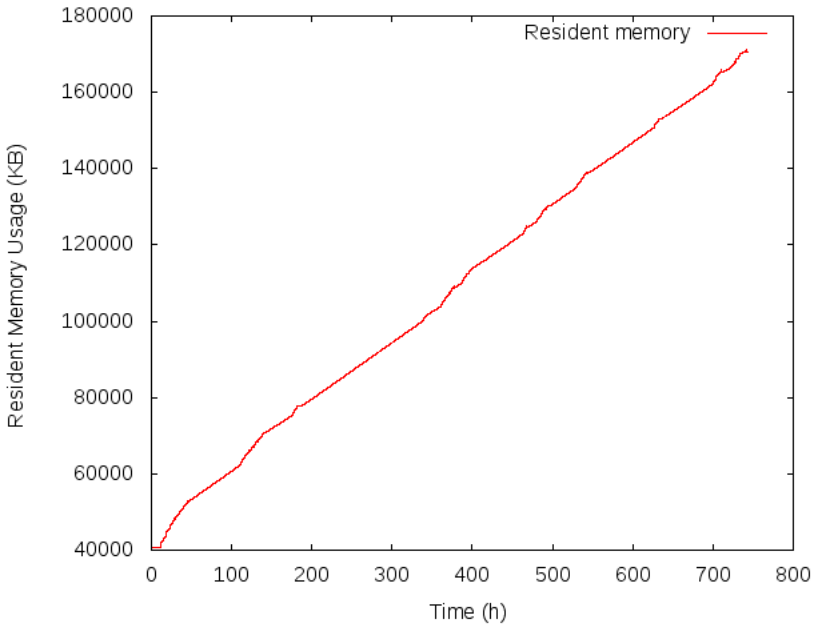

Figure 6. Resident memory used in the NC process in a 64-bits machine

Table 1. Regression-based estimates for resident memory usage in NC process

Time (months)	Estimative (MB)
2	283.591
4	524.060
6	764.528
8	1004.997
10	1245.466
12	1485.935

that holds the role of cloud controller. A zombie, or defunct, process is a process that is holding on an entry in the process table even after has completed its execution. Zombie processes do not use any memory space, but an uncontrolled increase of those processes may cause an exhaustion of process ID numbers in the operating system. Figure 8 shows that the number of zombie processes in cloud controller machine grows in some moments. The major part of processes that fall to the zombie state are apache processes that in its time are executed by the Eucalyptus services. Due to a Linux cleaning mechanism that is executed periodically, all zombies are removed from process table after a time period, so the amount of zombie processes detected in this experiment does not cause problems to system activities. Although, when the system is under heavy workload, the number of processes may reach much higher values, thence this metrics should be thoroughly monitored in such circumstances.

6.2. Rejuvenation Method for Eucalyptus Cloud Computing Environments

Based on the software aging symptoms observed in the Eucalyptus environment, we propose an automated method for triggering a rejuvenation mechanism in private cloud environments. Our method is based on the rejuvenation mechanism presented in [3], that sends a signal (SIGUSR1) to the apache master process, so that all slave idle processes are terminated and replaced by new ones. This action cleans up the accumulated memory and has a small impact on the service, since the master process waits for the established connections to be closed. Creating a new process to replace the old one

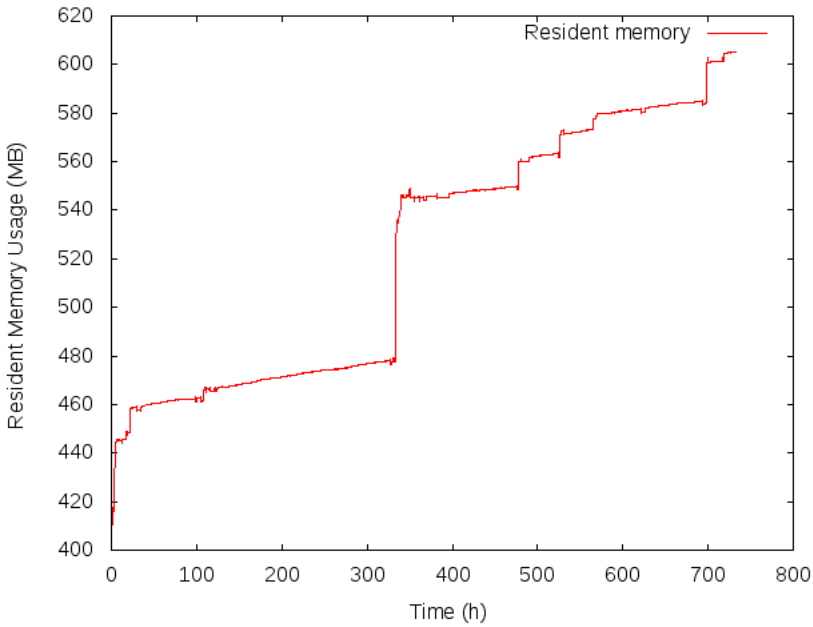

Figure 7. Resident memory used in the cloud controller process

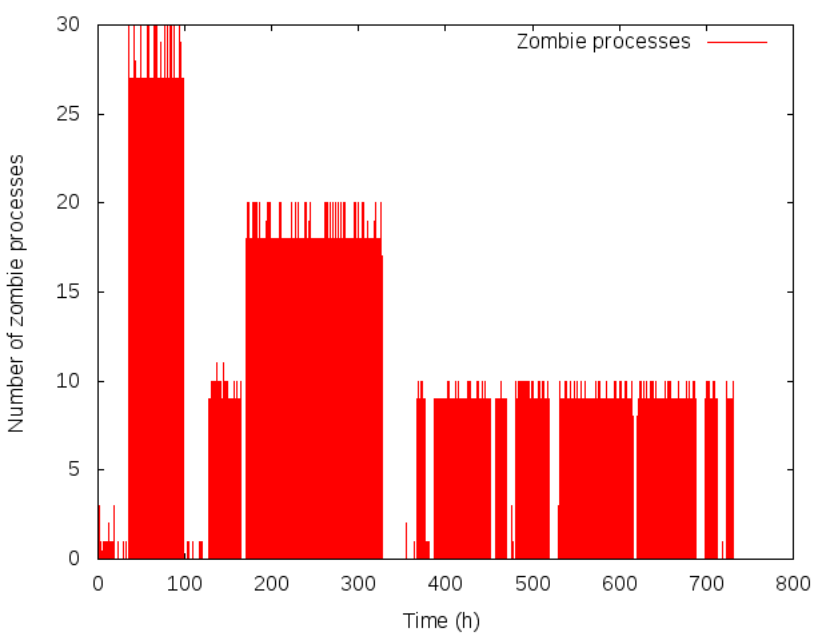

Figure 8. Number of zombie process in the cloud controller machine

causes a downtime of around 5 seconds, due to the load of eucalyptus configurations during process startup, as observed in previous experiments.

In a production environment, the interval between process restarts should be as large as possible, in order to reduce the effects of summing up small downtimes during a large runtime period. One approach to achieve that maximum interval is based on a high-frequency monitoring of process memory

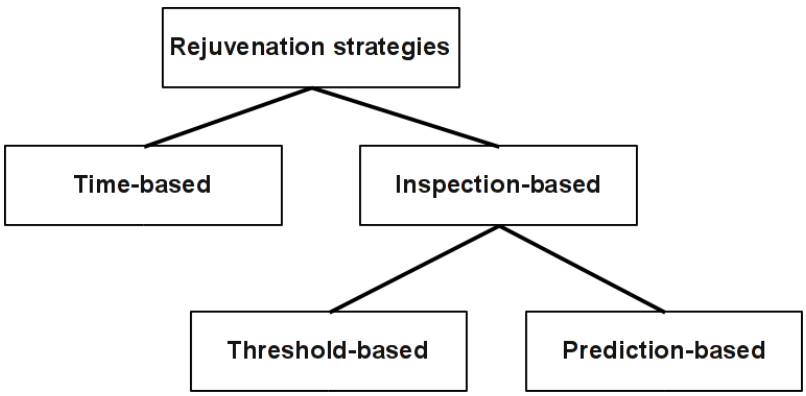

Figure 9. Classification of rejuvenation strategies

usage. At the exact moment when the memory limit is reached, the rejuvenation is triggered. Since a small sampling interval may affect system's performance, so we consider that a 1-minute interval is the minimum time duration to avoid interference in the system. Despite providing good results, we identified problems in such an approach. The NC process may reach its memory usage limit at some instant between two sampling points. Such an event introduces an additional downtime, which we will describe as "monitor-caused downtime".

The proposed triggering mechanism aim at removing this additional downtime, as it tries to keep the interval between process restarts as large as possible. The prediction about when the critical memory utilization (CMU) will be reached is used for this purpose. Therefore, considering the classification of rejuvenation strategies presented in Figure 9, our approach has characteristics of two categories, since it is a threshold based rejuvenation but it is aided by predictions. Time-series fitting enables us to perform a trend analysis and therefore to state, with an acceptable error, the time remaining until the process reaches the CMU. This information makes it possible to schedule the rejuvenation to a given time (T_{rej}), which takes into account a safe limit (T_{safe}) to complete the rejuvenation process before the CMU is reached. The safe limit should encompass the time spent during the rejuvenation and the time relative to the time-series prediction error. Therefore, we can state $T_{rej} = T_{CMU} - T_{safe} = T_{CMU} - (T_{restart} + T_{PredError})$.

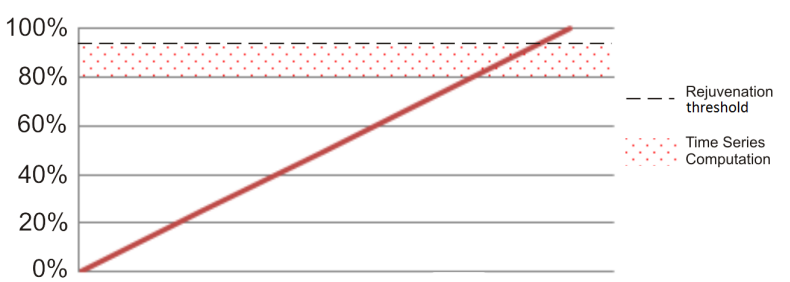

Figure 10. Rejuvenation forecasting

As seen in Figure 10, it is important to highlight that the trend analysis starts only after the monitoring script detects the node controller process has grown over a time series computation starting point, TSC_{SP}, that in our case is 80% of the critical memory utilization. This starting point was adopted to avoid unnecessary interference on the system due to the computation of time series fitting. When a limit of 95% of CMU is reached, the last prediction generated by the trend analysis is used to schedule the rejuvenation action, i.e., the last computed T_{CMU} will be used to assess the T_{rej} and the system will be prepared so that the rejuvenation occurs gracefully in time T_{rej}. We see that by

Table 2. Summary of the accuracy indices for each model (NC virtual memory)

Model	$\widehat{Y_t}$	MAPE	MAD	MSD
LTM	$159204 + 1764 * t$	0.01%	24743	$7.88 * 10^{10}$
QTM	$152104 + 1789.86 * t - 0.01589 * t^2$	0.01%	24713	$7.82 * 10^{10}$
GCM	$416421 * (1.00141^t)$	0.08%	210625	$7.51 * 10^{12}$
SCTM	$(10^7)/(3.42549 + 35.5216 * (0.996829^t))$	0.03%	79529	$7.90 * 10^{11}$
ARIMA	-	0.03%	77262	$7.76 * 10^{11}$

Table 3. Comparison of Experiments

	Experiment #1	Experiment #2
Availability	0.999584	0.999922
Number of nines	3.38	4.11
Downtime	108 seconds	20 seconds

reaching this time series computation final point, TSC_{FP}, there is no benefit in continuing computing new estimates, and it would be a risk to postpone the scheduling of rejuvenation mechanism. It is important to stress that the values adopted for TSC_{FP} and TSC_{SP} are specific to our environment, and therefore may vary if this strategy is applied to different systems.

6.3. Verifying the Proposed Rejuvenation Strategy

We carried out an experimental study to verify the effectiveness of proposed rejuvenation method on the described Eucalyptus cloud computing environment. We focused on the rejuvenation of Node Controller process, since it has shown the major aging effects among all monitored components.

It is noteworthy that the use of time series to reduce the system downtime during the execution of a rejuvenation action may also be applied to other computing environments. The time series itself is not involved directly in the system, but indicates the appropriate time at which an action should be taken.

We used data collected in the previous experiments to find out which kind of time series models has the better fitting for the growth of virtual memory usage in Node Controller processes.

Based on the empirical observation of virtual memory behavior (before process restarts) in Figures 3 and 4, we selected five models (LTM, QTM, GCM, SCTM, and ARIMA) to compare the accuracy of their trend analysis. A summary of the results of the models and their error rates are shown in Table 2, where $\widehat{Y_t}$ is the predicted value of the memory consumption at time t.

It can be seen from Table 2 that the QTM model have the smallest values for all used accuracy indices: MAPE, MAD and MSD. So the QTM model was chosen as the best fit for the trend analysis of virtual memory utilization in Eucalyptus node controllers, and therefore it was used for rejuvenation scheduling.

The actual experimental study for verifying the rejuvenation method was performed in two parts. In the first, the cloud environment was stressed with the workload described in Section 5, and the rejuvenation mechanism was triggered only when the monitoring script detected that the critical limit was reached. In the second experiment, the same workload was used, but the rejuvenation was scheduled based on the time-series predictions.

Figure 11 shows the trend analysis for the growth of virtual memory utilization, fit by a quadratic function $\widehat{Y_t} = 94429 + 3825.3t - 0.0686t^2$. Such analysis has provided a value of 809 minutes for the T_{CMU}, i.e., the predicted time to reach the 3 GB limit. By knowing the beginning time of the experiment, we scheduled the rejuvenation to a given $T_{rej} = 809 - (5/60 + 5)$, in minutes, counting up from the beginning of the experiment. After rejuvenation, the memory usage is reduced, and other trend analysis is carried out when 80% of the limit is reached again.

The results show that the proposed rejuvenation triggering method brings system availability to a higher level, when compared to the method which not relies on time series trend analysis. The number

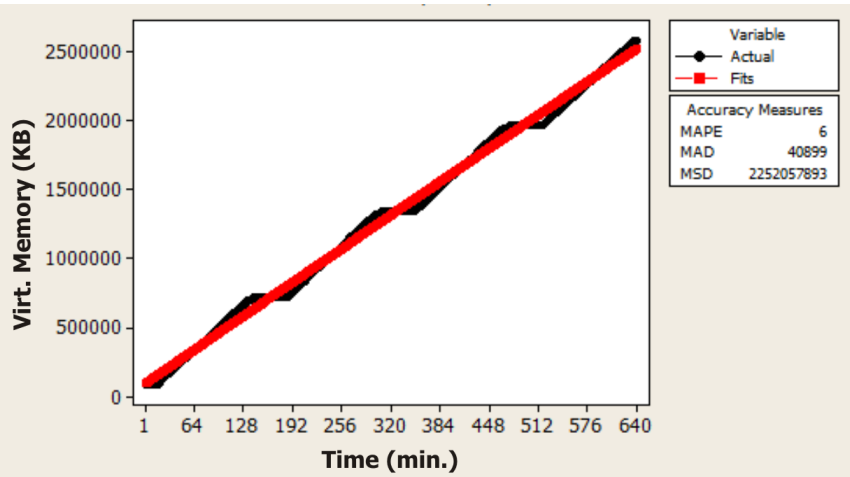

Figure 11. Quadratic Trend Analysis of Virtual Memory

Table 4. Comparison of accuracy indices for QTM and ARIMA predictions

Model	MAPE	MAD	MSD
QTM	2.01	54044.76	3690389330
ARIMA	2.206	59252.33	4254724867

of nines increases from 3.38 to 4.11 (see Table 3). In a time-lapse of one year, such a difference means a decrease from 218 to 40 minutes of annual downtime, i.e. the unavailability time was reduced about 80%, avoiding that requests for new virtual machine instantiations were rejected, even in the existence of enough computational resources to launch them.

Then, we compared the accuracy of the predicted values using the quadratic model and those ones provided by an ARIMA(2,1,2) model. Table 4 presents the error indices computed using only the values predicted from the time series computation starting point, i.e. 80% of memory utilization. The three indices (MAPE, MAD, and MSD) show that QTM has better accuracy than ARIMA for the environment used in this case study. Figure 12 confirms such conclusions since the QTM line is closer to the actual values measured in the experiment than ARIMA is in most analyzed points. We also noticed that the ARIMA model schedules the rejuvenation for a time instant approximately 5 minutes earlier than QTM does. The accumulation of such rejuvenations before the needed moment incurs in more restarts in a given large period, and a subsequent increase in downtime when compared to the rejuvenation based on the quadratic trend model.

7. Final Remarks

In this paper, we presented the software aging symptoms in the Eucalyptus cloud computing infrastructuree. Data collected during experiments indicated the presence of software aging, mainly related to virtual and resident memory usage. We propose an approach based on time series trend analysis to reduce downtime during the execution of rejuvenation actions. The rejuvenation scheduling adopted here is thus primarily threshold based but aided by predictions. We monitor a private cloud environment built with Eucalyptus framework running a specific workload, which reached critical limits of virtual memory usage in the node controllers. A rejuvenation strategy is used to avoid system unavailability, by scheduling the process restart to a proper time before the system stops due to the memory utilization limit.

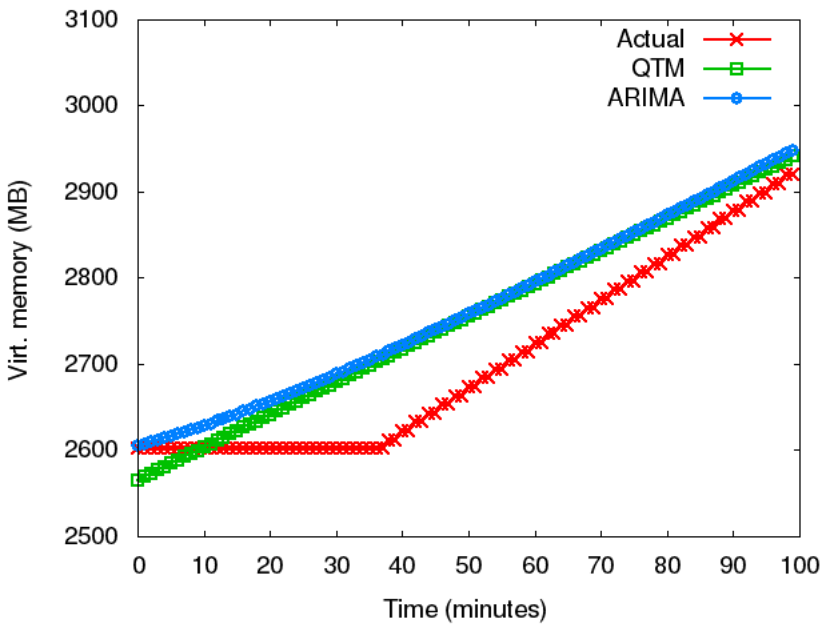

Figure 12. Comparison between predicted and actual values for virtual memory

In order to minimize downtime caused by the action of rejuvenation, we generate time series from sample data obtained at the beginning of the experiment. With the time series, we estimate in advance the time in which the rejuvenation action should be performed, considering that virtual memory usage follows a certain pattern of growth. The experimental results show the accuracy of our strategy, as well as a decrease of 80% in the downtime of the adopted Eucalyptus cloud computing environment. We showed that a quadratic trend model fits the behavior of Eucalyptus memory utilization better than an ARIMA model.

As future work, we intend to improve our approach using a larger range of time series models, including wavelets models. The minimization of downtime in different environments, such as data centers and traditional application servers is also planned.

Acknowledgements

This research was supported in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) under a JPL subcontract #1440119. Additionally, we would like to thank the CNPq, FACEPE and MoDCS Research Group for their support.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," UC Berkeley Reliable Adaptive Distributed Systems Laboratory, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.

[2] M. Grottke, R. Matias, and K. Trivedi, "The fundamentals of software aging," in *Proc 1st Int. Workshop on Software Aging and Rejuvenation (WoSAR), in conjunction with 19th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'08)*, Seattle, WA, Nov. 2008.

[3] R. Matias and P. J. F. Filho, "An experimental study on software aging and rejuvenation in web servers," in *Proc. of 30th Annual Int. Computer Software and Applications Conference (COMPSAC'06)*, Chicago, IL, Sep. 2006.

[4] Y. Bao, X. Sun, and K. S. Trivedi, "A workload-based analysis of software aging and rejuvenation," *IEEE Transactions on Reliability*, vol. 54, pp. 541–548, 2005.

[5] J. Gray, "Why do computers stop and what can be done about it?" in *Proc. 5th Int'l Symp. on Reliab. Distributed Software and Database Systems*. IEEE CS Press, Jun. 1985, p. 35.

[6] H. Okamura and T. Dohi, "Performance-aware software rejuvenation strategies in a queueing system," in *Proc. 2nd Int. Workshop on Software Aging and Rejuvenation (WoSAR), in conjunction with 21th IEEE Int. Symp. on Software Reliability Engineering (ISSRE)*, San Jose, CA, USA, Nov. 2010.

[7] D. L. Parnas, "Software aging," in *16th Int. Conf. on Software Engineering (ICSE-16)*, Sorrento, Italy, 1994, pp. 279–287.

[8] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[9] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. of 25th Symp. on Fault Tolerant Computing, FTCS-25*, Pasadena, CA, 1995, pp. 381–390.

[10] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *Proc. Int. Conf. on Dependable Systems and Networks (DSN'03)*, San Francisco, California, 2003, pp. 721 – 730.

[11] R. Matias, P. A. Barbetta, K. S. Trivedi, and P. J. F. Filho, "Accelerated degradation tests applied to software aging experiments," *IEEE Transaction on Reliability*, vol. 59(1), pp. 102 – 114, 2010.

[12] T. Thein and J. S. Park, "Availability analysis of application servers using software rejuvenation and virtualization," *Journal of Computer Science and Technology*, pp. 339–346, Mar. 2009.

[13] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, pp. 124–137, April 2005.

[14] Eucalyptus - the open source cloud platform. Eucalyptus Systems, Inc. [Online]. Available: http://open.eucalyptus.com/

[15] Nist. National Institute of Standards and Technology, Information Technology Laboratory, U.S. Department of Commerce. [Online]. Available: http://csrc.nist.gov)

[16] *Introduction to Cloud Computing Architecture*, 1st ed. Sun Microsystems, Inc., Jun. 2009.

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*. Washington, DC, USA: IEEE Computer Society, May 2009, pp. 124–131.

[18] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, "Dependability and security models," in *Proc. of 7th International Workshop on the Design of Reliable Communication Networks (DRCN 2009)*, Washington, DC, USA, Oct. 2009.

[19] M. Xie, K. Poh, and Y. Dai, *Computing System Reliability: Models and Analysis*. Kluwer Academic Publishers, 2004.

[20] J. D. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, 2nd ed. McGraw-Hill, 1998.

[21] D. Sun, G. Chang, Q. Guo, C. Wang, and X. Wang, "A dependability model to enhance security of cloud environment using systemlevel virtualization techniques," in *Proc. First International Conference on Pervasive Computing, Signal Processing and Applications (PCSPA 2010)*, Harbin Institute of Technology, China, Sep. 2010.

[22] B. Kedem and K. Fokianos, *Regression Models for Time Series Analysis*. John Wiley & Sons, Inc., Publication.

[23] P. Bloomfield, *Fourier Analysis of Time Series: An Introduction*. Wiley Series in Probability and Statistics, February 2000.

[24] H. Akaike, "Fitting autoregressive models for prediction," *Annals of the Institute of Statistical Mathematics*, vol. 21, no. 1, pp. 243–247, 1969.

[25] G. Box and G. Jenkins, *Time series analysis*, ser. Holden-Day series in time series analysis. San Francisco, CA: Holden-Day, 1970.

[26] C. Chatfield, *The Analysis of Time Series: An introduction*, 5th ed. New York, USA: Chapman & Hall/CRC, April 1996.

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

[28] G. Schwarz, *Estimating the Dimension of a Model*. Annals of Statistics, 1978.

[29] *Eucalyptus Open-Source Cloud Computing Infrastructure - An Overview*, Eucalyptus Systems, Inc., 130 Castilian Drive, Goleta, CA 93117 USA, Aug. 2009.

[30] Amazon elastic compute cloud. Amazon.com, Inc. [Online]. Available: http://aws.amazon.com/ec2/

[31] *Cloud Computing and Open Source: IT Climatology is Born*, Eucalyptus Systems, Inc., 130 Castilian Drive, Goleta, CA 93117 USA, 2010.

[32] J. D, K. Murari, M. Raju, S. RB, and Y. Girikumar, *Eucalyptus Beginner's Guide*, uec ed., May 2010, for Ubuntu Server 10.04 - Lucid Lynx, v1.0.

[33] Amazon elastic block store (ebs). Amazon.com, Inc. [Online]. Available: http://aws.amazon.com/ebs