Modeling and Analysis of Software Rejuvenation in a Server Virtualized System

Fumio Machida  
Service Platforms Research Laboratories  
NEC Corporation  
Kawasaki, Japan  
h-machida@ab.jp.nec.com

Dong Seong Kim, and Kishor S. Trivedi  
Department of Electrical and Computer Engineering  
Duke University  
Durham, NC 27708, United States  
dk76@duke.edu, kst@ee.duke.edu

Abstract—As server virtualization is used as an essential software infrastructure of various software services such as cloud computing, availability management of server virtualized system is becoming more significant. Although time-based software rejuvenation is useful to postpone/prevent failures due to software aging in a server virtualized system, the rejuvenation schedules for virtual machine (VM) and virtual machine monitor (VMM) need to be determined in a proper way for the VM availability, since VMM rejuvenation affects VMs running on the VMM. This paper presents analytic models using stochastic reward nets for three time-based rejuvenation techniques of VMM: (i) Cold-VM rejuvenation in which all VMs are shut down before the VMM rejuvenation, (ii) Warm-VM rejuvenation in which all VMs are suspended before the VMM rejuvenation and (iii) Migrate-VM rejuvenation in which all VMs are moved to the other host server during the VMM rejuvenation. We compare the three techniques in terms of steady-state availability and the number of transactions lost in a year. We find the optimal combination of rejuvenation trigger intervals for each rejuvenation technique by a gradient search method. The numerical analysis shows the interesting result that Warm-VM rejuvenation does not always outperform Cold-VM rejuvenation in terms of steady-state availability depending on rejuvenation trigger intervals. Migrate-VM rejuvenation is better than the other two as long as live VM migration rate is large enough and the other host server has a capacity to accept the migrated VM.

Keywords: Analytic model, optimization, server virtualization, stochastic reward net, time-based software rejuvenation

I. INTRODUCTION

Server virtualization becomes an essential software component for system infrastructure of various software services. Many enterprises introduce server virtualization to consolidate servers for enterprise applications and reduce the total cost of ownership. In recent years, server virtualization is also used for software infrastructure of cloud computing services which provide computing resources over the Internet. By using a virtual machine (VM), cloud service customer creates computing resources quickly on the cloud computing infrastructure. As the server virtualization becomes popular, availability management of server virtualized system grows in importance.

Software rejuvenation is one of the promising techniques for assuring high-availability of server virtualized system [1]. Since server virtualization is usually implemented by software called hypervisor or virtual machine monitor (VMM), it also faces the risk of software aging and related system failures. Software aging is caused by aging related bugs that are known to be difficult to fix completely in software development and test phases [2]. To postpone/prevent system failures caused by software aging in VMM, software rejuvenation can be applied to VMM. Software rejuvenation is a proactive recovery method that clears aging status by restarting or resetting the software. However, concrete availability analysis and optimization of rejuvenation schedule for this technique has not yet been discussed.

In this paper, we present a comprehensive availability model for a server virtualized system with time-based rejuvenations for VM and VMM. Our availability model captures software aging states of VM and VMM as well as their failures caused by aging. Numerical solution of the availability model enables us to find the optimal rejuvenation schedules for VM and VMM that maximize the steady-state availability of the VM. We make a comparison of three different rejuvenation techniques for VMM, namely Cold-VM rejuvenation, Warm-VM rejuvenation, and Migrate-VM rejuvenation. Cold-VM rejuvenation simply shuts down hosted VMs at the VMM rejuvenation and restarts the VMs after the rejuvenation. Warm-VM rejuvenation suspends VMs before the VMM rejuvenation and resumes the VMs after the rejuvenation. Migrate-VM rejuvenation moves the hosted VMs onto the other hosting server during the VMM rejuvenation by using live VM migration. As a result of the comparison, we see that Migrate-VM rejuvenation is the best rejuvenation method for VMM in terms of steady-state availability as long as live VM migration is fast enough. We also show that the steady-state availability achieved by Warm-VM rejuvenation is not always better than Cold-VM rejuvenation contrary to our intuitive understanding.
II. RELATED WORK

Warm-VM reboot was presented as a fast rejuvenation technique for VMM in [4][5]. When the VMM rejuvenation is triggered, all the hosted VMs become down since VMM manages the execution of hosted VMs. Warm-VM reboot reduces the down time of VM considerably by introducing the on-memory suspend technique and the quick reload mechanism. A simple availability analysis showed that Warm-VM reboot improves the availability of the application hosted on VMs compared to Cold-VM reboot that stops all the hosted VMs at the VMM rejuvenation [4]. The existing availability study was fairly simple as it did not consider the failures and recovery processes of VMs and VMM. In this paper, we provide a comprehensive availability model of a server virtualized system including Warm-VM reboot with time-based rejuvenation policy.

Availability model for a virtualized system including live VM migration and VM High-Availability feature was presented by using a hierarchical stochastic model in [6]. Some studies incorporated software rejuvenation for VM into availability model and computed the down time cost or steady-state availability of the system [7][8]. Time-based rejuvenation for VM was modeled as a continuous time-Markov chain for virtualized system composed of two physical servers and high-availability clustered VMs [7]. VMM failure and VMM rejuvenation were also taken into account in the availability model for virtualized system [8]. Compared to the existing modeling studies, our availability model focuses on the evaluation of the rejuvenation techniques for VMM including Warm-VM reboot and Cold-VM reboot. We also introduce the rejuvenation technique for VMM using live VM migration.

Time-based rejuvenation and the optimization issues were addressed in several previous studies. In the time-based rejuvenation policy, the optimum rejuvenation schedule that maximizes the steady-state availability can be determined by solving a Markov regenerative process [9]. Comprehensive availability model for cluster system with time-based rejuvenation was presented for determining the optimal rejuvenation trigger interval that minimizes the expected down time [10][11]. We leverage these availability models for time-based rejuvenation and apply to a server virtualized system to determine the optimum rejuvenation schedules for VMs and VMMs.

III. VMM REJUVENATION TECHNIQUES

When a VMM is rejuvenated for clearing the aging states of the VMM, the execution of VMs running on the VMM may also be interrupted. As the VMM rejuvenation techniques for minimizing the down time of the hosted VMs, three different approaches are discussed here.

A. Cold-VM rejuvenation

The simplest way to rejuvenate VMM is to shut down all the hosted VMs prior to the rejuvenation regardless of the execution states of the VMs. This technique forces a shutdown of the hosted VMs even if the VMs are running properly on the VMM. Although this technique involves unnecessary down time of the VMs and loses transaction running on the VMs, it clears all of aging states in VMs and VMM by a VMM rejuvenation.

B. Warm-VM rejuvenation

Instead of shutting down the hosted VMs, the VMM can suspend the hosted VMs before a VMM rejuvenation and resume the execution of the VMs after the rejuvenation. Warm-VM reboot technique enables VMM to save the execution states of VMs in memory instead of persistent storage and quickly resume after the rejuvenation [4][5]. We use the term Warm-VM rejuvenation as a general VMM rejuvenation technique using VM suspend including Warm-VM reboot. Although the VMs can resume the execution from the suspended states after a VMM rejuvenation, the aging states of applications and the VMs are not cleared by the VMM rejuvenation.

C. Migrate-VM rejuvenation

Live VM migration enables a running VM on a host server to move onto the other host server with very small interruption of the execution. By using live VM migration, the hosted VM can move onto the other host at a VMM rejuvenation and it can return back to the original host after the completion of the VMM rejuvenation by live VM migration again. The down time of a VM caused by live VM migration is very small and the VM continues the execution even while the original host is down. However the aging states of applications and the VMs are not cleared by the VMM rejuvenation. And live migration works only when there is capacity to accept the migrated VM on other hosts.

IV. AVAILABILITY MODELS

In order to study the effects of the VMM rejuvenation techniques on system availability, we present availability models of a server virtualized system using stochastic reward net (SRN) [12].

A. System architecture

A system configuration of a server virtualized system is shown in Figure 1. The server virtualized system consists of a hosting server (host1) installing a VMM and one hosted VM on which an OS and applications are running. To enable live VM migration, we assume that another hosting server (host2) is connected on the same network and the disk image...
of the VM is stored in a shared storage system. The availability of the system is continuously monitored by a monitoring tool on a management server. When a hosting server or a VM goes down unexpectedly, the monitoring tool in the management server detects the failed component and notifies an alert message to the system administrator.

B. Failure and recovery behaviors

Both the VM and the VMM are subject to software failures during their continuous execution due to residual aging bugs. VM contains operating system with some application programs for providing software services. Even in matured application software such as web server and application server, aging bugs may exist that accumulate software errors during the continuous operation and cause performance degradation or hangs [13][14]. VMM is usually embedded in operating system like Xen hypervisor. Xen hypervisor has also known to have aging bugs that cause memory leaks and system hangs [4]. To reset the aging states periodically, both of VM and VMM are rejuvenated by respective rejuvenation trigger intervals.

Besides the independent software component failures, the dependency between VM and VMM also need to be considered. When the VMM goes down due to failure or periodic rejuvenation, hosted VMs might go down as well.

All failures are detected by the monitoring tool on the management server within the predetermined monitoring interval. After an alert message issued by the monitoring tool, a system administrator investigates the cause of the failure and recovers the failed component.

To focus on the evaluation of rejuvenation techniques, we only consider the software failures on the server virtualized system (host2). We neglect the failures in the other hosting server (host1), the network, the shared storage system, and the management server. However, various hardware failures and recoveries also can be incorporated in availability model as presented in [6].

C. Cold-VM rejuvenation

Figure 2 shows the SRN for a server virtualized system with Cold-VM rejuvenation. The model consists of four submodels; (a) VMM model, (b) VMM clock model, (c) VM model and (d) VM clock model. The two clock models are used for triggering time-based rejuvenation of VM and VMM. A deterministic transition $T_{\text{interval}}$ fires with duration $1/\tau$ after the latest boot time and starts the rejuvenation as long as the immediate transition $T_{\text{policy}}$ is enabled. When the VM rejuvenation process finishes, the immediate transition $T_{\text{rej}}$ is enabled and a token is deposited in the place $P_{\text{clock}}$ again. Similarly, $T_{\text{interval}}$ is a deterministic transition for triggering VMM rejuvenation with duration $1/\tau$ after the VMM startup time.

In Figure 2(a), the VMM model represents the two-step failure process, two-step recovery process, and time-based rejuvenation of a VM. When the transition $T_{\text{halt}}$ fires, a token is deposited in $P_{\text{halt}}$. This transition represents the software aging of the VMM. If the transition $T_{\text{halt}}$ fires, a token is deposited in $P_{\text{halt}}$ which represents the VMM failure due to the software aging. The VMM failure can be detected by the monitoring tool at a certain monitoring interval and is manually recovered by a system administrator. $T_{\text{halt}}$ fires at
failure detection of VMM and $T_{vusd}$ fires when the VMM is recovered from the failure state. Whenever the VMM rejuvenation is triggered by the VMM clock model (see Figure 2(b)), the immediate transitions $T_{hejt}$ and $T_{bhejt}$ are enabled and a token is deposited in $P_{hejt}$. When the VMM rejuvenation cleans up the aging states, the token in $P_{hejt}$ is removed and a token is deposited in $P_{bhejt}$ by $T_{hejt}$ transition.

The VM model is based on two-step failure process, two-stage recovery process and time-based rejuvenation of a VM (see Figure 2(c)). The time-based rejuvenation is controlled by the VM clock model (in Figure 2 (d)). Since the execution of the VM depends on the underlying VMM, if there is no token in $P_{hup}$ nor in $P_{hdw}$, the immediate transitions $T_{cdw}$ and $T_{vtpms}$ are enabled and a token is deposited in $P_{cdw}$. $T_{restart}$ is enabled only when there is a token in $P_{hup}$ or $P_{hdw}$, and hence the VM cannot be restarted until the VMM becomes available.

Cold-VM rejuvenation makes hosted VM shutdown before starting the VMM rejuvenation. This policy is represented in the guard functions for the immediate transitions $T_{vpre}$ and $T_{vtpms}$, $T_{vpre}$ and $T_{vtpms}$ are enabled when a token is deposited in $P_{hpolicy}$ in the VMM clock model. In the VM model, a token is deposited in $P_{vsys}$ or $P_{vpsd}$. When a token is deposited in $P_{vsys}$ by firing $T_{vdw}$ and $T_{vtpms}$, the immediate transition $T_{hpolicy}$ is enabled and the VMM rejuvenation process starts. We assume that the VMM rejuvenation can start only in a token is deposited in $P_{vsys}$, $P_{vsys}$, $P_{vdw}$ or $P_{vpsd}$.

All transition times of timed transitions in the models are assumed exponentially distributed except for $T_{vinterval}$ and $T_{vinterval}$ which are deterministic because they represent the fixed rejuvenation trigger intervals of VM and VMM. In our implementation of the models on software package SPNP [15], we use 10-stage Erlang distribution for approximating the deterministic transition.

D. Warm-VM rejuvenation

The SRN model for a server virtualized system with Warm-VM rejuvenation consists of the VM model, the VM clock model, the VMM model and the VMM clock model. Figure 3 shows the VMM model which includes Warm-VM rejuvenation process. The other parts of the models are the same as shown in Figure 2 except the guard function of $T_{hpolicy}$, $T_{hpolicy}$ in the VMM clock model is modified for the Warm-VM rejuvenation policy as shown in the bottom of Figure 3.

Warm-VM rejuvenation uses suspend operation to stop the execution of VM at the VMM rejuvenation. This policy is represented in the guard functions for the immediate transitions $T_{vup}$ and $T_{vtpms}$. $T_{vup}$ or $T_{vtpms}$ are enabled when a token is deposited in $P_{hpolicy}$ in the VMM clock model (see Figure 2(b)). When a token is deposited in $P_{vsys}$ or $P_{vpsd}$ by firing $T_{vsys}$ or $T_{vpsd}$, the VMM can start the rejuvenation. The immediate transition $T_{vsys}$ and $T_{vtpms}$ are enabled, when the VMM rejuvenation is completed and a token is deposited in $P_{vsys}$ in the VMM model. The hosted VM resumes the execution when the token is deposited in $P_{vsys}$ or $P_{vpsd}$ by the transitions $T_{vsys}$ or $T_{vpsd}$. We assume that the VMM rejuvenation can start unless a token is deposited in $P_{vsys}$, $P_{vsys}$, $P_{vdw}$ or $P_{vpsd}$.

E. Migrate-VM rejuvenation

Migrate-VM rejuvenation is modeled in a similar way to Warm-VM rejuvenation. Figure 4 shows the VM model that includes live VM migration in place of VM suspend and resume. The guard function of $T_{hpolicy}$ in VMM clock model in Figure 2(b) is modified as well. If there is a token in $P_{vsys}$ or $P_{vdw}$ and $T_{vsys}$ and $T_{vtpms}$ are enabled by VMM clock model, the VM is migrated to the other host. The VM continues the execution on the other host during a token exists in $P_{vmig}$ or $P_{vpmig}$. When the VMM rejuvenation completes, the immediate transitions $T_{vsys}$ and $T_{vtpms}$ are enabled and the VM returns back to the original host by live VM migration. For simplicity, we neglect any failures during the VMM rejuvenation and VM migration. The VM can start rejuvenation unless a token is deposited in $P_{vsys}$, $P_{vsys}$, $P_{vdw}$ or $P_{vpsd}$. While a token is deposited in the
places $P_{\text{vfp}}$, $P_{\text{vfps}}$, $P_{\text{vfpd}}$, and $P_{\text{vfpres}}$, the VM is available and the transition $T_{\text{runtime}}$ is enabled.

V. NUMERICAL RESULTS

Three rejuvenation techniques are compared by numerical solution of SRNs using SPNP [15]. Table I shows the default parameter values that are based on our experimental studies and previous works in [4][5][6].

<table>
<thead>
<tr>
<th>Parameters Used in the Models</th>
</tr>
</thead>
</table>

**A. Optimal rejuvenation schedule**

Under the given parameter values, first we compute steady-state availability by varying the rejuvenation trigger intervals of VM and VMM. We define the reward function for computing steady-state availability, and Gauss-Seidel method [12] is used to compute the expected reward rate. Figure 5 shows the results of steady-state availability of Cold-VM rejuvenation system. At a certain combination of the rejuvenation trigger intervals, the steady-state availability of the VM is maximized. This is caused by the fact that more frequent rejuvenation increases the down time due to the rejuvenation and less frequent rejuvenation also increases the down time caused by software failure. By finding the point which maximizes the steady-state availability, we can solve the optimum combination of rejuvenation trigger intervals. The maximum values of steady-state availability for the three rejuvenation techniques in terms of steady-state availability are shown in Table II. To find the optimum combination, we used a gradient search method [16]. As a result, Migrate-VM rejuvenation outperforms the other two rejuvenation techniques in terms of the maximum value of steady-state availability of the VM. In Migrate-VM rejuvenation case, we find that the optimum rejuvenation trigger interval of VMM is smaller than that for the VM. Since the VMM rejuvenation has negligible effect on the VM availability, VMM rejuvenation can be performed more frequently than VM rejuvenation.

![Figure 5 Steady-state availability in Cold-VM rejuvenation system](image)

**B. Sensitivity analysis**

The impact of each parameter value on steady-state availability is studied by sensitivity analysis. Figure 6(a) and (b) show the results of steady-state availability by varying the rejuvenation trigger intervals of VM and VMM independently. Note that the steady-state availability of Warm-VM rejuvenation gets worse than that of Cold-VM rejuvenation if the rejuvenation trigger interval of VM increases over 50 hours (see Figure 6(a)). The result implies that Warm-VM rejuvenation is not always better than Cold-VM rejuvenation in terms of steady-state availability depending on the rejuvenation trigger intervals. This result is caused by the fact that the aging status of VM in Warm-VM rejuvenation case is not cleared by the VMM rejuvenation even though the VM is unavailable during the VMM rejuvenation. In the Warm-VM rejuvenation model of Figure 3, this behavior is represented by the loop consisting of the places $P_{\text{vfp}}$, $P_{\text{vfps}}$, $P_{\text{vfpd}}$, and $P_{\text{vfpres}}$. Although this loop includes down states of VM, it does not improve steady-state availability. On the other hand, Cold-VM rejuvenation clears the aging status of VM at every VMM rejuvenation. Cold-VM rejuvenation and Warm-VM rejuvenation need to be used separately depending on the situations.

Next, we observe the effect of suspend time of Warm-VM rejuvenation case on steady-state availability under the fixed rejuvenation trigger intervals. The result is shown in Figure 6(c). Under the given parameter values, Warm-VM rejuvenation is better than Cold-VM rejuvenation as long as the suspend time is less than 22 seconds. Similarly, the effect of live migration time of Migrate-VM rejuvenation is observed in Figure 6(d). As long as live VM migration completes within 60 seconds, Migrate-VM rejuvenation is more effective than Warm-VM rejuvenation.
C. Transactions lost

Warm-VM rejuvenation and Migrate-VM rejuvenation have the advantage of conserving the execution states of VM than Cold-VM rejuvenation. To show this, we compute the expected number of transactions lost in a year by analyzing the throughput of the transitions $T_{\text{res}}$, $T_{\text{rep}}$ and $T_{\text{queue}}$ in each VM model. The transaction processed by the application on the VM is lost when the VM goes down without preserving the execution states. The expected numbers of transactions lost due to VM restart, VM rejuvenation and VM repair are computed from the throughputs of corresponding transitions and results are summarized in Table III. In Cold-VM rejuvenation case, the VM is forcibly shutdown at the VMM rejuvenation, and hence the number of transactions lost due to VM restart is larger than the other two cases. Although Cold-VM rejuvenation reduces the number of VM rejuvenations, the total number of transactions lost in a year is higher than the other two cases. There is not much difference in the number of expected transactions lost between Warm-VM rejuvenation and Migrate-VM rejuvenation.

**TABLE III. EXPECTED NUMBER OF TRANSACTIONS LOST IN A YEAR**

<table>
<thead>
<tr>
<th>Causes of transactions lost</th>
<th>Cold-VM rejuvenation</th>
<th>Warm-VM rejuvenation</th>
<th>Migrate-VM rejuvenation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM restart due to VMM</td>
<td>53.95</td>
<td>4.41</td>
<td>4.41</td>
</tr>
<tr>
<td>rejuvenation or failure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM rejuvenation</td>
<td>335.64</td>
<td>357.64</td>
<td>357.71</td>
</tr>
<tr>
<td>VM repair due to VM failure</td>
<td>7.73</td>
<td>7.98</td>
<td>7.97</td>
</tr>
<tr>
<td>Total</td>
<td>397.32</td>
<td>370.03</td>
<td>370.09</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS

We have presented comprehensive availability models of three rejuvenation techniques for VMM and have shown some numerical results. Under the given parameter values, we have shown the optimum combination of rejuvenation trigger intervals of VM and VMM for each rejuvenation technique by a gradient search method. The result of sensitivity analysis shows that Warm-VM rejuvenation is not always better than Cold-VM rejuvenation in terms of steady-state availability. Migrate-VM rejuvenation achieves the best steady-state availability as long as VM live migration is fast enough and other servers have capacity to receive the migrated VM. In every case, the rejuvenation trigger intervals of VM and VMM need to be carefully determined so as to achieve higher steady-state availability of VM.

ACKNOWLEDGMENT

The authors would like to thank Kenichi Kourai for his technical comments on rejuvenation techniques for VMM.

REFERENCES