Software Aging Prediction and Availability Analysis

Ohnmar Nhway
University of Computer Studies
skynhway@gmail.com

Abstract

The 'software aging', is characterized by progressive performance degradation due to the exhaustion of operating system resources. Software rejuvenation should be planned and initiated in the face of the actual system behavior which requires measurement, analysis and prediction of system resource usage. In this paper, we present availability analysis and prediction for software aging in virtualized environment. The difficulty is that it can be due to two or more resources simultaneously involved in the service failure. So, we decide to evaluate the use of powerful Machine Learning algorithms to predict the time to crash (TTC) of a system which suffers from software aging phenomena. We also present time dependent software rejuvenation (TDSR) policy that can be applied with predictable data in virtualized environment. The behavior of the system is represented through a Stochastic Petri Net (SPN) model. Numerical analysis of the system availability is carried out the SHARPE tool simulation.

1. Introduction

In long running client-server, software faults are the major cause of computer system failure. Several recent studies have established that most system outages are due to software faults. Software rejuvenation is a technique for software fault tolerance which consists of stopping the executing software, 'cleaning' the 'internal state' and restarting. So, we compose virtualization technology for solving to protect software aging.

Virtualization technology, which allows multiple operating systems to run different applications on a single computer, has caught the attention of IT managers for its promise to let them better manage and utilize corporate IT resources. Moreover, availability is one of the key characteristics of virtualized data center.

In this paper, we focus on prediction for software aging due to resource exhaustion such as memory. Machine Learning Algorithms has been analyzed for predicting system crashes due to the resource exhaustion caused by software aging [4]. So, we measured Apache web server performance in such a system it is necessary to run a tool on the clients that generates a specific HTTP workload. We mainly emphasize the healing of software aging of web server in single physical machine and hardware failure is not considered.

The rest of the paper is organized as follows. Related research is highlighted in Section 2. We explain the system architecture and TDSR policy model in Section 3. We present the experimental evaluation using Nagios Monitoring Tool, Httperf Tool and Weka 3.5.8 is in Section 4. We conclude this work and discuss future avenues of research is Section 5.

2. Related Works

Many researchers have been constructed the models to help predict when the software rejuvenation should occur. The focus in this paper is on “how” to rejuvenate or prevent the crash failures, the gracefully termination of predictable model before it is restarted.

Y. Huang et al. [10] has suggested a complimentary technique which was preventive in nature. It involved periodic maintenance of the software so as to prevent crash failures. They called it Software Rejuvenation, and defined it as the periodic preemptive rollback of continuously running applications to prevent failures.D. Seong Kim et. al. [2] constructed non-virtualized and virtualized two hosts system models using a two-level hierarchical approach in which fault trees are used in the upper level and homogeneous Continuous Time Markov chains (CTMC) are used to represent sub models in lower level. Then, they incorporated not only hardware failures but also software failures including VMM, VM, and application failures.

J. Alonso et al. [5] focused on the software aging prediction model based on M5P (a well-known ML algorithm) and its evaluation in front of a varied and complex software aging scenarios. They evaluated three algorithms of Machine Learning (ML) (Linear Regression, Decision Trees, and M5P) to check whether they offered the right capabilities to model software aging phenomena, and M5P offered the best performance.

The authors proposed to predict the time until crash when the web server was suffering transient failures that consumed resources randomly [6]. The authors in [7] developed a measurement-based approach using time-series analysis to detect software aging and estimate resource exhaustion times due to aging in a web server.

The authors [9] introduced web server aging. To avoid the problem, it became an important issue to detect web server aging in web server maintenance. Grottke et al. [3] studied the development of resource usage in a web server while subjecting it to an artificial workload. They only focused on software rejuvenation using Apache based on Linux. The authors [8] analyzed resource usage data collected on an Apache web server.

In this paper, we focus on software aging caused by resource exhaustion (the most common cause of software aging), which causes finally a crash. We use Machine Learning to overcome the difficulty for predicting the time to crash due to software aging. We

consider not only software rejuvenation but also virtualization technology.

3. Proposed System

3.1. Overall System Architecture

In this section, we present the system architecture which consists of server and client running Apache 2.2.17 based on Linux platform on the top of Virtualization Middle ware in Figure 1. On the top of the virtualization middle layer, we create virtualized two hosts system which consists of two virtual machines running on the Virtual Machine Monitor in the host. Two virtualized hosts share a common storage (CPU, Memory, etc).

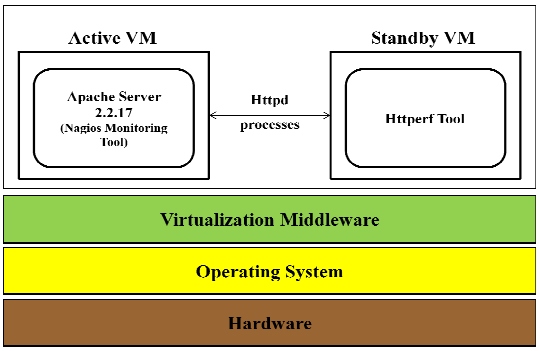

Figure1. System Architecture

To predict the Time To Crash (TTC) (due to resource exhaustion causing the software aging phenomena) based on a limited set of system metrics easily available in any operating system, we need to predict the resource usage nature with the effectiveness of Machine Learning algorithms. Thus we choose the two well-known algorithms: Linear Regression and M5P for prediction.

3.1.1. Linear Regression

In linear regression, data is modeled using linear predictor functions, and unknown model parameters are estimated from the data. A Linear regression of variables x_1, …..,x_n is a function of the form

$$y_i = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i \qquad (1)$$

for some set of coefficients α_0 , ... , α_n.It will work well to predict some variable y when (and only when) y depends linearly on the x_i variable. The α coefficients are calculated from the training data, trying to minimize the sum of the squares of these differences over all the training instances. It is an efficient, simple method for numeric prediction and widely used today to help for capacity planning analysis or detecting anomalies on the system.

3.1.2. M5P

The M5P or M5Prime algorithm [Wang & Witten, 1997] is a regression-based decision tree algorithm, based on the M5 algorithm by Quinlan [1992]. M5P builds a tree to predict numeric values for a given instance. The algorithm requires the output attribute to be numeric while the input attributes can be either discrete or continuous. Each leaf has a linear regression model associated with it in Equation (1). As the leaf nodes contain a linear regression model to obtain the predicted output, the tree is called a model tree.

3.2. Time-based Software Rejuvenation Policy using Stochastic Petri Net Model

3.2.1. State Transition of TDSR Policy

The state transition diagram of time dependent software rejuvenation policy for server virtualized system is shown Figure 2. In this model there are five states: the healthy state (up_i), the failure probable state (fp_i), the rejuvenation state (rej_i) and the down state ($D_1$and D_{own}) and the failure state (F).

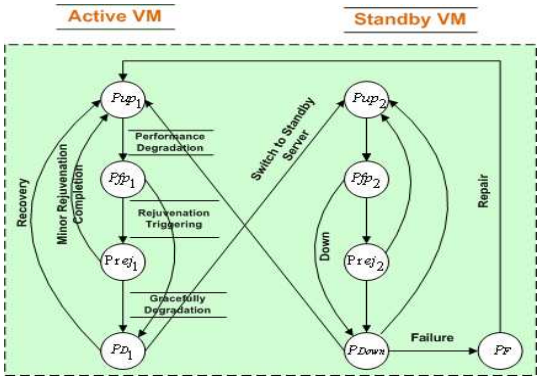

Figure2. State Transition Diagram of TDSR Policy

3.2.2. Stochastic Petri Nets Model for VM Aging

Figure 3 shows the SPN model for time dependent software rejuvenation policy in single physical server. The circles represent places with dots inside representing the tokens held inside that place. It consists of active-standby virtual machine servers. Both VMs are "healthy" working states, indicated by a token in place P_{up1} and P_{up2}. Active VM eventually transits to failure probably states in place P_{fp1} and P_{fp2} through the transition T_{fp1} and T_{fp2} due to the software's performance degradation.

The active VM is still operation in this state. At that time, VM can be entered the failure state (Place P_{D1}) through the firing transition T_{down1}. On the other hand, the VM can be rejuvenated (a token is placed in P_{FP1} or P_{FP2}), the transition T_{r1} or T_{r2} is enabled and the token moves to P_{rej1} or P_{rej2}. During the software restart, every other activity is suspended; from place P_{D1} and P_{Down} to transition T_{sw1} and T_{sw2} are used to model this fact. But VM can be switched over to another standby VM. When transition T_{sw} is enabled, the operation of active VM is

switched to standby VM and a token is moved to a place P_{up1}. When the place P_{Down} transfers to the place P_F using transition T_f , the place P_F is repaired using transition T_{repair}. After that operation will be restarted on standby VM. From a full system outage, the system can be repaired through the transition T_{repair}, all VMs are in healthy state in this model. The description of places is shown in Table 1.

Table1. TDSR policy model descriptions

Places	Description
P_{upi}	Healthy state of i^{th} virtual machines
P_{fpi}	Failure Probably state of i^{th} virtual machines
P_{Dl}	VMs or one active physical host failure state
P_{Ri}	Rejuvenation state of i^{th} virtual machines
P_{Down}	VMs or one active physical host down state
P_F	Failure state of i^{th} virtual machines (i= 1,2,3,4, … where i =number of VM)

3.2.3. Structure of Reachability Graph

In this section, we construct the reachability graph for the proposed model. This graph represents the active physical server with two virtual machines (2VMs on 1PM) at both active-standby virtual machines. Let the 9-tuple ($\#P_{up1}$, $\#P_{fp1}$, $\#P_{R1}$, $\#P_{D1}$, $\#P_{up2}$, $\#P_{fp2}$, $\#P_{R2}$, $\#P_{Down}$, $\#P_F$) denote the markings of the Petri net model. A marking is reachable from another marking if there is a sequence of transition firings starting from the original marking that result in the new marking. Let λ_1, λ_2, μ_r, μ_{up}, λ_d, λ_{down} ,λ_{sw} and μ_{rep} be the transition rates associated with T_{fp}, T_{rej}, T_r, T_{up}, T_d, T_{down}, T_{sw} and T_{repair} respectively. The steady-state balance equations of the system are as follows:

For marking up_1

$$\mu_{up}P_{D_1}+\mu_r P_{R_1}+\lambda_{sw}P_{D_2}+\mu_{rep}P_F=\lambda_{fp}P_{up_1} \quad (2)$$

For marking fp1

$$\lambda_{fp}P_{up_1}=\lambda_{rej}P_{fp_1}+\lambda_{down}P_{fp_1} \quad (3)$$

For marking R_1

$$\lambda_{rej}P_{fp_1}=\lambda_d P_{R_1}+\mu_r P_{R_1} \quad (4)$$

For marking D_1

$$\lambda_d P_{R_1}+\lambda_{down}P_{fp_1}=\lambda_{sw}P_{D_1}+\mu_{up}P_{D_1} \quad (5)$$

For marking up_2

$$\mu_{up}P_{D_2}+\mu_r P_{R_2}+\lambda_{sw}P_{D_1}=\lambda_{fp}P_{up_2} \quad (6)$$

For marking fp_2

$$\lambda_{fp}P_{up_2}=\lambda_{rej}P_{fp_2}+\lambda_{down}P_{fp_2} \quad (7)$$

For marking R_2

$$\lambda_{rej}P_{fp_2}=\lambda_d P_{R_2}+\mu_r P_{R_2} \quad (8)$$

For marking Down

$$\lambda_d P_{R_2}+\lambda_{down}P_{fp_2}=\lambda_{sw}P_{D_2}+\lambda_f P_{D_2}+\mu_{up}P_{D_2} \quad (9)$$

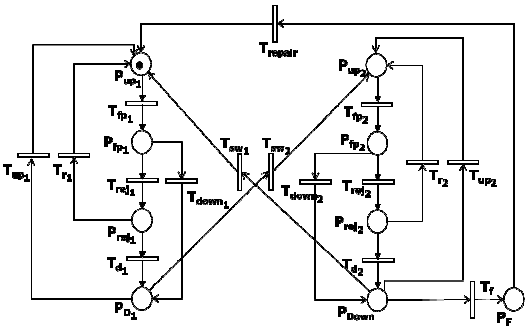

Figure 3. TDSR Policy using Stochastic Petri Net Model

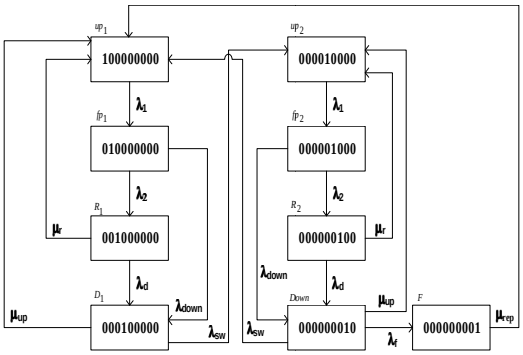

Figure 4. Reachability Graph for time based software rejuvenation policy using Stochastic Petri Net model

For marking F

$$\lambda_f P_{D_2}=\mu_{rep}P_F \quad (10)$$

The conservation equation of Figure 4 is obtained by summing the probabilities of all states in the system and the sum of equation is 1.

$$\sum_{i=1}^{n-1}P_{up_i}+\sum_{i=1}^{n-1}P_{fp_i}+\sum_{i=1}^{n-1}P_{rej_i}+\sum_{i=1}^{n-1}P_{D_i}+\sum_{i=1}^{n}P_{F_i}=1 \quad (11)$$

$$P_{up_2}=\{1+[\frac{(\lambda_{sw}\lambda_{fp}+\lambda_f\lambda_{fp})(\lambda_{rej}\lambda_d+\lambda_{down}\lambda_d+\lambda_{down}\mu_r)}{(\lambda_{sw}+\lambda_f+\lambda_{fp})}$$
$$\frac{(\lambda_{sw}+\mu_{up})}{\lambda_{sw}\lambda_{fp}\lambda_{down}(\lambda_d+\mu_r)+\lambda_{sw}\lambda_{fp}\lambda_d\lambda_{rej}}]\{[A]+[A][B]$$
$$+\frac{1}{(\lambda_{sw}+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]\}+[A]+[A][B]$$
$$+\frac{1}{(\lambda_{sw}+\lambda_f+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]+\frac{\lambda_f}{\mu_{rep}}$$
$$\frac{1}{(\lambda_{sw}+\lambda_f+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]\}^{-1} \quad (12)$$

Combining the above-mentioned balance equations with the conservation equation, and solving these simultaneous equations, the closed-form solution for the system is acquired.

$$P_{up_1} = \frac{(\lambda_{sw}\lambda_{fp}+\lambda_f\lambda_{fp})(\lambda_{rej}\lambda_d+\lambda_{down}\lambda_d+\lambda_{down}\mu_r)}{(\lambda_{sw}+\lambda_f+\lambda_{fp})} \frac{(\lambda_{sw}+\mu_{up})}{\lambda_{sw}\lambda_{fp}\lambda_{down}(\lambda_d+\mu_r)+\lambda_{sw}\lambda_{fp}\lambda_d\lambda_{rej}} P_{up_2} \quad (13)$$

$$P_{fp_1} = [A]P_{up_1} \quad (14)$$

$$P_{R_1} = [A][B]P_{up_1} \quad (15)$$

$$P_{D_1} = \frac{1}{(\lambda_{sw}+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]P_{up_1} \quad (16)$$

$$P_{fp_2} = [A]P_{up_2} \quad (17)$$

$$P_{R_2} = [A][B]P_{up_2} \quad (18)$$

$$P_{D_2} = \frac{1}{(\lambda_{sw}+\lambda_f+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]P_{up_2} \quad (19)$$

$$P_F = \frac{\lambda_f}{\mu_{rep}}\frac{1}{(\lambda_{sw}+\lambda_f+\mu_{up})}[A][\lambda_{down}+\lambda_d.B]P_{up_2} \quad (20)$$

Where,

$$A = \frac{\lambda_{fp}}{(\lambda_{rej}+\lambda_{down})} \qquad B = \frac{\lambda_{rej}}{(\lambda_d+\mu_r)}$$

4. Experimental Evaluation

4.1. Experimental Setup

In this subsection, we present the Machine Learning algorithms described previous section, we have conducted a set of experiments to evaluate the effectiveness of our approach for the software aging scenarios. Firstly, we have collected data from Apache server to predict for software aging caused by resource exhaustion in Figure 5. Apache server is one of the most popular used web server systems, and discovered that the Apache server shows evidence of aging after inserting memory leak into the web site running in it.

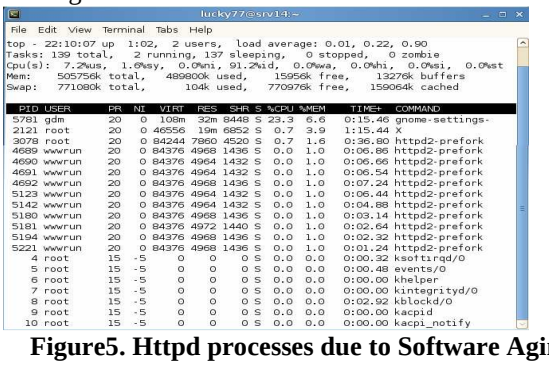

Figure5. Httpd processes due to Software Aging

Moreover, httperf is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance [12]. According to Figure 5, software aging has occurred 3078 PID and connection rates are used about 20000.

Figure6. Nagios monitoring process for critical state at server site

The monitoring subsystem's main task is to obtain the system activity report metrics. We use the monitoring agent that collects metrics from the whole system and specifically related to the web application server under monitoring such as memory used by the application server, CPU used, response time, etc. The current version of the system-level monitoring agent is based on the well-known and used monitoring tool called Nagios as shown in Figure 6. It offers a scalable and easy to upgrade monitoring framework used in several computer centers.

An implementation of Linear Regression and M5P extracted from the WEKA distribution is used for prediction purpose. The idea is to use ML classifiers to normal state (class=0) and aging detected critical state (class=1). Tuples in the training dataset were labeled 0 and 1 as it is presented in Table 2. We have used 104 instances from 9 parameter value.

Table2. Variables used in experiment to build TDSR policy model

Ex-time	L1 CPU	L5 CPU	L15 CPU	Response time	speed	TTC	Memory Usage	Class
8	2.48	9.4	5.53	0.86	44.8	3.4	358.4	0
17	2.86	18.23	15.64	0.119	21.08	7.2	358.4	0
26	2.62	9.29	12.49	0.128	13.78	11.1	358.4	0
50	21.29	18.68	18.3	0.07	7.168	21.4	358.4	0
179	27.08	36.49	20.11	0.201	2.002	76.7	358.4	0
232	27.08	36.49	20.11	10	1.98	25.77	460.8	1
288	1.45	3.09	7.34	10	1.6	32	460.8	1
795	72.42	85.51	56.59	10	0.57	88.3	460.8	1

4.2. Experimental Result

We express the two experimental results. One is prediction using Linear Regression and M5P from machine learning algorithms to predict the system crash due to software aging caused by resource exhaustion. Another is availability analysis with predictable data using Stochastic Petri net model to solve software aging. In this section, we predicted the Time to Crash (TTC) due to resource exhaustion causing the software aging based on limited set of system metrics (CPU, Memory, etc).

4.2.1. Prediction Results

Table3. Analysis Results for TTC prediction

Prediction Algorithm	Correlation Coefficient	Mean Absolute Error (MAE)
Linear Regression	0.9738	22.5598
M5P	0.9639	25.8052

According to experimental results in Table 3, both M5P and Linear Regression can be achieved good correlation coefficient.

4.2.2 Availability's Result

The availability of the proposed model is defined as:
Availability=1–Unavailability (21)

$$Availability = 1 - (\sum_{i=1}^{n-1} P_{D_i} + \sum_{i=1}^{n} P_{F_i}) \quad (22)$$

The expected total downtime with preventive maintenance in an interval of T time units is

$$Downtime(T) = Tx(\sum_{i=1}^{n-1} P_{D_i} + \sum_{i=1}^{n} P_{F_i}) \quad (23)$$

We illustrate the applicability of the SPN model and solution methodology through numerical examples. The exact model transition firing rates for the model are not known, a good estimate value for a range of model transition firing rates is assumed. For this purpose, we perform numerical analysis using the following failure profile mentioned in Table 4.

Table 4. Transitions Firing Rates

Transition	Description	Firing Rates(h^{-1})
T_{fp}	VM aging rate	Execution time for web server from experimental set up
$1/T_{sw}$	Mean time to switch over	5 sec
T_d	Failure rate	1time/ 90 days
$1/T_r$	Mean time to VM rejuvenation	10 mins
$1/T_{repair}$	Mean time to VM repair	30 mins
T_{rej}	VM rejuvenation rate	Time To Crash from experimental set up
$1/T_{up}$	VM healthy rate	30 mins
T_F	VM failure rate	1 time/3 months
T_{down}	Down rate	1 time/1 day

We expressed time to crash and execution time using 20000 connection rates from client through Httperf tool to 300 KB Apache server as shown in Figure 7. According to the figure, the higher execution time, the higher time to crash can be achieved. Therefore, time to crash is depended on the execution time.

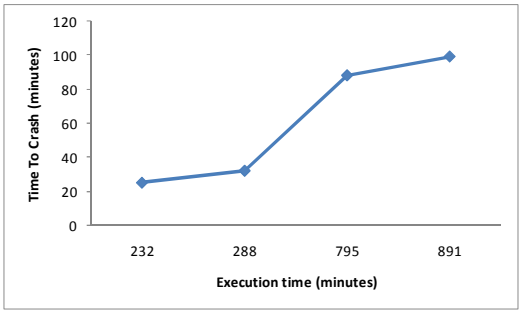

Figure7. Time To Crash vs different execution time

The change in the availability of system with different rejuvenation rates is graphed in Figure 8.

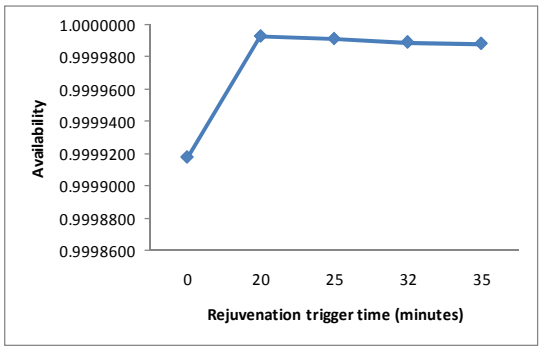

Figure8. Availability vs different rejuvenation triggers time

The change in the downtime of the system with the different rejuvenation rates is plotted in Figure 9. From this result, it is apparent that our predictable model is a cost-effective way to build high availability and virtualization technology can improve the software rejuvenation methodology.

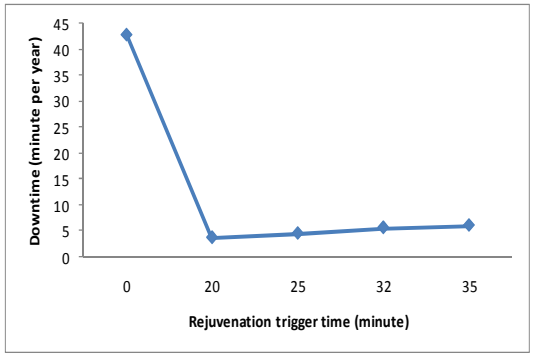

Figure9. Downtime vs different rejuvenation trigger time

5. Conclusion

In this paper, we presented availability analysis and web server software aging solution.
We present a Stochastic Petri Net Model by applying Time-based Software Rejuvenation Policy that can be applied in VDC. Numerical analysis of the system availability is carried out the SHARPE tool simulation. The numerical results are validated with the evaluation through SHARPE tool simulation. It is found that our results and SHARPE result are the same.

References

[1] A. Polumetla, "Machine Learning Methods for the Detection of RWIS Sensor Malfunctions", Master Thesis, July 2006, pp.30-33.
[2] D. Seong Kim, F. Machida and K. S. Trivedi, "Availability Modeling and Analysis of a Virtualized System", 15th IEEE Pacific Rim International Symposium on Dependable Computing, Pages 365-371, 2009.
[3] G.Reig, J.Alonso and J.Guitart, "Prediction of Job Resource Requirements for Deadline Schedulers to Manage

High-Level SLAs on the Cloud", IEEE Computer Society, NCA, 2010.
[4] J.Alonso, L.Belanche and D.R.Avresky, "Predicting Software Anomalies using Machine Learning Techniques", 2011.
[5] J.Alonso, J.Torres, J.L1.Berral and R.Gavalda`, "Adaptive on-line software aging prediction based on Machine Learning", IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp.507-516, 2010.
[6] J.Alonso, J.Torres and R.Gavalda`,"Predicting web server crashes: A case study in comparing prediction algorithms", Fifth International Conference on Autonomic and Autonomous Systems, pp.264-269, 2009.
[7] L.Li, Kalyanaraman Vaidyanathan and Kishor S. Trivedi, “An Approach for Estimation of Software Aging in a Web Server", ISESE, Proc Int'l Symp. Empirical Software Eng, pp. 91-100, 2002.
[8] M.Grottke, L. Li, K. Vaidyanathan and K. S. Trivedi, "Analysis of Software Aging in a Web Server", IEEE Transactions on Reliability, 55:441-420, 2006.
[9] Y. Haung, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: Analysis, module and applications", In Proc. Twenty-fifth International Symposium on Fault-Tolerant Computing, 1995, pp. 381-390.
[10] Y.Liang, H.Yang, J.Fu, C.Tan, A.Liu and S.Zhu, "The Effect of Real-valued Negative Selection Algorithm on Web Server Aging Detection", Journal of Software, Vol.7, No.4, April 2012.
[11] http://www.cs.waikato.ac.nz/ml/weka/
[12] http://http-performance-testing-with-httperf.html