The Annals of "Dunarea de Jos" University of Galati
Fascicle I – 2007. Economics and Applied Informatics. Years XIII - ISSN 1584-0409

Availability Analysis and Improvement of Software Rejuvenation Using Virtualization

Thandar THEIN
Sung-Do CHI
Jong Sou PARK
{thandar, sdchi, jspark}@kau.ac.kr
Computer Engineering Department
Korea Aerospace University, Seoul, Korea

Abstract. Availability of business-critical application servers is an issue of paramount importance that has received special attention from the industry and academia. To improve the availability of application servers, we have conducted a study of virtualization technology and software rejuvenation that follows a proactive fault-tolerant approach to counter act the software aging problem. We present Markov models for analyzing availability in such continuously running applications and express availability, downtime and downtime costs during rejuvenation in terms of the parameters in the models. Our results show that our approach is a practical way to ensure uninterrupted availability and optimize performance for even strongly aging applications.

Keywords: Software aging, Software rejuvenation, Virtualization, Availability, Markov process

1. Introduction

As business becomes increasingly dependent on information and computing technology, continuous availability is a universal concern. It has now been well-established that failures of computer systems are more often due to software faults than due to hardware faults [6, 11]. Recently, the wide-spread phenomenon of "software aging", one in which the state of a software system gradually degrades with time and eventually leads to performance degradation, transient failures or even crushes of applications, has been reported [7]. The primary causes of this degradation are the exhaustion of operating system resources, data corruption and numerical error accumulation. Some common examples of software aging are memory bloating and leaking, unreleased file-lock, data corruption, storage space fragmentation and accumulation of round-off errors [13]. Aging has not only been observed in software used on a mass scale but also in specialized software used in high-availability and safety critical applications. The most natural procedure to counteract such software aging is to apply the well-known technique of software rejuvenation. Virtualization techniques are getting more and more popular. They allow to run multiple virtual servers on a single physical machine. Introduced in the 1960s, virtualization has lately enjoyed a great surge of interest. The improved efficiency flexibility, and cost savings that virtualization of storage, networking, and computing resources enables in data centers have been the key drivers of this interest. The idea proposed framework in this paper is to exploit the advantage of virtualization technology to improve the software rejuvenation for addressing the software aging problem. The structure of this paper is as follows. In section 1 we address the problem issue. In section 2, we introduce the concepts of software rejuvenation and virtualization technology and address related research. Section 3 presents the proposed framework. Section 4 presents the models in which the operational states of the system in virtualized and non-virtualized scenario are described and in the following section, the model is analyzed and experimental results are

given to validate the model solution. Finally, we conclude that virtualization can be helpful for software rejuvenation and fail-over in the occurrence of application failures and software aging.

2. Software Rejuvenation and Virtualization

In this section, we introduce the concepts of software rejuvenation and virtualization technology.

2.1 Software Rejuvenation

Software rejuvenation is a proactive fault management technique aimed at cleaning up the system's internal state to prevent occurrence of more severe crash failure in the future [10]. It involves occasionally stopping the software application, cleaning its internal state and/or its environment, and then restarting it [8]. By removing the accrued error conditions and freeing up or decrementing operating system (OS) resources, this technique proactively prevents unexpected future system outages. Unlike downtime caused by sudden failure occurrences, the downtime related to software rejuvenation can be scheduled at the discretion of the user/administrator. According to the control mechanism, software rejuvenation can be categorized into two approaches, open-loop control and closed-loop control [10]. Open-loop approach is characterized by no feedback information from the system after the integration of software rejuvenation functionality. Time-based rejuvenation and its variants fall into this category. On the other hand, in closed-loop approach, rejuvenation trigger is dependent on some form of feedback from the system. The rejuvenation decision is made based on current system state and/or previous system behavior, which include workload, resource usage [5] and failure logs. Measurement-based rejuvenation belongs to this category [14]. Meanwhile, rejuvenation has been implemented in various types of systems, telecommunication system [1], transaction processing systems [3], cluster servers [4], and spacecraft systems [12]. Many research papers on this topic can be found at [10]. It is widely understood that this technique of rejuvenation provides better results, resulting in higher availability and lower costs.

2.2 Virtualization

Virtualization is a proven software technology that is rapidly transforming the IT landscape and fundamentally changing the way that people compute. 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.

Virtualization allows abstracting away the real hardware configuration of a system. One method of virtualizing the hardware resources of a computer involves using a layer of software, called the Virtual Machine Monitor (VMM), to provide the illusion of real hardware for multiple virtual machines (VMs). Inside each VM, the operating system (often called the guest OS) and applications run on the VM's own virtual resources such as virtual CPU, virtual network card, virtual RAM, and virtual disks [9]. VMs possess four key characteristics that benefit the user: compatibility, isolation, encapsulation and hardware independence [15]. A VMM can be hosted directly on the computer hardware (e.g., Xen [2] or within a host operating system (e.g., VMware).VMs offer a degree of flexibility that is not possible to obtain on physical machines. That is mainly because VM state, much like files, can be read, copied, modified, saved, migrated, and restored. As server consolidation using VMs is widely carried out. Recently, multiple server machines are consolidated into one machine using VMs. In such a machine, many VMs are running on top of virtualization middleware.

3. Proposed Rejuvenation Framework using Virtualization

We make the following contributions in our proposed framework. First we present new way of using virtualization to improve software rejuvenation for addressing the software aging issue.

With our proposed framework, it can be able to optimize the rejuvenation operation without requiring any additional hardware. This new approach is meant to be the less disruptive as possible for the running service and to get a zero downtime for most of the cases. Our framework of software rejuvenation is totally supported by software and can be easily deployed in existing IT infrastructures. Virtualization technology and software rejuvenation can be used to prolong the availability of the services.

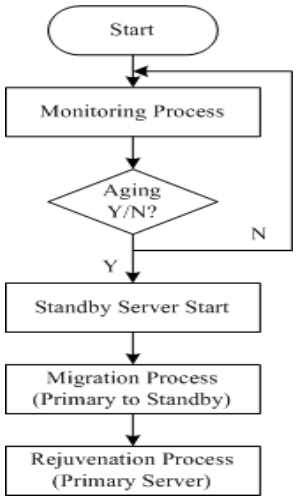

Figure 1. Proposed system flow

Our proposed rejuvenation framework using virtualization is shown in figure 1 and 2. In order to achieve optimize planned restarts in a single-server configuration; we adopt the virtualization technology in our proposed framework. On top of virtualization middleware layer, there are (n+1) virtual machines per application server shown in figure 2. A software load-balancer (VLM-LB) will be run on VM_0. The main application server will be run on one VM. The rest of VMs will work as hot-standby servers, where we instantiate a replica of the application server. In hot-standby configuration, component state is replicated to the standby on any change, i.e. the standby component is always up-to-date. In case of a failure, the standby component replaces the failed component and continues to operate based on the current state. The hot-standby configuration offers continuous availability without any interruption of service.

Aging-Detector and Anomaly-Detector will be installed for the detection of software aging or some potential anomaly such as protocol errors, log errors, application-level anomalies. When software aging or some potential anomaly happens VM-LB will trigger a rejuvenation operation. In VM_0, other necessary software modules such as SRA-Cood and Data-Collectors will be installed. To detect server outages Watchdog module will be applied in VM_0. In order not to lose any in-flight request or session data at the time of rejuvenation, first, all the new requests and sessions are migrated from the active server to standby server. When all the ongoing requests are finished in primary server, then the primary VM will be rejuvenated by using SRA (Software Rejuvenation Agent). SRA-Agent that is responsible for the rejuvenation operation will be installed in other VMs.

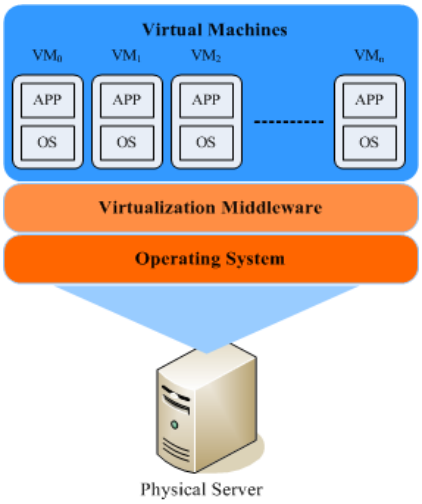

Figure 2. Rejuvenation framework using virtualization

4. System Model Analysis

In this section, we are interested in determining how virtualization technology can enhanced the software rejuvenation process and can reduce an application's downtime and the cost due to downtime. We also analyze the availability of system in non-virtualized and virtualized scenario. The state transition diagram of non-virtualized system is shown in figure 3. The virtualized state transition diagram of our framework is shown in figure 4. The assumptions used in the modeling are as follows:

- Failure rate (λ) and repair rate (μ) of the VM are identical at all states.
- Unstable rate (λ_u), the speed of escaping the healthy condition of VM is identical at all states.
- Rejuvenation rate (λ_r), the frequency of rejuvenation is identical at all states.
- Mean time spent during the rejuvenation process is constant ($1/\mu_r$).
- Mean switchover time, the time needed to transfer from the primary VM to the standby VM is constant ($1/\lambda_s$)
- The probability of going from normal state to failure state is negligible compared to other probabilities.
- During the rejuvenation process, the system can provide the continuous service except non-virtualized system.

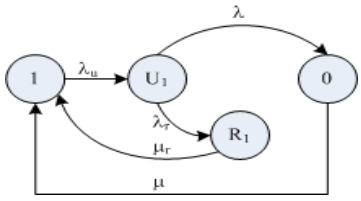

Figure 3. Non-virtualized state transition diagram

First the system is in normal state (1). In failure state (0), no operational server exists and service is not available during repair time ($1/\mu$). So it is not necessary to represent the switchover state. Under these assumptions and writing down the balance equations of figure 3, we get:

$$\mu_r P_R + \mu P_0 = \lambda_\mu P_1 \quad (1)$$

$$\lambda_u P_1 + (\lambda + \lambda_r) P_u \quad (2)$$

$$\lambda_r P_u = \mu_r P_R \quad (3)$$

$$\lambda P_u = \mu P_0 \quad (4)$$

After solving the above equations, we get the expressions for the state probabilities of figure 3.

$$P_0 = \frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}P_1 \quad (5)$$

$$P_u = \frac{\lambda_u}{\lambda+\lambda_r}P_1 \quad (6)$$

$$P_R = \frac{\lambda_r}{\mu_r}\frac{\lambda_u}{\lambda+\lambda_r}P_1 \quad (7)$$

The conservation equation of figure 2 is obtained by summing the probabilities of all states in the system and the sum of the equation is 1.

$$P_1 + P_u + P_R + P_S + P_0 = 1 \quad (8)$$

$$P_1 = \left[1+\frac{\lambda_u}{\lambda+\lambda_r}\left(1+\frac{\lambda_r}{\mu_r}+\frac{\lambda}{\mu}\right)\right]^{-1} \quad (9)$$

The non-virtualized system is not available in rejuvenation state (R_1) and the failure state (0). The system availability in the steady-state is defined as follows:

$$\text{Availability A} = 1 - (P_0 + P_{R_1}) \quad (10)$$

The expected total downtime and downtime cost of the application with rejuvenation in an interval of L time units are:

$$\text{Downtime DT (L)} = (P_0 + P_{R_1}) \text{ x L} \quad (11)$$

$$\text{Cost C (L)} = \left[(P_0 \times C_f) + (P_{R_1} \times C_r)\right] \times L \quad (12)$$

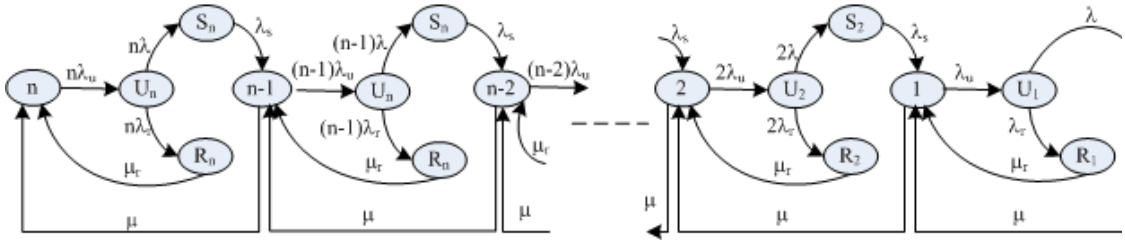

Figure 4. State transition diagram of virtualized system

Now consider the state transition diagram of our virtualized framework. According to the proposed framework, first active VM is in normal state (n) and standby VMs are in normal state (n-i). After a long mission time, normal states may change to unstable state (U_i) with rate i * λ_u. In the unstable states, VM performance is degraded and software-aging effects render the system unreliable.

If a VM is in an unstable state, the state can change to switchover state with rate i * λ_s and rejuvenation state with rate i* λ_r. If primary VM has encounter the software aging, the standby component takes the role of primary VM and continues to operate based on the current state. When all the requests are finished in the primary server, then primary VM will be rejuvenated. So service is not stopped even during the switchover process and rejuvenation process. In the failure state (0), all VMs stop running and no available VM remains.

After the rejuvenation, one of the healthy rejuvenated VM takes over the role of primary VM. Our state transition diagram in figure 4 can be described as a Markov process class. We can perform steady state analysis of the diagram. The steady-state balance equations of figure 4 are as follows:

$$(\mu + i\lambda_u)P_i = \lambda_s P_{s_{i+1}} + \mu_r P_{R_i} + \mu P_{i-1}$$

$$(i=2, 3, \ldots\ldots, n-1) \quad (13)$$

$$n\lambda_n P_n = \mu P_{n-1} + (n-1)\mu_r P_{R_n} \quad (14)$$

$$(\lambda_u + \mu)P_1 = \mu P_0 + \lambda_s P_{s_2} + \mu_r P_{R_1} \quad (15)$$

$$\mu P_0 = \lambda P_{u_1} \quad (16)$$

$$(\lambda + \lambda_r)P_{u_i} = \lambda_u P_i \quad \text{(i=1, 2, 3, ……., n)} \quad (17)$$

$$\mu_r P_{R_i} = i\lambda_r P_{u_i} \quad \text{(i=1, 2, 3, ……., n)} \quad (18)$$

$$\lambda_s P_{s_i} = i\lambda P_{u_i} \quad \text{(i=2, 3, ……., n)} \quad (19)$$

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

$$\sum_{i=0}^{n} P_i + \sum_{i=1}^{n} P_{u_i} + \sum_{i=1}^{n} P_{R_i} + \sum_{i=2}^{n} P_{S_i} = 1 \quad (20)$$

$$P_n = \left[n! \left\{ \sum_{i=1}^{n} \frac{1}{i!} \left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n-i} + \frac{\lambda_u}{\lambda+\lambda_r} \sum_{i=1}^{n} \frac{1}{i!} \left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n-i} + \frac{\lambda_r}{\mu_r}\frac{\lambda_u}{\lambda+\lambda_r} \sum_{i=1}^{n} \frac{1}{(i-1)!} \left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n-i} + \frac{\lambda}{\lambda_s}\frac{\lambda_u}{\lambda+\lambda_r} \sum_{i=2}^{n} \frac{1}{(i-1)!} \left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n-i} + \left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n} \right\} \right]^{-1} \quad (21)$$

$$P_i = \frac{n!}{i!}\left(\frac{\lambda}{\mu}\frac{\lambda_u}{\lambda+\lambda_r}\right)^{n-i} P_n \quad \text{(i = 0, 1, …, n)} \quad (22)$$

$$P_{u_i} = \left(\frac{\lambda_u}{\lambda+\lambda_r}\right) P_i \quad \text{(i = 1, 2, …., n)} \quad (23)$$

$$P_{R_i} = \frac{i\lambda_r}{\mu_r}\frac{\lambda_u}{\lambda+\lambda_r} P_i \quad \text{(i = 1, 2, …., n)} \quad (24)$$

$$P_{S_i} = \frac{i\lambda}{\lambda_s}\frac{\lambda_u}{\lambda+\lambda_r} P_i \quad \text{(i = 2, 3, …., n)} \quad (25)$$

The meaning of the probabilities is as follows:

P_i	the probability of the VM is in normal state
P_{ui}	the probability of the VM is in unstable state
P_{Ri}	the probability of the VM is in rejuvenation state
P_{Si}	the probability of the VM is in switchover state
P_0	the probability of the VM is in failure state

The virtualized system is not available in the rejuvenation processes in the normal state (1) and the failure state (0). The system availability in the steady-state is defined as follows:

$$\text{Availability } A_V = 1 - (P_0 + P_{R1}) \quad (26)$$

Predictable shutdown cost is far less than that of unexpected shutdown ($C_f >> C_r$). Where C_f is the unit cost of unexpected shutdown of a server, and Cr is the unit cost of rejuvenation process. The expected total downtime and downtime cost of our framework in an interval of L time units are as follows:

$$\text{DownTime } DT_V(L) = (P_0 + P_{R1}) \times L \quad (27)$$

$$\text{Cost } C_V(L) = (P_0 \times C_f + P_{R_1} \times C_r) \times L \quad (28)$$

4.1. Experimental Results

To acquire system dependability measures like availability and downtime cost, we perform experiments using the following failure profile [8] as shown in table 1.

Table 1. System operating parameters

Parameters	Values
n	1, 2, 3, 4
L	1 year
λ	1 time/year
μ	2 times/day
λ_r	1 times/month (λ_r=0, 1, 2, 3)
λ_u	2 times/month
$1/\mu_r$	10 min
$1/\lambda_S$	3 min
C_r	1 unit
C_f	100 units

The change in the availability of the non-virtualized and virtualized (3 VMs) system with the different rejuvenation rates is plotted in figure 5. In virtualized configuration, we use 3 VMs per application server; one VM to run a software load balancer, one VM where we run the main application server and a third VM that works as a hot standby. According to the result, system availability can increase by using software rejuvenation and virtualization technology.

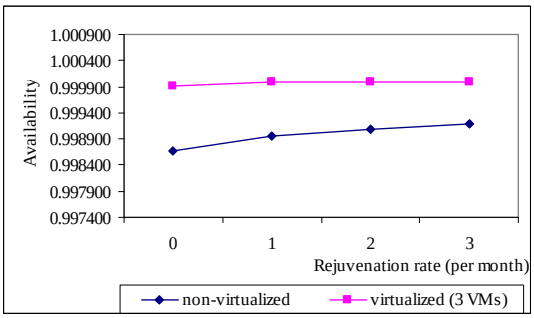

Figure 5. The plot of availability versus rejuvenation rate (with and without virtualization)

The change in the availability of virtualized system with different number of VMs and different rejuvenation rates is plotted in figure 6. As unstable states are removed frequently with high rejuvenation rates, the availability of the virtualized system is increased. However, as the degree of standby VMs is larger than or equal to 4, the improvement of availability is not significant. From this result, it is apparent that 3 VMs configuration is a cost-effective way and easy to manage to build high availability systems. The change in the downtime of the non-virtualized and virtualized (3 VMs) system with the different rejuvenation rates is plotted in figure 7. The change in the downtime of virtualized system with different number of VMs and different rejuvenation rates is plotted in figure 8.

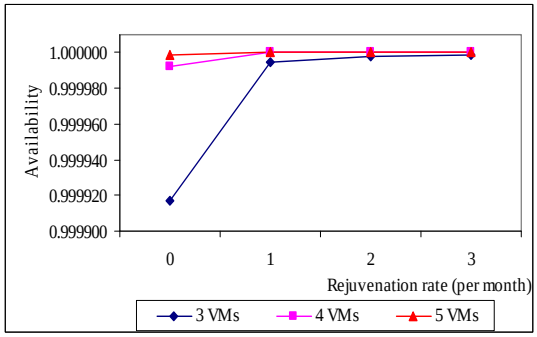

Figure 6. The plot of availability versus rejuvenation rate (virtualized)

The downtime cost of scheduled shutdown is much lower than that of an unscheduled shutdown. The downtime cost due to downtime with the different rejuvenation rates for non-virtualized and virtualized system is plotted in figure 9 and figure 10.

From this result, it is apparent that our rejuvenation framework is a cost-effective way to build high availability system and virtualization technology can improve the software rejuvenation process.

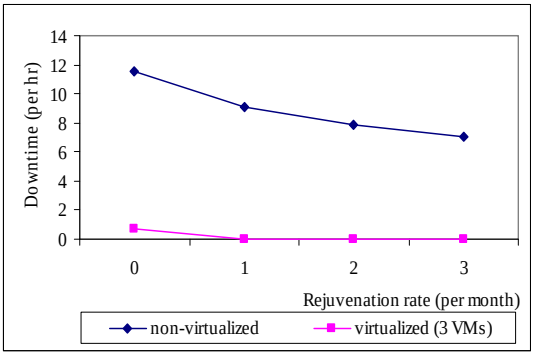

Figure 7. The plot of downtime versus rejuvenation rate (non-virtualized and virtualized)

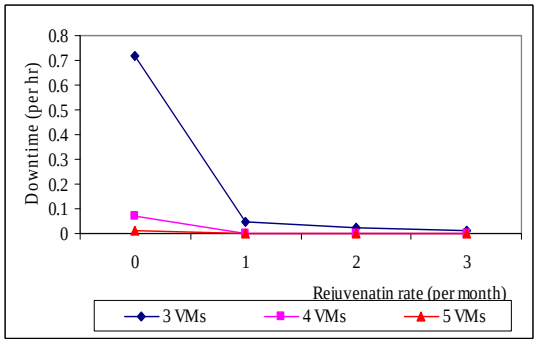

Figure 8. The plot of downtime versus rejuvenation rate (virtualized)

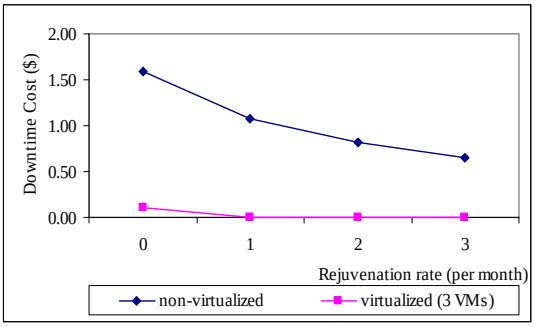

Figure 9. The plot of downtime cost versus rejuvenation rate (non-virtualized and virtualized)

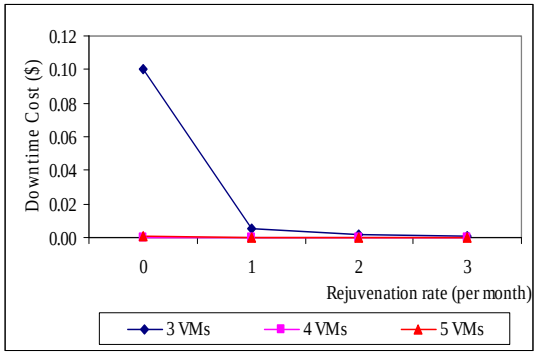

Figure 10. The plot of downtime cost versus rejuvenation rate (virtualized)

According to the results of figure 9 and 10, we can achieve a zero downtime even in case of VM restart. Software rejuvenation using virtualization provides a way to remove faults and vulnerabilities at run-time without affecting system availability. Virtualization can be used to prolong the availability of the service as much as possible while at the same time ensuring that the service is fail-safe. By using our proposed framework, the system does not require any additional hardware and can provide the less disruptive as possible for the running service and get a zero downtime for most of the case.

5. Conclusion

In this paper, we have presented for effective software rejuvenation using virtualization technology that has proved to be highly effective to counteract the software aging problem. We present a Markov model for analyzing software rejuvenation in such continuously running applications and express availability, downtime and costs in terms of the parameters in the models. Our results show that virtualization and software rejuvenation can be used to prolong the availability of the services. Our framework can be applied to single-server or cluster configurations without any additional cost. We conclude that virtualization can be helpful for software rejuvenation and fail-over in the occurrence of application failures and software aging.

References

[1] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability," Empirical Software Engineering, vol. 2, no. 1, pp. 59–77, 1997.

[2] P. T. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," In Proc. 19th ACM Symposium on Operating Systems Principles (SOSP-2003), pages 164-177, October 2003.

[3] K. J. Cassidy, K. C. Gross, and A. Malekpour, "Advanced Pattern Recognition for Detection of Complex Software Aging in Online Transaction Processing Servers," in Proc. International Conference on Dependable Systems and Networks, 2002, pp. 478–482.

[4] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive Management of Software Aging," IBM Journal of Research and Development, vol. 45, no. 2, pp. 311–332, 2001.

[5] S. Garg, A. van Moorsel, K. Vaidynathan and K. S. Trivedi, "A Methodology for Detection and Estimation of Software Aging," Proc. Ninth Int. Symp. On Software Reliability Engineering, Paderborn, Germany, November 1998

[6] J. Gray and D. P. Siewiorek, "High-Availability Computer Systems," IEEE Computer, vol. 24, no. 9, pp. 39–48, 1991.

[7] *M. Grottke, Lei Li, K. Vaidyanathan and K. S. Trivedi, " Analysis of Software Aging in a Web Server'" IEEE Transactions on Reliability, Vol. 5, No. 3, September 2006.*

[8] *Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software Rejuvenation: Analysis, Module and Application," In Proc. Of FTCS-25, Pasadena, CA, Jun 1995.*

[9] *H. V. Ramasamy and M. Schunter, "Architecting Dependable Systems Using Virtualization," Online Available: www.opentc.net/publications/OTC_Architecting_Dependable_Systems.pdf*

[10] *"Software Rejuvenation," Department of Electrical and Computer Engineering, Duke University Online Available: http://www.softwarerejuvenation. com/*

[11] *M. Sullivan and R. Chillarege, "Software Defects and their Impact on System Availability—a Study of Field Failures in Operating Systems," in Proc. Twenty-First International Symposium on Fault-Tolerant Computing, 1991, pp. 2–9.*

[12] *A. T. Tai, L. Alkalaj, and S. N. Chau, "On-Board Preventive Maintenance: A Design-Oriented Analytic Study for Long-life Applications," Performance Evaluation, vol. 35, no. 3–4, pp. 215–232, 1999.*

[13] *K.s. Trivedi, K. Vaidyanathan and K.Goseva-Popstojanova, "Modeling and Analysis of Software Aging and Rejuvenation," Proc. Of 33rd Annual Simulation symp., Grece, Apr 2000.*

[14] *K. Vaidyanathan and K. S. Trivedi, "A Measurement-Based Model for Estimation of Software Aging in Operational Software Systems." Proc. Tenth IEEE Intl. Sym. On Softwae Reliability Engineering, pp.88-93, Boca Raton, FL, November 1999.*

[15] *Virtual Machines: Online Available: http://www.vmware.com*

Acknowledgements

This research was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2007-210-D00006).