Availability Modeling and Analysis of a Single-Server Virtualized System with Rejuvenation

Jian Xu, Xuefeng Li, Yi Zhong and Hong Zhang
Institute of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing, 210094,China
dolphin.xu@njust.edu.cn

***Abstract*—Availability of business-critical application servers is an issue of paramount importance that has received special attention from the industry and academia in the last decade. This paper presents two stochastic reward net based availability models for a single-server virtualized system. The similarity in both models is that software rejuvenation is applied at not only virtual machine monitor (VMM) level using a time-base policy but also at virtual machines (VMs) using a prediction-based policy. The key difference is that the passive software replication and the active software replication are respectively adopted at the VM level of both models. We compare these models in terms of steady-state system availability by numerical analysis. Results show steady-state system availability with the active replication style gets a bit better than that of the passive one. Further, we study the impact of two critical parameters, the VMM rejuvenation interval and the VM aging detection probability, on downtime and on the number of transaction lost by sensitivity analysis.**

***Index Terms*—Analytic model, virtualization, availability, stochastic reward net, software rejuvenation**

I. Introduction

Software aging is the phenomenon of progressive degradation of running software image which might lead to performance reduction, hang ups or even crashes [1]. The primary causes are exhaustion of systems resources, like memory-leaks, unreleased locks, non-terminated threads, shared-memory pool latching, storage fragmentation, or data corruption. This undesirable phenomenon has been observed in enterprise clusters [2,3], telecommunications systems [4,5], web servers [6] as well as other software. It is most likely to manifest itself in long-running or always-on applications such as web and applications servers, components of web services, and complex enterprise systems.

The primary method to fight aging is software rejuvenation proposed by Huang et al. [1]. The main idea behind software rejuvenation is to gracefully terminate an application and periodically or adaptively restart it at a clean internal state. Garbage collection, flushing operating system kernel tables, reinitializing internal data structures are some examples of what cleaning the internal state of a software might involve. An extreme, but well known example of rejuvenation is a hardware reboot. Rejuvenation has been implemented in various types of real life systems – telecommunication systems [4,5], transaction processing systems [7], web servers [6] and cluster servers [2, 3]. In this paper, we focus on the virtualized system, which is becoming more and more important as the emergence of cloud computing.

Virtualization technology is the key element in cloud computing[18,19], which is used for software infrastructure of cloud computing services to provide computing resources over the Internet. A virtual machine monitor (VMM) or hypervisor is a thin software layer that virtualizes machine resources to allow multiple virtual machines (VMs) to be multiplexed on a single physical machine and ensures VMs functionally isolated from one another. The use of virtualization to consolidate servers for enterprise applications is currently used widely as a solution to improve system availability [8-10,17]. However, both VMM and VMs in a virtualized system are subject to software failures during their continuous execution due to residual aging bugs. Especially for the VMM, it plays a critical role of a single-server virtualized system and often becomes the single point of failure. Thus, there is an urgent need to apply software rejuvenation to such systems to further improve system availability.

The main contribution of this paper is to propose new availability models for a single-server virtualized system and to discover novel results of comparing existing models. For a single-server virtualized system hosting two VMs in a passive or active replication mode, stochastic reward net (SRN) models of the virtualized system are developed by applying a combinatory rejuvenation technique that uses a time-based policy for a VMM and a prediction-based policy for VMs, which capture aging states of VMs and the VMM as well as their failures caused by software aging. And then we demonstrate that the availability of the virtualized system with an active replication mode is higher than the one with a passive replication mode, and analyze the combined effect of the rejuvenation interval (defined as the time between successive rejuvenations) in the time-based policy and the detection probability in the prediction-based policy on the steady state expected availability, downtime and transaction lost by numerical solutions. Further, comparing with the existing models for a single-server virtualized system, we discover some novel results.

The rest of paper is organized as follows. Section 2 presents some related work. Section 3 focuses on some motivations for this work. Section 4 presents a considered architecture for a single-server virtualized system with virtualization and software rejuvenation and provides availability models with different rejuvenation policies using SRN. To validate the technique, the analysis results are presented in Section 5. Finally Section 6 concludes the paper.

II. Related Work

There are too many works to improve system availability in the literature of software rejuvenation, which aim at determining an appropriate rejuvenation technique and rejuvenation schedules to minimize system downtime or maximize system availability. These works can be approximately divided into two categories: periodical rejuvenation based on time or work performed, and adaptive or proactive rejuvenation where the time to resource depletion or performance degradation is estimated. Countless studies have shown that the latter approach is more efficient, resulting in higher availability and lower cost. Our work also falls into the latter approach.

Among the methods to apply proactive software rejuvenation two are dominant: time-based approach [1-5,7], and the prediction-based (or measurement-based) approach [6,11]. The first method attempts to obtain an analytic model of a system taking into consideration various system parameters such as workload, MTTR and also distributions of failure. On this basis, an optimized rejuvenation schedule is obtained. The tools used here include CTMC models [1], MRGP models [3], SRN [2,4,5] and others [7,20,21]. In the prediction-based approach, the behavior of running software is monitored and the rejuvenation process is only triggered upon detection of any anomalies in the behavior of software. In our work, we integrate the analytic-based approach with the prediction-based approach in a model.

As the emergence of cloud computing, there are a few works focusing on software rejuvenation in a virtualized system. Combining virtualization and rejuvenation to achieve high availability is used by Machida et al.[8], Thein et al.[9,13] and Kourai et al.[10,14,15]. These works benefit from consolidation property of system-level virtualization.

Thein et al. [13] present a continuous-time Markov chain based analytical model to capture the behavior of the virtualized clustering system with software rejuvenation. They analyze system availability with the time-based rejuvenation policy under different cluster configurations, 2 VMs hosted on a single physical server and 2 VMs per a physical server in dual physical servers. Analysis results show that it is possible to benefit from increased availability by integrating virtualization, clustering and software rejuvenation. Based on the previous work [15], Thein et al. [9] further present a virtual machine based software rejuvenation framework named VMSR to offer high availability for application server systems. They again use a continuous-time Markov chain to model a single physical server hosting multiple virtual machines in the scheme of hot standby with the VMSR framework. Both works [9,13] only use the time-based rejuvenation policy for the VM failure without considering the VMM failure and its rejuvenation issues. However, VMM as the single point of failure of a consolidated system plays a critical role in improving system availability. Hence we propose a combinatory rejuvenation technique here for a single server virtualized systems considering the failures and rejuvenation of both VMs and the VMM.

Due to the critical role of VMM, a fast software rejuvenation technique for VMM named Warm-VM reboot was presented by Kourai et al. [14,15]. Compared with Cold-VM reboot that stops all the hosted VMs at the VMM rejuvenation, Warm-VM reboot improves the availability of the application hosted on VMs by introducing the on-memory suspend technique and the quick reload mechanism. However, their work only focuses on rejuvenating a parent component when the parent component is a VMM and the subcomponents are VMs. In this paper, we not only take the failures and rejuvenation of both VMs and the VMM into account but also adopt a different rejuvenation policy for them.

A comprehensive availability model for a server virtualized system with time-based rejuvenations for VM and VMM was presented by Machida et al.[8]. Their model focuses on the evaluation of the rejuvenation techniques for VMM including Warm-VM rejuvenation, Cold-VM rejuvenation and Migrate-VM rejuvenation. Further, they leverage existed availability models for time-based rejuvenation and apply them to a server virtualized system to determine the optimum rejuvenation schedules in terms of downtime. It is important to note that our approach is different from their works in terms of rejuvenation policy. They use the time-based rejuvenation policy for both VMs and VMM while we are proposing a new combinatory rejuvenation technique that uses a time-based policy for the VMM and a prediction-based policy for VMs. Moreover, our work discusses the impact of the configuration of VMs on rejuvenation process. For the VM level of a single-server virtualized system, both VMs can work in an active or passive software replication mode and provide similar services. Thus, we develop two SRN models for the cases of using the active and passive replication at the VM level respectively to analyze system performance from the aspects of transaction lost as well as availability and downtime. Moreover, we discover some novel results through comparing the existing models.

III. Motivation

In the following text, we focus on the motivation of using a combinatory policy and applying SRN to model system behaviors respectively.

In a single-server virtualized system, there are two critical components, the VMM and the VM hosted on the

VMM. Both of them may be failure because of aging-related bugs. Because of the VM hosting on the VMM, the VM can also be rejuvenated by the rejuvenation of its underlying VMM. But downtime cost due to the VMM rejuvenation is much more than downtime cost due to the VM rejuvenation. This motivates us to handle the VM rejuvenation as well as the VMM rejuvenation. Further, to rejuvenate any one of the VMM and the VM in a proactive way, we have two choices for rejuvenation policy, namely the time-based policy and the prediction-based one. For the former, we need only build an analytical model using assumptions about its operational profiles or failure distribution, to make the model as close as possible to a real life system taking no consideration of its real operation process. So in whatever situation, the time-based approach is easy to be used. But the approach is not precise to some extent. For the latter, we need monitor the behavior of running software, collect some resource-related or time-related data, validate the existence of aging, analyze system performance level and trigger the rejuvenation. The above actions can not be taken if absence of the detailed configuration of a running system and some embedded functionalities in the system. So the prediction-based approach is the sole way to apply the optimal rejuvenation schedule produced from the time-based approach to a real life system, which makes rejuvenation come true. Of course, comparing with the time-based approach, the prediction-based approach is relatively complex, but more precise. In our work, we have designed a component named Software Rejuvenation Agent (SRA) inside each VM to monitor consumable resources and states and carry out rejuvenation operations. The existence of SRA makes the adoption of a prediction-based policy to the VM possible. This motivates us to use a combinatory rejuvenation policy, namely the prediction-based policy at the VM level and the time-based policy at the VMM level respectively.

And then we focus on the motivation of adopting SRN to model system behaviors dynamics. Models are system abstractions in one or more specific aspects, and provide the basis for the analysis of system behavior. There are several model types that have been used for modeling the performability of complex systems. However, the most common used model is the state-space based model, which can capture the complicated system dependencies. This kind of model demands the collection of system variables, the values of which define the system state at a given point. The state-space based model can further be divided into two sub-types: the low-level model and the high-level one.

The low-level model usually uses a stochastic process to characterize system behavior dynamics since there are significant uncertainties and unpredictable variations either inherent in the system or in its inputs. Such uncertainties can be taken into consideration via stochastic modeling and solution techniques. Several stochastic models, such as CTMC (Continuous time Markov chain)[1], SMP (Semi-Markov process)[20], MRGP (Markov Regenerative Process)[3] and MRM (Markov reward model)[21] and so on, has been widely used in the field of software rejuvenation modeling. To achieve the solutions of these models to analyze system performance, we usually make the resulting process a homogeneous CTMC by some converting techniques regardless of the original type of processes. For example, in order to solve the MRGP, we choose to approximate the deterministic transitions in MRGP using an r-stage Erlang distribution, so the resulting process becomes a homogeneous CTMC and the solution techniques for Markov chains can be applied. But, this approach has a significant shortcoming that the state space of the CTMC increases by r due to the approximation. And if we wish to model a single-server virtualized system with n standby VMs where each VMs has less than six states (more details in section 4.1), the overall state space will become unmanageable if we were to build the CTMC by hand. Thus, in order to avoid the manual construction of a high fidelity model we may resort to the following higher level model.

The high-level model usually has not only a rigorous mathematical basis, but also a powerful graphical representation. With a rigorous mathematical basis, they can be automatically transformed into a certain stochastic process, and then use the same way as the low-level model to analyze system performance. With the powerful graphical representation, they allow the designer to focus more on the system characteristics being modeled rather than on the error-prone specification of system state space. From this perspective, the modeling power of the high-level model is remarkably greater than the low-level one. Due to the graphic nature of the high-level model, the changes to models are done easily, while minor changes in a Markov chain require redefining all the states in the model. This motivates us to use the high-level model in our work.

The widely used high-level models in the field of software rejuvenation modeling include SPN(Stochastic Petri Net), GSPN(General Stochastic Petri Net)[22,23], DSPN(Deterministic and Stochastic Petri Net)[3,7] and SRN(Stochastic Reward Net)[2,4,5,8,16], etc. SPN only allows exponentially distributed firing times for transitions. GSPN extend the SPN by allowing zero firing time for some transitions, in which transitions with exponentially distributed firing times are called timed transitions while the transitions with zero firing times are called immediate transitions. For SPN and GSPN, they contain only immediate transitions and/or timed transitions with exponentially distributed firing time, so the underlying stochastic process is a homogeneous CTMC. However, we design a transition with deterministic distributed firing time for VMM rejuvenation clock in our models. Obviously, SPN and GSPN can be applicable no longer. As for DSPN, it requires all transitions are deterministic. However, our models contain not only deterministic transitions, but also exponentially distributed firing time transitions. Obviously, DSPN can also be inapplicable. As our choice in this paper, SRN is a variant of GSPN with a more powerful modeling power by several extensions. The first extension is its ability to allow extensive marking dependency, which can specify parameters, such as the firing rate of the timed transitions, the multiplicities of input/output arcs and the reward rate in a marking, as functions of the number of tokens in any place in the SRN. In our models, the output measures are expressed in terms of the expected values of the reward rate functions. Another important extension is the ability to express complex enabling/disabling conditions through guard functions. This can greatly simplify the graphical representations of complex systems. In our models, almost each immediate transition is assigned with a guard function. Moreover, as same as the other high level models, an SRN can be automatically transformed into a Markov reward model (MRM), steady-state and/or transient analysis of the MRM by the tool SPNP [24] solves the SRN models. This motivates us apply the tool SRN to model the process of software aging and software rejuvenation in this paper.

IV. Availability Models

The benefits of software replicas created by virtualization technology are to reduce service interruption and optimize the rejuvenation process without any additional physical nodes. The proactive software rejuvenation with virtualization is just based on this principle. In order to study the effects of virtualization and software rejuvenation on availability for a single-server virtualized system, we provide two high level availability models using SRN to depict the process of software aging and software rejuvenation. Further, we analyze these models in terms of system availability, downtime and the number of transaction lost by sensitivity analysis.

A. System Architecture

The system architecture of a single-server virtualized system is shown in Fig.1. The single-server virtualized system consists of a hosting server installing a VMM and three VMs running in a dispatcher-worker mode on top of this VMM.

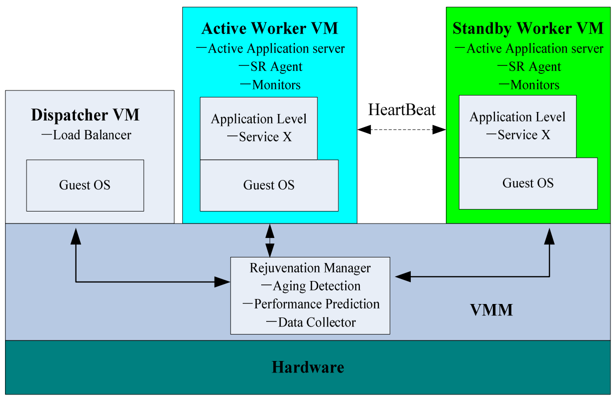


Figure 1. The considered architecture for a single-server virtualized system.

Dispatcher VM will be used for providing fail-over capabilities and a myriad of load balancing policies. The remaining VMs are in operation and deploy the same services, named Worker VM. Our idea is to hold multiple replicas of an application in a single physical server by virtualization technology, in which one replica is designated as primary and all others are designated as standbys (or backups). If the primary fails, one of the backups takes over as the new primary. This approach uses VMs as containers for the replica in order to avoid the need for additional hardware and it can provide continuous services during rejuvenation. In this paper, we create two Worker VMs. The primary application server and the standby one will be running on the active Worker VM and the standby Worker VM respectively. The VMs inform each other about their health using a heartbeat mechanism. We also designs a component named Software Rejuvenation Agent (SRA) inside each VM to monitor consumable resources and states, and another component named Rejuvenation Manager (RM) inside the VMM to analyze VMs' behavior, detect their anomalies and trigger the rejuvenation of a VM when it detects anomalies in the behavior of that VM. The existence of SRA and RM makes a prediction-based policy for VM possible. Similarly, due to absence of the above components, the time-based rejuvenation policy is the only one applicable for the VMM.

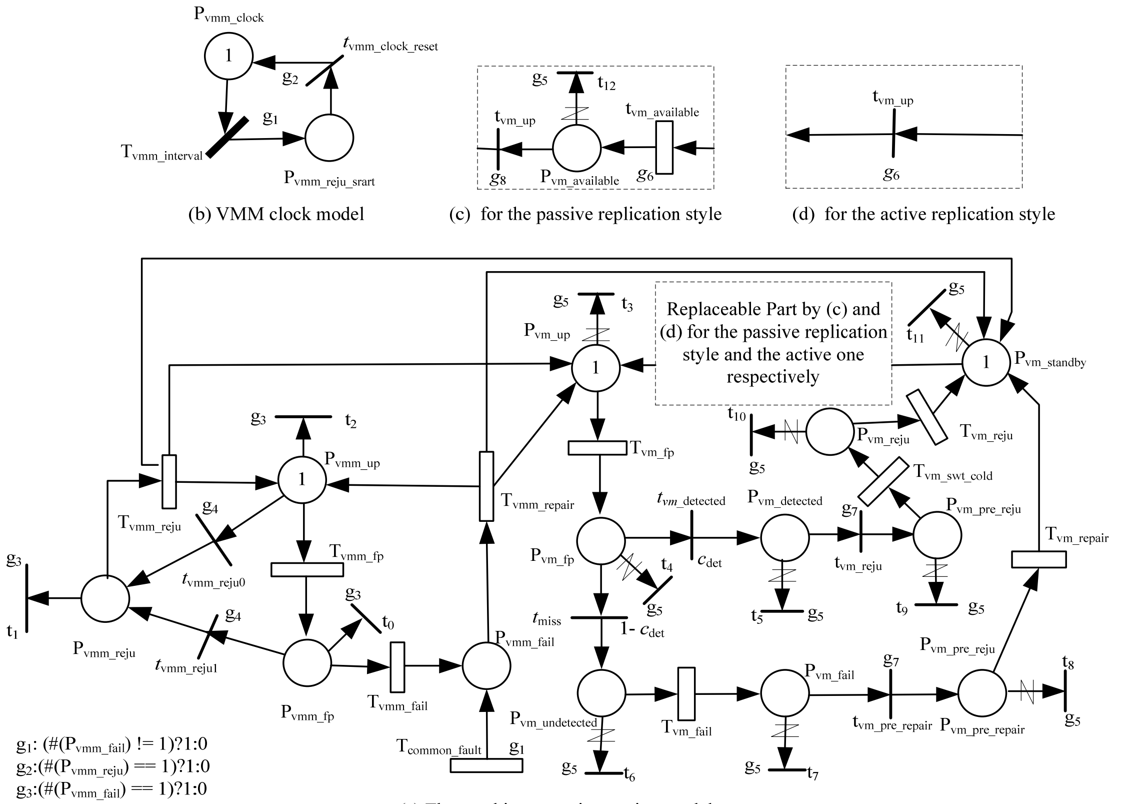
B. SRN Models

For a VMM rejuvenation, we use the simplest rejuvenation approach that shuts down all the hosted VMs prior to the rejuvenation regardless of the execution states of the VMs. The greatest strength of this approach is to clear all of aging states in VMs and VMM by a VMM rejuvenation. Of course, this weaknesses of this approach are also obvious. Because of the dependency between VMs and VMM, the execution of VMs running on the VMM may also be interrupted, which results in transactions lost and unnecessary downtime of the VMs.

For a VM rejuvenation, because of the replication mode used at the VM level in our considered architecture, there are also two different styles, a passive replication style and an active one. The former has no backup replicas in memory. Upon primary replica failure, a backup replica is loaded into memory and assumes the role of the new primary replica. The new primary replica's state is initialized from the last checkpoint to ensure its state is identical to the state of the old primary replica before its failure. The latter, in contrast, has all backup replicas created, initialized, and loaded in memory. The primary replica state is transferred to all backup replicas at the end of every operation on the primary replica. When the primary replica fails, a new primary replica is chosen from the backup replicas. Intuitively, application of the passive replication style will result in lower VM availability, longer downtime and higher the number of transaction lost. Two rejuvenation models using these two policies respectively will be built in this section, and this assumption will be verified through experiments in section 4.

In our proposed models, a Petri Net (PN) is a bipartite directed graph whose nodes are divided into two disjoint sets - places and transitions. Input arcs are directed arcs which connect places to transitions and output arcs are directed arcs which connect transitions to places. A cardinality may be associated with these arcs. An arc with multiplicity *m* is represented by an '/m' on the arc. The distribution of tokens on the places of the PN is called a marking of the PN. When representing a PN graphically, places are represented by circles and immediate transitions, timed transitions and deterministic transitions are shown by narrow bars, hollow rectangles and filled rectangles, respectively. The number of tokens *n* in place *P* is represented graphically as the number *n* inside the circle for place *P*. Variable cardinality of an arc is represented by adding a *Z* on the arc. Each transition can also be associated with a guard function, which is usually a function of a marking. The transition is enabled only when the guard function is evaluated to TRUE and all other conditions relating to priorities and input arc conditions are met. The reader is referred to [25] for a detailed description of SRNs.

The SRN model with a combinatory rejuvenation policy is shown in Fig. 2. The model is represented by four SRNs, where Fig.2(a) is the SRN using the prediction-based policy at the VM level and the time-based policy at the VMM level, and Fig. 2(b) is the SRN for the VMM clock model, which is used for triggering time-based rejuvenation of VMM. Two different replication styles for the standby VMs are modeled by the Fig. 2(c) and Fig. 2(d) respectively.



g_1: $(\#(P_{vmm\_fail}) != 1)?1:0$
g_2: $(\#(P_{vmm\_reju}) == 1)?1:0$
g_3: $(\#(P_{vmm\_fail}) == 1)?1:0$
g_4: $(\#(P_{vmm\_reju\_start}) == 1)?1:0$
g_5: $(\#(P_{vmm\_fail}) == 1 \,||\, \#(P_{vmm\_reju}) == 1)?1:0$
g_6: $(\#(P_{vm\_detected}) == 1 \,||\, \#(P_{vm\_fail}) == 1)?1:0$
g_7: $(\#(P_{vm\_standby})==1\&\&\#(P_{vmm\_fail})!=1\&\&\#(P_{vmm\_reju})!=1)?1:0$
g_8: $(\#(P_{vm\_reju}) == 1 \,||\, \#(P_{vm\_pre\_repair}) == 1)?1:0$

(e) guard functions

Figure 2. SRN models with different replication styles at the VM level

Combining part (a), part (b) and part (c) of Fig. 2, the integrated SRN model represents the failure process, recovery process, and combinatory rejuvenation of VMM and VMs with the passive replication style. Initially there is one token in the $P_{\mathrm{vm\_up}}$, $P_{\mathrm{vm\_standby}}$ and $P_{\mathrm{vmm\_up}}$ respectively, which demonstrates fully stable states of VMs and VMM. As time passes, each active VM eventually transits to an unstable state (place $P_{\mathrm{vm\_fp}}$) through the transition $T_{\mathrm{vm\_fp}}$ representing the software aging of the VM. The active VM still works in this state but its failure chances increase. It is assumed that the RM can detect the aging with probability c_{det}. The $P_{\mathrm{vm\_fp}}$ place has two immediate transitions with the appropriate probability for detecting aging or failures during detection process. If the aging is detected, a token is fired through $t_{\mathrm{vm\_detected}}$ and will be deposited in $P_{\mathrm{vm\_detected}}$. If the detection process fails, a token is put in $P_{\mathrm{vm\_undetected}}$. As time progresses, the active VM eventually transits to a failure state (place $P_{\mathrm{vm\_fail}}$) through the transition $T_{\mathrm{vm\_fail}}$. Whenever a token is deposited in $P_{\mathrm{vm\_detected}}$ or $P_{\mathrm{vm\_fail}}$, the active VM suspends all user requests and provides no longer services.

The recovery process consists of the standby VM loaded in memory, the standby VM's internal state catch up and the switching of the current active VM with the standby VM. The transition $P_{\mathrm{vm\_standby}} \rightarrow T_{\mathrm{vm\_available}} \rightarrow P_{\mathrm{vm\_available}}$ in the SRN model represents the standby start time required to perform the first step. On a token in place $P_{\mathrm{vm\_available}}$, the standby VM is available. The latter two steps of this rejuvenation process are performed by using the timed transitions $T_{\mathrm{vm\_swt\_cold}}$. The rejuvenation and repair of the active VM can begin when the standby VM is in an available state (place $P_{\mathrm{vm\_available})}$ and the underlying VMM is neither in the failure state nor in the rejuvenation state. The guard function g_7 ensures this condition. In passive replication mode, the standby VM's internal state is initialized from the last checkpoint to ensure its state is identical to the state of the active VM before the failure. Hence, the time for the final catch up step mainly includes the time to transfer all pending requests and current sessions from the active VM to the standby one. After the switch is completed, a token is deposited in place $P_{\mathrm{vm\_reju}}$ to mark the completion of rejuvenation. When the active VM is completely rejuvenated, it is placed back in service as the new standby VM. The process can repeat continuously. Because of the dependency of VMs and VMM, all tokens representing VMs' operational nodes (i.e. tokens in places $P_{\mathrm{vm\_up}}$, $P_{\mathrm{vm\_fp}}$, $P_{\mathrm{vm\_fail}}$, $P_{\mathrm{vm\_pre\_repair}}$, $P_{\mathrm{vm\_reju}}$, $P_{\mathrm{vm\_pre\_reju}}$, $P_{\mathrm{vm\_detect}}$, $P_{\mathrm{vm\_undetected}}$) are removed when the system is considered failed (i.e. token in place $P_{\mathrm{vmm\_fail}}$ or $P_{\mathrm{vmm\_reju}}$). This is accomplished by the guard g_5, which enables the immediate transitions t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{13}, t_{12}, when it detects a token in place $P_{\mathrm{vmm\_fail}}$ or $P_{\mathrm{vmm\_reju}}$. In this case, it is inevitable to involve unnecessary downtime and lose transactions.

Similarly, when the transition $T_{\mathrm{vmm\_fp}}$ fires, a token is deposited in $P_{\mathrm{vmm\_fp}}$. If the transition $T_{\mathrm{vmm\_fail}}$ fires, a token is deposited in $P_{\mathrm{vmm\_fail}}$ which represents the VMM failure due to the software aging. The effects of common mode faults are applied using $T_{\mathrm{common\_fault}}$. When this transition is fired, VMM failure also occurs. When the VMM is recovered from the failure state, the token in $P_{\mathrm{vmm\_fail}}$ is removed and a token is deposited in $P_{\mathrm{vmm\_up}}$, $P_{\mathrm{vm\_up}}$ and $P_{\mathrm{vm\_standby}}$ respectively by $T_{\mathrm{vmm\_repair}}$ transition. The time-based periodic rejuvenation is driven by a deterministic clock shown in Fig.2(b). When the deterministic transition $T_{\mathrm{vmm\_interval}}$ fires and deposits a token in place $P_{\mathrm{vmm\_reju\_start}}$ each d time units, the VMM rejuvenation is triggered and the immediate transitions $t_{\mathrm{vmm\_reju0}}$ and $t_{\mathrm{vmm\_reju1}}$ are enabled by guarding function g_4 and a token is deposited in $P_{\mathrm{vmm\_reju}}$. Similarly, when the VMM rejuvenation cleans up the aging states, the token in $P_{\mathrm{vmm\_reju}}$ is removed and a token is deposited in $P_{\mathrm{vmm\_up}}$, $P_{\mathrm{vm\_up}}$ and $P_{\mathrm{vm\_standby}}$ respectively by $T_{\mathrm{vmm\_reju}}$ transition. Whenever in case of VMM rejuvenation or VMM repair, both leads to rejuvenation of the whole system and those VM related places flushed out because of VMs running on the VMM.

Combining part (a), part (b) and part (d) of Fig. 2, the new integrated SRN model represents the failure process, recovery process, and combinatory rejuvenation of VMM and VMs with the active replication style. The most noteworthy difference is that there is no need to take some time to make the standby VM activated. Hence we use the immediate transition t_{vm_up} to represent this step instead of the timed transition $T_{vm\_available}$ in Fig. 2(c). The rejuvenation process in practice only consists of standby replica's internal state catch up and the switching of the current active replica with the standby replica, which can be represented by timed transition $T_{vm\_swt\_hot}$. Moreover, the standby VM's internal state is continuously updated with the active VM internal state changes. Hence, the total time including the time for the final catch up step and the replica switch time in active replication style is shorter than the total one in passive replication style. After the switch is completed, a token is deposited in place $P_{\mathrm{vm\_reju}}$ to mark the completion of rejuvenation.

The guard functions used by these two models are summarized in Fig. 2(e), where the guard function g_8 is only suitable for the model with the passive replication style.

V. Analysis and Results

We used the stochastic Petri net package (SPNP) tool [24] to have an evaluation of two rejuvenation models applied to the target system. We assumed that all transition times of timed transitions in the models are exponentially distributed except for $T_{\mathrm{vmm\_interval}}$ which is deterministic because it represents the fixed rejuvenation trigger intervals of VMM. There are many previous studies [3-5,8-9,13,16] supporting the use of exponential distributions. For examples, in a CTMS (Cable Modem Termination System) cluster system, Yun Liu. [4,5] have assumed that the distribution of time between hardware failures and software failures caused by Heisenbugs are exponential, the distribution of time between software failures caused by aging-related faults is hypo-exponential, and the distribution of failure detection time and node switchover/reboot/rejuvenation/giveback time are exponential. Of course, the assumption of the exponential distribution is not always well-suited for all real-world applications. For examples, in Salfner [7], it has been concluded from data of a commercial telecommunication platform that the distribution of time-to-failure can be approximated best by a lognormal distribution. In our experiments, the assumption of the exponential distribution makes the modeling and computation easier. However, our models are not restricted to exponential distributions; other well-known distributions such as lognormal and Weibull could be used as well. The model parameters need be varied with different distributions, while the model solutions can be left to the tool SPNP.

Because of the existence of the deterministic transition, we used 10-stage Erlang distribution for approximating it. The model parameter values used were based on prior investigations [8,16] of software aging related failures and the application of the software rejuvenation as the solution, as shown in Table 1. For an example of $\lambda_{\mathrm{vm\_fp}}$, the calculation of the default value is as follows. We assume that the mean duration time observed from the robust state to the unstable state is one week, namely 7 × 24 hours. Thus, the VM aging rate is 1/(7 × 24)=0.005952381, which means the number of aging appearance is 005952381 per hour.

TABLE I.
PARAMETERS USED IN ALL MODELS

Parameter		values	
Symbol	Description	Default Value	Mean Time
λ_{vm_fp}	VM aging rate	0.005952381	1 week
λ_{vm_fail}	VM failure rate after aging	0.013888889	3 days
$\lambda_{vm\_recovery}$	VM failure recovery rate	2	30 mins
λ_{vm_reju}	VM rejuvenation rate	60	1 min
$\lambda_{vm\_available}$	Standby VM activation rate in cold-standby	60	1 min
$\lambda_{vm\_swt\_cold}$	VM state transition rate in cold-standby	120	30 secs
$\lambda_{vm\_swt\_hot}$	VM state transition rate in hot-standby	1200	3 secs
λ_{vmm_fp}	VMM aging rate	0.001388889	1 month
λ_{vmm_fail}	VMM failure rate after aging	0.005952381	1 week
$\lambda_{vmm\_recovery}$	VMM failure recovery rate	1	1 hour
λ_{vmm_reju}	VMM rejuvenation rate	30	2 mins
$\lambda_{vmm\_reju\_inter}$	VMM rejuvenation trigger rate	--	--
c_{det}	VM aging detection probability	--	--

For the simplicity of description, we call the model using the combinatory rejuvenation policy with the passive replication style “Model A”. Similarly, we call the model using the active replication style “Model B”.

A. Model A VS. Model B

In this section, we compare these two models only from the aspect of system steady-state availability regardless of downtime and transaction lost. The main causes are as follows. For downtime, the expected total downtime over time interval T is calculated as $T*(1-P_{\text{avail}})$, where P_{avail} is system steady-state availability probability. Thus, it is easy to find that the variation tendency of the expected total downtime is opposite to that of steady-state availability. For transaction lost, when the active VM is rejuvenated or repaired, there are no transactions lost because of the current transactions transferred to the standby VM and the subsequent user requests queued by the Dispatcher VM. Thus, the replication style using by both models should be no impact on whether transactions lose or not.

Availability models capture the failure, repair and rejuvenation behavior of systems and their components. In the above two models, the impact of critical parameters, such as the rejuvenation trigger intervals of VMM and aging detection probabilities of VM, on steady-state availability was studied. Service is available when VMM is in the robust state or the failure probable state and the active VM pertains to one of states: the robust state, the failure probable state and the failure detection missing state. Fig.3 shows the results of steady-state availability by varying the VMM rejuvenation trigger interval and the VM aging detection probabilities independently.

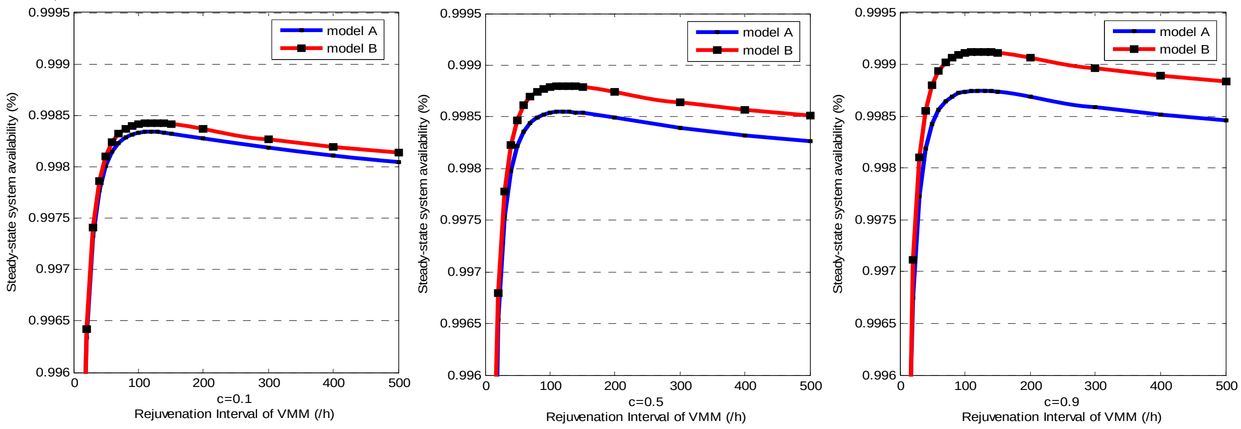


Figure 3. Steady-state availability for both models by varying the rejuvenation interval of VMM and aging detection probability of VM

In the first case, the VM aging detection probability is fixed at 0.1, 0.5 and 0.9 respectively. For an example of the aging detection probability c=0.5, when varying the VMM rejuvenation interval from 5 hours to about 100 hours, we can find that the larger rejuvenation interval lead to the higher system steady-state availability. At a certain rejuvenation interval, in this case $\lambda_{vmm\_reju\_inter} \approx 100$, system steady-state availability is maximized. System availability appears to drop very slowly with the rejuvenation interval greater than 100 hours. We may directly see that there is such a variation tendency for both models. This is caused by the fact that more frequent rejuvenation increases the availability due to the rejuvenation and less frequent rejuvenation also increases the availability caused by software failure. Further, we see that system steady-state availability of model B is almost better than that of model A if the rejuvenation interval increases over 20 hours. This result is caused by the fact that the time to rejuvenation for the passive replication style is longer than that of the active one because of no time needed for the standby VM loaded in memory with the active replication style.

In the second case, when the VMM rejuvenation interval is fixed, we can find that the higher aging detection probability is better than the lower one in terms of system steady-state availability. This result shows that it is very important to make the aging detection probability more close to 1 in the prediction-based rejuvenation policy. In fact, it is also very difficult and impossible to do this. So in the following discussions, we assume that the highest detection probability is 0.9.

From the above discussions, we can draw a conclusion that model B using the active replication style is better than model A using the passive one in terms of steady-state system availability.

B. *Sensitivity Analysis*

For an example of Model B, we focus on investigating the impact of two critical parameters, aging detection probability and the VMM rejuvenation interval, on the downtime in this section. The expected total downtime consists of the downtime due to software rejuvenation and the downtime due to system failure. Hence, we also analyze the two downtime components that contribute to the expected total downtime here. In case of software rejuvenation or software failure, the expected downtime over time interval T is calculated as $T * P_{unavail}$, where $P_{unavail}$ is system steady-state unavailability probability due to software rejuvenation or software failure, T is time interval and its value is set to 1,440 minutes.

In the first case of the VMM rejuvenation interval is set to 120 hours, execution results of model B by varying values of the aging detection probability c_{det} are shown in Fig. 4. As the probability c_{det} increases, the expected downtime due to software rejuvenation only slightly increases, however the expected downtime due to software failure and the expected total downtime markedly decrease. Further, we find that the expected downtime of software failure is always greater than that of software rejuvenation regardless of values of c_{det}. Hence, we can draw a conclusion that at a given rejuvenation interval, the recovery process is the main contributor to the expected downtime and the higher the aging detection probability leads to the lower the expected total downtime.

In the second case of the aging detection probability c_{det} is set to 0.9, execution results of model B by varying the VMM rejuvenation interval are shown in Fig. 5. As the rejuvenation interval increases, the downtime due to software rejuvenation firstly rapidly drops and then slightly increases, meanwhile the downtime due to software failure firstly slightly increases and then slightly drops. The variation tendency of the expected total downtime is almost the same as that of the downtime due to rejuvenation. When the rejuvenation interval is 80 hours or less, the rejuvenation process is the main contributor to the expected total downtime with 51.1% to 99.9% of the expected total downtime. Especially when the rejuvenation interval are less than 30 hours, the rejuvenation process contributes to the expected total downtime with 91.7% to 99.9% of the expected downtime. On the other hand, when the rejuvenation interval is more than 80 hours, the failed system with 52.9% to 80.7% of the expected downtime is the greater contributor. This finding is inline with the premise that the use of proactive rejuvenation reduces the chance of system failures caused by software aging. The above results are caused by the fact that more frequent rejuvenation increases the downtime due to the rejuvenation and less frequent rejuvenation also increases the downtime caused by software failure. By finding the point which minimizes the expected downtime, we can solve the optimal rejuvenation interval.

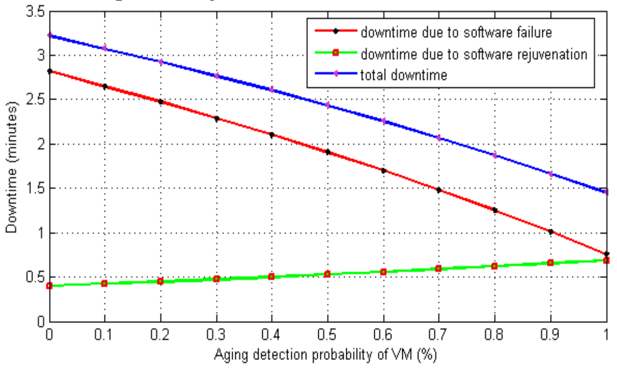


Figure 4. The expected downtime due to software rejuvenation and failure with different aging detection probabilities of VM

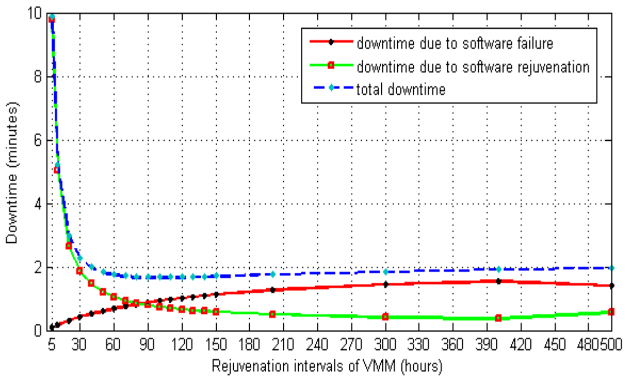


Figure 5. The expected downtime due to software rejuvenation and failure with different rejuvenation intervals of VMM

As analyzed before, the aging detection probability should be no impact on whether transactions lose or not. Hence, we focus on the impact of the VMM rejuvenation interval on the number of transactions lost. When VMM is rejuvenated or repaired, transactions lose because of VMM shutting down all the hosted VMs prior to the rejuvenation or repair regardless of the execution states of the VMs. The expected number of transactions lost due to VMM rejuvenation and VMM repair can be computed from the throughputs of the transitions T_{vmm_reju} and T_{vmm_repair} for each SRN model. We will execute model B again to analyze the expected number of transactions lost with different VMM rejuvenation intervals. Results with varying rejuvenation intervals and the fixed value of 0.9 for detection probability c_{det} are shown in Fig. 6.

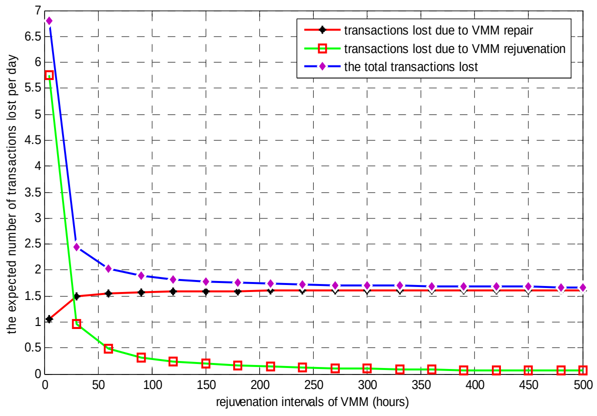


Figure 6. The expected number of transactions lost with different rejuvenation intervals

As seen from Fig.6, the expected number of transactions lost due to VMM rejuvenation is monotone decreasing with the increasing rejuvenation interval. The higher rejuvenation rate (i.e. the rejuvenation interval is less than 30 hours) has a significant impact on the expected number of transactions lost. However, when the rejuvenation interval increases to more than 30 hours, VMM repair instead of VMM rejuvenation is the new greater contributor to the expected number of transactions lost. Further, when the rejuvenation interval increases from 80 to 500, the expected number of transactions lost due to rejuvenation gradually levels off at about 0.159. When the rejuvenation interval increases to infinity, this means that VMM has no rejuvenation operation. The expected number of transactions lost will be equal to the number of transactions lost associated with VMM repair. Form the above analysis, we can find that the rejuvenation interval has almost no impact on the expected number of transactions lost only if it is greater than a proper value got from the solution of SRN model.

C. *Model B VS. Other Models*

To further validate the effect of our models and rejuvenation policies, we consider the existing models with different policies including no rejuvenation, just the VMs rejuvenation using the prediction-based rejuvenation policy, just the VMM rejuvenation using the time-based rejuvenation policy, and both the VMM and the VMs rejuvenation with the time-based policy. Further, we compare the effect of these policies with model B, and the results are shown in Fig. 7, where the aging detection probability of the VMs used in the prediction-based policy is fixed at 0.9 and the VM interval used in the time-based policy is 120 hours.

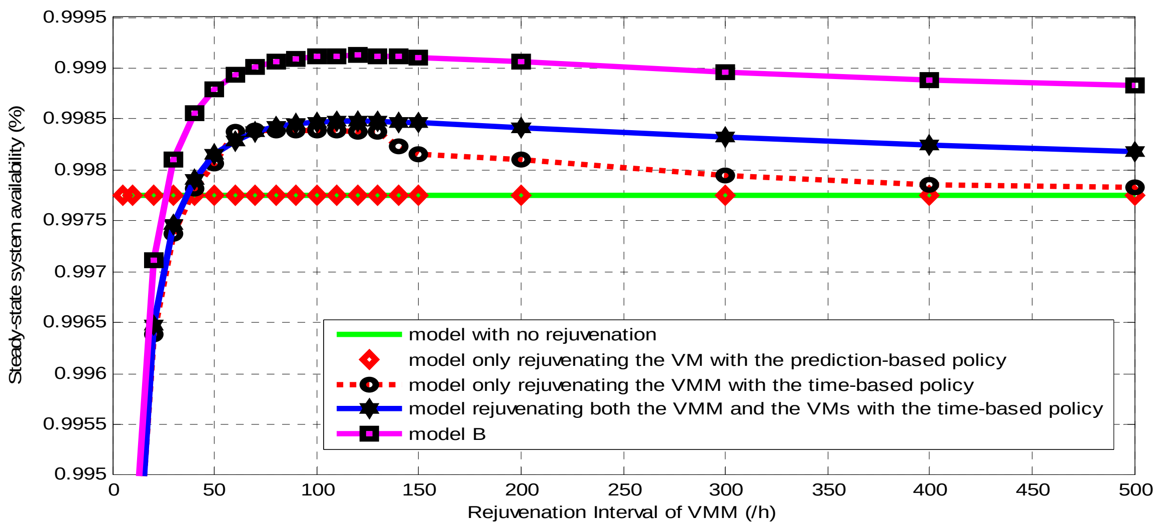


Figure 7. Comparison of rejuvenation effects using different rejuvenation policies

From Fig. 7, the findings we got are as followings:

- There is no system availability improvement in case of just rejuvenating the VMs by comparison to the case of no rejuvenation. However, there is an obvious system availability improvement in case of just rejuvenating the VMM by comparison to the case of no rejuvenation. The result shows that due to the availability of the upper-level VMs depending on the availability of the lower-level VMM, the extreme limit of system availability depends on the maximum VMM availability. Thus, rejuvenating the VMM is the most efficient way of improving system availability.
- There is an obvious system availability improvement in case of rejuvenating both the VMM and the VMs regardless of rejuvenation policies used by comparison to the case of just rejuvenating the VMM. The results show that the further system availability improvement depends on simultaneously improving the availability of both of the VMM and the VMs. Further, comparing our policy with the time-based policy applied to both the VMM and the VMs, we find that our policy is much better than the competitor.

VI. Conclusions

We have presented and analyzed comprehensive availability models for a class of system with deployment of virtualization technology and software rejuvenation. Results of the first experiment showed two models were with the same variation tendency and the model with the active replication style is obviously better than that of the model with the passive one in terms of system steady-state availability. Further, for an example of model B, we finished two other experiments in order to analyze system rejuvenation and repair process by varying two critical parameters, the VM aging detection probability and the VMM rejuvenation interval. Experimental results showed that the aging detection probability had a very important impact on the expected downtime and the higher the aging detection probability led to the lower the expected total downtime. However, we also showed the aging detection probability had no impact on whether transactions lost or not. Under a given parameter value for the aging detection probability, we observed that more frequent rejuvenation increases the downtime due to the rejuvenation and less frequent rejuvenation also increases the downtime caused by software failure. By finding the point which minimizes the expected downtime, we could solve the optimal rejuvenation interval. Moreover, we also observed that a proper VMM rejuvenation interval has almost no impact on the number of transactions lost and the repair process is the main contributor to the expected number of transactions lost.

In the future work, we will discuss some methods to carefully determine the optimal combinatory of the VMM rejuvenation interval and the VM aging detection probability so as to achieve higher steady-state availability and lower downtime.

Acknowledgement

This work is funded by a grant from the Natural Science Foundation of Jiangsu Province, China under grant No. BK2011023.

Reference

[1] Y. Huang, C. Kintala, N. Kolettis, and N.Fulton, “Software Rejuvenation: Analysis, Module and Applications,” *Proc. Int’l Symp. Fault-Tolerant Computing*, pp. 381-391,1995.

[2] K. Vaidyanathan, R.E. Harper, S.W. Hunter, K.S. Trivedi, “Analysis and implementation of software rejuvenation in cluster systems,” *ACM SIGMETRICS Performance Evaluation Review, Joint International Conference on Measurement and Modeling of Computing Systems*, pp. 62-71, 2001.

[3] Wang,D.Z., Xie, W, Trivedi, K.S., “Performability analysis of clustered systems with rejuvenation under varying workload,” *Performance Evaluation*, vol. 64, pp.247-265,2007.

[4] Y. Liu, K. Trivedi, Y. Ma, J. Han, and H. Levendel, “Modeling and analysis of software rejuvenation in cable modem termination systems,” *The 13th International Symposium on Software Reliability Engeineering (ISSRE)*, pp.159-172, 2002.

[5] Y. Liu, K. Trivedi, Y. Ma, J. Han, and H. Levendel, “A proactive approach towards always-on availability in broadband cable networks,” *Computer Communications*, vol. 28, pp.51-64,2005.

[6] L. Li, K.Vaidyanathan, and K.S. Trivedi, “An Approach to Estimation of Software Aging in a Web Server,” *Proc. Int’l Symp. Empirical Software Eng. (ISESE 2002)*, pp.45-52, 2002.

[7] F. Salfner, K. Wolter, “Analysis of service availability for time-triggered rejuvenation policies,” *Journal of Systems and Software*, vol. 83, pp.1579-1590, 2010.

[8] F. Machida, D. Kim, and K. Trivedi, “Modeling and Analysis of Software Rejuvenation in a Server Virtualized System,” *Proc. of 2nd Workshop on Software Aging and Rejuvenation (WoSAR2010)*, pp.1-6, 2010.

[9] Thein T, Park J S, “Availability Analysis of Application Servers Using Software Rejuvenation and Virtualization,” *Journal of Computer Science and Technology*, vol. 24, no.2, pp. 339-346, 2009.

[10] K. Kourai, “Fast and Correct Performance Recovery of Operating Systems Using a Virtual Machine Monitor,” *Proc. of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, vol.46, no.7, pp. 99-110,2011.

[11] Y. Bao, X. Sun, and K. S. Trivedi, “A Workload-Based Analysis of Software Aging, and Rejuvenation,” *IEEE Transactions on Reliability*, vol.54, no. 3, pp. 541-548, 2005.

[12] K. Trivedi, K. Vaidyanathan, K. Goseva-Popstojanova, “Modeling and Analysis of Software Aging and Rejuvenation,” *Proc. of 33rd Annual Simulation Symposium*, IEEE Computer Society, pp. 9-24,2000.

[13] T. Thein, S. Chi and J. Park, “Availability Modeling and Analysis on Virtualized Clustering with Rejuvenation,” *Int’l Journal of Computer Science and Network Security*, vol.8, no. 9, pp.72-80, 2008.

[14] K. Kourai and S. Chiba, “A fast rejuvenation technique for server consolidation with virtual machines,” *Proc. Int’l Conf. on Dependable Systems and Networks (DSN 2007)*, pp. 245-255, 2007.

[15] K. Kourai and S. Chiba, “Fast software rejuvenation of virtual machine monitor,” *IEEE Trans. on Dependable and Secure Computing*, vol.8,no.6,pp.839-851,2011.

[16] A. Rezaei and M. Sharifi, “Rejuvenation high available virtualized systems,” *Proc. of 5th International Conference on Availability, Reliability and Security (ARES2010)*, pp. 289-294, 2010.

[17] Yi Yang, Lichao Wang, Rui Kang, “Discretization model of instantaneous availability for continuous-time systems”, *Journal of Computers*, vol.7, no.4,pp.977-981,2012.

[18] Yanyan Hu, Xiang Long, Jiong Zhang, “I/O Behavior Characterizing and Predicting of Virtualization Workloads”, *Journal of Computers*, vol.7, no.7, pp. 1712-1725,2012.

[19] Yunpeng Xiao, Guangxia Xu, Yanbing Liu, Bai Wang, “A Metadata-driven Cloud Computing Application Virtualization Model,” *Journal of Computers*, vol.8, no.6, pp. 1571-1579,2013.

[20] Okamura,H., Dohi, T., “Comprehensive evaluation of aperiodic checkpointing and rejuvenation schemes in operational software system,” *Journal of Systems and Software*, vol. 83, pp. 1591-1604,2010.

[21] K. Vaidyanathan, K. S. Trivedi, “A Comprehensive model for software Rejuvenation,” *IEEE Transaction on Dependable and Secure Computing*, vol.2, no.2, pp. 124-137, 2005.

[22] M.M. Hosseini, R.M. Kerr and R.B. Randall, “An

Inspection Model with Minimal and Major Maintenance for a system with Deterioration and Poisson Failures," *IEEE Trans. Reliability*, vol. 49, no.1, pp.88-98, March 2000.

[23] Letian Jiang, Guozhi Xu, "Modeling and analysis of software aging and software failure," *Journal of Systems and Software*, vol.80,no.4, pp. 590-595, 2007.

[24] C. Hirel, B. Tuffin, K. Trivedi, "SPNP: Stochastic Petri Nets. Version 6.0," *Proc. of the 11th International Conference on Computer Performance Evaluation: Modeling Techniques and Tools*, pp. 354-357, 2000.

[25] J. K. Muppala, G. Ciardo and K. S. Trivedi, "Stochastic Reward Nets for Reliability Prediction," *Communications in Reliability, Maintenability and Serviceability*, pp. 9-20, July 1994.

Jian Xu received a Ph.D. in Computer Science in 2007 from Nanjing University of Science and Technology, Nanjing, China. Now he holds the position of an associate professor at Nanjing University of Science and Technology. His research interests are software engineering, particularly monitoring and rejuvenation of smoothly degrading systems, and he has published about 20 papers in journals and refereed conference proceedings in those areas. He is a member of ACM and IEEE.

Xuefeng Li is currently a master candidate of Nanjing University of Science and Technology, Nanjing, China. His major research interests are software rejuvenation and virtualization.

Yi Zhong is currently a Ph.D. candidate of Nanjing University of Science and Technology, Nanjing, China. Her major research interests are software rejuvenation and virtualization.

Hong Zhang is a professor at Nanjing University of Science and Technology, China. His research interests are information security, specially in wireless sensor network security and cloud security, and he has published more than 100 papers in journals and refereed conference proceedings in those areas.