Proactive Fault Management with Virtualization for Software Aging

Thandar Thein, Sung-Do Chi, and Jong Sou Park

Korea Aerospace University,
200-1, Hwajeon-Dong Dukyang-Gu, Goyang City Gyeonggi-Do 412-791, Korea
{thandar,sdchi,jspark}@kau.ac.kr

Summary. Unplanned computer system outages are more likely to be the result of software failures than of hardware failures. Moreover, software applications executing continuously for a long period of time show a degraded performance and/or an increased occurrence rate of hang/crash failures. This phenomenon has been referred to as software aging. In this paper, we have conducted a study of virtualization technology and software rejuvenation that follows a proactive fault management approach to counter act the software aging. We present a model to evaluate the effectiveness of proactive fault management approach with the use of virtualization technology in operational software systems, and express downtime and costs due to downtime during rejuvenation in terms of the parameters in that model. We perform mathematical derivation and use SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool to evaluate the feasibility of our model. Our results show that proposed approach can provide uninterrupted availability of the services.

1 Introduction

Software failures are now known to be a dominant source of system outages. The consequences of software failure can lead to huge economic losses or risk to human life. Several studies and much anecdotal evidence point to software aging as a common phenomenon. Software aging is gaining in significance because of the growing economic importance of software and the fact that increasingly, software is a major part of the capital of many high-tech firms. Software aging usually observed as a progressive degradation through time, which can lead to system crashes or undesirable hang ups. Not only software used on a mass scale, but also specialized software used in high-availability and safety-critical applications suffer from aging. This phenomenon is particularly troublesome in long-running applications. Software aging has also been observed in many real software systems [3, 5, 16]. For this reason, in this paper we propose a new solution to counter act the aging problem.

Software rejuvenation technique has been widely used to avoid the occurrence of unplanned failures, mainly due to the phenomena of software aging or caused by transient failures [8]. Most current fault-tolerant techniques are reactive in nature. Proactive fault management, on the other hand, takes suitable corrective action to prevent a failure before the system experiences a fault. Software

R. Lee (Ed.): Soft. Eng., Arti. Intel., Net. & Para./Distri. Comp., SCI 149, pp. 189–200, 2008.
springerlink.com

rejuvenation is a specific form of proactive fault management which can be performed at suitable times. Proactive rejuvenation techniques have been studied in [2, 3, 6, 13, 17] and it is widely understood that this technique of rejuvenation provides better results, resulting in higher availability and lower costs.

Traditional fault-tolerant techniques work with recovery overhead because it carries out the restoration operation after system failure. The software rejuvenation technique works with the planed restart mechanism, which can lower the system recovery overhead by a grate extent. System availability can be further enhanced by taking a proactive approach to detect and predict an impending outage of a specific server in order to initiate planned failover in a more orderly fashion. This approach not only improves the end user's perception of service provided by the system, but also gives the system administrator additional time to work around any system capacity issues that may arise.

Although the fault in the application program still remains, performing the rejuvenation occasionally or periodically prevent failures due to that fault. Any rejuvenation typically involves an overhead, but it prevents more severe crash failures from occurring. Hence, an important issue in analyzing rejuvenation policies is to determine their usefulness in terms of availability, downtime, and cost and to provide an optimal criterion to decide when and how often to recover the system from the degraded state. The side effect of software rejuvenation is the temporary outage of service during rejuvenation. To solve this problem, we adopt the virtualization technology to provide uninterrupted service.

Virtual machine concept was first developed by IBM in the 1960's and popular in the 1970s [4]. At that time, computer systems were large and expensive, so IBM invented the concept of virtual machines as a way of time-sharing for mainframes, partitioning machine resources among different users. Virtualization is a proven software technology that is rapidly transforming the IT landscape and fundamentally changing the way that people compute. Virtualization technologies find important applications over a wide range of areas such as server consolidation, secure computing platforms, supporting multiple operating systems, kernel debugging and development, system migration, etc, resulting in widespread usage. Virtualization is a hot topic in the technology world. The technology enables a single computer to run multiple operating systems simultaneously. Many fields, such as autonomic computing, service consolidation, security and education publish results that praise the benefits of virtualization. Virtualization has proved as a successful tool for management of complex IT-environments and it is emerging as a technique to increase system reliability and availability [9, 12].

An approach for software rejuvenation based on automated self-healing techniques presented in [12], which exploit the usage of virtualization to optimize the self recovery actions. Software aging in virtual machine monitors (VMMs) has been studied in [10]. The main contribution of that paper is the development of a methodology for proactive management of software systems which are prone to aging, and specifically to resource exhaustion.

The idea proposed in this paper is to hold the multiple virtual machines which are running aging applications, and trigger the rejuvenation action of

each virtual machine when something anomalous is detected. Software rejuvenation using virtualization provides a way to remove faults and vulnerabilities at run-time without affecting system availability. By coupling proactive software rejuvenation and virtualization, significant increases in system availability and performance can be achieved.

The structure of the paper is as follows: Section 1 discusses problem issue and describes the methods to counteract the software aging problem. Section 2 addresses the related background. Section 3 presents our proposed approach. In section 4, we present a state transition model to describe the behaviors of virtualized cluster system and in the following section, the models are analyzed and experimental results are given to validate the model solution. Finally, we conclude with a summary of our results in section 5.

2 Background

2.1 Software Aging

The term software aging describes the phenomena of progressive degradation of the running software that may lead to system crashes or undesired hang ups [8]. Gradual performance degradation may also accompany software aging. Failures of both crash/hang type as well as those resulting in data inconsistency have been reported. Memory bloating and leaking, unreleased file-locks, data corruption, storage space fragmentation and accumulation of round-off errors are some typical causes of slow degradation. This undesired behavior is especially visible in long-running software such as web and application servers and enterprise always-on applications.

2.2 Software Rejuvenation

Software rejuvenation is a proactive fault management technique aimed at cleaning up the system internal state to prevent the occurrence of more severe crash failures in the future. It involves occasionally terminating an application or a system, cleaning its internal state and restarting it [14]. Some examples of cleaning the internal state of software are garbage collection, flushing operating system kernel tables and reinitializing internal data structures. Extreme example of rejuvenation might be a simple hardware reboot. Software rejuvenation is a cost effective technique for dealing with software faults that include protection not only against hard failures, but also against performance degradation as well.

2.3 Virtualization

Virtualization is a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partitioning or aggregation, partial or complete machine simulation, emulation, time-sharing, and others. A virtualization layer, thus, provides

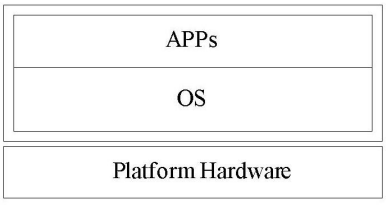

Fig. 1. (a) Without Virtualization

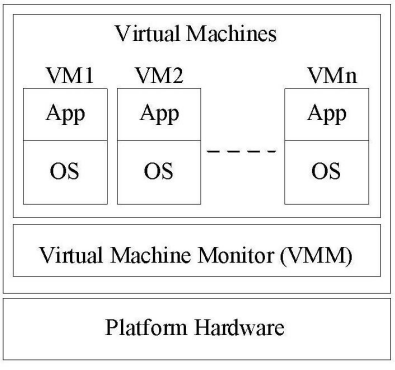

Fig. 1. (b) With Virtualization

infrastructural support using the lower-level resources to create multiple Virtual Machines (VMs) that are independent of and isolated from each other. Sometimes, such a virtualization layer is also called Virtual Machine Monitor (VMM).

The VMM is the essential part of the virtual machine implementation, because it performs the translation between the bare hardware and virtualized underlying platform: providing virtual processors, memory, and virtualized I/O devices [1]. Since all the virtual machines share the same bare hardware, the VMM should also provide appropriate protection so that each VM is an isolated replica. Traditionally, the VMM sits between the bare system hardware and operating systems. Non-virtualized and virtualized system architectures are shown in figure 1(a) and 1(b).

3 Proposed Approach

In this section we describe our proposal to offer the high availability mechanism for aging applications. This approach has been designed to use over any server or service. Our approach makes use of several concepts: a virtualization layer that is installed in every application server; the use of primary-backup scheme for application server replication; and the adoption of software rejuvenation for software aging problem. By merging all of these concepts and techniques, we can get the cost-effective and high availability solution for aging applications without requiring additional servers and load-balancing machines.

3.1 Virtualized Cluster Architecture

Figure 2 represents the conceptual architecture of our approach. It is just necessary to install a virtualization layer and install some software modules. We have adopted virtualized clustering architecture in our proposed approach. Our approach requires the creation of three virtual machines on top of the virtualization layer. In VM1, we will install software load-balancer module (VM-LB)

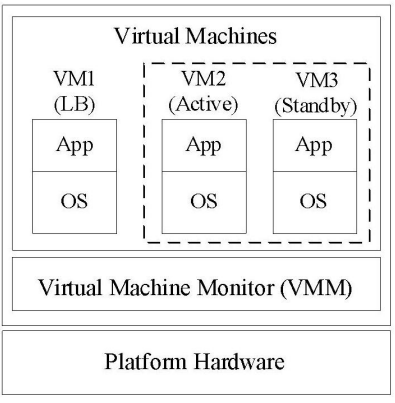

Fig. 2. VMM and VMs relationship

to collect system data from the application and some other software modules that will be responsible for the detection of software aging or other potential anomalies and to do as a coordinator of the self-recover actions. VM2 will be used to run main application server and VM3 where we create a standby replica of the application server.

We have adopted virtualized clustering architecture in our proposed approach. This setup builds an HA cluster between 3 virtual machines on a single physical machine. The cluster manager itself is running directly in the virtual machines.

3.2 Rejuvenation Process

The rejuvenation process of our approach is shown in figure 3. When software aging or some potential anomaly is detected in active VM, we should apply

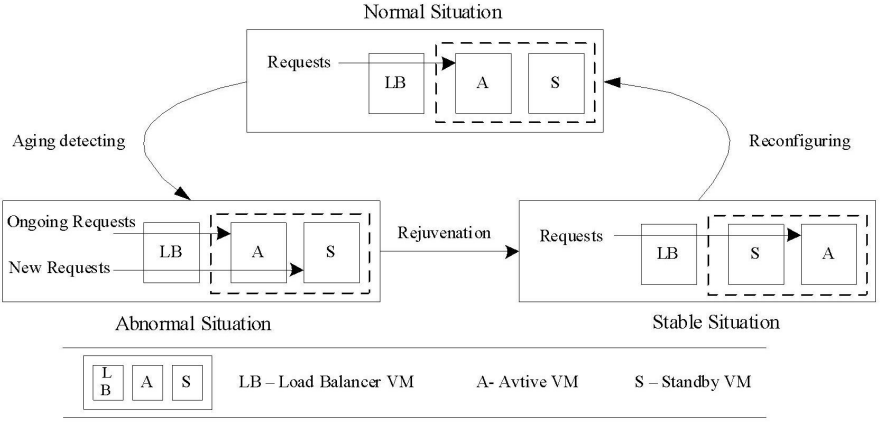

Fig. 3. Rejuvenation process

rejuvenation action. First we start the standby VM and all the new requests and sessions are migrated from the active VM to standby VM. When all the ongoing requests are finished in primary VM, then the primary VM will be rejuvenated and the VM becomes as good as new.

4 System Model and Analysis

We construct the state transition model to describe the behavior of our proposed system, and it is shown in figure 4. Suppose that the system is started with the normal operation state. The system can smoothly degrade in time. We denote failure rate (λ) and repair rate (μ) for all respective states such as error detection rate (λ_d), rejuvenation triggering rate (λ_r), switchover rate (λ_s), failure rate (λ), rejuvenation service rate (μ_r) and repair rate (μ).

The state transition diagram consists of seven states. The explanation of the states for the state transition diagram of figure 4 is as follows:

(M,1,1): VM-LB is in Monitoring state, Active VM and Standby VM are in nor-mal state
(M,D,1): VM-LB is in Monitoring state, Active VM is in detection state and standby VM is in normal state
(M,R,1): VM-LB is in Monitoring state, Active VM is in rejuvenation state and another VM is in normal state
(M,0,1): VM-LB is in Monitoring state, one VM is in failure state and one active VM is in normal state
(M,0,D): VM-LB is in Monitoring state, one VM is in failure state and active VM is in detection state
(M,0,R): VM-LB is in Monitoring state, one VM is in failure state and active VM is in rejuvenation state
(M,0,0): VM-LB is in Monitoring state, Active VM and Standby VM are in failure state.

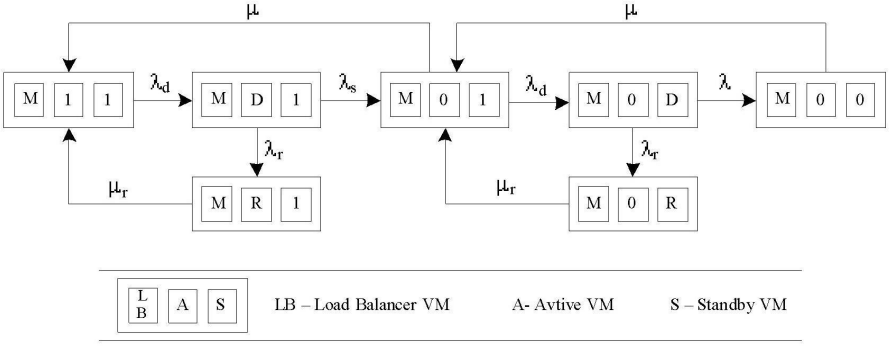

Fig. 4. A state transition diagram of system's behavior

And let the steady-state probabilities of the state of the system are as follows:

P_1=the probability that the system is in (M,1,1) state
P_2=the probability that the system is in (M,D,1) state
P_3=the probability that the system is in (M,0,1) state
P_4=the probability that the system is in (M,R,1) state
P_5=the probability that the system is in (M,0,D) state
P_6=the probability that the system is in (M,0,R) state
P_7=the probability that the system is in (M,0,0) state

The assumptions used in the modeling are as follows:

- Failure rate and repair rate of the VM are identical at all states.
- Detection rate is identical at all states.
- Rejuvenation rate is identical at all states.

After long mission time, the normal states of active VM may change to detection state with rate λ_d. In detection states, server performance is degraded and software-aging effects render the system unreliable. If a server is in detection state (M,D,1)/(M,0,D), the state can change to either a rejuvenation state (M,R,1)/(M,0,R) with rate λ_r or switchover to (M,0,1) state with rate λ_s. In failure state (M,0,0), all servers stop running and no available server remains. When software aging or some potential anomaly is detected in active VM, the system can still operate by using secondary VM and simultaneously rejuvenating the primary VM. This situation is depicted in the model of figure 4 by the transition from state (M,D,1) to (M,0,1) at rate λ_s.

Our state transition diagram can be described as a Markov process class. So we can perform steady-state analysis of the diagram easily. The steady-state balance equations of the system are as follows:

$$\mu P_3 + \mu_r P_4 = \lambda_d P_1 \tag{1}$$
$$\lambda_d P_1 = (\lambda_s + \lambda_r) P_2 \tag{2}$$
$$\lambda_r P_2 = \mu_r P_4 \tag{3}$$
$$\lambda_s P_2 + \mu_r P_6 + \mu P_7 = (\mu + \lambda_d) P_3 \tag{4}$$
$$\lambda_d P_3 = (\lambda + \lambda_r) P_5 \tag{5}$$
$$\lambda_r P_5 = \mu_r P_6 \tag{6}$$
$$\lambda P_5 = \mu P_7 \tag{7}$$

Solving the steady-state balance equations, we find,

$$P_2 = \frac{\lambda_d}{(\lambda_s + \lambda_r)} P_1 \tag{8}$$
$$P_3 = (\frac{\lambda_d}{\mu} - \frac{\lambda_r \lambda_d}{\mu(\lambda_s + \lambda_r)}) P_1 \tag{9}$$
$$P_4 = \frac{\lambda_r \lambda_d}{\mu_r(\lambda_s + \lambda_r)} P_1 \tag{10}$$

$$P_5 = \frac{\lambda_d}{(\lambda + \lambda_r)}(\frac{\lambda_d}{\mu} - \frac{\lambda_r \lambda_d}{\mu(\lambda_s + \lambda_r)})P_1 \tag{11}$$

$$P_6 = \frac{\lambda_r}{\mu_r}\frac{\lambda_d}{(\lambda + \lambda_r)}(\frac{\lambda_d}{\mu} - \frac{\lambda_r \lambda_d}{\mu(\lambda_s + \lambda_r)})P_1 \tag{12}$$

$$P_7 = \frac{\lambda}{\mu}\frac{\lambda_d}{(\lambda + \lambda_r)}(\frac{\lambda_d}{\mu} - \frac{\lambda_r \lambda_d}{\mu(\lambda_s + \lambda_r)})P_1 \tag{13}$$

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.

$$\sum_{i=1}^{n} P_i = 1 \tag{14}$$

$$P_1 = [1 + \frac{\lambda_d}{(\lambda_s + \lambda_r)}(1 + \frac{\lambda_r}{\mu_r}) + (\frac{\lambda_d}{\mu} - \frac{\lambda_r \lambda_d}{\mu(\lambda_s + \lambda_r)})(1 + \frac{\lambda_d}{(\lambda + \lambda_r)}(1 + \frac{\lambda_r}{\mu_r} + \frac{\lambda}{\mu}))]^{-1} \tag{15}$$

Using system-operating parameters, first we obtain the probability P_1, and then we calculate the probabilities of being in (M,D,1), (M,R,1), (M,0,1), (M,0,D), (M,0,R) and (M,0,0) states. The system is not available in rejuvenation state (M,0,R) and failure state (M,0,0). The availability of the system can be defined as follows:

$$Availability = 1 - (P_6 + P_7) \tag{16}$$

The expected total downtime of the system in an interval of T time units is:

$$DownTime = (P_6 + P_7) \times T \tag{17}$$

When the system is down, no service is provided and no revenue is received. There is a business cost due to service being unavailable during downtime. Predictable shutdown cost is far less than that of unexpected shutdown ($C_f \gg C_r$). If C_f is the average per unit cost of unscheduled downtime and C_r is the average per unit cost of downtime during rejuvenation, then the total expected downtime cost in an interval of T time units is:

$$DownTimeCost = ((P_6 \times C_r) + (P_7 \times C_f)) \times T \tag{18}$$

4.1 Experiments

To acquire system dependability measures like availability, downtime and downtime cost, we perform experiments using the system-operating parameters shown in table 1 [11, 14]. We perform mathematical derivation and use SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool to evaluate the feasibility of our model. SHARPE is a well known package [7, 15] in the field of reliability and performability. It is possible to use different kinds of models hierarchically for different physical or abstract levels of the system and to use different kinds of models to validate each other's results.

Table 1. System-operating parameters

Parameters	Values
T	1 year
λ_d	1 time/month
λ_r	1 time/month
$\frac{1}{\lambda_s}$	3 min
λ	1 time/year
μ	2 times/day
$\frac{1}{\mu_r}$	10 min
C_r	20 units
C_f	1000 units

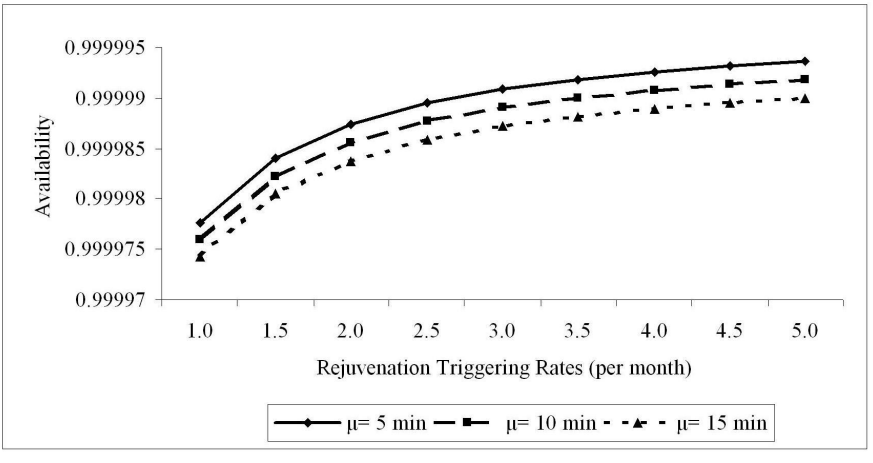

Fig. 5. Availability vs rejuvenation triggering rates and rejuvenation service rates

To examine the influence of rejuvenation triggering rates and recovery rates (rejuvenation service rates) on system availability, we set the value of recovery rate is range from 1 to 5 and rejuvenation service time is range from 5 min to 15 min. The plot of system availability in the case of performing rejuvenation is shown in figure 5.

The graph (figure 5) shows that the larger the rejuvenation triggering rates, the larger the system availability that is expected. We also find out that the faster the rejuvenation service time, the larger the availability of the system. According to the result, system availability can increase by using software rejuvenation and vir-tualized clustering technology.

The change in the downtime of virtualized clustering system with the different rejuvenation triggering rates and different rejuvenation service rates is plotted in figure 6. The change in the downtime cost of virtualized clustering system

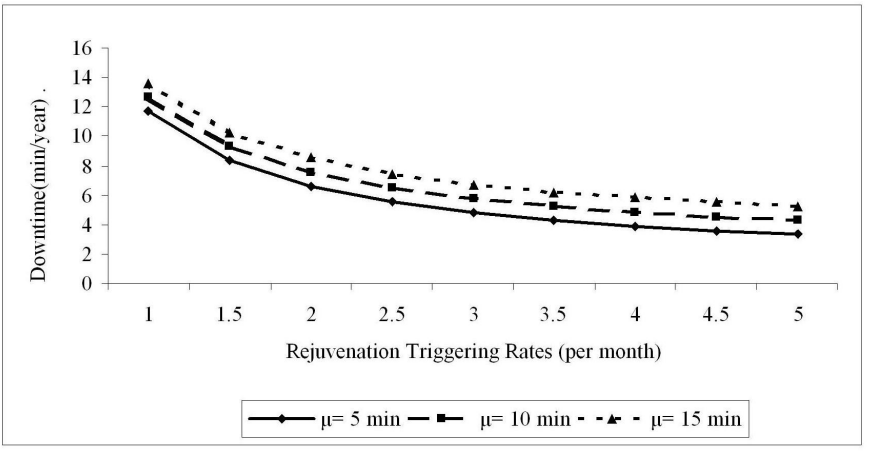

Fig. 6. Downtime vs rejuvenation triggering rates and rejuvenation service rates

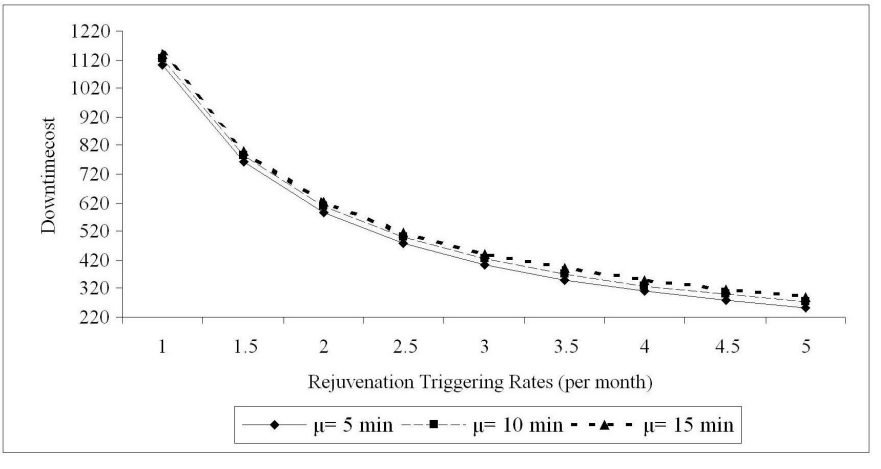

Fig. 7. Downtime cost vs rejuvenation triggering rates and rejuvenation service rates

with the dif-ferent rejuvenation triggering rates and different rejuvenation service rates is plot-ted in figure 7.

The downtime cost of scheduled shutdown is much lower than that of an unscheduled shutdown. According to the figure 6 and 7, rejuvenation can decrease both the downtime and the cost due to down time in this example and hence it is beneficial.

From our results, it is apparent that our proposed approach is a cost-effective way to build high availability system and virtualized clustering technology can improve the software rejuvenation process. For our proposed approach's validity

we use SHARPE tool. And the evaluation results through SHARPE are same with our mathematical results.

5 Conclusion

In this paper, we proposed a new approach to solve software aging problem through the use of software rejuvenation and virtualization. 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 model. We validated our experiment results with the evaluation results through SHARPE tool. It is found that our analytical results and SHARPE results are same. Our results show that our approach can be used to prolong the availability of the services. Our approach can be applied to single-server or cluster configurations without any additional cost. Future work will include experiments with an implementation under real world conditions to verify the practical efficacy of the approach.

Acknowledgement. His research was supported by the Advanced Broadcasting Media Technology Research Center (ABRC) in Korea Aerospace University, Korea, under the Gyeonggi Regional Research Center (GRRC) support program supervised by Gyeonggi Province.

References

1. Alonso, J., Silva, L., Andrzejak, A., Silva, P., Torres, J.: High-available grid services through the use of virtualized clustering. In: Proc. of the 8th IEEE/ACM International Conference on Grid Computing, pp. 34–41 (2007)
2. Cassidy, K., Gross, K., Malekpour, A.: Advanced pattern recognition for detection of complex software aging phenomenain online transaction processing servers. In: Proc. of the Int. Conf. on Dependable Systems and Networks, pp. 478–482 (2002)
3. Castelli, V., Harper, R.E., Heidelberger, P., Hunter, S.W., Trivedi, K.S., Vaidyanathan, K., Zeggert, W.P.: Proactive management of software aging. IBM Journal of Research and Development 45(2), 311–332 (2001)
4. Creasy, R.J.: The origin of the VM/370 time-sharing system. IBM Journal of Research and Development 25(5), 483 (1981)
5. Dohi, T., Popstojanova, K.G., Vaidyanathan, K., Trivedi, K.S., Osaki, S.: Software rejuvenation modeling and applications, Springer Reliability Engineering Handbook, pp. 245–263. Springer, Heidelberg (2003)
6. Garg, S., van Moorsel, A., Vaidyanathan, K., Trivedi, K.: A methodology for detection and estimation of software aging. In: Proc. of the 9th Int. Symp. on Software Reliability Engineering, pp. 282–292 (1998)
7. Hirel, C., Sahner, R.A., Zang, X., Trivedi, K.S.: Reliability and performability modeling using SHARPE 2000. In: Proc. of the Int. Conf. on Computer Performance Evaluation: Modelling Techniques and Tools, pp. 345–349 (2000)
8. Huang, Y., Kintala, C., Kolettis, N., Fulton, N.D.: Software rejuvenation: analysis, module and application. In: Proc. Of the Fault Tolerance Computing Symp., pp. 381–390 (1995)

9. Jagarajan, A., Mueller, F., Engelmann, C., Scott, S.: Proactive fault tolerance for HPC with Xen virtualization. In: Proc. of the Int. Conf. on Supercomputing 2007, pp. 23–32 (2007)
10. Kourai, K., Chiba, S.: A fast rejuvenation technique for server consolidation with virtual machines. In: Proc. of the 37th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN 2007), pp. 245–255 (2007)
11. Park, K., Kim, S.: Availability analysis and improvement of active/standby cluster systems using software rejuvenation. The Journal of Systems and Software 61, 121–128 (2002)
12. Silva, L.M., Alonso, J., Silva, P., Torres, J., Andrzejak, A.: Using virtualization to improve software rejuvenation. In: Proc. of the 6th IEEE Int. Symp. on Network Computing and Applications, pp. 33–44 (2007)
13. Silva, L., Madeira, H., Silva, J.G.: Software aging and rejuvenation in a SOAP-based server. In: Proc. of the IEEE Network Computing and Applications, pp. 56–65 (2006)
14. Software Rejuvenation. Department of Electrical and Computer Engineering, Duke University, `http://www.software-rejuvenation.com/`
15. Trivedi, K.S.: Symbolic hierarchical automated reliability and performance evaluator (SHARPE). In: Proc. of the Dependable System and Networks, p. 544 (2002)
16. Trivedi, K.S., Vaidyanathan, K., Postojanova, K.G.: Modeling and analysis of software aging and rejuvenation. In: Proc. of the 33rd Annual Simulation Symp., pp. 270–279 (2000)
17. Vaidyanathan, K., Trivedi, K.: A comprehensive model for software rejuvenation. IEEE Trans. on Dependable and Secure Computing 2(2), 124–137 (2005)