Research on rejuvenation analytical models for a virtualized system with live VM migration

Yi Zhong*, Jian Xu, Jing Zhong, Fengyu Liu

[1] *School of Computer Science and Engineering Nanjing University of Science and Technology, Nanjing, Jiangsu, China, 210094*

Received 6 October 2013,www,cmnt.lv

Abstract

With the widespread use of virtualization technology, availability of Virtual Machines (VMs) in server virtualized systems became an important issue. In aspect of availability of virtualized systems, software rejuvenation is a favourable technology, which can postpone or prevent failures caused by software aging in VMs and underlying Virtual Machine Monitor (VMM). During the rejuvenation of the VMM, live VM migration can further improve availability of virtualized systems. In this paper, we will analysis rejuvenation process of virtualized system with live VM migration, propose a time and load based rejuvenation analytical model in Stochastic Reward Net (SRN) to describe state change during system rejuvenation, find the optimum combinations of rejuvenation trigger intervals that maximize the availability of VM, and do a experience to analyze and compare the model in this paper and the time based model. The experience result shows the time and load based rejuvenation analytical model is better than the time based model in respect of system availability and throughput rate, and is more stable in face of the dynamic change of system load.

Keywords: Virtual Machine (VM); Virtual Machine Monitor (VMM); Software Aging; Software Rejuvenation; VM Migration; Stochastic Reward Net (SRN)

1 Introduction

As a common software and hardware resources sharing framework, cloud computing gets very fast growing due to it can provide high available and high performance computing and storage service to both Internet and intranet users recently. Server virtualization is the fundamental part of cloud computing to support dynamic volume users' request via virtual machines (VMs) whenever necessary. To assure the high quality service to win users' satisfaction on their business, how to keep high-availability of VMs running on server virtualized systems becomes a common concern to many organizations, service providers and users.

After long online execution, because of age-related bugs, which are difficult to find and remove during developing and testing, software system may meet memory leaking, unreleased file lock and data corruption, until the whole system failure or crashes [1]. Software rejuvenation is one of common used techniques to assure high-availability of server virtualized systems by rebooting or resetting software environment to clean internal error state for continuing service [2, 3]. In a virtualized system, multi-VMs are hosted on hypervisor or virtual machine monitor (VMM) for resource management and monitoring. When VMM fails or suspends for aging issue, all the hosted VMs will have to halt, so software rejuvenation is both applicable to VMM and VMs. To minimize the down time cost, live VM migrate are widely adopted which means live migrate a running VM to another physical machine without client application awareness. In this paper, we will model and describe software rejuvenation process for server virtualized systems with live VM migration by a stochastic reward net (SRN) rejuvenation model, and analyze the relationship between different composite live VM migration actions and system performance matrix.

The rest of this paper is organized as following: Section 2 lists existing different rejuvenation policies and their corresponding implementation models. Section 3 introduces a time and load based analytical model for a virtualized system with migration-VM rejuvenation model in SRN. To validate the technique, the analysis results on system steady-state availability, sensitivity and system throughput rate are presented in Section 4. Finally section 5 concludes this paper and shows the further work.

2 Related work

VMM rejuvenation for server virtualized system was firstly introduced in [4]. If VMM rejuvenation is directly performed on a host, the execution environments of all the hosted VMs are cleared because they are running on a VMM. So based on the different approaches on hosted VMs, VMM rejuvenation techniques can be divided into three categories [3, 9]: Code-VM rejuvenation, Warm-VM rejuvenation and Migrate-VM rejuvenation.

*Corresponding author's E-mail: zhongyi@njust.edu.cn

- Code-VM Rejuvenation: Before triggering VMM rejuvenation, the simplest way to handle the hosted VMs is to just shut down all of them regardless of their execution states. Those VMs will be restarted in clean environment after the VMM rejuvenation. While it forces shutdowns of the hosted VMs which may bring potential risks. In case some VMs are still in running state, all the transactions running on them are lost by the Cold-VM rejuvenation. An advantage of the Cold-VM rejuvenation, however, is that the rejuvenation action clears all the aging states of the VMs in addition to the aging states of the VMM.
- Warm-VM Rejuvenation: All the hosted VMs are suspended and the corresponding execution states are archived before VMM rejuvenation is triggered. Then after the VMM rejuvenation, all VMs are resumed back to origin states. Warm-VM reboot enables a VMM to suspend hosted VM execution in memory and resume the execution after VMM rejuvenation [4]. Due to the execution states of VMs are kept in memory instead of persistent storage, it can fast the recovery VM quickly without transactions loss risk and the overall down time are greatly reduced.
- Migrate-VM Rejuvenation: It's a combination rejuvenation, which combines VMM rejuvenation and live VM migration in a fitful way. Live VM migration keeps services on a VM continue working and migrates the VM among different physical machines. Using live VM migration, a hosted VM are moved to another before VMM rejuvenation. Once VMM completes rejuvenation, the VM could choose to return back to the original host or continue stay on current one. In such rejuvenation method, the VM can still continue the execution even the original VMM is being rejuvenated. However, the aging states in the hosted VMs are not cleared by the VMM rejuvenation. Meanwhile it only works when the migration target server has a capacity to accept the migrated VMs.

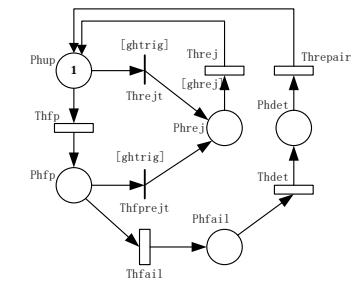
In order to completely analyze the availability of a server virtualized system, D. Kim et al. leveraged a hierarchical stochastic model to present the system architecture and the details of failure/recovery [5]. T.Thein, et al. built a continuous time-Markov chain (CTMC) for time-based periodical rejuvenation for VMs on a virtualized system which included two physical servers and clustered VMs, but VMM rejuvenation is not covered [6]. A.Rezaei, et al. added VMM failure and VMM rejuvenation in the availability model for a virtualized system [7]. Melo et al. introduced the Stochastic Petri Net (SPN) model for time-based rejuvenation in cloud computing, so as to evaluate the impacts to system availability on different VM migration policies [8]. Mechida.F, et al. proposed time-based analytical models in SRN for virtualized system with cold-VM rejuvenation and warm-VM rejuvenation respectively, and gave the comparisons of these models [3]. In [9], Mechida. F, et al. further considered the effects of the failures of the target host and VM migration process, proposed and analyzed a time-based model for virtualized systems with migrate-VM rejuvenation with emphasis, compared availability of the VM on the type of live VM migration (stop-and-copy or pre-copy) and the policy for migration back to the original host after VMM rejuvenation (return-back or stay-on).

A time and load based analytical model was firstly proposed by Garg S, et al.[10]. They assumed system may have different system loading in different software rejuvenation time window, so the down time cost and transactions loss numbers varied a lot during rejuvenation. Y.Bao, et al. consolidated time-based and measure-based approaches into rejuvenation framework with additional consideration for load and resource leak indexes [11,12]. K.Vaidyanathan, et al. simplified his model to be a semi-Markov load model and tried to find out the best rejuvenation schedule with maximum availability and minimum down time cost [13]. In our previous work, we constructed a rejuvenation analytical model for a single-server virtualized system with a combinatory rejuvenation technique that uses a time-based policy for a VMM and a measurement-based policy for VMs [14].

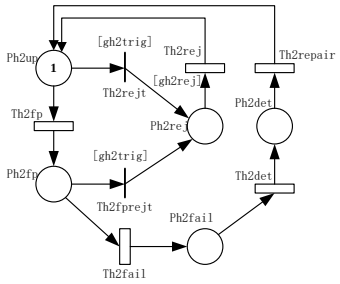
3 Time and load based analytical model for a virtualized system with Migrate-VM rejuvenation

As one extended branch of SPN, SRN makes system's steady-state availability can be measured by defining the corresponding reward functions, and has quite a few enhancements such as guard functions, transition priorities, variable cardinality arcs, and so on, so SRN has been wide adopted in rejuvenation modeling. We leverages SRN models multiple services' environment with live VM migration, so as to assure the system still keep service online by means of transiting the corresponding live VMs to another available host during VMM rejuvenation.

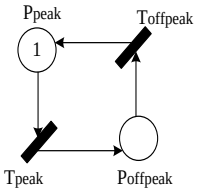
In this paper, for some graphical representation, we have conventions as follows: a narrow bar for an immediate transition, an empty rectangle for a time transition, a filled rectangle for a deterministic transition and # for token numbers.



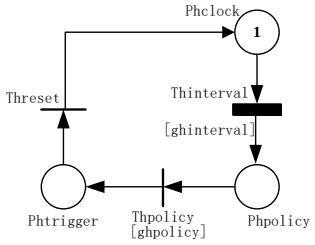
(a) VMM1 model



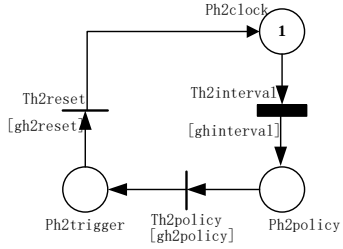
(b) VMM2 model



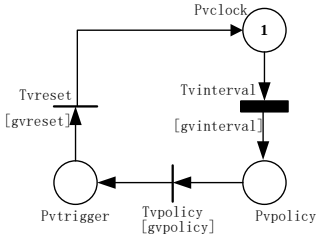
(c) Peak hour model



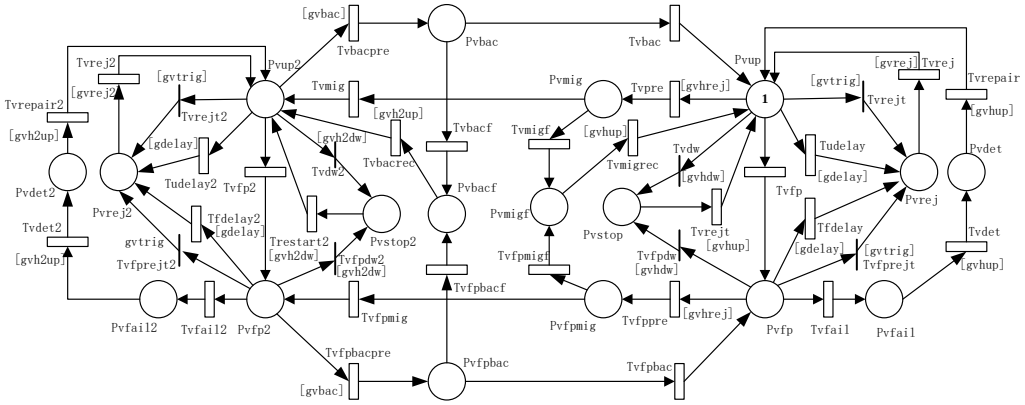
(d) VMM1clock model



(e) VMM2 clock model



(f) VM clock model



(g) VM model

FIGURE 1 Time and load based analytical model for a virtualized system with Migrate-VM rejuvenation

TABLE 1 Guard functions

Guard	Definition	Guard	Definition
g_{vtrig}	if(#$P_{vtrigger}$==1&&# $P_{offpeak}$==1) 1 else 0	$g_{vinterval}$	if(#P_{vup}==1||#P_{vfp}==1||#P_{vup2}==1||#P_{vfp2}==1) 1 else 0
g_{delay}	if(#$P_{vtrigger}$ ==1&&# P_{peak}==1) 1 else 0	$g_{vpolicy}$	if(#P_{vup}==1||#P_{vfp}==1||#P_{vup2}==1||#P_{vfp2}==1) 1 else 0
g_{vrej}	if(#P_{vclock}==1&&(#P_{hup}==1||#P_{hfp}==1)) 1 else 0	g_{vrej2}	if(#P_{vclock}==1&&(#P_{h2up}==1||#P_{h2fp}==1)) 1 else 0
g_{vreset}	if(#P_{vrej}==1||#P_{vrej2}==1) 1 else 0	g_{vbac}	Return-back: if(#P_{hup}==1||#P_{hfp}==1) 1 else 0
g_{vhrej}	if(#$P_{hpolicy}$==1&&(#P_{h2up}==1||#P_{h2fp}==1)) 1 else 0		Stay-on: if(#$P_{h2policy}$==1&&(#P_{hup}==1||#P_{hfp} ==1))1 else 0
g_{vhup}	if(#P_{hup}==1||#P_{hfp}==1) 1 else 0	g_{vh2up}	if(#P_{h2up}==1||#P_{h2fp}==1) 1 else 0
g_{vhdw}	if(#P_{hfail}==1) 1 else 0	g_{vh2dw}	if(#P_{h2fail}==1) 1 else 0
$g_{hinterval}$	if(#P_{hup}==1||#P_{hfp}==1) 1 else 0	$g_{h2interval}$	if(#P_{h2up}==1||#P_{h2fp}==1) 1 else 0
$g_{hpolicy}$	if(#P_{vup}==0&&#P_{vfp}==0&&#P_{vmig}==0&&#P_{vfpmig}==0&&#P_{vbac}==0&&#P_{vfpbac}==0) 1 else 0	$g_{h2policy}$	if(#P_{vup2}==0&&#P_{vfp2}==0&&#P_{vmig}==0&&#P_{vfpmig}==0&&#P_{vbac}==0&&#P_{vfpbac}==0) 1 else 0
g_{hreset}	if(#P_{hrej}==1) 1 else 0	$g_{h2reset}$	if(#P_{h2rej}==1) 1 else 0
g_{htrig}	if(#$P_{htrigger}$ == 1) 1 else 0	g_{h2trig}	if(#$P_{h2trigger}$==1) 1 else 0
g_{hrej}	if(#P_{hclock}== 1) 1 else 0	g_{h2rej}	if(#$P_{h2clock}$==1) 1 else 0

The SRN model comprises seven sub-models as shown in Figure 1: (a) VMM1 model, (b) VMM2 model, (c) Peak hour model, (d) VMM1 clock model, (e) VMM2 clock model, (f) VM clock model and (g) VM model. Initially, VM is hosted on the VMM on host 1 which called VMM1. Before VMM1 is rejuvenated, VM will be live migrated to the VMM on host 2, i.e., VMM2. Both of the two VMMs are rejuvenated either periodically by each individual clock models (VMM1 and VMM2 clock models) or when system load achieved the threshold value (Peak model). In Figure 1 (g), the right side describes behavior of the VM on VMM1, the left side captures the behavior of the VM on VMM2 and the middle is for live VM migration. The corresponding guard functions are listed in table 1.

The process of live VM migration can be illustrated as follows: In the beginning, the VM hosts on VMM1. When the clock for VMM1 requests rejuvenation for VMM1, g_{vhrej} will enable T_{vpre} or T_{vfppre} for live VM migration as long as VMM2 is available (which means a

token is deposited in P_{vup2} or P_{vfp2}). It represents the live VM migration begins. When the live VM migration is finished, a token is deposited in P_{vmig} or P_{vfpmig}. While the live VM migration may have certain probability of failure for some reasons (such as network is temporary down, or target host does not have enough capability to host the VM, or wrong configuration for virtualization…). If the VM migration fails, transition T_{vmigf} or $T_{vfpmigf}$ will be enabled, and then a token arrives at P_{vmigf}. Then, a token is deposited in P_{vup} by firing the transition $T_{vmigrec}$ if VMM1 is available. Otherwise if a token is deposited in P_{vup2} or P_{vfp2} by firing the transition T_{vmig} or T_{vfpmig}, the VM has been successfully migrated to VMM2 and continue servicing. When a token is deposited in P_{vup2} (VM is migrated from its up-state) or P_{vfp2}(VM is migrated from its fail-probable-state) or P_{vmigf} (VM migration failed), VMM1 rejuvenation starts. According to $g_{hpolicy}$ definition in table 1, if a token is deposited in P_{vup}, P_{vfp} , P_{vmig}, P_{vfpmig}, P_{vbac} or P_{vfpbac}, VMM1 is still in use, and can not be rejuvenated.

In peak hour model, a token is deposited in P_{peak} or $P_{offpeak}$ represents the VMM runs at peak load (system load over a threshold value) or off-peak load (system load under a threshold value). The aging speed of a VM at peak load is faster than it at off-peak load. T_{peak} and $T_{offpeak}$ are deterministic because they represent periodical inter-transitions between peak load and off-peak load. By the transition T_{peak}, the token in P_{peak} is removed and a token is deposited in $P_{offpeak}$, while by the transition $T_{offpeak}$, the token in $P_{offpeak}$ is removed and a token is deposited in P_{peak}. When a token is deposited in $P_{vtrigger}$ (in VM clock model) and a token is deposited in $P_{offpeak}$ (in Peak hour model), one of immediate transitions T_{vrejt}, $T_{vfprejt}$, T_{vrejt2} and $T_{vfprejt2}$ will be fired. While if a token is deposited in P_{peak}, one of immediate transitions T_{udelay}, T_{fdelay}, $T_{udelay2}$ and $T_{fdelay2}$ will be enabled, which means VMM1 or VMM2 rejuvenation will start after a time.

After live VM migration and VMM rejuvenation are successfully performed, VM may choose to stay on current host (Stay-on policy) or return back to origin host (Return-back policy). The only difference is whether VM will be back to origin host. If the current host is only for temporary usage or the VM is forced to go back to origin host, then it has to follow return-back policy. Otherwise either of the two policies can be used. In VM model, the guard function g_{vbac} for transitions $T_{vbacpre}$ and $T_{vfpbacpre}$ represents both policies of VM migration. For return-back policy, $T_{vbacpre}$ or $T_{vfpbacpre}$ is enabled after the original VMM is available (VMM in up-state or fail-probable-state). While for stay-on policy, $T_{vbacpre}$ or $T_{vfpbacpre}$ is enabled until the VMM2 rejuvenation is required. Then a token is deposited in P_{vbac} or P_{vfpbac} by either policy. If live VM migration from VMM2 to VMM1 fails, the VM will be restarted on VMM2. Table 1 gives the details of all guard function for migrate-VM rejuvenation, including g_{vbac} implementation both for 'return-back' and 'stay-on' policies.

The clock for VM rejuvenation works independent of where the VM is hosting. When a token is deposited in $P_{vtrigger}$ in the VM clock model, one of the immediate transitions T_{vrejt} , $T_{vfprejt}$, T_{vrejt2}, or $T_{vfprejt2}$ is enabled in VM model. This ensures that the VM rejuvenation is performed on the other hosting server if required when the VM is on VMM2.

4 Experiments

TABLE 2 Default values of transitions in the model

Transition	Description	Default Value
$\mathbf{T_{vfp}}$	aging rate under peak hour	1/3 day^{-1}
	aging rate under offpeak hour	1/7 day^{-1}
$\mathbf{T_{vfail}}$	VM failure rate after aging under peak hour	1/24 h^{-1}
	VM failure rate after aging under offpeak hour	1/3 day^{-1}
$\mathbf{T_{vdet}}$	VM failure detection rate	12 h^{-1}
$\mathbf{T_{vrepair}}$	VM failure recovery rate	2 h^{-1}
$\mathbf{T_{vrej}}$	VM rejuvenation trigger rate	1 day^{-1}
$\mathbf{T_{vrestart}}$	VM restart rate	120 h^{-1}
$\mathbf{T_{vsd}}$	VM shutdown rate	120 h^{-1}
$\mathbf{T_{vpre}}$	VM migration pre-copy rate	90 h^{-1}
$\mathbf{T_{vmig}}$	VM migration rate by pre-copy	3600 h^{-1}
$\mathbf{T_{udelay}}$	VM peakhour delay period at up state	0.5 h^{-1}
$\mathbf{T_{fdelay}}$	VM peakhour delay period at failure-possible state	0.5 h^{-1}
$\mathbf{T_{hfp}}$	VMM aging rate	1/30 day^{-1}
$\mathbf{T_{hfail}}$	VMM failure rate after aging	1/7 day^{-1}
$\mathbf{T_{hdet}}$	VMM failure detection rate	12 h^{-1}
$\mathbf{T_{hrepair}}$	VMM reactive recovery rate	1 h^{-1}
$\mathbf{T_{hrej}}$	VMM rejuvenation rate	30 h^{-1}
$\mathbf{T_{hinterval}}$	VMM rejuvenation trigger rate	1/7 day^{-1}
$\mathbf{T_{vhinterval}}$	VM rejuvenation trigger rate	1 day^{-1}
$\mathbf{T_{peak}}$	peak hour rate	1/12 h^{-1}
$\mathbf{T_{offpeak}}$	off peak hour rate	1/12 h^{-1}

For the SRN model in this paper, we choose SPNP to implement its Markov regenerative process [15]. Except deterministic transitions $T_{hinterval}$, $T_{h2interval}$, $T_{vinterval}$, T_{peak} and $T_{offpeak}$, all other transitions are assumed to be exponentially distributed. All default values of transition rate are listed in Table 2, while other default parameters are listed Table 3.

TABLE 3 Default parameter values in the model

Cv	**VM migration coverage**	**0.9**
$\mathbf{R_1}$	request incoming rate in peak hours	1000 s^{-1}
$\mathbf{R_2}$	request incoming rate in offpeak hours	10 s^{-1}
x	request processing rate per VM	275 s^{-1}

Given parameters listed in Table 2 and Table 3, this section will give detail analysis and comparison on system availability, sensitivity and throughput rate between time-based VM rejuvenation model[9] and our enhanced time and load balance based model.

4.1 OPTIMUM REJUVENATION TRIGGER INTERVALS AND SYSTEM STEADY-STATE AVAILABILITY

System steady-state availability varies for different combination of rejuvenation trigger intervals of the VM and the VMM. Frequent rejuvenation increases the

downtime for too much cost of rejuvenation actions, while low frequent rejuvenation may also increase the system downtime for frequent software failures. Only at a certain trigger interval, the steady-state availability of the VM can be maximized. We use a gradient search algorithm to find the exact point for optimal system performance, so as to get the optimum combination of rejuvenation trigger intervals of VM and VMM [9].

Table 4 presents the definition of reward functions for steady-state availability. Using those functions along with parameters in Table 2, we can compare the two models' system steady-state availability. Then Table 5 summarizes the two models' availability under default parameter sets. Table 6 shows the optimum rejuvenation trigger interval by the gradient search algorithm.

Consolidate data in Table 5 and Table 6, when VM and VMM take the fitful rejuvenation interval, the two models both can achieve their maximum availability. The comparison of steady-state availability between stay-on policy and return-back policy depends on parameter setting. Under default values in this section, our model gets higher availability.

TABLE 4 Reward functions for steady-state availability analysis

Model	Definition
Time based model	if((#P_{vup}==1)||(#P_{vfp}==1)||(#P_{vup2}==1)||(#P_{vfp2}==1))) 1 else 0
Time and load based model	if((#P_{vup}==1)||(#P_{vfp}==1)||(#P_{vup2}==1)||(#P_{vfp2}==1))) 1 else 0

TABLE 5 System steady-state availability with default parameter values

Model	**Time based model**		**Time and load based model**	
	Stay-on	Return-back	Stay-on	Return-back
Steady-state availability	0.996361	0.996533	0.997033	0.996535

TABLE 6 Optimum combinations of rejuvenation trigger intervals

Model		**VM rejuvenation trigger interval (hour)**	**VMM rejuvenation trigger interval (hour)**	**Steady-state availability**
Time based model	Stay-on	20.45	30.72	0.996808
	Return-back	20.95	50.36	0.996771
Time and load based model	Stay-on	110.2	20.32	0.9977
	Return-back	20.85	50.78	0.996782

TABLE 7 VM request processing speed under different system loads

Peak Hours per day(h)	**Time based model**		**Time and load based model**	
	stay on	return back	stay on	**return back**
4	190497	190530.2	190572.8	190533
8	348908.8	348969.7	349094.6	348969.3
12	507178.3	507265.8	507518.8	507266.8
16	665300.8	665415	665843.2	665419.7
20	823269.5	823411.7	824064.3	823425
24	**981073.2**	**981240.8**	**982171.6**	**981272.3**

4.2 MIGRATION SUCCESSFUL RATE

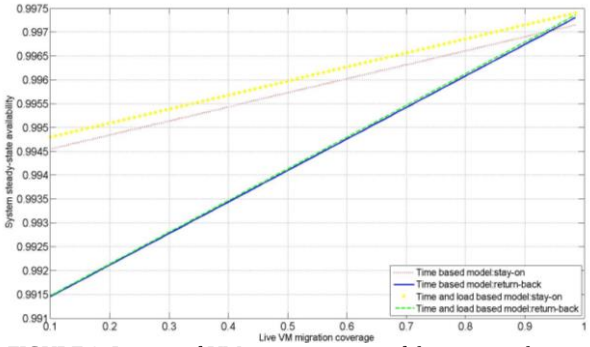


FIGURE 2 Impacts of VM migration successful rate to steady-state availability

Live VM migration failure may cause the down time of systems increase. Figure 2 presents the impact of migration successful rate to steady-state availability. As the increasing of migration successful rate, the steady-state availability will also be improved accordingly. Due to return-back policy requires VM should be migrated to origin host after VMM rejuvenation, it has much stricter requirement. While with high migration successful rate, the stead-state availability of return-back policy has advantage than the stay-on policy. Based on the data in Figure 2, comparing to time-based model, our time and load based model achieves better in overall steady-state availability, sensitivity of successful rate on return-back policy.

4.3 PEAK DURATION PER DAY

Figure 3 presents the impacts of different peak duration per day to system steady-state availability. It's obvious that as the peak duration extends, all the models will has performance drop issue accordingly. The reason is longer peak duration will fast the VM aging and increase the probability of VM failure, as a result the system availability will decrease. Among all the models, time based model with stay-on policy is most sensitive to system load, the system availability will drop sharply when system load increasing. Since our new model already takes system load as one of the major factor, it can reduce the load impact to system availability as much as possible. As showed in Figure 3, our model does not drop so obviously comparing to original time based model. Which means the new model can still keep

high performance and availability under high system load, even for stay-on policy.

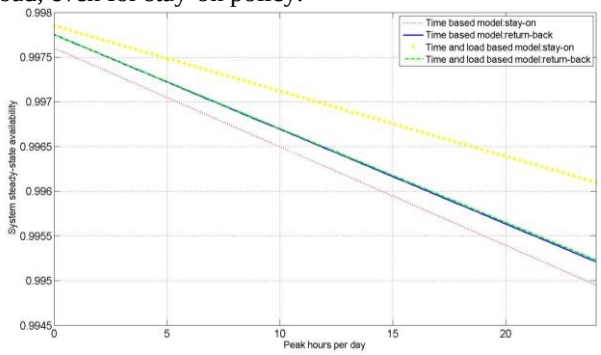


FIGURE 3 Impacts of system load to steady-state availability

4.4 THROUGHPUT RATE

Beside availability measurement factor, the experiment also adds system throughput rate as another performance factor according to system load attribute in our time and load based model. Regarding to definition of system load, the state space Ω will be divided into two parts: Ω_{peak} as system peak load and $\Omega_{offpeak}$ as system off-peak load, then $\Omega=\Omega_{peak} \cup \Omega_{offpeak}$, $\Omega_{peak} \cap \Omega_{offpeak} = \phi$.

Set R1 as the incoming speed of requests during peak time, R2 as as the incoming speed of requests during off-peak time. System throughput rate, as the processed request number per unit time, can be defined as:

$$E[T] = \sum_{i \in \Omega} p(i) \times \min(R(i), x) \quad (1)$$

where x is the speed of processing request for a VM, $p(i)$ is the steady-state probability at system available state i, i.e., a token is deposited in one of P_{vup}, P_{vfp}, P_{vup2}, P_{vfp2}. $R(i)$ is the corresponding R value at state i. If state i is during peak time (a token deposited in P_{peak}), $R(i)$ is R1, otherwise it will be R2.

Table 7 presents the average processed request numbers for two models within one hour under different peak durations. Obviously when the peak duration extends, the system load gets much heavier. The same happens to the request number of each model. When the peak duration is less than 20 hours, our time and load based model can get much better performance. Only when peak duration is more than 20 hours, the time-based model can perform better than ours. While roughly for real system, over 20 hours peak time is very rare.

5 Conclusion

To analyze a virtualized system with Migrate-VM rejuvenation, this paper introduces a new time and load based rejuvenation analytical model in SRN, and then it also presents corresponding experiments to compare our model with existing time based model in system availability, sensitivity and throughput. The result proves our model works much better than time-based one for availability and throughput, and is more stable with the dynamic change of system load. In further research, we will keep analyzing the aging process in real virtual environment, adjusting parameters to make our model is more adapted to changes in the load, and build a load sensitive and adaptive rejuvenation framework, so as to find the best rejuvenation policy which can maximize system availability.

Acknowledgement

This paper is based upon work supported by the National Natural Science Foundation of China (No. 61300053) and the Jiangsu Provincial Natural Science Foundation of China (No. BK2011023).

References

[1] Grottke M, Nikora A P, Trivedi K S 2010 An empirical investigation of fault types in space mission system software *DSN 2010 Conf. on Dependable Systems and Networks* Chicago IL USA June 28-July 1

[2] Cotroneo D, Natella R, Pietrantuono R, Russo S 2010 Software aging analysis of the linux operating system *ISSRE 2010 IEEE International Symposium on Software Reliability Engineering* San Jose CA USA Nov 1-4

[3] Machida F, Kim D, Trivedi K S 2010 Modeling and analysis of software rejuvenation in a server virtualized system *WoSAR 2010 IEEE Second International Workshop on Software Aging and Rejuvenation* San Jose CA USA Nov 2

[4] Kourai K, Chiba S 2007 A fast rejuvenation technique for server consolidation with virtual machines *DSN2007 Int. Conf. on Dependable Systems and Networks* Edinburgh UK June 25-28

[5] Kim D, Machida F, Trivedi K S 2009 Availability modeling and analysis of a virtualized system *PRDC 2009 IEEE International Symposium on Pacific Rim Dependable Computing* Shanghai PRC Nov 16-18

[6] Thein T, Park J 2009 Availability analysis of application servers using software rejuvenation and virtualization *Journal of Computer Science and Technology* **24** (2) 339-46

[7] Rezaei A, Sharifi M 2010 Rejuvenation high available virtualized systems *ARES 2010 International Conference on Availability, Reliability and Security* Krakow Poland Feb 15-18

[8] Maciel Melo M P 2013 Availability study on cloud computing environments: Live migration as a rejuvenation mechanism *DSN 2013 Annual IEEE/IFIP International Conference on Dependable Systems and Networks* Budapest Hungary June 24-27

[9] Machida F, 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) 212-30

[10] Garg S, Huang Y, Kintala C 1995 Time and load based software rejuvenation: policy, evaluation and optimality *FFTS 1995 Proceedings of the First Fault-Tolerant Symposium* Madras IN Dec 20-22

[11] Bao Y, Sun X, Trivedi K S 2003 Adaptive software rejuvenation: degradation model and rejuvenation scheme *DSN 2003 Int'l Conf. On Dependable Systems and Networks* San Fransisco CA USA June 22-25

[12] Bao Y, Sun X, Trivedi K S 2005 A Workload-Based Analysis of Software Aging, and Rejuvenation *IEEE Transactions on Reliability* **54**(3) 541-8

[13] Vaidyanathan K, Trivedi K S 2005 A Comprehensive model for software Rejuvenation *IEEE Transaction on Dependable and Secure Computing* **2**(2) 124-37
[14] Zhong Yi, Xu Jian, Zhang Hong, Liu Fengyu 2013 Research on Measurement-based Rejuvenation Analytical Models for a Single-server Virtualized System *Journal of Computational Information Systems* **9** (23) 9611-8
[15] Ciardo G, Muppala J K, Trivedi K S 1989 SPNP: Stochastic Petri Net Package *PNPM 1989 Proc. Int'l Workshop on Petri Nets and Performance Models* Kyoto JPN 1989 Dec 11-13

Authors

Yi Zhong , 1979.07,Nanjing, Jiangsu, P.R. China

Current position, grades: M.Sc., lecturer
University studies: received her B.Sc. in Computer Science & Technology from Nanjing University of Science and Technology in China. She received her M.Sc. from Nanjing University of Science and Technology in China.
Scientific interest: Information security, software rejuvenation
Publications: more than 10 papers published in various journals.
Experience: teaching experience of 12 years, has completed two scientific research projects.

Jian Xu , 1979.07,Nanjing, Jiangsu, P.R. China

Current position, grades: phD, associate professor
Scientific interest: Software rejuvenation, virtualization technologies

Jing Zhong , 1988.11,Nanjing, Jiangsu, P.R. China

Current position, grades: Graduate student
University studies: School of computer science and engineering, Nanjing University of Science and Technology, China
Scientific interest: Software rejuvenation, virtualization technologies

Fengyu Liu , 1943.07,Nanjing, Jiangsu, P.R. China

Current position, grades: professor
Scientific interest: Information security, high-confidence software