

# HIGH AVAILABILITY SOLUTION: RESOURCE USAGE MANAGEMENT IN VIRTUALIZED SOFTWARE AGING

Aye Myat Myat Paing<sup>1</sup> and Ni Lar Thein<sup>1</sup>

<sup>1</sup>University of Computer Studies, Yangon  
amyat2p@googlemail.com

## ABSTRACT

*As businesses increasingly rely on IT for their mission-critical operations, continuous availability is a universal concern. Nowadays, virtualized platform has become the popular option to deploy complex enough services. Deployed services are expected to be always, available, but these long running services are especially sensitive to suffer from software aging phenomenon. Software aging of virtual machine monitors (VMMs) is becoming critical because performance degradation or crash failure of a VMM affects all virtual machines (VMs) on it. To counteract this issue, we deploy software rejuvenation methodology. To eliminate the service outage during software rejuvenation process, we combine the rejuvenation with live migration technology. Live VM migration enables a running VM on a host server to move onto the other host server with very small interruption of the execution. Live VM migration depends on VMs placement and efficient resource available is required. The idea behind our paper is two-fold. First, we present the optimization of the resource usage as accepting as many services as in virtualized environment which support of VM live migration. Second, to demonstrate how much it can improve system availability, the stochastic Petri nets model a virtualized server system in case of using time based software rejuvenation for VMM is presented. Finally, we perform the numerical analysis to evaluate the model.*

## KEYWORDS

*Availability, Clustering Technology, Software Aging, Software Rejuvenation, Stochastic Petri Nets, Virtualization.*

## 1. INTRODUCTION

Availability has long been a critical issue for online computer systems whose failure can halt business process. This is particularly more pertaining to high demanded high available (HA) computing systems based on off-the-shelf components. Exhaustion of system resources, data corruption, and numerical error accumulation are the primary symptoms of the degradation, which may eventually lead to performance degradation of the software, crash/hang failure, or other undesirable effects. That degradation of software is known as software aging. Software 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 [3].

System Virtualization techniques are getting popular and gaining significant interest in the enterprise and personal computing spaces. The majority of today's high availability (HA) clusters is based on real physical hardware and virtualization is coming more and more popular nowadays, one has to think about possible combinations of virtualization and high availability clustering. Modern computers are sufficiently powerful to use VMs. Many fields, such as autonomic computing, server consolidation, security and education publish results that praise the benefits of

virtualization. Software rejuvenation can be applied so as to mitigate adverse effects of software aging. Software rejuvenation [3] 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. To eliminate the service outage during software rejuvenation process, we combine rejuvenation with virtualization technology. A virtualization layer also called virtual machine monitor (VMM) is a software layer that abstracts the physical resources for use by the VMs. Recently, software aging of VMMs is becoming critical because many VMs run on top of a VMM in one machine consolidating multiple servers and aging of the VMM directly affects all the VMs. Without any rejuvenation, both the OS software and the application software running on top of VM degrade in performance with time due to the exhaustion of system resources such as free physical memory, and eventually crash, which is very undesirable to high availability (HA) systems and fatal to mission-critical applications.

One of the promises of virtualization is the ability to allow applications to dynamically move from one physical server to another as the demands and resource availabilities change, without service interruption. More recently, Xen [1] and VMWare [8] have implemented "live" migration of VMs that involve extremely short downtimes. To migrate, we must know which virtual machine needs to be migrated and when this relocation has to be done and, moreover, which host must be destined.

The main contributions of this paper are two-fold. Firstly, we present the optimization of the resource usage as accepting as many services as possible on the virtualized environment. Secondly, we present VM migration based rejuvenation policy to offer the high availability by preventing failures due to software aging. We use a stochastic Petri nets based approach to build models and evaluate through both analytical and SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) tool simulation [6].

The organization of this paper is as follows. In Section 2 we discuss the related work. The proposed resource management algorithm for enhancing system availability and stochastic Petri nets model for system availability are follow in Section 3 and 4. Finally we conclude our paper in Section 5.

## **2. RELATED WORK**

In this section we review selected publications related to our work. There are many benefits to virtualization technologies. The authors [2] presented a mixed software rejuvenation policy for an operational software system with multiple degradation states, which considered both the history information and the current running state. By this policy, the system is rejuvenated when it achieves to a degradation threshold or it comes to pre-determined rejuvenation interval. Continuous-time Markov chains were used to describe the multiple degradation states model.

Machida et. al [7] presented comprehensive availability models of three rejuvenation techniques for a server virtualized system with time-based rejuvenations for VM and VMM. The result of sensitivity analysis showed that Migrate-VM rejuvenation achieves the best steady-state availability as long as VM live migration is fast enough and other servers have capacity to receive the migrated VM. However they do not consider the VMs placement on other physical servers when VM migration is needed.

Rezaei et al. [9] proposed a new rejuvenation technique for high available virtualized systems that is applied at both VM and VMM levels and yet it does not require any modifications to applications. They have considered a typical virtualized system that acts as a hot-standby machine containing two VMs that run on a single VMM.

High available systems using virtualization technology and software rejuvenation methodology are proposed by Thein et al. [11, 12]. In [12], they provide stochastic process based models to evaluate availability of the system in case of without virtualization technology and in case when virtualization and software rejuvenation are used. Virtual machine based software rejuvenation framework to offer high availability for application servers is presented in [11].

Kim et. al [4] have presented availability models of a nonvirtualized and virtualized system using two level hierarchical stochastic models with fault tree and CTMC. In virtualized system model, HA service and VM live migration are incorporated. In order to minimize the downtime of a server virtualized system, a fast software rejuvenation technique called Warm-VM Reboot for VMM [5] was presented by introducing the on-memory suspend technique and quick reload mechanism.

In our work, to analyze the availability of virtualized server system we present stochastic Petri nets model for time-based rejuvenation policy for VMM. To support live VM migration, resource usage management algorithm is proposed.

### **3. PROPOSED RESOURCE MANAGEMENT ALGORITHM FOR ENHANCING SYSTEM AVAILABILITY**

To counteract the software aging phenomenon for VMM in virtualized cluster system, we combine software rejuvenation and live VM migration. During the live VM migration, we need to consider the process of dynamically allocating VMs to other PMs in resource pool and to meet their resource requirements. Therefore, resource management algorithm is proposed for guaranteeing the availability of potential services and is accepted as many services as possible according to the available resource capacity. This section presents system architecture and resource management in virtualized environment.

#### **3.1. System Architecture**

The system consists of virtualized cluster servers and management server. Example system architecture is shown in Figure 1. Clustering supports two or more servers running multiple VMs. To enable live VM migration, all PMs are connected on the same network and the shared disk image. A heartbeat keep-alive system is used to monitor the health of the PM between them. The management server contains three service providers: resource manager, aging detector and rejuvenation manager. The aging detector of management server is responsible for the detection of software aging. When software aging happens in active VMM, the rejuvenation manager will trigger the rejuvenation action. Before rejuvenation, the resource manager is responsible for getting resource information, and how to allocate the VMs to accept the maximum number of services in virtualized environment according to the resources available.

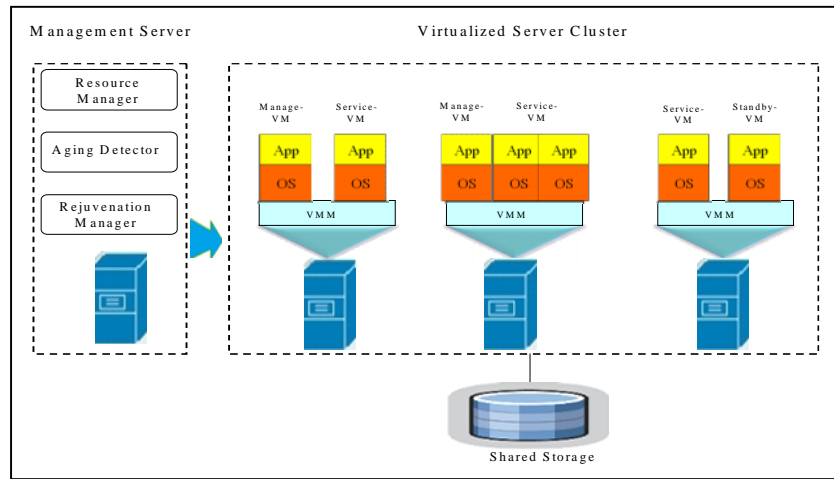


Figure 1. System Architecture

The sequence diagram for proposed system is shown in Figure 2. When some potential anomaly happens, in order not to lose any in-flight request or session data at the time of rejuvenation for VMM, all the VMs on that aging affected PM are migrated to one of the selected PM from virtualized cluster system using resource management algorithm. After the VM migration, a rejuvenation operation will be triggered for VMM.

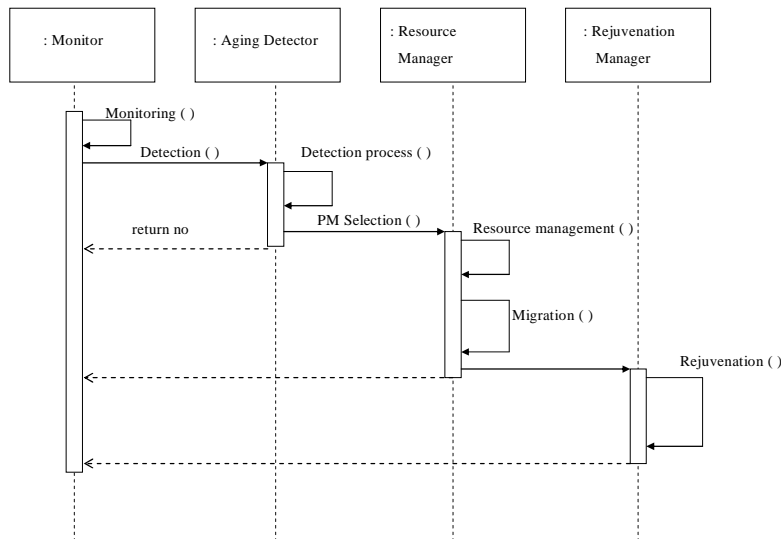


Figure 2. Sequence Diagram for Proposed System

### 3.2. Resource Management in Virtualized Environment

This sub-section describes how to solve the software aging failure in VMM using software rejuvenation and live VM migration. Live VM migration technology allows us to migrate a running VM without any or minimal outage, from one physical machine to another.

As time degrades, the VMM running on the PM becomes software aging. When the software aging or some potential anomaly happens in one of the PM in the resource pool, the rejuvenation manager of management server will trigger a rejuvenation operation. If the aging affected PM is about to be rejuvenated, all the new requests and sessions are migrated from the VMs from the aging affected PM to VMs from other PMs in the resource pool. When the ongoing requests are finished in aging infected PM, these VMM will be rejuvenated.

With the advent of the virtualization, live migration can be performed without service interruption. In order to carry out the appropriate VM live migrations, the proposed Resource Management Algorithm carries out how to allocate the VMs to accept the maximum number of services on the virtualized infrastructure according to the resources available. The resource management algorithm is presented in Figure 3.

The assumptions used in the resource management algorithm are as follow.

- It does not allowed same service in one PM because it can be decided to which PM of each VM must be allocated such that number of VMs is maximized the services and VMs requirements are satisfied without exceeding PM's capacity and whole infrastructure capacity limits.
- One of the PMs in the resource pool can suffer software aging at a time.

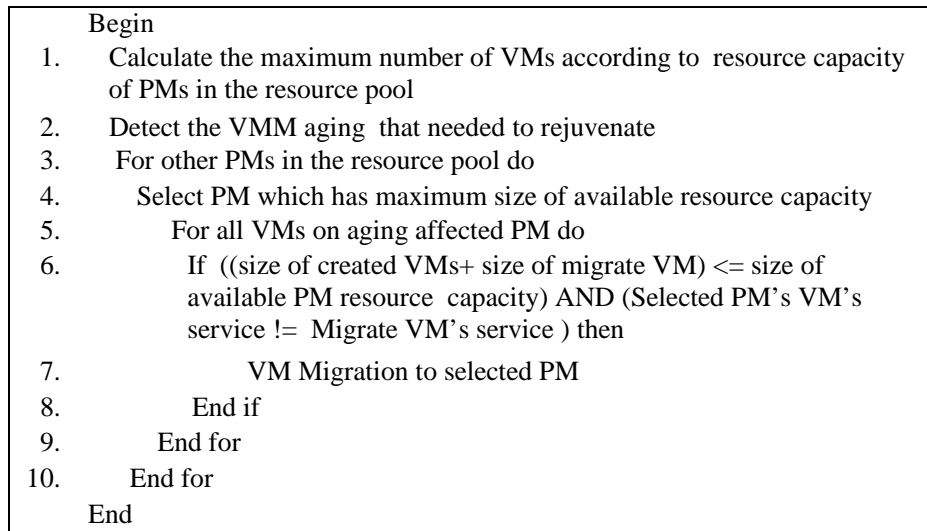


Figure 3. Resource Management Algorithm

The example scenario is described for calculating the maximum number of services as the input for the resource management algorithm taking into account the VM and the PM features as shown in Table 1.

Table 1. Example Resource Capacity Description for VM and PM

Role	Memory Size
PM1, PM2, PM3	16384 MB
Virtualization Layer (VMWare ESX)	4096 MB
Service VM	2048 MB
Manage-VM	512 MB
Standby VM	2048 MB

In this basic scenario, the VM size is only computed with the memory required and based on virtualized clustering architecture. The VM's size is predefined and fixed on the available physical resources.

The example infrastructure is composed of three physical machines, and Manage-VM is deployed separately with service VMs: then the maximum numbers of VMs is computed as followings:

$$\text{Available Memory for each PM} = \text{Memory size of each PM} - \text{Memory size usage of VMM} \tag{1}$$

$$\text{Total Available Memory} = \sum_{i=1}^m \text{memorysize of PM}_m \tag{2}$$

(m= number of PMs)

$$\text{Maximum number of VMs} = \frac{\text{Total Available Memory size}}{\text{Memory size of each VM}} \tag{3}$$

According to the Table I, we can calculate available memory for each PM: 12288 MB and the example infrastructure is composed of 3 PMs then we get 36864 MB total memory available. If we only deploy service VM (basic scenario):  $36864/2048 = 18$  VMs are available to be run simultaneously as shown in Table 2.

For the scenario of Manage VM associated with every Service VM, and a load balanced scenario of Manage-VM and two service VMs, the maximum number of services can be deployed as shown in Table 2.

Table 2. Maximum Number of VMs accepted by different scenario

	Maximum Allowed VMs
Basic Scenario	18
Scenario with Manage -VM	14
Load Balanced Scenario with LB(with Manage-VM and Standby VM)	8

#### 4. STOCHASTIC PETRI NETS MODEL FOR SYSTEM AVAILABILITY

In this section, we present stochastic Petri nets (SPN) model with time-based rejuvenation policy for VMM in virtualized cluster system as shown in Figure 4. It has  $n$  tokens which represented  $n$  VMs and  $m$  tokens which represented  $m$  PMs. In this policy, the rejuvenation interval is determined by using clock. If the active VMM is about to be rejuvenated cause of software aging, all the VMs on aging affected PM are migrated to other PMs according to VMs placement through resource management algorithm and then will be started for the new requests and sessions. It can return back to the original PM after the completion of the VMM rejuvenation through live VM migration.

In initial condition all PMs are “healthy” working states, indicated by a token in place  $P_{up}$ . As time progresses, each VMM eventually transits to failure probably states in place  $P_{FP}$  through the transition  $T_{fp}$  but the failure probably rates are different for each VMM. The VMM are still operation in this state but VMs need to migrate when token is placed in  $P_{FP}$  and before VMM rejuvenation. When the VMs migration transition  $T_{mig}$  is enabled, the VMs from  $P_{VM}$  are migrated to another PM and a token is moved to a place  $P_{Mig}$ . After the migration, VMs will be restarted on another PM.

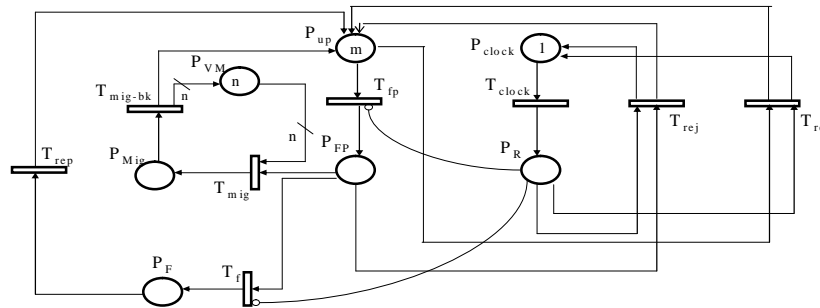


Figure 1. Stochastic Petri Nets Model for VMM Rejuvenation

Places	Description
$P_{up}$	: Healthy state of $i^{th}$ PM
$P_{FP}$	: Failure Probably state of $i^{th}$ PM $i= 1,2,3$ (number of PM)
$P_F$	: Failure state
$P_R$	: Rejuvenation state of $i^{th}$ PM
$P_{Mig}$	: VMs Migration state from aging PM to selected PM for migration
$P_{clock}$	: Rejuvenation Interval state of $i^{th}$ PM
$P_{VM}$	: Available VMs state on $i^{th}$ PM

In this model, rejuvenation interval is determined by using a clock and there are tokens in the place  $P_{clock}$ . There are VMMs to be rejuvenated (a token is placed in  $P_{FP}$  and  $P_{up}$ ), the transitions  $T_{clock}$  is enabled. This transition is competitively enabled with  $T_{fp}$  and fires when the clock expires

if  $T_{fp}$  has not fired by that time. Once it fires, token moves in the place  $P_R$  and the activity related with software rejuvenation. During the rejuvenation, every other activity in the VMM is suspended. This is modeled by inhibitor arcs from  $P_R$  to transitions  $T_{fp}$  and  $T_f$ . After VMMs have been rejuvenated, it goes back to healthy state with transition  $T_{rej}$  and a token is also reset to  $P_{clock}$ . When there is a system crash (i.e., there is a token is placed in place  $P_F$  by using transition  $T_f$ . From a full system outage, the system can be repaired through the transition  $T_{rep}$  and all VMMs are in healthy state in place  $P_{up}$ .

### 4.1. Reachability Graph

We construct the reachability graph for SPN model with 2VMs and 3PMs ( $n=2, m=3$ ). Let 7 tuple  $(P_{up}, P_{FP}, P_{clock}, P_R, P_{Mig}, P_F$  and  $P_{VM})$  denote the marking with  $P_x = 1$ , if a token is presented in place  $P_x$ , and zero otherwise. A marking is reachable from another marking if there exists a sequence of transition firings starting from the original marking that result in the new marking. The marking process is mapped into a continuous time Markov chain (CTMC) with state space isomorphic to the reachability graph of the SPN as shown in Figure 5.

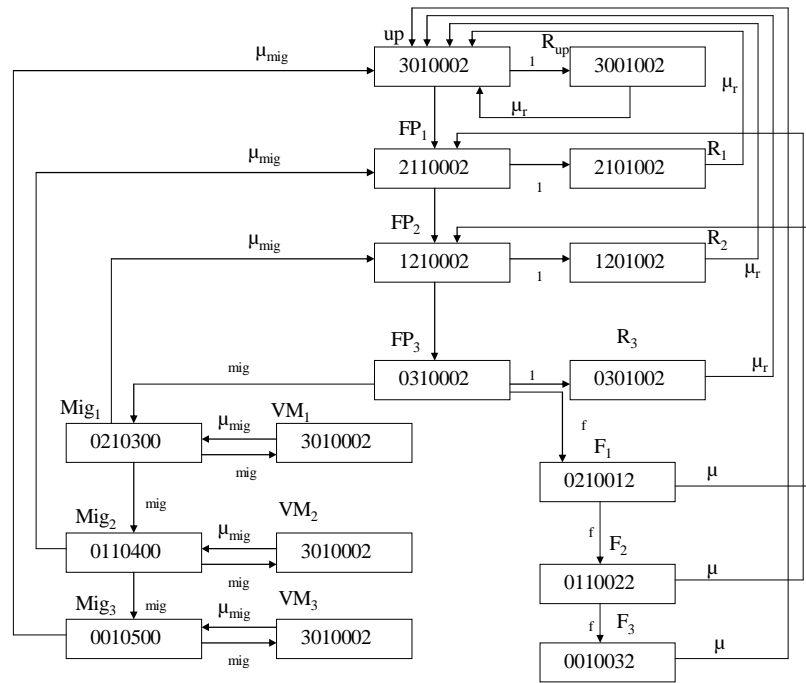


Figure 2. Reachability Graph for SPN Model

Figure 5 illustrates the reachability graph with squares representing the markings and arcs representing possible transition between the markings. Let  $\mu, \mu_1, f, \mu_{mig}, \mu_r,$  and  $\mu$  be the transition rates associated with  $T_{fp}, T_{clock}, T_f, T_{mig}, T_{rej}$  and  $T_{rep}$  respectively.

By mapping through actions to this reachability graph with stochastic process, we get mathematical steady-state solution of the chain. We may compute the steady-state probability by first writing down the steady-state balance equations of figure as follows.



$$\mu_r P_{R_2} + \mu_r P_{R_1} + \mu_r P_{R_{up}} + \mu_r P_{R_3} + \mu P_{F_3} + \mu_{mig} P_{Mig_3} = \lambda P_{up} + \lambda_1 P_{up} \quad (4)$$

$$\lambda_1 P_{up} = \mu_r P_{R_{up}} \quad (5)$$

$$\lambda P_{up} + \mu_{mig} P_{Mig_2} + \mu P_{F_2} = \lambda P_{FP_1} + \lambda_1 P_{FP_1} \quad (6)$$

$$\lambda_1 P_{FP_1} = \mu_r P_{R_1} \quad (7)$$

$$\lambda P_{FP_1} + \mu_{mig} P_{Mig_1} + \mu P_{F_1} = \lambda P_{FP_2} + \lambda_1 P_{FP_2} \quad (8)$$

$$\lambda_1 P_{FP_2} = \mu_r P_{R_2} \quad (9)$$

$$\lambda P_{FP_2} = \lambda_{mig} P_{FP_3} + \lambda_f P_{FP_3} + \lambda_1 P_{FP_3} \quad (10)$$

$$\lambda_1 P_{FP_3} = \mu_r P_{R_3} \quad (11)$$

$$\lambda_{mig} P_{FP_3} + \mu_{mig} P_{VM_1} = \mu_{mig} P_{Mig_1} + \lambda_{mig} P_{Mig_1} + \lambda_{mig} P_{Mig_1} \quad (12)$$

$$\lambda_{mig} P_{Mig_1} = \mu_{mig} P_{VM_1} \quad (13)$$

$$\lambda_{mig} P_{Mig_1} + \mu_{mig} P_{VM_2} = \lambda_{mig} P_{Mig_2} + \lambda_{mig} P_{Mig_2} + \mu_{mig} P_{Mig_2} \quad (14)$$

$$\lambda_{mig} P_{Mig_2} = \mu_{mig} P_{VM_2} \quad (15)$$

$$\lambda_{mig} P_{Mig_2} + \mu_{mig} P_{VM_3} = \lambda_{mig} P_{Mig_3} + \mu_{mig} P_{Mig_3} \quad (16)$$

$$\lambda_{mig} P_{Mig_3} = \mu_{mig} P_{VM_3} \quad (17)$$

$$\lambda_f P_{FP_3} = \mu P_{F_1} \quad (18)$$

$$\lambda_f P_{F_1} = \mu P_{F_2} \quad (19)$$

$$\lambda_f P_{F_2} = \mu P_{F_3} \quad (20)$$

The conservation equation of Figure 5 is obtained by summing the probabilities of all states in the system and the sum of equation is 1.

$$P_{up} + P_{R_{up}} + \sum_{i=1}^3 P_{FP_i} + \sum_{i=1}^3 P_{Mig_i} + \sum_{i=1}^3 P_{R_i} + \sum_{i=1}^3 P_{F_i} + \sum_{i=1}^3 P_{VM_i} = 1 \quad (21)$$

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

$$P_{R_{up}} = \frac{\lambda_1}{\mu_r} P_{up} \quad (22)$$

$$P_{FP_2} = \frac{\lambda_{mig} + \lambda_f + \lambda_1}{\lambda} BP_{up} \quad (23)$$

$$P_{R_2} = \frac{\lambda_1}{\mu_r} \left( \frac{\lambda_{mig} + \lambda_f + \lambda_1}{\lambda} \right) BP_{up} \quad (24)$$

$$P_{FP_1} = \frac{1}{\lambda} ABP_{up} \quad (25)$$

$$P_{R_1} = \frac{\lambda_1}{\mu_r} \left( \frac{1}{\lambda} A \right) BP_{up} \quad (26)$$

$$P_{FP_3} = BP_{up} \quad (27)$$

$$P_{R_3} = \frac{\lambda_1}{\mu_r} BP_{up} \quad (28)$$

$$P_{Mig_1} = \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) BP_{up} \quad (29)$$

$$P_{VM_1} = \frac{\lambda_{mig}}{\mu_{mig}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) BP_{up} \quad (30)$$

$$P_{Mig_2} = \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 BP_{up} \quad (31)$$

$$P_{VM_2} = \frac{\lambda_{mig}}{\mu_{mig}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 BP_{up} \quad (32)$$

$$P_{Mig_3} = \frac{\lambda_{mig}}{\mu_{mig}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 BP_{up} \quad (33)$$

$$P_{VM_3} = \left( \frac{\lambda_{mig}}{\mu_{mig}} \right)^2 \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 BP_{up} \quad (34)$$

$$P_{F_1} = \frac{\lambda_f}{\mu} BP_{up} \quad (35)$$

$$P_{F_2} = \left( \frac{\lambda_f}{\mu} \right)^2 BP_{up} \quad (36)$$

$$P_{F_3} = \left( \frac{\lambda_f}{\mu} \right)^3 BP_{up} \quad (37)$$

$$P_{up} = \left\{ \begin{aligned} & 1 + \frac{\lambda_1}{\mu_r} + B + \frac{\lambda_{mig} + \lambda_f + \lambda_1}{\lambda} B + \frac{\lambda_1}{\mu_r} \left( \frac{\lambda_{mig} + \lambda_f + \lambda_1}{\lambda} \right) B + \frac{1}{\lambda} AB + \frac{\lambda_1}{\mu_r} \left( \frac{1}{\lambda} A \right) B \\ & + \frac{\lambda_1}{\mu_r} B + \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) B + \frac{\lambda_{mig}}{\mu_{mig}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) B \\ & + \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) B + \frac{\lambda_{mig}}{\mu_{mig_2}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) B + \frac{\lambda_{mig}}{\mu_{mig}} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 B \\ & + \left( \frac{\lambda_{mig}}{\mu_{mig}} \right) \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) B + \frac{\lambda_f}{\mu} B + \left( \frac{\lambda_f}{\mu} \right) B + \left( \frac{\lambda_f}{\mu} \right) B \end{aligned} \right\}^{-1} \tag{38}$$

Where

$$A = \left[ (\lambda_{mig} + \lambda_f + \lambda_1) + \lambda_1 \left( \frac{\lambda_{mig} + \lambda_f + \lambda_1}{\lambda} \right) - \mu_{mig} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right) - \lambda_f \right]$$

$$B = \frac{\lambda}{A + \frac{\lambda_1}{\lambda} A - \mu_{mig} \left( \frac{\lambda_{mig}}{\lambda_{mig} + \mu_{mig}} \right)^2 - \mu \left( \frac{\lambda_f}{\mu} \right)^2}$$

The meaning of the probabilities as follows:

- $P_{up}$  : The probability of the VMM is in healthy state
- $P_{R_{up}}$  : The probability of the VMM is in rejuvenation state from VMM healthy state
- $P_{R_i}$  : The probability of the VMM is in rejuvenation state from VMM failure probably state (i=1,2,3 where i= number of PM)
- $P_{FP_i}$  : The probability of the VMM is in failure probably state
- $P_{Mig_i}$  : The probability of the VMs migration state
- $P_{F_i}$  : The probability of VMM is in failure state
- $P_{VM_i}$  : The probability available VMs on PM state

### 4.2. Availability and Downtime Analysis

Availability is a probability of a system which provides the services in a given instant time. Based on this, system availability and unavailability can be computed. In the proposed model, services are not available when both VMMs are in failure state  $P_{F_i}$ . We also define the availability of the SPN model as:

Availability = 1 - Unavailability

$$Availability = 1 - \sum_{i=1}^3 P_{F_i} \tag{39}$$

Downtime is the expected total downtime of the application with rejuvenation in an interval of T time units is:

$$\text{Downtime} = T \times \left( \sum_{i=1}^3 P_{F_i} \right) \tag{40}$$

### 4.3. Numerical Analysis

In this subsection, we illustrate the applicability of the SPN model and solution methodology through numerical analysis. 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 3 [3,10].

Table 3. Transition Firing Rates

Transitions	Firing Rates (h <sup>-1</sup> )
T <sub>fp</sub>	1 time/a day
T <sub>f</sub>	2times/ 3 months
T <sub>clock</sub>	1time/ month
1/T <sub>mig</sub>	3 mins
1/T <sub>rej</sub>	10 mins
T <sub>repair</sub>	1time/ 2 hours

The influence of VM migration transition firing rates along with different rejuvenation rates on availability is shown in Figure 6. The mean time to VM migration transition is assumed 2 mins and 3 mins. The higher VM migration rates, the higher availability of our system model can be achieved. Therefore, the availability is dependent on the mean time to VM migration transition.

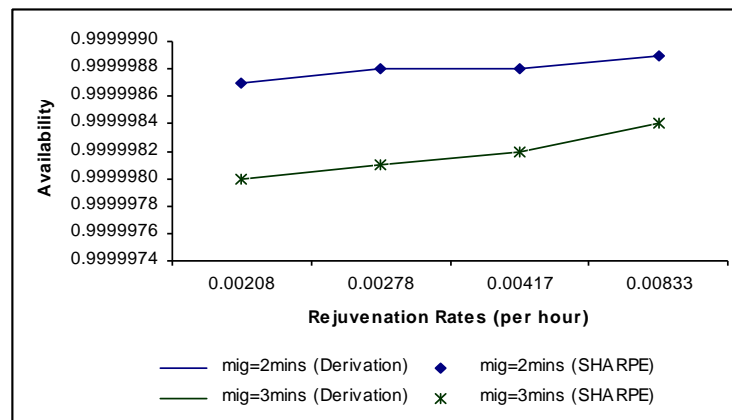


Figure3. Availability vs. different rejuvenation rates and different mean time to VM migration

The Figure 7 shows the differences in downtime with different mean time to VM migration transition and different rejuvenation transition firing rates. From the result, it is apparent that by combining virtualization technology and software rejuvenation mechanism can enhance the availability of virtualized IT infrastructure.

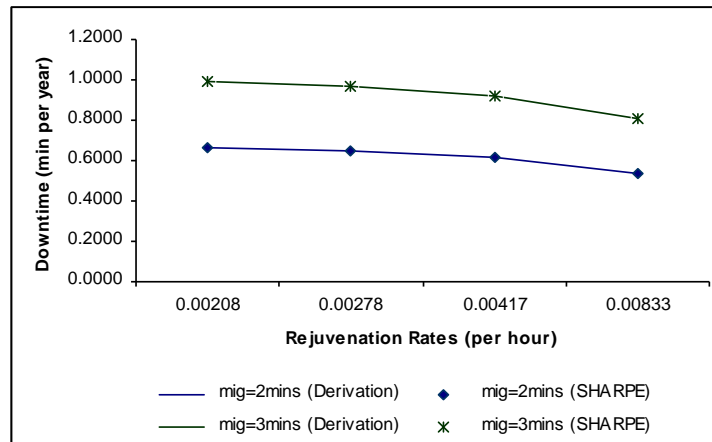


Figure 4. Downtime vs. different rejuvenation rates and different mean time to VM migration

The change in availability of a system with different rejuvenation rates and different VMM failure rates is plotted in Figure 8. The influence of failure rates along with different rejuvenation rates on availability can be seen from this figure. We assume the failure rates are one time per month and two times per three months. High availability systems require fewer failures and faster repair. We observe from Figure 8 that the failure rate of the system acts as an important factor in the availability of the virtualized system.

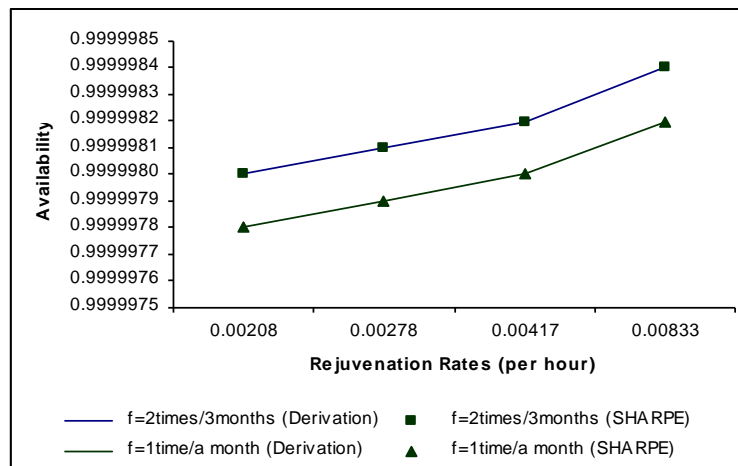


Figure 5. Availability vs. different rejuvenation rates and different VMM failure rates

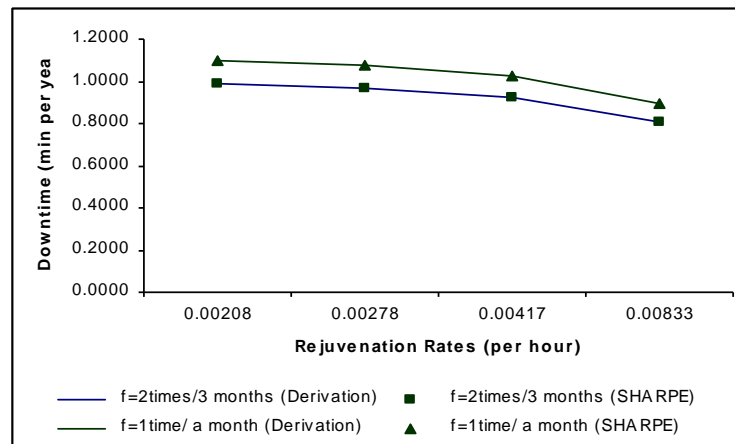


Figure 6. Downtime vs. different rejuvenation rates and different VMM failure rates

Figure 9 plotted the downtime as a function of the different rejuvenation rates. According to the Figure 9, we can see that the fewer failure, the lower downtime can be achieved.

SHARPE [6] is well known package 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. So for our models' validity we use this tool. According to Figure 6, 7, 8 and 9, it is found that the derivation results and SHARPE tool simulation results are the same.

## 5. CONCLUSION

Software can cause software aging as time degrades. Software aging decreases the availability of the system. It is widely understood that the technique of rejuvenation provides better results, resulting in higher availability and lower cost. In this paper we presented possible combinations of virtualized clustering and software rejuvenation in order to counteract the software aging and also presented a resource management algorithm to guarantee the availability of the services deployed and optimize the resources available in the infrastructure. Stochastic Petri nets model for analyzing VMM time based software rejuvenation in continuously running applications are presented and express availability and downtime and in terms of the transitions firing rates in the model. The numerical results are validated with the evaluation results through SHARPE tool. It is found that the derivation results and the SHARPE result are the same. The proposed combine method can be applied to any cluster server configurations without any additional cost. The obtained results show that the use of virtualization, clustering technology and software rejuvenation mechanism can provide a very fast recovery to cut down the mean time to recovery to the minimum. It can achieve minimize downtime even in case of service restart.

## REFERENCES

- [1] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In Proc. NSDI '05, May 2005.
- [2] X. Du, Y. Qi, D. Hou, Y. Chen, X. Zhong, "A Mixed Software Rejuvenation Policy for Multiple Degradations Software System," Proceedings of 11th IEEE International Conference on High Performance Computing and Communications 2009, pp.376-383,2009

- [3] Y. Huang, C. Kintala, N. Kolettis, N. D. Fulton, "Software rejuvenation: Analysis, module and application," Proc. the 25th Int. Symp., Fault-Tolerant Computing, Pasadena, CA, June. 1995, pp.381-390. doi: 10.1109/FTCS.1995.466961
- [4] D. Kim, F. Machida, and K. S. Trivedi, Availability modeling and analysis of a virtualized system, In Proc. of IEEE Int'l Symp. Pacific Rim Dependable Computing (PRDC 2009), 2009.
- [5] K. Kourai and S. Chiba, Fast software rejuvenation of virtual machine monitor, In IEEE Trans. on dependable and secure computing, 2010.
- [6] K. S. Trivedi, SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator. In Proc. Int. Conference on Dependable Systems and Networks, 2002, pp. 544.
- [7] F. Machida, D. Kim, and K. S. Trivedi, Modeling and Analysis of Software Rejuvenation in a Server Virtualized System. In Proc. of the 2rd International Workshop on Software Aging and Rejuvenation, (WoSAR2010), 2010.
- [8] F. Machida, D. Kim, J. Park and K. S. Trivedi, Toward Optimal Virtual Machine Placement and Rejuvenation Scheduling in a Virtualized Data Center, In Proc. of the 1st Int'l Workshop on Software Aging and Rejuvenation (WoSAR2008), 2008.
- [9] A. Rezaei and M. Sharifi, Rejuvenating High Available Virtualized Systems", The 2010 International Conference on Availability, Reliability and Security, IEEE, 2010
- [10] Software rejuvenation. Department of Electrical and Computer Engineering, Duke University, [Online] Available form: <http://www.software-rejuvenation.com>
- [11] T. Thein and J. S. Park, Availability Analysis of Application Servers Using Software Rejuvenation and Virtualization, J. Computer Science and Technology, 24(2), 2009, pp. 339-346.
- [12] T. Thein, S. Chi, J. Park, "Improving Fault Tolerance by Virtualization and Software Rejuvenation", In Proceedings of the Second Asia International Conference on Modeling & Simulation, Kuala Lumpur, Malaysia, 2008, pp. 855-860.