Modeling and Analysis of High Availability Techniques in a Virtualized System

Xiaolin Chang[ab*], Tianju Wang[ab], Ricardo J. Rodríguez[c], Zhenjiang Zhang[d]

[a]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, P. R. China

[b]School of Computer and Information Technology, Beijing Jiaotong University, P. R. China

[c]Centro Universitario de la Defensa, Academia General Militar, Zaragoza, Spain

[d]School of Electronics and Information Engineering, Beijing Jiaotong University, P. R. China

Abstract

Availability evaluation of a virtualized system is critical to the wide deployment of cloud computing services. Time-based, prediction-based rejuvenation of virtual machines (VM) and virtual machine monitors (VMM), VM failover, and live VM migration are common high-availability (HA) techniques in a virtualized system. This paper investigates the effect of combination of these availability techniques on VM availability in a virtualized system where various software and hardware failures may occur. For each combination, we construct analytic models rejuvenation mechanisms improve VM availability; (2) prediction-based rejuvenation enhances VM availability much more than time-based VM rejuvenation when prediction successful probability is above 70%, regardless failover and/or live VM migration are also deployed; (3) failover mechanism outperforms live VM migration, although they can work together for higher availability of VM. In addition, they can combine with software rejuvenation mechanisms for even higher availability; (4) and time interval setting is critical to a time-based rejuvenation mechanism. These analytic results provide guidelines for deploying and parameter setting of HA techniques in a virtualized system.

***Keywords**:* Live VM migration, Software aging, Software rejuvenation, Stochastic Reward Nets, Virtual Machine, Virtual Machine Monitor

[*] Corresponding author, *Email address:* xlchang@bjtu.edu.cn (Xiaolin Chang)

1. Introduction

System virtualization technology has been widely adopted for academic and industrial purposes. In a virtualized system [1], a virtual machine monitor (VMM) is a software layer between (one or more) operating systems and a physical hardware able to emulate hardware of a physical machine. Thus, it plays a critical role in the virtualized system, often becoming the single point of failure. A virtual machine (VM) emulates a particular computer system, running on the top of VMM. Like traditional software, VM and VMM are also subject to software-related problems as software aging, bugs, crashes, and so on [2]-[4]. These problems clearly reduce the VM availability and increase VM downtime. Without loss of generality, we use application availability and VM availability interchangeably. Large downtime of applications may lead to productivity loss and even revenue loss [5][6]. Software rejuvenation, failover, and live VM migration are common high availability (HA) techniques used in a virtualized system [7][8]. The tremendous growth in the deployment of virtualized systems demands the availability analysis of these systems with HA techniques [7].

State-space models are expressive and popular models applied to availability analysis in different domains, such as cluster computing systems, telecommunication systems, or air control systems, among others [9]-[13]. In particular, they are found also effective for VM availability analysis [14]. The existing model-based VM availability analysis ignored the existence of VMM failures [15]-[17], assumed that one type failure exists in the considered system [15][16][18][19], considered only rejuvenation mechanisms [20][21], or considered only a physical host [19][22][23]. Thus, their analyses did not capture the effect of live VM migration on VM availability. Similarly, to assume only one VM in a host [20][24][25] cannot capture the effect of VM failover on VM availability.

In this paper, we consider a virtualized system composed with three main components: *Main host*, *Backup host*, and *Management host*. *Main host* includes active and standby (or backup) VMs. *Backup host* contains only standby VMs. Applications are deployed in the active VM. When the active VM fails, different actions may happen according to the restoring policy, such as the use of a standby VM on the same host, the migration to the other host, or simply the failed VM is restarted. This paper aims to investigate the effect of software rejuvenation, failover, and live VM migration techniques on the VM availability in a virtualized system with a variety of failures. We assume that these HA techniques are ready to be used and their implementations are out of the scope of this paper.

The contribution of this paper is three-fold: first, we investigate VM availability in a virtualized system with several co-existing failures, including hardware, shared storage, live VM migration, non-aging Mandelbug-related, and aging-related failures (in both VM and VMM). Second, we construct stochastic reward nets (SRN) models for each combination of software rejuvenation, failover, and live VM migration in order to analyze the induced effect of these techniques over VM availability. We also investigate whether some of these HA techniques could work together to improve the VM availability and the capability of this cooperation. Third, we carry out sensitivity analysis to investigate the effect of model parameters on the ability of software rejuvenation, failover, and live VM migration mechanisms in improving VM availability.

The proposed SRN models help to select the combination of failure recovery techniques and the parameter settings of a given scenario. Our numeric results indicate that:

(1) Both VMM rejuvenation and VM rejuvenation mechanisms enhance system availability when various failures co-exist.

(2) Prediction-based VM rejuvenation mechanism improves the VM availability in a higher degree than time-based VM rejuvenation mechanism, when prediction successful probability is above 70% and regardless failover and/or live VM migration are deployed.

(3) Failover mechanism performs better than live VM migration and they can work together for higher availability. In addition, they can work with software rejuvenation mechanisms for achieving even higher availability.

(4) Rejuvenation time interval setting is critical to a time-based rejuvenation mechanism. VMM clock interval is critical for the ability of live VM migration technique in improving VM steady-state availability.

The rest of the paper is organized as follows. In Section 2, we discuss the related work about HA techniques and model-based VM availability analysis. Section 3 introduces the system architecture considered in this paper. Section 4 describes SRN models constructed for analysis. The numerical analysis and discussion are presented in Section 5. Section 6 concludes this paper and discusses the future work.

2. Related Work

Both software failures and hardware component faults may lead to failures into a virtualized system and then reduce VM availability. Software failures are caused by inherent software design bugs. In [34], the authors classified software bugs into the following three main categories:

(1) Bohrbug, which manifests a failure when certain fixed set of conditions are met.

(2) Non-aging related Mandelbug, whose activation and/or error propagation is complicated and uncertain.

(3) Aging related Mandelbug, whose activation process is related to an accumulation of errors or resources consumption.

Bohrbug can be easily fixed. The left two kinds of bugs are hard to mitigate. In this paper, we ignore Bohrbugs and focus on non-aging related Mandelbugs. Both of the non-aging related Mandelbugs and the aging-related bugs occur on the VMMs and VMs subsystems [24]. When the non-aging related Mandelbugs failures happen, the VMM or VM would be in crash and need to be repaired. For long-running VMMs and VMs, software aging is one of the major causes of software failures [3]. Software aging has been observed in many systems, including web servers and enterprise clusters [26][27]. Software aging not only increases the failure rate and thus degrades the system performance, but also leads to system crashes [2]. Software rejuvenation [3] is a software fault tolerance technique to defend against software aging. This technique gracefully stops the execution of an application/system and periodically restarts it at a clean internal state in a proactive manner. Two main kinds of software rejuvenation approaches are distinguished:

- *Time-based rejuvenation.* Rejuvenation is triggered by a clock counting time. Analytical models help finding out the optimal interval to maximize availability and minimize downtime cost.
- *Prediction-based rejuvenation.* Rejuvenation is triggered when the system behaviors meet some predefined criteria or particular conditions. Machine learning, statistical approaches, structural models, and other techniques have been applied to define such conditions.

Besides software rejuvenation, failover solution and live VM migration are the most common techniques used for achieving VM high availability in virtualized systems, such as VMware ESXi [28]. Failover is a backup operational mechanism, in which the functions of a system component (e.g., a processor, server, network, or database) are assumed by secondary system components when the primary component becomes unavailable due to failure or shut-down scheduled. In a virtualized system, failover is achieved by creating an active VM and a standby VM. When the active VM suffers a failure or gets ready to be rejuvenated, the standby VM takes over the role of the active VM to continue task execution. Live VM migration refers to the process of moving a running VM or application between different physical machines without affecting the execution of applications. The information of memory, storage, and network connectivity of the original VM is transferred from the original host to the destination host.

Recently studies have been carried out for the VM availability analysis by adopting analytic modeling approach, specially using state-space models. A single server virtualized system with multiple VMs was modeled and analyzed in [15][16], where it was shown that the combination of failover mechanism with VM software rejuvenation technique enhanced VM availability in these systems. In [17], a continuous-time Markov chain (CTMC) based analytical model to capture the behavior of the virtualized clustering system with VM software rejuvenation was presented. In particular, system availability with the VM time-based rejuvenation mechanisms under different cluster configurations were analyzed, and results showed that the integration of virtualization, clustering, and software rejuvenation improved system availability. All these works [15]-[17] neglected the existence of VMM failures in the virtualized system. However, the VMM plays a critical role in improving system availability. In [18], only VMM software aging-triggering failures were considered.

The design of effective approaches for software rejuvenation on a virtualized system in order to improve VM availability have also been addressed. In [24], three VM rejuvenation techniques (namely cold-VM rejuvenation, warm-VM rejuvenation, and migrate-VM rejuvenation) were proposed for virtualized systems with VMM and VM aging-related failures. Their numerical results indicated that migrate-VM rejuvenation outperformed the others as long as the VM migration rate was fast enough. Unlike our paper, they assumed only one VM on *Main host* and no prediction-based rejuvenation techniques. Besides the failures mentioned in [24], in this paper we consider shared storage failures and non-aging Mandelbugs-related failures. We also consider that the virtualized system is composed by two VMs on a host and investigate the ability of failover mechanism. Moreover, we compare the abilities of time-based VM rejuvenation and prediction-based rejuvenation techniques in improving VM availability.

A single-server virtualized system where several VMs are instantiated on a VMM is considered in [19]. However, only VMM aging-related failure and time-based VMM rejuvenation techniques were considered. In [20], a system architecture with two hosts is analyzed where each host disposed a VM running on the VMM. They proposed a hierarchical stochastic model based on Fault Tree and CTMC that described hardware failures of different nature (e.g., CPU, memory, power, etc.), software failures (VMs, VMM, application) and corresponding recovery behaviors. However, this model does not cover completely the dependencies of behaviors between hardware and software subsystems (see Section 3.2). In [21], Nguyen et al. proposed a comprehensive availability model for a virtualized system with two hosts where each host runs two VMs on the VMM. They considered diverse failures, such as hardware, shared storage, aging-related, and non-aging Mandelbugs-related failures, as well as corresponding recovery behaviors modeled with SRN. They used a cold-VM rejuvenation to drastically push a VM in the running state into DOWN state. Failover and live VM migration mechanisms were both ignored in [20]and [21]. Furthermore, these works only considered the time-based rejuvenation technique. In this paper, we consider software rejuvenation, failover and live VM migration techniques, as well as time-based and prediction-based rejuvenation techniques.

A new hybrid rejuvenation technique which combined time-based rejuvenation mechanism for VMM and prediction-based rejuvenation mechanism for VMs was presented in [22]. They demonstrated that such combination produced higher system availability and lower downtime cost than using just prediction-based or time-based rejuvenation for VMs. SRN models for availability of a single-server virtualized system were presented in [23], where the abilities of VM time-based rejuvenation and VM prediction-based rejuvenation were compared. Both works considered only one host, and did not consider non-aging Mandelbug-related failures and hardware failures. These works used a failover mechanism (active VM and standby VM on the same host), but did not consider live VM migration as this paper does. As in [23], we compare VM time-based rejuvenation with VM prediction-based rejuvenation, but in a scenario with a variety of failures.

An availability model of a data center (DC) with live VM migration and failover mechanisms was introduced in [24] to ensure the high availability of cloud based businesses. Different failures were considered, such as hardware, shared storage, virtual DC failures, among others. Unlike our work, they did not distinguish non-aging Mandelbug-related failures from aging-related failures for VM. Furthermore, the effect of VMM failure on VM availability was analyzed by investigating host hardware failure. Note that VM live migration is performed only when VMM runs in virtualized systems. Thus, it is difficult, if not impossible, to analyze the effect of live VM migration on VM availability in a virtualized system.

In [35], the authors proposed a cloud availability model for the cloud data center with three PM pools of switched on, standby, and switched off PMs. They only considered PM failures and applied backup PMs for improving availability. In [36], the authors applied SRNs to analyze the system availability, which was defined as the probability that a job did not traverse any failure states during its execution. They considered as failure any deviation of a job execution from the correct life cycle, including queuing failures, running failures, aborts and exiting failures. Actually, these failures can be classified to aging-related failure or Non-aging Mandelbug related failure. Job checkpointing and job replication were adopted for improving the availability. In our paper, we consider not only software failures but also hardware failures. In addition, the recovery techniques considered in our paper include checkpointing, failover and live VM migration. Furthermore, we consider the difference of VM software failures on the system availability from VMM.

3. System Architecture and Component Interaction

This section first relates the system architecture that we consider in this paper, and then the interplay between its components. Note that current cloud data centers may easily adapt their architectures to the proposed scheme when enough physical resources are available.

3.1. Description of the System Architecture

Figure 1 depicts the system considered in this paper. It is mainly composed of three components: *Main host*, *Backup host*, and *Management host*. *Main host* contains a VMM, which runs an active VM with a desired application, and a standby VM. *Backup host* is a spare host that performs the *Main host* role when a VM migration occurs. It disposes a standby VM used by the failover mechanism after *Backup host* takes the role of *Main host*. Finally, *Management host* is a component responsible for detecting VMM failures and host hardware failures by means of specific management tools. VM images or VMM code files are stored in a shared storage within the system. *Software rejuvenation agent* (SRA), installed in each VM, is responsible for the VM rejuvenation operation. *Rejuvenation manager* (RM) is installed in the VMM in order to analyze the behaviors of VMs deployed on this VMM, detect anomalies and trigger the rejuvenation of malfunctioning VM.

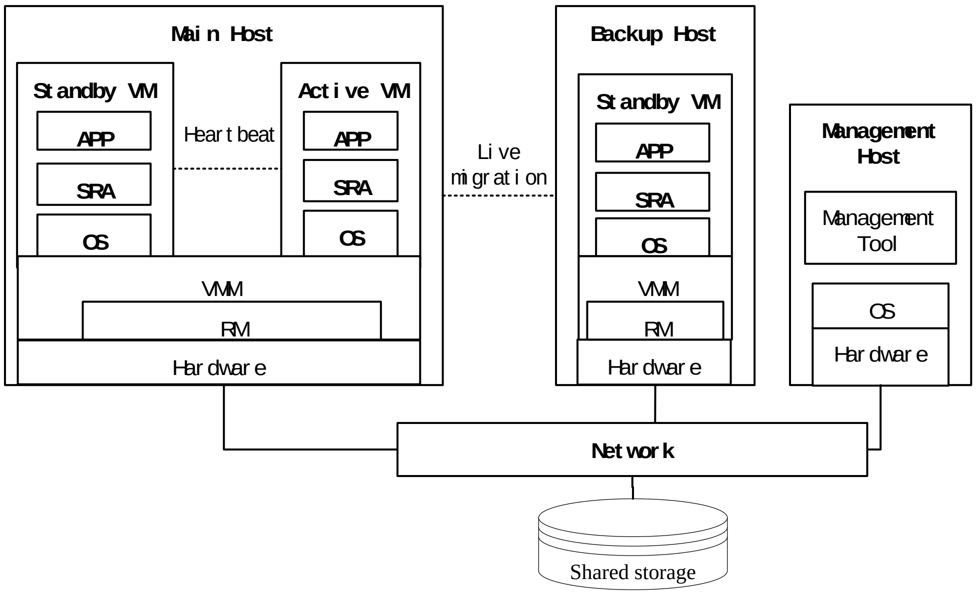


Figure. 1. System architecture considered in this paper

The VMs on the same host inform each other about their health using a heartbeat mechanism when failover is deployed. The running state of an active VM can be stored in the shared storage and later sent to the standby VM to be recovered. We consider that an active VM may suffer non-aging Mandelbug-related and aging related failures. When an active VM suffers from these failures, the active VM stops accepting requests and needs to be repaired. Thus, the standby VM plays the role of the active VM during its repair period and takes charge of the tasks. Let us remark this change in the standby VM is performed very quickly due to failover mechanism. The malfunctioning VM will turn to be standby VM after reparation, rejuvenation, or recovery is completed.

Live VM migration (when deployed) can be used to migrate the active VM to *Backup host* when the VMM needs to be rejuvenated. This technique moves an active VM, with all the requests and sessions, from *Main host* to *Backup host* without loss in any in-flight request or session data during the rejuvenation or repair. Pre-copy [29] and stay-on methods have been proved to be effective methods in the live VM migration process [24]. Herein, we consider pre-copy migration because it causes less downtime. Thus, as response to a VM migration request, the memory of the active VM is copied to *Backup host* without interrupting its operation.

3.2. Interplay between Components

Instead of analyzing the overall effect of hardware, non-aging Mandelbug-related, and software aging-related failures, we consider the effect of each failure respectively in order to capture realistic behavior of a virtualized system. This section introduces the state transitions for each component within the system. In particular, we focus on failures in *Main host*, *Backup host*, shared storage, VMM, and VMs. Failures occurring on *standby VM* and *Management host* are ignored.

Main host and shared storage have two states: UP and FAIL state. For VMM, there exist five states when no rejuvenation mechanisms are used: UP state (healthy state without software aging), FP state (probable failing state but VMM still runs), FAIL state (a software aging-related failure occurs and then VMM stops running), CRASH state (a non-aging Mandelbug-related failure occurs and then VMM stops running), and DOWN state (a hardware failure occurs in the host and VMM stops running). When using time-based rejuvenation for VMM, we define an additional REJU state to identify when the VMM is ready to be rejuvenated.

Finally, VM states are different depending on the rejuvenation mechanisms used. In this paper, we consider time-based and prediction-based rejuvenation mechanisms. When no rejuvenation is applied, an active VM has five states similar to states of a VMM: UP state, FP state, FAIL state, CRASH state, and DOWN state. In the latter state, the

VMM is not in UP or FP state, or shared storage is in FAIL state. When time-based rejuvenation is applied, there is an extra REJU state as VMM does. Similarly, prediction-based rejuvenation mechanism introduces three additional states: DETECT state (software aging in VM is detected), UNDETECT state (software aging is undetected) and REJU state.

Since there exists dependencies between components, the change of a component state clearly triggers the change of state of its dependent component. In the following, we enumerate the dependencies between the components and the related state transitions:

(1) **Between host and VMM.** When a host fails into FAIL state, the running VMM (in UP or FP state) moves to DOWN state in which the VMM subsystem becomes unavailable. The VMM in DOWN state can restart to UP state when the host returns to UP state. When the host goes into FAIL state, the VMM in CRASH, REJU, or FAIL states is suspended and reloaded from the memory after the host is repaired.

(2) **Between VMM and VM.** When the VMM is in FAIL, DOWN, or CRASH states, the active VM (in UP or FP state) then turns to DOWN state and the standby VM is suspended. When the VMM of *Main host* needs to be rejuvenated (e.g., it is REJU state), rejuvenation techniques such as warm-VM reboot [30] and VM migration (detailed SRN models are given in Section 4) can be applied to lead VM to UP state again.

(3) **Between shared storage and VM.** Recall that VM image files are stored on the shared storage. Thus, the state of the shared storage has a great impact on the VM states. When the shared storage fails, the VMs in running states (either UP or FP state) move to DOWN state. When the shared storage completes its repairing, the VMs are restarted to UP state. When the current state of a VM is not in running state, its operation state is temporarily suspended and resumed after the shared storage returns to UP state.

4. Availability SRN Models

All time intervals are assumed to follow exponential distributions, except for rejuvenation-triggered intervals. As in [38], this paper uses a 10-stage Erlang distribution for approximating these deterministic transitions. SRN [31][32] is a formalism widely used in rejuvenation modeling. In order to understand the ability of each HA mechanism in improving the VM availability, we develop SRN models for nine policies, described in Table 1. Each policy represents a combination of the HA mechanisms. Let *ABCD* be used to represent a policy. Each item is defined as follows:

$$A = \begin{cases} = T\text{: denotes that VMM implements a time-based rejuvenation mechanism} \\ = N\text{: denotes that no HA mechanism is implemented in VMM} \end{cases}$$

$$B = \begin{cases} = T\text{: denotes that VM implements a time-based rejuvenation mechanism} \\ = P\text{: denotes that VM implements a prediction-based rejuvenation mechanism} \\ = N\text{: denotes no rejuvenation mechanism is implemented in VM} \end{cases}$$

$$C = \begin{cases} =F\text{: denotes that failover is implemented} \\ =N\text{: denotes that no failover is implemented} \end{cases}$$

$$D = \begin{cases} =M\text{ : denotes that live VM migration is deployed} \\ =N\text{ : denotes that no live VM migration is deployed} \end{cases}$$

Table 1. Description of nine policies used in this paper

Policy	Description
NNNN	No HA technique deployed for VMM and VM
TNNN	Only time-based rejuvenation used for VMM
TNFN	Time-based rejuvenation for VMM and failover mechanism for VM
TNNM	Time-based rejuvenation for VMM and live VM migration for VM
TTNM	Time-based rejuvenation for VMM, live VM migration and time-based rejuvenation for VM
TPNM	Time-based rejuvenation for VMM, live VM migration and prediction-based rejuvenation for VM
TNFM	Time-based rejuvenation for VMM, failover and Live VM migration and no rejuvenation for VM
TTFM	Time-based rejuvenation for VMM, failover and Live VM migration and time-based rejuvenation for VM
TPFM	Time-based rejuvenation for VMM, failover and Live VM migration and prediction-based rejuvenation for VM

4.1. Stochastic Reward Nets

A Stochastic Reward Net is a stochastic Petri net with many advanced structural and stochastic characteristics [31][32]. In SRN, an enabling function (also called a *guard*) allows to define the enabling function of a transition as a marking dependent function. In addition, both arc multiplicities and firing rates are allowed to be marking-dependent. SRN allow to compute measures of interests by defining reward rates at net level.

In the following, we first introduce the models for host, shared storage, VMM, VMM clock, and VM clock sub-models. These models are unchangeable regardless of policies. Next, we describe the models for the policies considered in this paper. Table 2 summarized the parameters used in the rest of this section, as well as the corresponding transitions and values used for numerical analysis. Most values are set according to [21, 23, 24] and the corresponding references are given in the last column. The values without references are set by ourselves according to the values with references.

Table 2. Default parameters used in the models

Symbol	Description	Transition	Values(/h)	Mean time
λ_{vfp}	VM aging-related rate	T_{vfp}, T_{v1fp}, T_{v2fp}	0.005952381	1 week [24]
λ_{vfail}	VM aging-related failure rate	T_{vfail}, T_{v1fail}, T_{v2fail}	0.013888889	3days [24]
λ_{crash}	VM non-aging Mandelbug related failure rate	T_{vcrash}, $T_{v1crash}$, $T_{v2crash}$	0.00034722	120days
λ_{reup}	VM non-aging Mandelbug -related repair rate	T_{vreup}, T_{v1reup}, T_{v2reup}	2	30mins
$\lambda_{vrepair}$	VM repair rate	$T_{vrepair}$,$T_{v1repair}$, $T_{v2repair}$	2	30mins [24]
λ_{vreju}	VM rejuvenation rate	T_{vreju}, T_{v1reju}, T_{v2reju}	60	1min [24]
λ_{swt}	VM switch rate	$T_{swt1reju}$, $T_{swt2reju}$	1200	3s [23]
λ_{vstart}	VM restart rate	T_{vstart}, $T_{v1start}$, $T_{v2start}$	120	30s [24]
$\Lambda_{interval}$	VM clock interval	-	0.04166667	1day [24]
λ_{detect}	VM detect probability	-	0.9	N/A [23]
λ_{pre}	VM migration prepare rate	T_{vpre}, T_{vback}, T_{vfpre}, $T_{vfpback}$,	90	40s [[24]
λ_{migs}	VM migration successful probability	-	0.9	N/A [24]
β_{hfp}	VMM aging rate, namely rate of transition	T_{h1fp},T_{h2fp}	0.001388889	1 month [24]
β_{hfail}	VMM aging-related failure rate	T_{h1fail}, T_{h2fail}	0.005952381	1 week [24]
β_{crash}	VMM non-aging Mandelbug- related failure rate	$T_{h1crash}$, $T_{h\ 1crash}$	0.00046296	90days
β_{reup}	VMM non-aging Mandelbug- related repair rate	T_{h1reup}, T_{h2reup}	0.5	2hours
$\beta_{hrepair}$	VMM repair rate	$T_{h1repair}$, $T_{h2repair}$	1	1 hour [24]
β_{hreju}	VMM rejuvenation rate	T_{h1reju}, T_{h2reju}	30	2mins [24]
β_{hstart}	VMM restart rate	$T_{h1start}$, $T_{h2start}$	60	1min [24]
$\beta_{interval}$	VMM clock interval	-	0.005952381	1week [24]
β_{wfail}	Host failure rate	T_{w1fail}, T_{w2fail}	0.00023148	180days [21]
$\beta_{wrepair}$	Host repair rate	$T_{w1repair}$, $T_{w2repair}$	0.01388889	3days [21]
B_{ssf}	Shared storage failure rate	T_{fail}	0.00011574	360 days [21]
B_{ssr}	Shared storage repair rate	T_{repair}	0.01388889	3 days [21]

4.2. SRN Models for System Components

Host model, depicted in Figure 2(a), expresses the occurrence of hardware failure and repair process of *Main host*. We consider a system with two hosts. At the beginning, *host*$_1$ is *Main host* and *host*$_2$ is *Backup host*. After live VM migration, *host*$_2$ becomes *Main host* and *host*$_1$ becomes *Backup host* after repair. When *host*$_1$ (*host*$_2$) is *Main* *host*, *host*$_1$ (*host*$_2$) is at first in UP state, represented by one token in P_{w1up} (P_{w2up}) place. When a hardware failure occurs, the transition t_{w1fail} (t_{w2fail}) fires and the token in P_{w1up} (P_{w2up}) is taken out and deposited in P_{w1fail} (P_{w2fail}). After the hardware repair process completes, the token is moved from P_{w1fail}(P_{w2fail}) to P_{w1up} (P_{w2up}) by firing the transition $T_{w1repair}$ ($T_{w2repair}$), representing the host in UP state again.

The shared storage failure and repair process is similar to the host model, as depicted by Figure 2(b). We assume that shared storage is in UP state in the general case. Due to unexpected failures, the shared storage shuts down and then falls into FAIL state (namely, the token is moved from place P_{up} to place P_{fail}). A delay is required to detect the fault position of the shared storage and its repairing. Then, the shared storage returns to UP state.

Figure 2(c) depicts VMM failure and VMM recovery process for *Main host*$_1$. When *host*$_1$ (*host*$_2$) is *Main host*[1], at first there is a token in place P_{h1up} (P_{h2up}), denoting no software aging exists in the VMM. The VMM software aging occurs after continuously running for a period of time. After a while, the transition T_{h1fp} (T_{h2fp}) fires and a token is moved from P_{h1up} (P_{h2up}) to P_{h1fp} (P_{h2fp}). When the transition T_{h1fail} (T_{h2fail}) fires, the token from P_{h1fp} (P_{h2fp}) is deposited in P_{h1fail} (P_{h2fail}) which represents the VMM failure due to software aging. When a non-aging Mandelbug-related failure occurs, the VMM in UP state turns directly to CRASH state, represented by the removal of token from place P_{h1up} (P_{h2up}) and placed in $P_{h1crash}$ ($P_{h2crash}$). When the failure of VMM is detected by the management tool that executes in Management host, then VMM enters into the repair process. After VMM is repaired, the transition $T_{h1repair}$ ($T_{h2repair}$) or T_{h1reup} (T_{h2reup}) – depending on the type of failure fires and the token is moved from P_{h1fail} (P_{h2fail}) or $P_{h1crash}$ ($P_{h2crash}$) to p_{h1up} (P_{h2up}), denoting that the VMM enters UP state.

When a hardware failure occurs, the VMM in UP or FP state shuts down at once by firing the transition t_{hidw}, and the token is deposited in P_{h1dw} (P_{h2dw}). As soon as the hardware failure is removed, a token is taken from P_{h1dw} (P_{h2dw}) to P_{h1up} (P_{h2up}), denoting that VMM restarts and enters into UP state. When the VMM is in CRASH, FAIL, or REJU state), the VMM is suspended in memory and quickly reloaded to continue normal execution after the hardware failure is repaired.

When the VMM needs to be rejuvenated, the token is moved from P_{h1up} (P_{h2up}) or P_{h1fp} (P_{h2fp}) to P_{h1reju} (P_{h2reju}) by firing immediate transitions t_{h1rejt} (t_{h2rejt}) or $t_{h1fprejt}$ ($t_{h2fprejt}$), depending on the current state before rejuvenation

[1]As before, we use $i = 1$ notation to refer to VMM in *host*$_1$ which performs as *Main host* and $i = 2$ to refer to VMM in *host*$_2$.

takes place. After the rejuvenation finishes, a token in the P_{h1reju} (P_{h2reju}) is moved to P_{h1up} (P_{h2up}) only when host is in UP state (places P_{w1up}, P_{w2up}) and there exists a token in place $P_{h1clock}$ ($P_{h2clock}$).

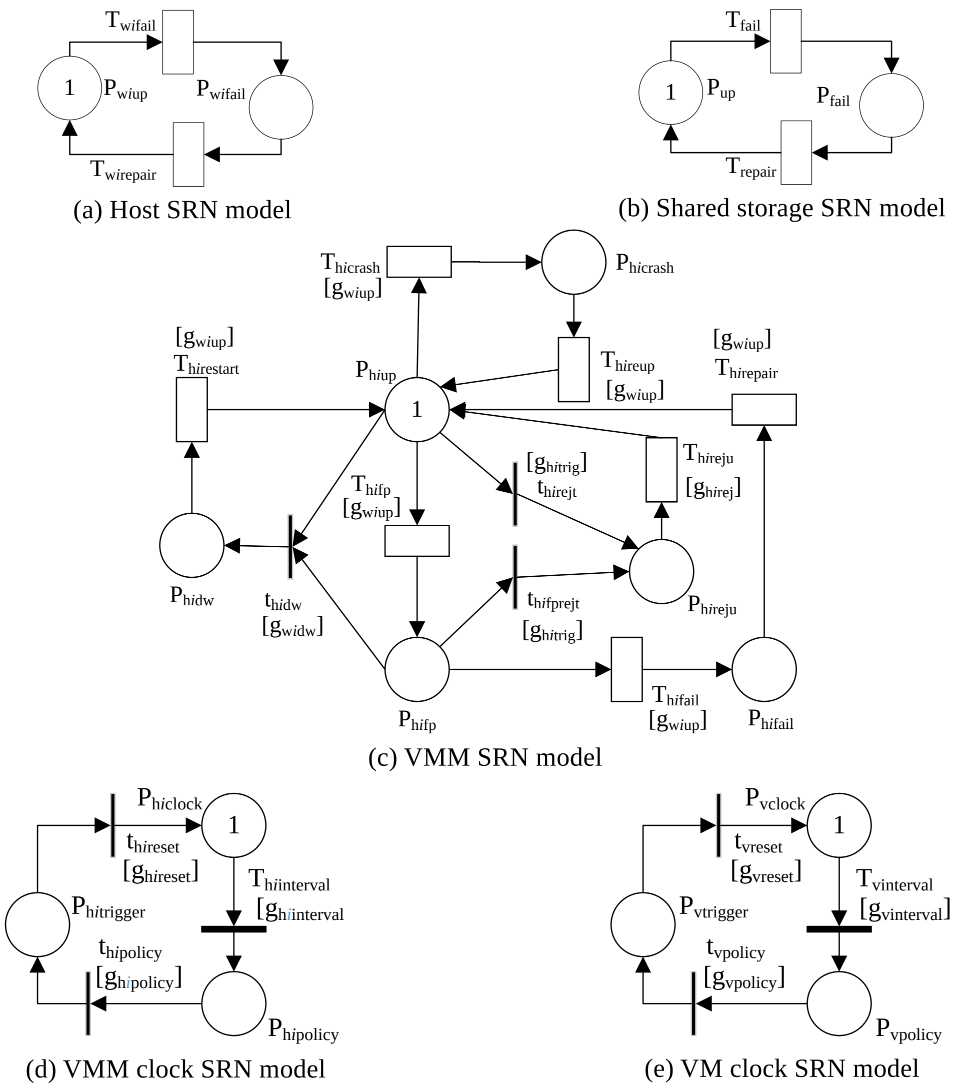


Figure. 2. SRN models for different system components

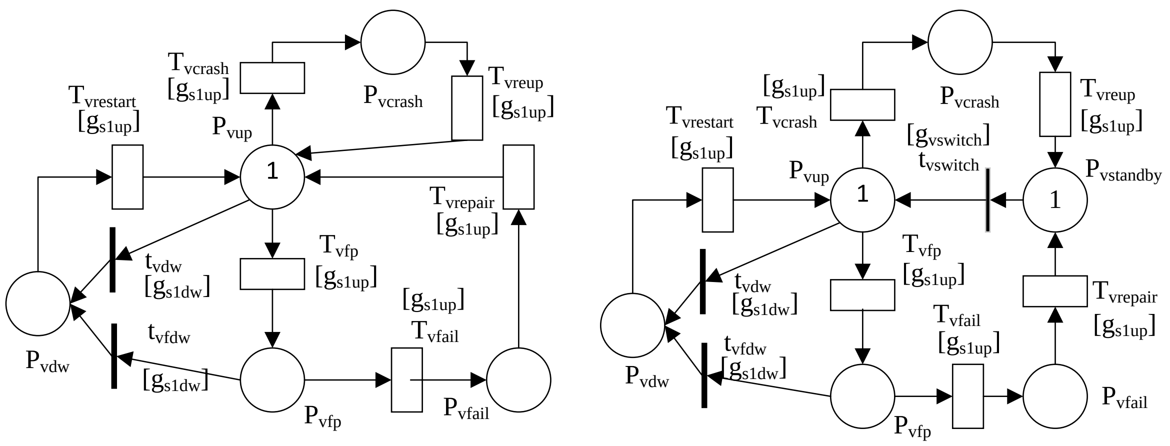
Figure 2(d) depicts the SRN of the VMM clock, used to trigger VMM time-based rejuvenation with a rate of $\beta_{interval}$. When the transition $T_{h1interval}$ ($T_{h2interval}$) fires after the last boot time, a token is moved from $P_{h1clock}$ ($P_{h2clock}$) to $p_{h1policy}$ ($p_{h2policy}$). The guard function $g_{h1policy}$ in $t_{h1policy}$ ($t_{h2policy}$) ensures the VM has been shut down or migrated to *Backup host*. Then, the token in $P_{h1policy}$ ($P_{h2policy}$) is moved to $P_{h1trigger}$ ($P_{h2trigger}$) and the VMM rejuvenation process begins. After the VMM rejuvenation completes, the immediate transition $t_{h1reset}$ ($t_{h2reset}$) enables and the token in $P_{h1trigger}$ ($P_{h2trigger}$) is moved to $P_{h1clock}$ ($P_{h2clock}$) again, which denotes counting the time for executing the next VMM rejuvenation period.

Similarly, Figure 2(e) depicts the VM clock model, used to trigger VM time-based rejuvenation with a rate of $\Lambda_{interval.}$ When transition $T_{vinterval}$ fires after the last boot time, the token is moved from P_{vclock} to $P_{vpolicy}$. The immediate transition $t_{vpolicy}$ is enabled when the VM in UP or FP state and shared storage is in UP state. Then, the token in $P_{vpolicy}$ is deposited in $P_{vtrigger}$ and the VM rejuvenation process begins. After this process completes, the immediate transition t_{vreset} fires and the token in $P_{vtrigger}$ is placed in P_{vclock}, which denotes counting the time for executing the next VM rejuvenation period.

4.3. SRN Models for Combination of Policies

We first describe SRN models for NNNN, TNNN, and TNFN policies, since they have similar sub-models and features. As live VM migration is not implemented (i.e., *D = N*), *Backup host* is ignored in these policies. Policy NNNN consists of four sub-models: (1) Host1 model shown in Figure 2(a); (2) shared storage model shown in Figure 2(b); (3) VMM1 model shown in Figure 2(c); and (4) VM model shown in Figure 3(a). Since there is no rejuvenation process for VM and no failover mechanism in both policies NNNN and TNNN, they have the same VM sub-model. Policy TNNN has an additional sub-model compared to policy NNNN. Namely, the VMM1 clock model shown in Figure 2(d). Policy TNFN differs from TNNN just for a special VM model that uses a failover mechanism, as depicted in Figure 3(b). Guard functions used in the above models are summarized in Table 3.

In the following, we explain the Figure 3(a). A VM may suffer from non-aging Mandelbug-related and aging-related failures. Failure and recovery processes are similar to those in VMM model. Initially, one token exists in P_{vup}, representing the VM is in fully stable state. Later on, the VM transits to FP state, i.e., the token is moved from P_{vup} to P_{vfp} through the transition T_{vfp} representing VM software aging. Note that the VM still works in FP state, but its failure likelihood increases. Since no VM rejuvenation process is deployed in these policies, the aging VM then turns to FAIL state after a certain period of time. This state transition is represented by the token being moved from P_{vfp} to P_{vfail}, after firing of the transition T_{vfail}. After VM is repaired, it changes to UP state. That is, the token is moved from P_{vfail} to P_{vup}. When the VM suffers from non-aging Mandelbug-related failures, the VM falls to CRASH state (i.e., the token is moved from P_{vup} to place P_{vcrash}) and waits for repair. When places P_{h1fail}, P_{h1dw}, or $P_{h1crash}$ are marked, or shared storage is failed, the token is moved from P_{vup} (or P_{vfp}) to P_{vdw}. When the VMM is in UP state (i.e., the token is in P_{h1up}) or FP state (i.e., the token is in P_{h1fp}) and the shared storage is in UP state, the VM can be restarted to UP state (i.e., the token is in place P_{vup}) by firing the timed transition $T_{vrestart}$. When VM is in FAIL or CRASH states, it is suspended and a warm-VM reboot mechanism is used later on.



(a) NNNN and TNNN

(b) TNFN

Figure. 3. SRN models for VM with (a) NNNN and TNNN and (b) TNFN policies

Table 3. Guard functions for NNNN, TNNN and TNFN policies

Guard	Definition
g_{w1up}	if(#(P_{w1up})==1) then 1 else 0
g_{w1dw}	if(#(P_{w1fail})==1) then 1 else 0
g_{s1up}	if(#(P_{h1up})==1||#(P_{h1fp})==1&&#(P_{up})==1) then 1 else 0
g_{s1dw}	if(#(P_{h1fail})==1||#(P_{h1dw})==1||#($P_{h1crash}$)==1||#(P_{fail})==1) then 1 else 0
g_{h1trig}	if(#($P_{h1trigger}$)==1&&#(P_{w1up})==1) then 1 else 0
g_{h1rej}	if(#($P_{h1clock}$)==1&&#(P_{w1up})==1) then 1 else 0
$g_{h1interval}$	if(#(P_{h1up})==1||#(P_{h1fp})==1) then 1 else 0
$g_{h1policy}$	if(#(P_{vdw})==1) then 1 else 0
$g_{h1reset}$	If(#(P_{h1reju})==1) then 1 else 0
$g_{vswitch}$	If (#(P_{hup})==1||#(P_{hfp})==1&&#(P_{up})==1&&(#(P_{vcrash})==1||#(P_{vfail})==1)) then 1 else 0

Policy TNFN uses failover mechanism (see Figure 3(b)). Thus, there are two VMs (active and standby VM) on the VMM. Using a heartbeat mechanism, the running state of the active VM can be stored in the shared storage and later be sent to the standby VM. When the active VM suffers from non-aging Mandelbug-related or aging-related failures, the active VM stops accepting requests. That is, the token is moved from $P_{vstandby}$ to P_{vup} by firing the immediate transition $t_{vswitch}$. Then, the standby VM is responsible of the tasks and plays the role of the active VM during its repair period. At the same time, the token is moved from P_{vcrash} (or P_{vfail}) to $P_{vstandby}$ by firing the timed transition T_{vreup} (or $T_{vrepair}$). That is, the primary active VM changes to be standby VM after the repair or recovery processes complete.

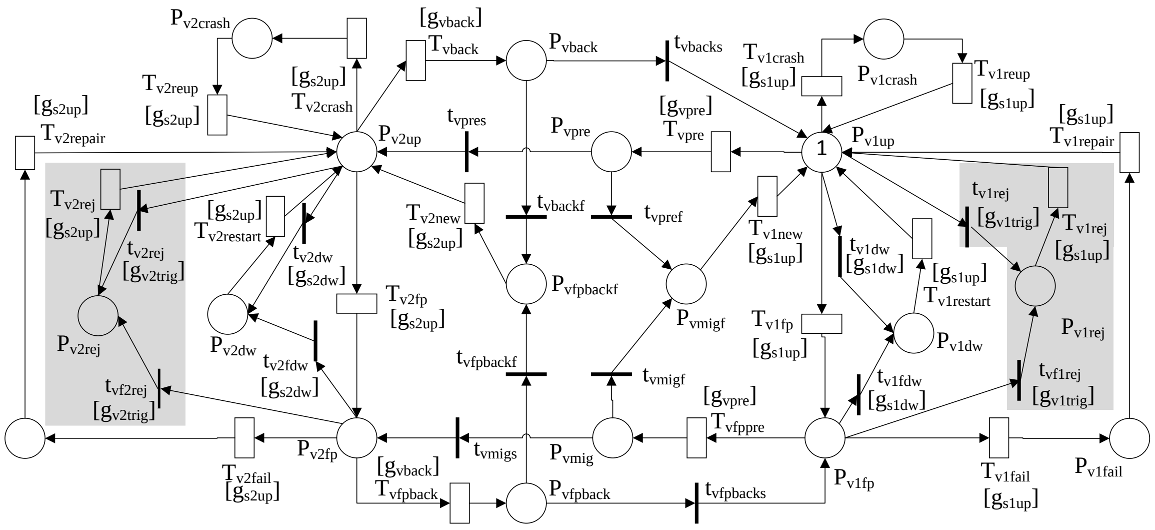


Figure. 4. SRN models for VM with TNNM and TTNM policies (places and transitions added by TTNM policy are highlighted in gray)

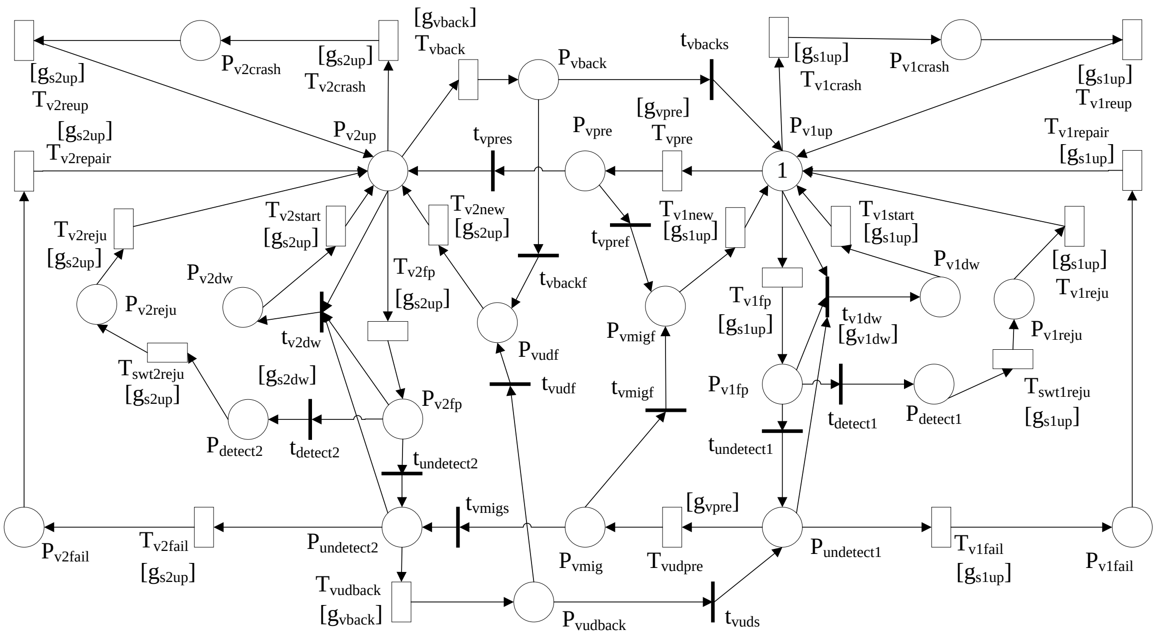


Figure. 5. SRN models for VM with TPNM policies

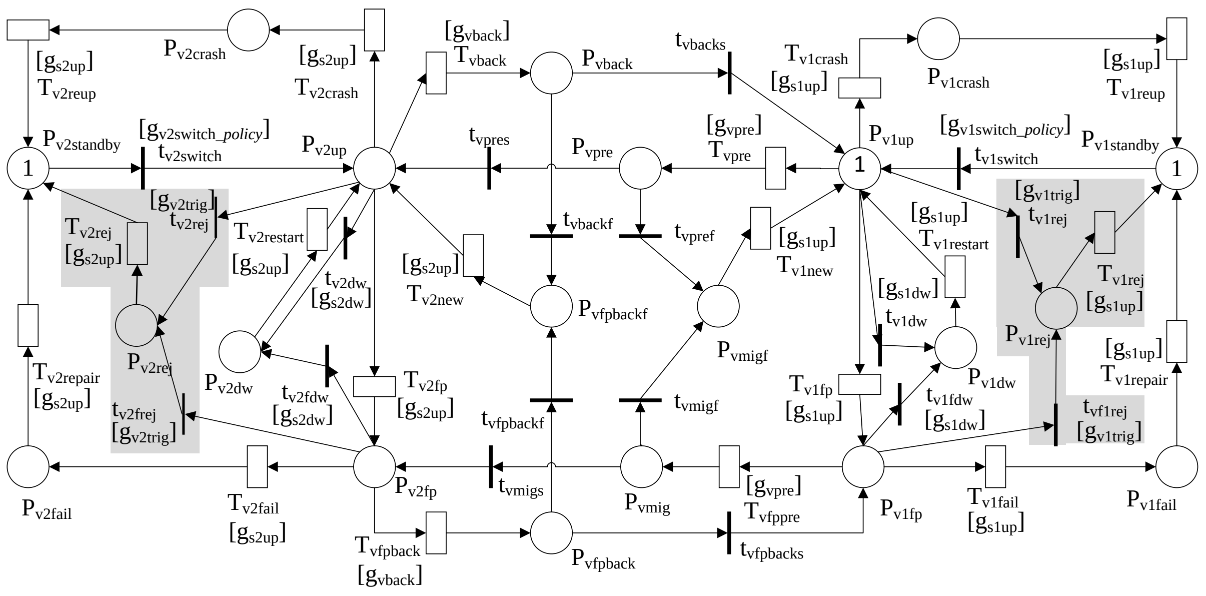


Figure. 6. SRN models for VM with TNFM and TTFM policies (places and transitions added by TTFM policy are highlighted in gray)

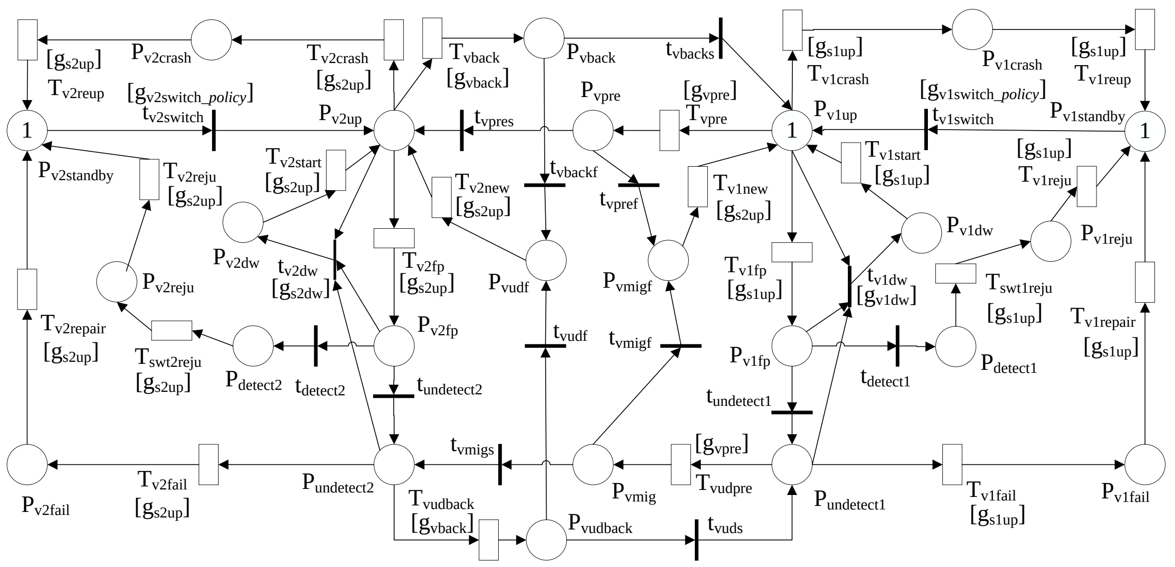


Figure. 7. SRN models for VM with TPFM policy

TNNM, TTNM, and TPNM policies use live VM migration to counteract the VMM rejuvenation. The VM rejuvenation mechanism is the difference between these policies. Since the failover mechanism is unimplemented, there is only one active VM on *Main host* and there is no standby VM on both *Main host* and *Backup host* for each policy. TNNM policy consists of eight submodels: (1) Host1 and (2) Host2 models as depicted in Figure 2(a); (3) shared storage model shown in Figure 2(b); (4) VMM1 and (5) VMM2 models as depicted in Figure 2(c); (6) VMM1 and (7) VMM2 clock models shown in Figure 2(d); and (8) VM model shown in Figure 4 (without considering places and transitions highlighted in gray area). Policies TTNM and TPNM consist of nine and eight submodels, respectively. The first seven submodels are the same as in TNNM policy. TTNM policy adds two submodels: VM model depicted in Figure 4 (considering places and transitions highlighted in gray area) and VM clock model shown in Figure 2(e). On the contrary, TPNM policy only adds the VM model depicted in Figure 5. Guard functions used in VM models of these policies are summarized in Table 4.

Finally, we describe the submodels for TNFM, TTFM, and TPFM policies. TNFM policy consists of eight submodels, where (1)-(7) submodels are the same as TNNM policy and (8) VM model as depicted in Figure 6. Similarly, TTFM policy consists of nine submodels, while TPFM policy has eight. In both cases, (1)-(7) submodels are the same as in TNNM policy. TTFM policy adds two submodels: VM clock model shown in Figure 2(e) and the VM model depicted in Figure 6. Note that transitions highlighted in the gray are have a different guard with respect to TNFM policy. TPFM policy has the VM model shown in Figure 7. The explanations of these models are similar to the ones described previously. Particular guard functions used in VM models of these three policies are summarized in Table 5. The reward functions for the nine models are shown in Table 6. Since the failure and recovery processes of host, shared storage, and VMM are the same as in those in Section 4.2, in the following we focus on the VM models.

VM model using time-based rejuvenation. First, we explain the VM model for TTNM policy, which uses time-based rejuvenation for VM. Initially, there exists one token in P_{v1up}, representing the active VM in fully stable state. Later, the active VM transits to FP state (namely, the token is placed in P_{v1fp}) through the transition T_{v1fp} representing the VM software aging. When VM fails due to software aging occurrence but the VM rejuvenation process is not triggered, the token is moved to P_{v1fail} by firing the transition T_{v1fail}. After VM completes its repair, it changes to UP state. That is, a token is taken from P_{v1fail} to P_{v1up}. Finally, when the VM suffers from a non-aging Mandelbug-related failure, the VM turns to CRASH state (i.e., a token is taken from P_{v1up} to $P_{v1crash}$) and waits for repair. When the VM is rejuvenated, the token is moved from P_{v1up} (or P_{v1fp}) to P_{v1rej}. As soon as the VM rejuvenation process completes, the token returns back to p_{v1up} by firing the timed transition T_{v1rej}.

When the VMM is in FAIL, DOWN, or CRASH states or if the shared storage fails, the VM in UP or FP state changes to DOWN state (i.e., a token is moved from P_{v1up} or P_{v1fp} to place P_{v1dw}). When VM is in FAIL or CRASH state at this time, it is suspended and later a warm-VM reboot mechanism is used. When the VMM is in UP or FP state (places P_{h1up} or P_{h1fp}, respectively) and if the shared storage is UP state, the VM is restarted to UP state (place P_{v1up}) by firing the timed transition $T_{v1restart}$. When the VMM starts rejuvenation and the VM is in UP or FP states, the VM will migrate to *Backup host* whether it is in UP or FP states. If the VM has been migrated to *Backup host*, the failure, recovery, and rejuvenation behaviors of the VM on the VMM2 is equal to the ones on the VMM1. Details of the live VM migration process are described in Section 4.4.

Table 4. Guard function of VM Models for TNNM, TTNM and TPNM policies

Guard	Definition
g_{w1up}	if(#(P_{w1up})==1) then 1 else 0
g_{w1dw}	if(#(P_{w1fail})==1) then 1 else 0
g_{h1rej}	if(#($P_{h1clock}$)==1&&#(P_{w1up})==1) then 1 else 0
g_{h1trig}	if(#($P_{h1trigger}$)==1&&#(P_{w1up})==1) then 1 else 0
g_{w2up}	if(#(P_{w2up})==1) then 1 else 0
g_{w2dw}	if(#(P_{w2fail})==1) then 1 else 0
g_{h2rej}	if(#($P_{h2clock}$)==1&&#(P_{w2up})==1) then 1 else 0
g_{h2trig}	if(#($P_{h2trigger}$)==1&&#(P_{w2up})==1) then 1 else 0
$g_{h1interval}$	if(#(P_{h1up})==1||#(P_{h1fp})==1) then 1 else 0
$g_{h1policy}$	if(#(P_{v1up})==0&&#(P_{v1fp})==0&&#(P_{vpre})==0&&#(P_{vmig})==0&&#(P_{vback})==0&&#($P_{vfpback}$)==0) then 1 else 0
$g_{h1reset}$	If(#(P_{h1reju})==1) then 1 else 0
$g_{h2interval}$	if(#(P_{h2up})==1||#(P_{h2fp})==1) then 1 else 0
$g_{h2policy}$	if(#(P_{v2up})==0&&#(P_{v2fp})==0&&#(P_{vpre})==0&&#(P_{vmig})==0&&#(P_{vback})==0&&#($P_{vfpback}$)==0) then 1 else 0
$g_{h2reset}$	If(#(P_{h2reju})==1) then 1 else 0
g_{s1up}	if(#(P_{h1up})==1||#(P_{h1fp})==1&&#(P_{up})==1) then 1 else 0
g_{s1dw}	if(#(P_{h1fail})==1||#(P_{h1dw})==1||#($P_{h1crash}$)==1||#(P_{fail})==1) then 1 else 0
g_{s2up}	if(#(P_{h2up})==1||#(P_{h2fp})==1&&#(P_{up})==1) then 1 else 0
g_{s2dw}	if(#(P_{h2fail})==1||#(P_{h2dw})==1||#($P_{h2crash}$)==1||#(P_{fail})==1) then 1 else 0
g_{vpre}	if(#($P_{h1policy}$)==1&&(#(P_{h2up})==1||#(P_{h2fp})==1)&&#(P_{up})==1) then 1 else 0
g_{vback}	if(#($P_{h2policy}$)==1&&(#(P_{h1up})==1||#(P_{h1fp})==1)&&#(P_{up})==1) then 1 else 0
g_{v1trig}	if(#($P_{vtrigger}$)==1&&#(P_{up})==1) then 1 else 0
g_{v2trig}	if(#($P_{vtrigger}$)==1&&#(P_{up})==1) then 1 else 0
$g_{vinterval}$	if(#(P_{h1up})==1||#(P_{h1fp})==1||#(P_{h2up})==1||#(P_{h2fp})==1&&#(P_{up})==1) then 1 else 0
$g_{vpolicy}$	if(#(P_{h1up})==1||#(P_{h1fp})==1||#(P_{h2up})==1||#(P_{h2fp})==1) then 1 else 0
g_{vreset}	if(#(P_{v1rej})==1||#(P_{v2rej})==1) then 1 else 0

Table 5. Guard functions of VM models for TNFM, TTFM and TPFM policies

Guard	Definition
$g_{v1switch\_TNFM}$	If(#(P_{h1up})==1||#(P_{h1fp})==1&&#(P_{up})==1&&(#($P_{v1crash}$)==1||#(P_{v1fail})==1)) then 1 else 0
$g_{v2switch\_TNFM}$	If(#(P_{h2up})==1||#(P_{h2fp})==1&&#(P_{up})==1&&(#($P_{v2crash}$)==1||#(P_{v2fail})==1)) then 1 else 0
$g_{v1switch\_TTFM}$	If(#(P_{h1up})==1||#(P_{h1fp})==1&&#(P_{up})==1&&(#($P_{v1crash}$)==1||#(P_{v1fail})==1)||#(P_{v1rej})==1)) then 1 else 0
$g_{v2switch\_TTFM}$	If(#(P_{h2up})==1||#(P_{h2fp})==1&&#(P_{up})==1&&(#($P_{v2crash}$)==1||#(P_{v2fail})==1)||#(P_{v2rej})==1)) then 1 else 0
$g_{v1switch\_TPFM}$	If(#(P_{h1up})==1||#(P_{h1fp})==1&&#(P_{up})==1&&(#($P_{v1crash}$)==1||#(P_{v1fail})==1)||#(P_{v1reju})==1)) then 1 else 0
$g_{v2switch\_TPFM}$	If(#(P_{h2up})==1||#(P_{h2fp})==1&&#(P_{up})==1&&(#($P_{v2crash}$)==1||#(P_{v2fail})==1)||#(P_{v2reju})==1)) then 1 else 0

Table 6. Reward rates for computing the steady-state unavailability of different models

Policy	Unavailability condition
NNNN, TNNN	if(#(P_{vdw})==1||#(P_{vfail})==1||#(P_{vcrash})==1) then 1 else 0
TNFN	if(#(P_{vdw})==1) then 1 else 0
TNNM, TTNM	if(#(P_{v1dw})==1||#(P_{v1fail})==1||#($P_{v1crash}$)==1||#(P_{v2dw})==1||#(P_{v2fail})==1||#($P_{v2crash}$)==1||#(P_{vmigf})==1||#($P_{vfpbackf}$)==1) then 1 else 0
TPNM	if(#(P_{v1dw})==1||#(P_{v1fail})==1||#($P_{v1crash}$)==1||#($P_{detect1}$)==1||#(P_{v1reju})==1#($P_{detect2}$)==1||#(P_{v2reju})==1||#(P_{v2dw})==1||#(P_{v2fail})==1||#($P_{v2crash}$)==1||#(P_{vmigf})==1||#($P_{vfpbackf}$)==1) then 1 else 0
TNFM	if(#(P_{v1dw})==1||#(P_{v1fail})==1||#($P_{v1crash}$)==1||#(P_{v2dw})==1||#(P_{v2fail})==1||#($P_{v2crash}$)==1||#(P_{vmigf})==1||#($P_{vfpbackf}$)==1) then 1 else 0
TTFM	if(#(P_{v1dw})==1||#(P_{v2dw})==1||#(P_{vmigf})==1||#($P_{vfpbackf}$)==1) then 1 else 0
TPFM	if(#(P_{v1dw})==1||#(P_{v2dw})==1||#(P_{vmigf})==1||#(P_{vudf})==1||#($P_{detect1}$)==1||#($P_{detect2}$)==1) then 1 else 0

VM model using prediction-based rejuvenation. In this part we detail the VM model for TPNM policy, which uses prediction-based rejuvenation for VM (see Figure 5). Initially, there exists a token in P_{v1up}, representing the fully stable state of VM. At time passes, the VM eventually transits to a FP state, that is, a token is taken from P_{v1up} and placed in P_{v1fp} through the transition T_{v1fp} representing the software aging of the VM. It is assumed that RM detects VM software aging with probability λ_{detect}. Thus, place P_{v1fp} has two immediate transitions with appropriate probabilities for detecting aging or failures. When the aging is detected, the immediate transition $t_{detect1}$ fires and the token is deposited in $P_{detect1}$. Otherwise (i.e., detection fails), the token in place P_{v1fp} is deposited in $P_{undetect1}$. In DETECT state (represented by $P_{detect1}$), the VM finishes its tasks in hand and stops receiving requests. After a preparing process, the VM gets ready to be rejuvenated, which is represented by firing of transition $T_{swt1reju}$ (that is, the token is deposited in P_{v1reju}). When the VM ends rejuvenation, the transition T_{v1reju} fires and the token is placed in P_{v1up}, which represents the VM is in UP state. In UNDETECT state (place $P_{undetect1}$), the token is moved to P_{v1fail} by firing the transition t_{v1fail} when the VM failure occurs due to software aging. Otherwise (i.e., when the VM suffers from a non-aging Mandelbug-related failure), the VM falls to CRASH state (i.e., the token is taken from place P_{v1up} to place $P_{v1crash}$).

As before, when the VMM is in FAIL, DOWN, or CRASH state, or if the shared storage fails, the VM in UP, FP, or UNDETECT state changes to DOWN state (i.e., the token is moved from P_{v1up}, P_{v1fp}, or $P_{undetect1}$ to place P_{v1dw}). When VM is in FAIL, DETECT, REJU, or CRASH states at this time, its execution is suspended and later a warm-VM reboot mechanism is used. When the VMM is in UP or FP state and the shared storage is in UP state, then the VM is restarted to UP state by firing the timed transition $T_{v1start}$. When the VMM is going to be rejuvenated and the VM is in UP or FP state, the VM migrates to *Backup host* whether this host is in UP or FP state as well. Once the VM was migrated, failure, recovery, and rejuvenation behaviors of the VM on the VMM2 are similar to the ones on the VMM1. The process of live VM migration is described in the sequel.

4.4. Live VM Migration Process

At the beginning, the VM is on the VMM1 of *Main host*. When the clock of VMM1 requests rejuvenation for VMM1, the guard g_{vpre} enables T_{vpre} and T_{vudpre} (T_{vfppre}) for live VM migration as long as VMM2 is available. When the live VM migration is completed, the token is deposited in P_{v2up} and $P_{undetect2}$ (P_{v2fp}) each. Note that the live migration may fail with probability $(1 - \lambda_{migs})$. When the VM migration fails, transition t_{vmigf} and t_{vpref} are fired and the token arrives at P_{vmigf}. Then, the token is deposited in p_{v1up} by firing the transition T_{v1new} when the VMM1 is in UP or FP state. Otherwise, if the token is deposited in P_{v2up} (VM is migrated from its UP state) or $P_{undetect2}$ (VM is migrated from its UNDETECT state) or P_{v2fp} (VM is migrated from its FP state), VMM1 rejuvenation starts. On the contrary, when the token is deposited in P_{vpre}, P_{vmig}, $P_{vudback}$ ($P_{vfpback}$) and P_{vback}, the VMM1 cannot be rejuvenated, since VMM1 is still being used and the migration process has not completed yet.

5. Numerical Analysis and Discussions

We use the Stochastic Petri Net package (SPNP) tool [31] to carry out numerical analysis of the nine policies models. The Gauss-Seidel method [33] is used to improve computation precision. Model parameter values are set according to the existing related literature [20]-[24] and summarized in Table 2. In the following, we analyze *steady-state availability*, *downtime*, and parameter sensitivity of the system under aforementioned policies. Note that *steady-state availability* is often specified as a number of nines in Service Level Agreement (SLA) documents as a marketing feature [37]. However, using the ‘nines’ has been in question because it could not appropriately reflect the variations of *steady-state availability* with its time of occurrence [37]. Moreover, it is hard to apply the number of nines in modeling and formula. Therefore, as in the existing research papers, this paper uses the probability to denote *steady-state availability*.

5.1. Steady-State Availability Analysis

This section performs numerical analysis by using default settings. Table 7 shows the steady-state availability (SSA) for each policy. As expected, we observe that the more HA mechanisms deployed, the higher the VM availability is. In the sequel, we investigate the effect of VMM clock interval on SSA.

We first compare NNNN, TNNN, and TNFN policies. As Table 7 indicates, NNNN policy with no rejuvenation for VMM and VM achieves the lowest SSA of 0.97168871533. When VMM time-based rejuvenation mechanism is considered (TNNN policy), SSA improves to 0.989856441223 and further to 0.991734992389 with VM failover mechanism for VM (TNFN policy). Note that the significance of the SSA improvement depends on the concrete scenario where it is used. For example, the formula for calculating Downtime (measured in hours) is normally (1-SSA)·8760. Here, 8760 is the number of hours of 365 days. In this scenario, the SSA improvement 0.018167726 from NNNN to TNNN becomes significant.

We now vary the VMM clock interval from 10 hours to 300 hours. Note that this variation has no effect on SSA of NNNN policy but effectively affects to SSA of TNNN and TNFN policies. Figure 8 plots the SSA of both TNNN and TNFN policies while varying the VMM clock interval from 10 hours to 300 hours. Based on our results, we observe that:

(1) SSA of both TNNN and TNFN policies are improved with the increasing interval of VMM clock when the interval is small. For example, less than 90h. This is because the frequently VMM rejuvenation leads to VM shut down more often, and then makes the VM SSA lower.

(2) SSA of both TNNN and TNFN policies stops increasing and starts to decrease when the VMM clock interval reaches a certain value. In our numerical analysis, this value is 260 hours for TNNN policy and 160 hours for TNFN policy. The reason behind this fact is that the less frequent rejuvenation, the more frequent VMM software aging-related failure occurrence. Accordingly, the SSA of VM decreases due to the close dependencies between VMM and VM, as mentioned in Section 3.2.

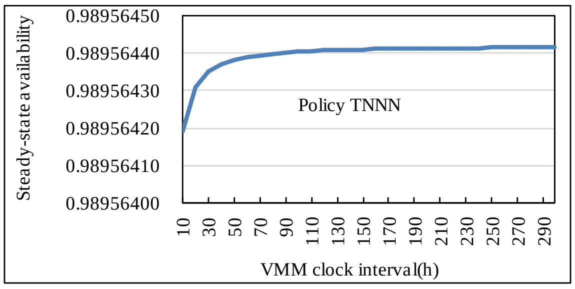
(3) Failover helps to improve SSA. When the VMM clock interval is set to 10 hours, there is an obvious SSA improvement.

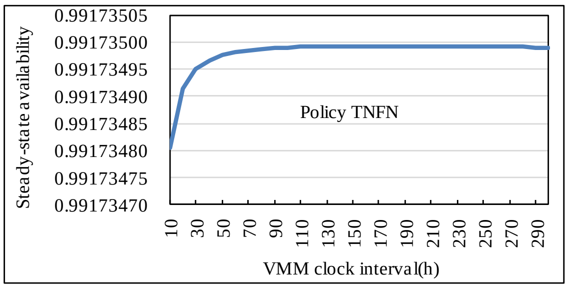
We then compare the failover mechanism with live VM migration by comparing TNFN, TNNM, and TNFM policies. Results are shown in Figure 9. Note that the results of TNFN policy (see Figure 8) are also depicted in this figure to highlight the difference between TNFN and TNNM policies. We observe that the SSA of TNNM policy increases from 0.989520273 to 0.989847214 when the VMM clock interval ranges from 5 hours to 15 hours. The reason is that frequent VMM rejuvenation leads to frequent VM migration and hence, the SSA decreases due to VM migration failures. These results suggest to migrate VM so often can be counterproductive. But as the VMM clock interval further increases, SSA under TNNM decreases and approximates to SSA under TNNN policy. The reason is that the chance of triggering a live VM migration caused by VMM rejuvenation is reduced when the VMM clock interval is large.

Table 7. Steady-state availability (SSA) of the system for each policy under given parameters

Policy	NNNN	TNNN	TNFN	TNNM	TTNM
SSA	0.97168871533	0.98956441223	0.991734992389	0.989617052647	0.998677663722

Policy	TPNM	TNFM	TTFM	TPFM
SSA	0.999436751527	0.998314139248	0.99968371	0.999927083





(a) (b)

Figure. 8. Steady-state availability of (a) TNNN and (b) TNFN policies

It is also observed that the SSA of TNFM policy increases from 0.997935120797 to 0.998092670444 when the VMM clock interval ranges between 20 to 60 hours, since less VMM rejuvenation leads to less VM migration failure occurrences and thus, there is a less effect of migration failures. Similarly, when the VMM clock interval is larger than 60 hours, the SSA of TNFM policy decreases and even becomes smaller than under small VMM clock interval, but still outperforms to TNFN and TNNM policies. The reason is that when the VMM rejuvenation is less frequent the VMM may fail easily due to VMM software aging. Hence, the VM has to shut down due to VMM aging failure leading to a decreasing of SSA.

These results also indicate that failover mechanism performs better than live VM migration mechanism, in terms of improving SSA. Recall that TNFN policy with failover mechanism reduces downtime caused by VM aging or non-aging Mandelbug-related failures, but however, VM is shut down when VMM fails or rejuvenates. Hence, this leads to great transition loss similar to cold-VM rejuvenation. TNNM policy using live VM migration reduces the downtime caused by VMM rejuvenation, but downtime caused by VM aging or non-aging Mandelbug related failures cannot be overcome. On the contrary, TNFM policy outperforms both TNNM and TNFN policies by using failover and live VM migration mechanisms together. In this way, we avoid downtime caused by VM aging and non-aging Mandelbug-related failures and also minimize the impact of VMM rejuvenation. As summary, TNFM policy outperforms the others, in terms of SSA.

Finally, we investigate the effect of VM rejuvenation mechanisms on SSA using TNNM, TTNM, and TPNM policies. Figure 10 plots the SSA of these policies while varying the VMM clock interval, assuming only live VM migration enabled. Both VM time-based and prediction-based rejuvenation mechanisms improve SSA compared with no rejuvenation. In addition, the results show that predication-based mechanism outperforms time-based mechanism under high software aging detection probability.

Figure 11 plots the SSA of these policies while varying the VMM clock interval, assuming both VM failover and live migration are deployed. These results confirm that: (1) VM rejuvenation is an effective way to improve the SSA; and (2) predication-based mechanism performs better than time-based mechanism. In addition, these results indicate that the SSA of TTFM and TPFM policies increase slowly as long as the VMM clock interval increases. However, after 260 hours, the value of VMM clock interval has little impact on the SSA of these policies (Figure 12 shows the precise variation of SSA). Figure 11 also shows how the SSA of TNFM policy decreases as the VMM clock interval increases, similar to the previous case (see Figure 11). As shown in Figure 10 and Figure 11, the combination of failover and live VM migration mechanisms performs better than when individual mechanisms are applied.

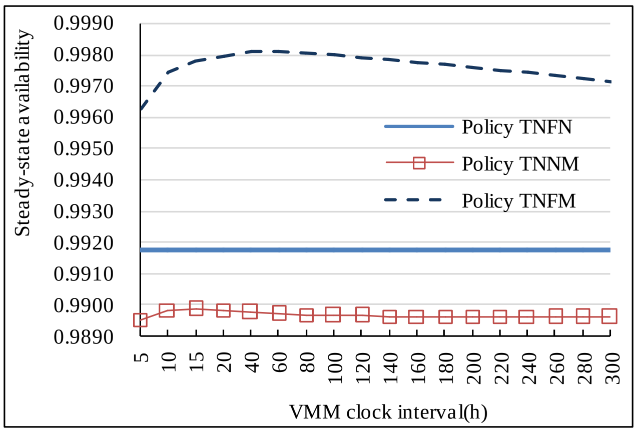


Figure. 9. Steady-state availability of TNFN, TNNM and TNFM policies by varying the VMM clock interval

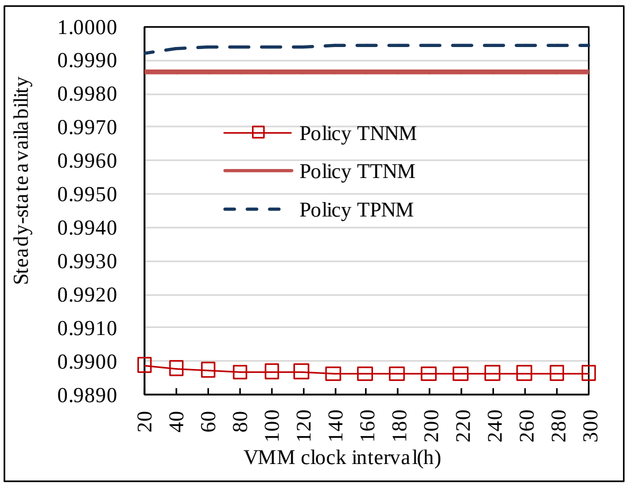


Figure. 10. Steady-state availability of TNNM, TTNM and TPNM policies by varying the VMM clock interval

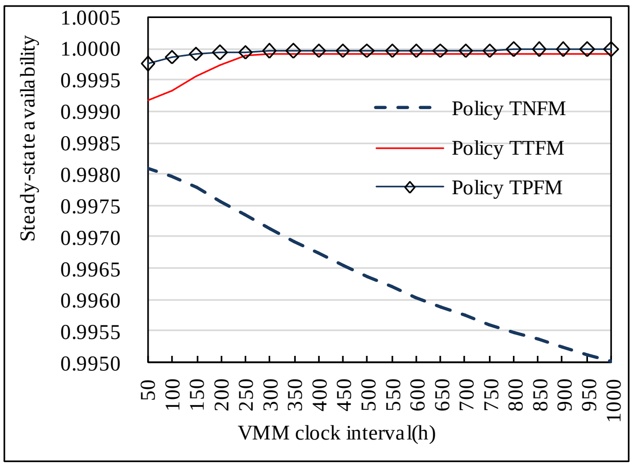


Figure. 11. Steady-state availability of TNFM, TTFM and TPFM policies by varying the VMM clock interval

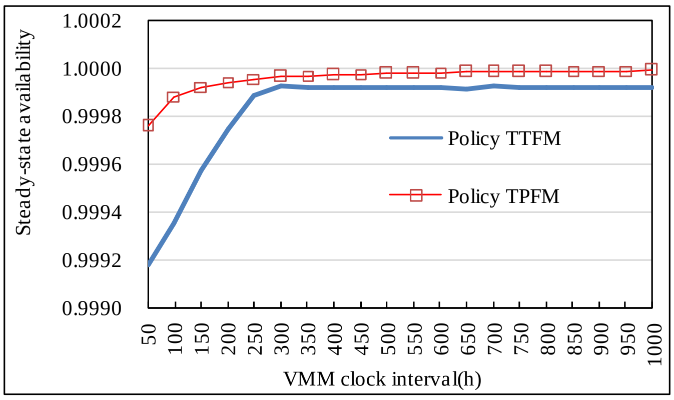


Figure. 12. Detail of steady-state availability of TTFM and TPFM policies by varying the VMM clock interval

5.2. Downtime Analysis

Figure 13 shows the downtime per year of the nine policies. Downtime is measured in the order of hours. Results show that the best policy is TPFM, achieving a downtime near to 40 minutes, while the worst policy is NNNN that achieves a downtime about 248.07 hours. Using failover mechanism reduces the downtime caused by VM non-aging Mandelbug-related or aging-related failures in active VM, since it is less transaction loss as the tasks reload to standby VM quickly. Similarly, using live VM migration technique can also reduce the downtime caused by the VMM rejuvenation by migrating the active VM to other host, while the active VM can nearly keep running due to pre-copy policy.

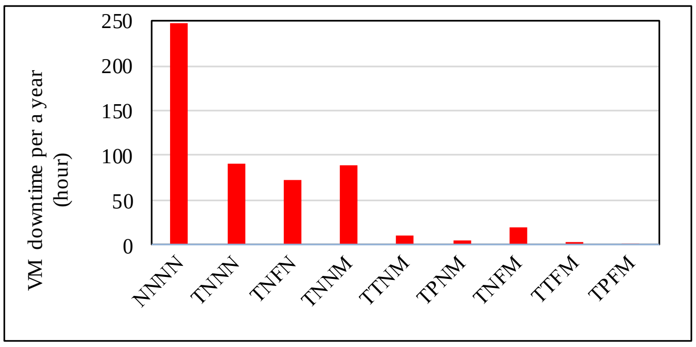


Figure. 13. Downtime per year for the nine policies

5.3. Sensitivity Analysis

Previous results show that software rejuvenation, failover, and live migration techniques are effective HA mechanisms for improving VM availability. Parameter values can, however, produce influence on model results as Section 5.1 shown by the effect of VMM clock interval. In this section, we aim to perform the sensitivity analysis in terms of live migration successful probability, VM rejuvenation clock interval, aging detection probability, VM and VMM aging rates. This analysis becomes critical to choose an appropriate combination of these HA techniques.

5.3.1 Live migration successful probability

We first remove VM rejuvenation and failover mechanisms to investigate how probability of a successful live VM migration affects SSA. We then add VM rejuvenation considering time-based and prediction-based VM rejuvenation. These results are shown in Figure 14. We notice that the larger live VM migration successful probability, the higher SSA is. In addition, we observe that prediction-based still outperforms time-based VM rejuvenation.

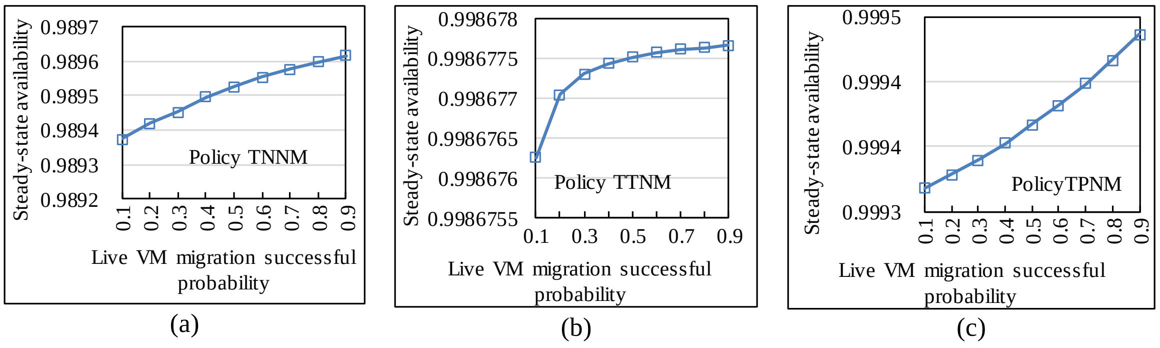


Figure. 14. SSA of (a) TNNM, (b) TTNM and (c) TPNM policies by varying the probability of a live VM successful migration

5.3.2 VM clock interval

We now further examine the effect of other parameters. Figure 15 depicts how the VM clock interval, ranging from 10 to 200 hours, affects SSA in TTNM policy. SSA increases from 0.997940680365 to 0.998704632303 when VM clock interval ranges from 10 to 40 hours. Note that in TTNM policy without failover mechanism, frequent VM rejuvenation may lead to more transaction loss and hence, the SSA will decrease. Similarly, SSA starts to decrease when the VM clock interval is bigger than 40 hours since by increasing VM clock interval, the VM fails more easily due to VM software aging before its rejuvenation. Hence, the appropriate settings of the VMM and VM rejuvenation triggering intervals will maximize the VM SSA.

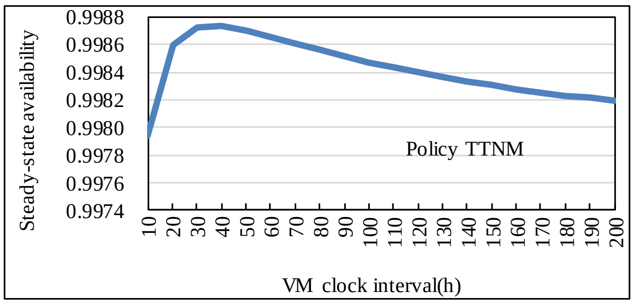


Figure. 15. Steady-state availability of TTNM policy by varying the VM clock interval

5.3.3 Aging detection probability

Figure 16 describes SSA of TPNM policy by varying aging detection probability. As expected, analytical results show that a higher aging detection probability leads to a higher SSA. In summary, we observe that a proper combination of VMM and VM rejuvenation becomes useful to gain high levels of SSA.

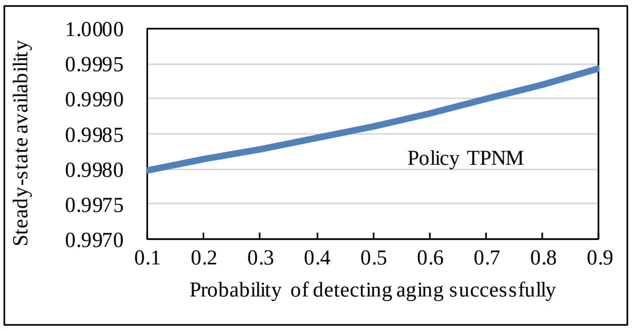


Figure. 16. Steady-state availability of TPNM policy by varying the probability of aging detection

5.3.4 VM and VMM aging rates

This section shows the effects of both VM and VMM aging rates on SSA. All VMs on the same VMM share the physical memory. Thus, both VM and VMM aging rates depend on the workload running on the system, in terms of VMs and the software running on them. Thus, this section results could reflect the effect of varying workload on the capability of each policy. The above numerical analysis results validate the ability of predication-based VM rejuvenation mechanism. Thus, we focus on investigating NNNN, TNNN, TNFN, TNNM, TPNM, TNFM, and TPFM. Numerical analysis is carried out by varying VM aging time from 2 days to 12 and varying VMM from 15 days to 45 days, respectively. Figure 17(a) and Figure 18 (a) show the results. Figure 17(b) and Figure 18 (b) detail the variation of steady-state availability under TPFM, TNFM and TPNM. These results confirm the conclusion of Section 5.1 about the capability of each policy. We could observe that

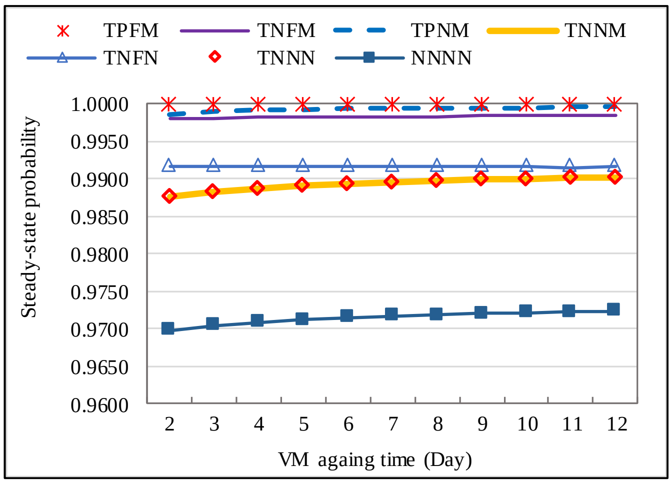
(1) VMM time-based rejuvenation mechanism significantly improves VM steady-state availability.

(2) Failover mechanism works better than live migration mechanism, verified by Figure 9 results.

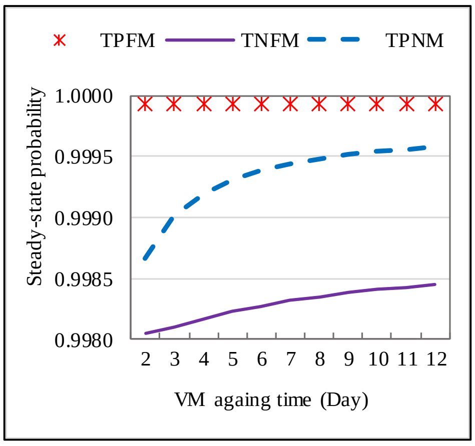
(3) VM predication-based rejuvenation technique could further improve steady-state probability.

(4) The SSA improvement from TNNM to TPNM is larger than from TNNM to TNFM, suggesting probability of detecting aging successfully is more important than failover mechanism parameter in terms of improving SSA.

(5) The SSA improvement from TNNN to TNNF is larger than from TNNN to TNNM, suggesting failover mechanism parameter is more important than probability of successful live VM migration in terms of improving SSA.

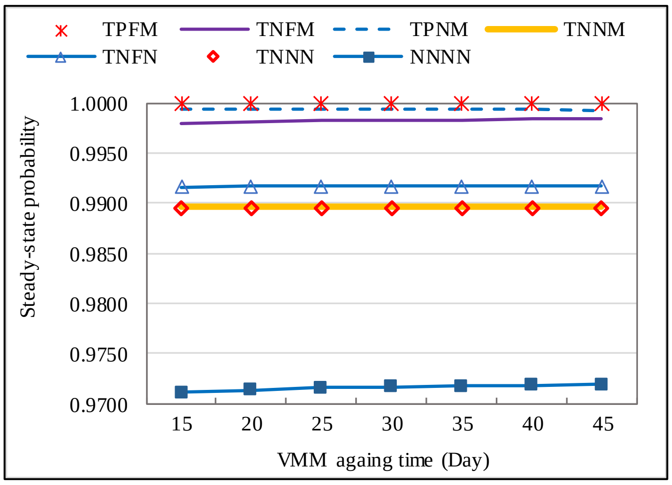


(a)

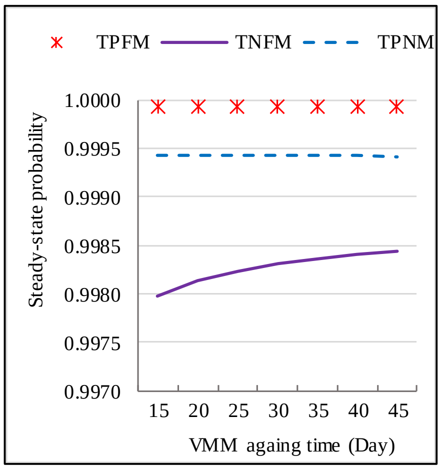


(b)z

Figure. 17. Steady-state availability under different policies by varying the VM aging time



(a)



(b)

Figure. 18. Steady-state availability under different policies by varying the VMM aging time

6. Conclusions and Future Work

This paper applies Stochastic Reward Nets to analyze the effect of diverse high-availability techniques on the VM steady-state availability in a virtualized system composed of three components (a main host and a backup host containing a VMM and VMs, respectively, and a management host) connected to a shared storage, under a variety of failures. In particular, we consider main host, backup host, shared storage, VMM, and VM failures. As high-availability techniques, in this paper we study VM and VMM time-based and prediction-based rejuvenation, VM failover, and live VM migration. Our numerical results indicate that: (i) VMM clock interval is a critical factor for the ability of live VM migration technique in improving VM SSA; (ii) the combination of failover mechanism with live VM migration significantly improve VM availability; and (iii) VM prediction-based rejuvenation outperforms VM time-based rejuvenation, in terms of steady-state availability.

In this paper, we focus on the VM steady-state analysis. It is known that transient availability evaluation is important for a highly dependable system in some cases. We plan to expand the proposed models to analyze the VM survivability when a system failure occurs. It is noticed that, as the existing work on analyzing the availability of a virtualized system, this paper considered a simple system aging model, namely exponential distribution with fixed rate. However, the aging phenomena of a complex system is a result of a variety of factors, including memory utilization increase due to memory leaks, resources saturation, and error accumulation, etc.. Thus, we plan to extend the proposed modeling and analysis to the virtualized system with a more realistic aging process. In addition, we will consider multiple instances of standby and active VM running in the same active host, as well as multiple instances of standby VM maintained in the same backup host. Moreover, we will use these analysis results to design an automatic parameter setting mechanism for finding the appropriate values to maximize SSA while minimizing the cost of each HA technique implementation and deployment.

Acknowledgements

The research of Xiaolin Chang was supported by National Natural Science Foundation of China (No. 61572066). The research of Ricardo J. Rodríguez was supported in part by the EU Horizon 2020 research and innovation program under grant agreement no. 644869 (DICE), by Spanish MINECO project CyCriSec (TIN2014-58457-R), and by University of Zaragoza and Centro Universitario de la Defensa under project number UZCUD2016-TEC-06.

References

[1] Ruest, N. and Ruest, D. (2009) Virtualization, A Beginner's Guide. 1 Edition. McGraw-Hill Education.

[2] Grottke, M. and Trivedi, K.S. (2007) Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate, Computer, 40(2), 107-109.

[3] Grottke, M., Matias, R. and Trivedi, K.S. (2008) The fundamentals of software aging. Proceedings of the 2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp), pp. 1-6.

[4] Carrozza, D., Cotroneo, D., Natella, R., Pecchia, A. and Russo, S. (2010) Memory leak analysis of mission-critical middleware. Journal of Systems and Software, 83 (9), 1556-1567.

[5] Tankard, C. (2011) Advanced Persistent threats and how to monitor and deter them. Network Security 2011, 8, 16-19.

[6] http://www.stratus.com/assets/aberdeen-maintaining-virtual-systems-uptime.pdf.

[7] Mina, N., Maria, T. and Ferhat, K. (2016) Availability in the cloud: state of the art. J. Network and Computer Applications, 60, 54-67.

[8] VMWare, Automating High Availability (HA) Services with VMware HA, Tech. rep., VMWare, Inc., available at https://www.vmware.com/pdf/ vmware_ha_wp.pdf (accessed November 21, 2016) (2006).

[9] Kanoun, K., Borrel, M., Morteveille, T. and Peytavin, A. (1999) Availability of CAUTRA, a subset of the French air traffic control system. IEEE Transactions on Computers, 48 (5), 528-535.

[10] Ramani, S., Trivedi, K.S. and Dasarathy,B. (2000) Performance analysis of the corba event service using stochastic reward nets. Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 238-247.

[11] Leangsuksun, C., Shen, L., Liu, T., Song, H. and Scott, S.L. (2003) Availability prediction and modeling of high mobility OSCAR Cluster. Proceedings of the 2003 IEEE International Conference on Cluster Computing, pp. 380-386.

[12] Lanus, M., Yin, L. and Trivedi, K.S. (2003) Hierarchical composition and aggregation of state-based availability and performability Models. IEEE Transactions on Reliability, 52 (1), 44-52.

[13] Longo, F., Ghosh, R., Naik, V.K. and Trivedi, K.S. (2011) A scalable availability model for Infrastructure-as-a-Service cloud. Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN), pp. 335-346.

[14] Trivedi, K.S., Andrade, E. and Machida, F. (2012) Chapter 1 - Combining Performance and Availability Analysis in Practice, in: A. Hurson, S. Sedigh (Eds.), Dependable and Secure Systems Engineering, Vol. 84 of Advances in Computers, Elsevier, 1-38.

[15] Thein, T. and Chi, S.D. Park, J.S. (2007) Availability Analysis and Improvement of Software Rejuvenation Using Virtualization, Economics and Applied Informatics, (1), 5-14.

[16] Thein, T., Chi, S.D. and Park, J.S. (2008) Improving Fault Tolerance by Virtualization and Software Rejuvenation. Proceedings of the 2nd Asia International Conference on Modeling Simulation (AICMS), pp. 855-860.

[17] Thein, T. and Chi, S.D. Park, J.S. (2008) Availability modeling and analysis on virtualized clustering with rejuvenation. International Journal of Computer Science and Network Security, 8 (9), 72-80.

[18] Melo, M., Maciel, P., Araujo, J., Matos, R. and Ara´ujo, C. (2013) Availability study on cloud computing environments: live migration as a rejuvenation mechanism. Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1-6.

[19] Bruneo, D., Distefano, S., Longo, F., Puliafito, A. and Scarpa, M. (2013) Workload-Based Software Rejuvenation in Cloud Systems. IEEE Transactions on Computers, 62 (6), 1072-1085.

[20] Kim, D.S., Machida, F. and Trivedi, K.S. (2009) Availability modeling and analysis of a virtualized system. Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp.365-371.

[21] Nguyen, T.A., Kim, D.S. and Park, J.S. (2014) A comprehensive availability modeling and analysis of a virtualized server system using stochastic reward nets, The Scientific World Journal 2014, 18.

[22] Rezaei, A. and Sharifi, M. (2010) Rejuvenating high available virtualized systems. Proceedings of the International Conference on Availability, Reliability, and Security 2010 (ARES), pp.289-294.

[23] Xu, J., Li, X., Zhong, Y. and Zhang, H. (2014) Availability modeling and analysis of a single-server virtualized system with rejuvenation, Journal of Software, 9(1), 129-139.

[24] Machida, F. and Kim, D.S. Trivedi, K.S. (2013) Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration, Performance Evaluation, 70 (3), 212-230.

[25] Nguyen, T.A., Kim,D.S. and Park, J.S. (2016) Availability modeling and analysis of a data center for disaster tolerance. Future Generation Computer Systems, 56, 27-50.

[26] Grottke, M., Li, L., Vaidyanathan, K. and Trivedi, K.S. (2006) Analysis of software aging in a web Server. IEEE Transactions on Reliability, 55 (3), 411–420.

[27] Alonso, J., Bovenzi, A., Li, J., Wang, Y., Russo, S. and Trivedi, K.S. (2012) Software Rejuvenation: Do IT & Telco Industries Use It? Proceedings of the IEEE 23rd International Symposium on Software Reliability Engineering

Workshops (ISSREW), pp. 299-304.

[28] Stephen, A.H. (2010) Systems research and development at VMware. Operating Systems Review, 44(4), 1-2.

[29] Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C. and Pratt, I. Warfield, A. (2005) Live migration of virtual machines. Proceedings of the 2^{nd} Conference on Symposium on Networked Systems Design & Implementation-Volume 2, NSDI'05, USENIX Association, Berkeley, CA, USA, pp. 273-286.

[30] Machida, F., Kim, D.S., Park, J.S. and Trivedi, K.S. (2008) Toward optimal virtual machine placement and rejuvenation scheduling in a virtualized data center. Proceedings of the 2008 IEEE International Conference on Software Reliability Engineering Workshops, pp. 1-3.

[31] Ciardo, G., Muppala, J. and Trivedi, K.S. (1989) SPNP: stochastic Petri net package. Proceedings of the 3rd International Workshop on Petri Nets and Performance Models (PNPM), pp. 142-151.

[32] Muppala, J., Ciardo, G. and Trivedi, K.S. (1994) Stochastic Reward Nets for Reliability Prediction. Communications in Reliability, Maintainability and Serviceability, 1 (2), 9-20.

[33] Hazewinkel, M. (Ed.) Encyclopaedia of Mathematics Springer, 1994.

[34] Grottke, M. and Trivedi, K.S. (2005) A classification of software faults. The Journal of Reliability Engineering Association of Japan, vol 27, 425-438.

[35] Ghosh R., Francesco, L., Flavio, F., Stefano, R. and Trivedi, K.S. (2014) Scalable analytics for IaaS cloud availability. IEEE Trans. Cloud Computing 2(1), 57-70.

[36] Marcello, C., Domenico, C., Flavio, F. and Stefano, R. (2016) To Cloudify or Not to Cloudify: The Question for a Scientific Data Center. IEEE Trans. Cloud Computing 4(1), 90-103.

[37] Marcus, E. The myth of the nines, [Online; accessed on November 21,2016], available at http://searchstorage.techtarget.com/tip/The-myth-of-the-nines (Aug. 2003).

[38] Wang, D., Xie, W. and Trivedi, K.S. (2006) Performability analysis of clustered systems with rejuvenation under varying workload. Performance Evaluation (64), 247-265.