A Nonlinear Dynamic Model for Software Aging in Service-Oriented Software

Yun-Fei Jia, Hui Xu, Ren-Biao Wu
Tianjin Key Laboratory for Advanced Signal Processing, Civil Aviation University of China, Tianjin, China
Email: yunfeijia1979@gmail.com, HuiXu@163.com，rbWu@cauc.edu.cn

***Abstract*—Software aging results from runtime environment degradation, and is significantly correlated with available computing resources. A set of variables evolving with time can describe the running state of computer system. Consequently, software aging is analogous to evolution of a dynamic system in this paper. We construct a nonlinear dynamic model based on the experimental observations. First, we assume the mathematical form of nonlinear dynamic equations. Then, we select resource parameters which can reflect the "health" of the whole computer system as variables of our model. Finally, we estimate the values of each parameters in our model using nonlinear inversion. Our approach is validated by two different datasets. The dynamic model can describe the evolution of software aging and interpret the interplay of various resource parameters. Moreover, this model can be used to forecast abrupt state degradation and help us to explore the root cause of software aging. For example, by comparing the output of our model against real values, with a suspected "aging factor" as input, we can identify which resource variable is the root cause of injuring the stability of computer system.**

***Index Terms*—Software aging; software maintenance; dynamic system**

I. Introduction

Software aging results from runtime environment degradation. [1] analyzes the time delay and available resources of computer system. For example, the process space or kernel space accumulate many error conditions after longtime execution, such as memory leak, round-off error, and out-of-order concurrent processes/threads [2]. These error conditions propagate and accumulate with time. Eventually, they will result in performance degradation even downtime. It should be noted that the code of software dose not degrade. The root origin of runtime error conditions is defects included in software. Usually, we cannot guarantee the large-scale and/or complex service-oriented software system dose not contain any defects. [3] states that diagnosing program error is a time-consuming hard work. Inherently, software aging refers to loss of available computing resources.

So far, less than ten publications discussing experimental studies on software aging can be found on major software and reliability journals [4-6]. This is contrast with the growing awareness and widely accepted importance of experiment-based studies [7]. The main advantage of experimental studies is that theories of concern can be validated or invalidated by hard evidence with respect to practical software processes. [5] analyzes two important resource variables representing system activity, i.e., memory and swap space, using trend estimation method. Their finding lies in that software aging is usually accompanied with resource exhaustion. Consequentially, they propose a metric "time to exhaustion". Nevertheless, there are several lacks in their study. First, too less resource variables are addressed. There are additional resource variables contribute to software aging, such as CPU usage, cache or buffer. Second, the interplay among these resource variables is not interpreted. For example, when memory is used up, the swapping rate will greatly increase. Consequently, swapping rate increase will consume much CPU time. Finally, software aging usually show nonlinear characteristics, as is overlooked in their study. To sum up, a challenge is to construct such a comprehensive model that it can describe the interplay among various resource variables, and forecast the paroxysmal state degradation of software system. This paper is aimed to meet this challenge.

Taking into account the fact that software state can be described by a set of evolving resource variables, we can analog the software system as a dynamic system. The advantage of this idea lies in that we can exploit the rich achievements in dynamic system in physics. The variables refer to resource variables with regard to software system. Likewise, for dynamic system such as thermodynamic system, the variables refer to temperature, pressure, volume, etc. In addition, nonlinear phenomenon researches in physics are usually based on established dynamic equation set. As for software system, the dynamic equation set cannot be known because it is a new object and not researched adequately. Inspired by nonlinear dynamic inversion method [8], we construct a dynamic model based on massive observations from software experiments. Our dynamic model can describe the evolution of software aging and can forecast specific resource variables. In addition, it can help us to understand the origin of software aging

The rest of this paper is organized as follows. Related studies are provided in Section 2. Section 3 introduces our method. The experimental set-up and observations are described in Section 4. Section 5 applies our model to forecasting paroxysmal state change and exploring the cause of software aging. Section 6 concludes this paper.

doi:10.4304/jsw.9.9.2260-2266

II. Related Studies

Model-based studies are the majority of software aging research. In [9], Huang et al. proposed a three-state stochastic model, including a robust state, a failure-prone state and a failure state. This model is solved to validate the effectiveness of software rejuvenation and determine the optimal schedules. This model was extended and studied in detail by many researchers to answer similar question [10,11].

Unlike model-based studies, measurement-based studies focus on practical software system, in which the data of interest are generated, collected, analyzed with the purpose of forecasting resource exhaustion time. The rationale behind measurement-based studies lies in that aging phenomenon is significantly related to resource usage of computer system [4-6]. Shereshevsky et al. [12] monitors the Hölder exponent (a measure of the local rate of fractality) of the system parameters and find that system crashes are often preceded by the second abrupt increase in this measure. In [5], a reward function is defined based on the rate of resource consumption to estimate time to exhaustion for each resource. Further, a metric "estimated time to exhaustion" is proposed to predict the approximate time of system resource depletion. A comprehensive evaluation function is proposed in [4] to measure the mean aging speed of the Apache server. [13] extracts two primary components from seven important resource variables via Principal Component Analysis (PCA) method.

To sum up, there is a large gap between model-based researches and measurement-based researches. This paper will bridge the gap by constructing a dynamic model based on experimental observations.

III. Introduction of Dynamic Inversion

A. Dynamic System

Roughly speaking, a dynamic system is a mathematical formalization for a set of interplayed objects (real or virtual objects), which are changing following a fixed "rule". Characteristics of dynamic system can be described by a set of state variables that measure concerned properties of these objects. Usually, dynamic system can be described by differential equation, integral equation or difference equation. For example, in a thermaldynamic system, thermaldynamic equations are used to formulate the rule of changing properties of the objects in the system. To take an example of gas, temperature, pressure and density of the gas are often used as state variables. Likewise, the evolving process of software system can be treated as a dynamic system. In this dynamic system, we are interested in the performance of Apache. Hence, we can use the parameters related with the performance of the server as state variables. In addition, we adopt different state variables for experiment I and experiment II as cross-validation of our conclusion, because the experimental configuration and subject software in both experiments are different, and key parameters of concern are different consequently.

B. Nonlinear Dynamic Inversion

The main rationale behind nonlinear dynamic inversion method is to determine the equations of the dynamic system using a set of input data. Nonlinear dynamic inversion has been employed in physics research extensively [8]. A common form of dynamic equations is as follows:

$$\frac{dx_j}{dt} = f_j(x_1, x_2, \cdots, x_n) \;\; j = 1,2, \cdots, n \tag{1}$$

Where $x_1, x_2, \cdots, x_n$, are state variables of the system. Given the formulation of $f_j(x_1, x_2, \cdots, x_n)$ and the value of $x_j(t_i)(i = 1,2, \cdots, q, j = 1,2, \cdots, n)$, we can estimate the values of various parameters in equation (1). Most nonlinear characteristics can be described by polynomials. Thus, we employ polynomials in our method. The discrete form of equation (1) can be written as:

$$\frac{dx_j}{dt} = \sum_{k=1}^{K} Q_k b_k \qquad j = 1,2, \cdots, n \tag{2}$$

In which, b_k denotes the coefficients of polynomials function Q_k; Q_k is a term of the polynomials. In our case, the discrete form of equation (2) is employed, which is shown in equation (3).

$$\mathrm{D} = (d_1, d_2, \cdots, d_{N-2})^T = \begin{bmatrix} \dfrac{x_j(3\Delta t) - x_j(\Delta t)}{2\Delta t} \\ \dfrac{x_j(4\Delta t) - x_j(2\Delta t)}{2\Delta t} \\ \vdots \\ \dfrac{x_j(N\Delta t) - x_j\big((N-2)\Delta t\big)}{2\Delta t} \end{bmatrix} = \sum_{k=1}^{K} Q_k b_k \tag{3}$$

In this paper, we need to improve the calculation accuracy of the D in equation (3). We exploit a type of non-parametric algorithm developed by Sen to estimate the slope of variables [14].This method is not affected by outliers, and it is robust to missing data. This approach focuses on all pairs of data points y_k, y_l with $\mathrm{k} < l$. For each of these pairs, the slope $q_{kl} = (y_l - y_k)/(l - k)$ is calculated. Sen's slope estimate is defined as the median of the $\mathrm{n}' = \mathrm{n}(\mathrm{n} - 1)/2$ slopes obtained.

Substitute equation (3) into the right items of equation (2), then substitute $x_j(t_i)$, the obtained Q is shown as follows:

$$\mathrm{Q} = \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1N} \\ Q_{21} & Q_{22} & \cdots & Q_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ Q_{N-2,1} & Q_{N-2,2} & \cdots & Q_{N-2,N} \end{bmatrix} \tag{4}$$

Let $\mathrm{b} = \{b_1, b_2, \cdots, b_k\}^T$, then equation (1) can be written as:

$$\mathrm{D} = \mathrm{Qb} \tag{5}$$

Equation (5) is overdetermined equations with regard to vector b. This can be solved by generalized linear inversion method, as is omitted in this paper.

IV. EXPERIMENTS

A. Experimental Setup

The experimental setup consists of a server running Apache httpd 2.0 and three clients connected via an Ethernet local area network. Apache is deployed on a computer (CPU: Pentium III 776 Mhz, RAM: 256 Megabytes, NIC 100 Mbps, OS: Fedora Core 6). Three other computers with the same hardware configuration are used as clients to generate artificial concurrent requests to access static web pages on the Apache server. They are all connected via a switch. Figure 1 shows the schematic diagram of our experimental set-up.

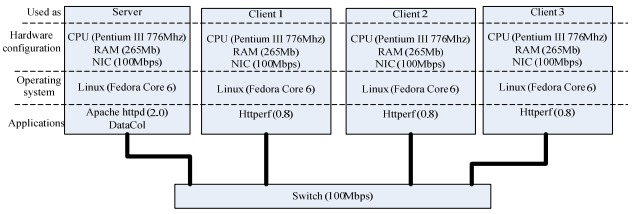

Figure 1. Experimental setup

In this paper, we focus on the software aging phenomenon of Apache httpd 2.0. The early version of this software system was also used in the studies on software aging by other researchers [4]. However, the contemporary Apache httpd 2.0 provides many new features, and its aging phenomenon should be reviewed. Since Apache has been well tested in practice, it is difficult for us to observe its aging symptoms in a short period under a normal runtime environment and the default parameter settings. It is necessary to find some way to expedite the aging of Apache. In the experiments, we adjust two parameters that are related to the accumulation of the effects of software errors: **MaxRequestPerChild** and **MaxSpareServers**. The first parameter limits the number of requests handled by each child process of Apache. For example, when it is set to 10, a child process of Apache will be killed after it has handled 10 requests. After the old child process is killed, a new one will be created to replace the old one to handle subsequent requests. This periodical cleaning mechanism reduces the accumulation of runtime memory leak. In our experiments, this parameter is set to zero which means runtime errors will accumulate all through each experiment. The second parameter, **MaxSpareServers**, sets the maximum number of idle child processes. When the number of requests is low, some of existing child processes may be at idle state. If there are more than **MaxSpareServers** idle processes, Apache will kill excess ones. By setting it to zero, we can turn off this mechanism so that no child processes will be killed during runtime.

Apart from above set-up, we use the method in [4] to determine the capacity of the server. We set **MaxRequestsPerChild** and **MaxClients** to 0 and 250 respectively, to maximize the capacity of the server. The number of requests coming from the three clients is gradually increased. The reply rate, error rate and response time with respect to different connection rates are recorded. Result shows that the capacity of the web server is about 390 requests per second. This parameter is used by *httperf* [15], a web server test tool, deployed on the three clients, to generate artificial connection requests for static html pages with exponential time intervals to the web server.

We implemented a program named *DataCol* to monitor the resource usage of the operating system. It can online collect the metrics that are related with the performance of the server, such as available memory, size of caches, and CPU usage, etc.

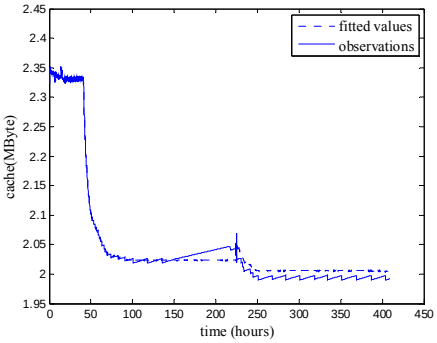

Figure 2. Real value and output of our model of cache

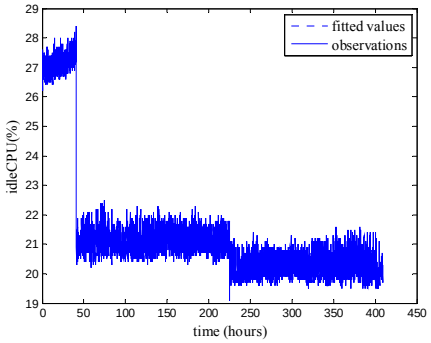

Figure 3. Real value and output of our model of available CPU

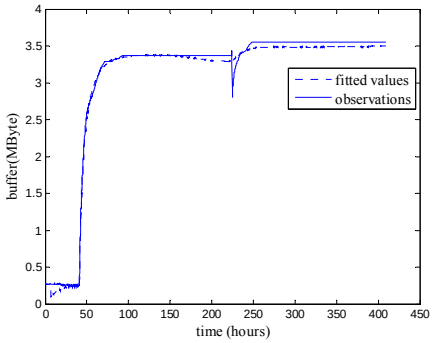

Figure 4. Real value and output of our model of buffer

B. Observations of Experiment I

Experiment I lasts for more than 400 hours. Three resource variables change obviously with time. They are cache, available CPU and buffer, as are shown by solid line in Figure 2, Figure 3 and Figure 4 respectively.

From Figure 2 we can see that, cache usage decreases with time. Rather, cache usage is approximately proportional to system performance. This is to say, higher cache usage will provide better system performance. Thus, cache usage decrease may be a cause of software aging. From Figure 3 we can see idle CPU decreases with time, showing that the system is more and more busier

with time. In Figure 4, buffer usage increases greatly with time, this may be attributed too much data are blocked in buffer.

Previous studies often focus on trend or characteristics analysis of various resource variables independently, as is contrary to practice. In next section, we will employ dynamic model to describe the entire software system, with purpose of describing the state degradation of the whole software system.

C. Nonlinear Dynamic Model

We adopt three variables in equation (2), and let X, Y and Z denote cache, buffer and available CPU respectively. In equation (2), the number of the highest order of Q, denoted by m, can be an arbitrary integer. The higher m is, the more accurate the model is. However, too large m will cause over-fitting problem that prejudices the generalization of the model. In our case, we got the best trade-off when m is set to 2. Then, equation (2) can be written as follows:

$$\begin{aligned}\frac{dX}{dt} &= a_1X + a_2Y + a_3Z + a_4X^2 + a_5Y^2 + a_6Z^2 \\ &\quad + a_7XY + a_8XZ + a_9YZ + a_{10} \\ \frac{dY}{dt} &= b_1X + b_2Y + b_3Z + b_4X^2 + b_5Y^2 + b_6Z^2 \\ &\quad + b_7XY + b_8XZ + b_9YZ + b_{10} \\ \frac{dZ}{dt} &= c_1X + c_2Y + c_3Z + c_4X^2 + c_5Y^2 + c_6Z^2 \\ &\quad + c_7XY + c_8XZ + c_9YZ + c_{10} \end{aligned} \quad (5)$$

In which, $a_1, a_2, \cdots, a_{10}$, $b_1, b_2, \cdots, b_{10}$, $c_1, c_2, \cdots, c_{10}$, are parameters to be determined. Following the method in section III, we get the dynamic model and compare the output of our model against real values, which are shown by dotted line in Figure 2, Figure 3 and Figure 4.

From Figure 2, Figure 3 and Figure 4 we can see that the output of our dynamic model can fit the real values with high accuracy. In addition, the output of our dynamic model has filtered the vibrations of real vale, which enable us to study the nature of software aging.

D. Experiment II

To validate the effectiveness of our dynamic model, we apply our modeling method to the data set in [4]. The data collection process in [4] can be described as follows: The SNMP (Simple Network Management Protocol)-based distributed resource monitoring tool developed by Garg et al. was used for data collection. The resource monitoring tool was used to collect operating system computing-resource usage data (physical/virtual memory usage, file/process table usage, etc.) and system activity data (paging activity, CPU utilization, etc.) from nine het erogeneous UNIX-like workstations. These workstations were connected by an Ethernet LAN at the Duke Department of Electrical and Computer Engineering. These workstations provide various services, and inputs from clients are unknown. The objects or parameters collected on the workstations include those that describe the state of the operating system resources, state of the processes running, information on the /tmp file system, availability and usage of network related resources, and information on terminal and disk I/O activity. More than 100 such parameters were collected at regular intervals (10 mins) for more than three months. In this paper, we focus on the data collected from the workstation named Rossby as cross-validation.

There are several reasons to select the data in [4] as cross-validation: 1) The data set is collected at software system runtime rather than controlled experiment, so the conclusion is more helpful for software process. 2) The experiment II lasts for more than 1000 hours; 3) The subject software, Operating system and hardware configuration are quite different from experiment I.

In experiment II, we find the available CPU does not increase obviously, whereas storage-related variables, such as memory usage, net service memory, or swap usage, show obvious growing trend. Accordingly, we include these resource variables in our model. In order to obtain better fitting accuracy, we scale the three variables in the range [0-100]. The comparison of output of model against real value is provided in Figure 5, Figure 6 and Figure 7.

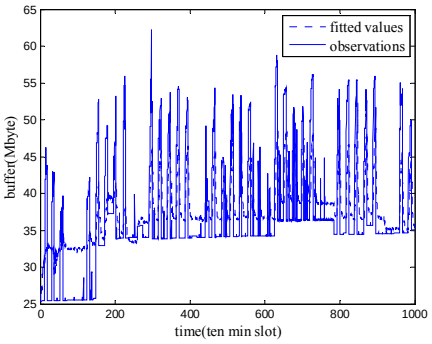

Figure 5. buffer usage

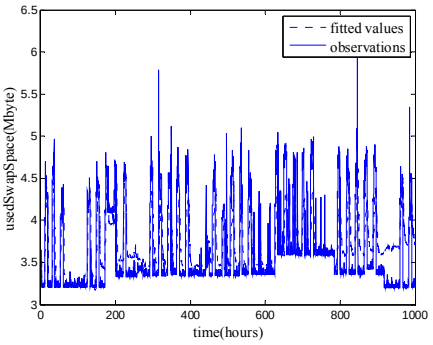

Figure 6. swap usage

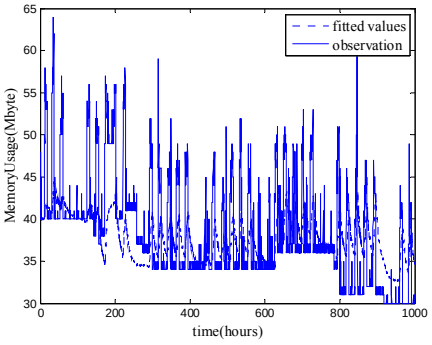

Figure7. net service memory usage

We can see from Figure 5 to Figure7 that, our model fits the observations with high accuracy. Particularly, the spike shown in all three resource variables can be tracked by our model. To sum up, our model can be generalized to describe the behavior of more software systems.

V Application

A. Forecasting Software State Jump

In some scenarios, the usage of some resources must be limited under a threshold. For example, if the CPU usage of a web service system is over 40% long time, the system will crash with high probability. However, at the runtime of software system, its state often jumps paroxysmally, rather than degrades smoothly as we imagine. A question naturally arise is that how to forecast the paroxysmal jump of certain resource variable. In this subsection, we use the data set in experiment I to illustrate the effectiveness of our method.

First, we need to construct the dynamic model with some samples from the observations in experiment I. There are 5259 observations in experiment I. This dataset is equally partitioned into two subsets, the first subset is used for constructing our dynamic model, and the latter subset is used for testing our approach. The forecasting results are listed in Figure 8, Figure 9 and Figure 10.

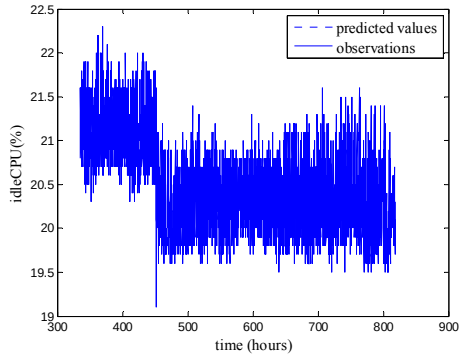

Figure 8 Available CPU

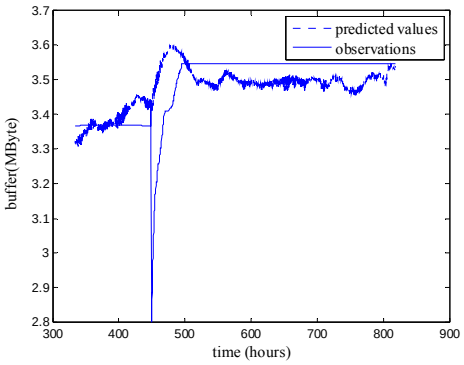

Figure 9 buffer usage

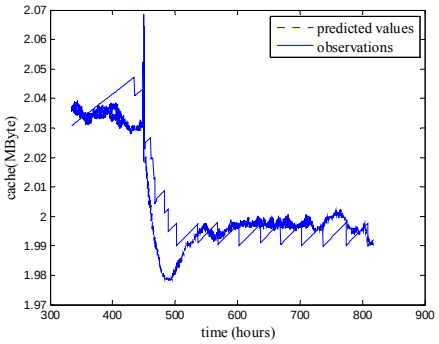

Figure 10. cache usage

From Figure 8, Figure 9 and Figure 10 we can see that the output of dynamic model can track the real value with high accuracy. Particularly, the state jump can be tracked.

B. Analysis of Possible Causes of Software Aging

When the computing resources are nearly exhausted, variation of a resource variable will cause other resource variables to change consequently. We have observed that three important resource variables change obviously. A following question is how to identify which resource variable is the root cause of injuring the stability of resource distribution of computer system. We will simulate the dynamics of the software system in experiment I. We assume the root cause of software aging is buffer increase, idleCPU decrease and cache decrease respective in our three times of simulation, and test whether the output of our model can fit the observations.

More specifically, we input the real value of one parameter and the initial value of other two parameters into the model, and calculate the value of the other two parameters with time. For example, we take the real value of buffer and the initial values of idleCPU and cache as input parameters. Then we calculate the idleCPU and cache of our dynamic model at next interval by recurrence, with real value of buffer as the input value. The result is illustrated in Figure 11 and Figure 12. In addition, we also take cache or idleCPU as input parameters respectively and calculate the other two parameters. The results are not listed because the output of dynamic model cannot fit the observations.

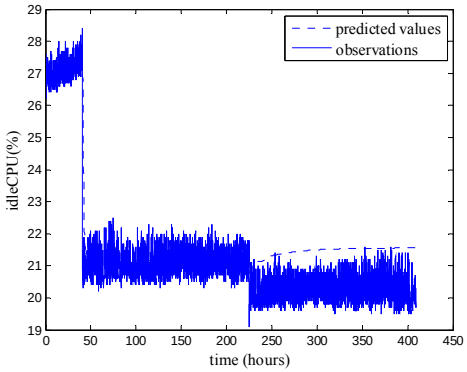

Figure 11. Forecasting available CPU

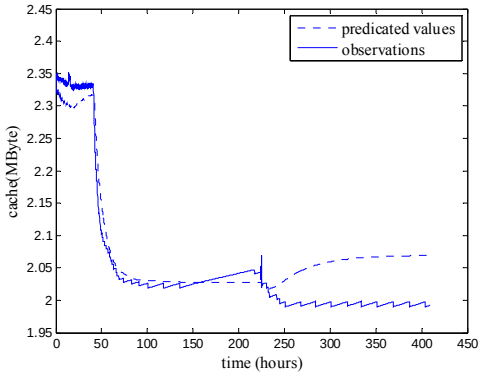

Figure 12. Forecasting of cache usage

From Figure 11 and Figure 12, we can see that our assumption that, buffer usage increase incurs change of other parameters, will output similar dynamics as experimental observations. Obviously, we can see there is a large error in Figure 12. There are several reasons: 1) Our model employs only three variables, so the relationship between the three variables is not accurately described; 2) We assume that buffer usage increase is the only cause of the other two variables, many secondary factors are not included in our model; 3) More higher order of polynomials will bring higher accuracy, whereas we just employ quadratic polynomials.

Despite of the large errors shown in Figure 11 and Figure 12, we think our model is effective, because our object in the simulation is to explore which resource variable is the root cause of software aging, instead of accurately forecasting the value of cache or available CPU. Our analysis can help us to understand the behavior of software system when it gradually ages.

VI CONCLUSION

In this paper, we report the software aging phenomena observed in two service-oriented software systems, which are described in experiment I and experiment II. Both experiments show nonlinear aging signs. Further, the resource variables of software system in experiment I are mutually interplayed during their evolving process, and shows state jump. This observation is validated by experiment II. For the first time, software aging is analogous as state evolution of a dynamic system. Consequently, we formulate the experimental observations in a nonlinear dynamic model via method of dynamic inversion. Using the dynamic model, we can explore the causes of software aging, i.e., which resource variable is the root origin of software aging, and forecast the paroxysmal state change of software system. This dynamic model can help us understand software aging phenomenon and dynamic behavior of software system at runtime. In addition, this dynamic model can forecast the nonlinear state jump of software system. Finally, our method can be easily generalized to other software systems.

ACKNOWLEDGEMENT

The authors would like to thank Professor Kishor S. Trivedi, the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, and Michael Grottke for providing the datasets that are analyzed in this paper. This work is supported by the The National Key Technology R&D Program (Grant No. 2011BAH24B12) and the Fundamental Research Funds for the Central Universities (Grant No. 3122013P006).

REFERENCES

[1] Liang F., Ma S., Luckow A. and Schnor B.“Advance Reservation-based Computational Resource Brokering using Earliest Start Time Estimation,” *Journal of Computers*, 2012 V7(6):1329-1336

[2] http://www.software-rejuvenation.com

[3] Pu F., “Assumption-based Reasoning with Constraints for Diagnosing Program Errors,” *Journal of Computers*,2014 V9(1):1-11

[4] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, “Analysis of Software Aging in a Web Server,” *IEEE Transaction on Reliability*, 55(3):411-420, September 2006.

[5] A. Vaidyanathan, and K.S. Trivedi, “A Comprehensive Model for Software Rejuvenation,” *IEEE Transaction on Dependable and Secure Computing*, 2005, V2: 124-137

[6] 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,* 59(1):102-114, December 2009

[7] Cai K.Y. “Software Reliability Experimentation and Control,” *Journal of Computer Science and Technology*, 2006, V21(5): 697-707

[8] Sara D.C., Caroline F.J., Sophie P. and Raul M., “Dynamic inversion of the 2000 Tottori earthquake based on elliptical subfault approximations,” *Journal of Geophysical Research: Solid Earth*, 115(B12)(1978-2012), December, 2010

[9] Huang Y., Kintala C., Kolettis N., et al., “Software Rejuvenation: Analysis, Module and Applications,” *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing*, pp.381-390,1995

[10] Zhao J., Wang Y.B., Ning G.R., Trivedi K.S., Matias R. Jr., Cai K.Y., “A comprehensive approach to optimal software rejuvenation,” *Performance Evaluation*, 2013

[11] Liang Y.W., Yang H., Fu J., Tan C.Y., Liu A.L., Zhu S.W., “The Effect of Real-valued Negative Selection Algorithm on Web Server Aging Detection,” *Journal of Sofware*, pp.849-855,2012

[12] Shereshevsky M., Crowell J., Cukic B., et al., “Software Aging and Multifractality of Memory Resources,” *Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pp:721-730, 2003

[13] Jia Y.F., Chen X.E., Zhao L. et. al., “On the Relationship between Software Aging and Related Parameters,” *Proceedings of the 8th International Conference on Quality Software*, pp.: 241- 246, 2008

[14] Sen P.K., “Estimates of the Regression Coefficient Based on Kendall’s Tau,” *Journal of the American Statistical Association*, 63:1379–1389, 1968.

[15] Mosberger D. and Jin T., “Httperf - A Tool for Measuring Web Server Performance”, *In First Workshop on Internet Server Performance,* Madison, Jun. 1998.

Yun-Fei Jia is currently a lecturer in the School of Electronical & Information Engineering at Civil Aviation University of China. He received a B.E.(2001) and a M.S. (2004) degrees from HeBei University of Technology, and completed a PhD in software testing at BeiHang University in 2010. His research interests include software testing and software measurement.

Hui Xu is a Master student in School of Electronic Information Engineering at Civil Aviation University of China.He gained the B.E. Degree from Civil Aviation University of China in 2011.

Ren-Biao Wu was born in Wuhan, China, in 1966. Currently, he is a professor at Civil Aviation University of China. His interests are signal processing and image processing.