



19th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

Simulating Resilient Server using XEN Virtualization

Idris Winarno, Yoshiteru Ishida

*Department of Computer Science and Engineering,
Toyohashi University of Technology
Tempaku, Toyohashi 441-8580, Japan*

Abstract

Since servers play a critical role in data processing and data transmission for serving many clients, failures in servers cause not only performance degradation of the server itself but also threats to all the computers connected to the server. Resilient servers that can self-recognize failures, self-repair failures and self-replace failed parts are required when computer systems and networks become such huge-scale as witnessed by data centers and the cloud computing. As a preliminary study, we report simulations restricting ourselves to demonstrate the self-repair network (SRN) model in a homogeneous environment realized by a virtualization technique. Simulations are conducted using native and hosted VMM on a single physical computer. Three scenarios: hang faulty, DoS attack and virus infection, are simulated. These simulations demonstrated how a server with a homogeneous environment (realized by the self-repair network model using the virtualization technique) can or cannot keep the resilience, and even suggested a possibility and necessity of using a self-reconfiguration model to create diversity.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Peer-review under responsibility of KES International

Keywords: self-repair network; resilient server; virtual machine

1. Introduction

A computer which has a high specification and run a service that can be accessed by multiple users over a network is called a server¹⁰. Email, web server, and other services are usually run by a computer server since it needs high computation to serve in multiple connections. The server takes an important role along its role to deliver the information to users. Therefore, server reliability must always take precedence in order for it to perform in a variety of situations. Each server in the data center must be highly available for serving the data processing and transmission. To maintain the system availability, it is important to repair the server in the data center when the failure occurs.

Server failure could happen by many problems which affect the server performance; for example hang faulty, virus spreading, and Denial of Service (DoS) attack. The simplest way to repair such failures as virus infection is by installing an anti-virus and an anti-rootkit on each server in the data center. However, when the server failure is complex, we have to reinstall or reset the server. Moreover, the reinstalling and the resetting the server need much time to do. It is difficult to repair from the failure using this scenario.

* Corresponding author. Tel.: +81-80-3685-7576.
E-mail address: idris@sys.cs.tut.ac.jp

The other scenario to repair the system is using a self-repairing network model⁷, which is derived primarily from the concept of an immunity-based system⁹. In the self-repair network model, the self-repair (intra-node repair visualized as a loop) and the mutual-repair (inter-node repair visualized as a cycle) are used to recover the system from failures among the servers (also called nodes). As a sensor to detect anomaly of the system, several tools such as *Tripwire*, *ClamAV* and *rkhunter* may be used. We report simulations aiming at resilient servers that can recover by themselves from failures by incorporating the self-repairing network (SRN)⁶ model with the XEN virtualization. XEN is one of the Virtual Machine Monitor (VMM). There are two types of VMM (Section 3.3).

2. Related Work

Nowadays, servers run in a virtualization environment, and are collected to a data center. A data center is a collection of servers, which have been evolved and still evolving. Getting started from a centralized technology known as the mainframe machine (1.0 version), decentralized technology (2.0 version) known as the mechanism of client-server and distributed computing is replacing the centralized technology. Currently, the virtualization technology (3.0 version) that prioritizes service-oriented mechanism⁵ and is based on web 2.0 (Fig. 1) is used. Virtualization technology takes an important position in the simulations of the resilient server against the failure scenario. Some projects^{1,3} use XEN to increase availability and dependability of a server, and the other project¹³ uses VMware ESXi to build a backup system performed on a virtual machine. Many researches focus on server performance involving virtualization (virtual machine) technology. Detection system of stealthy malware has been proposed using VMM-Based “out-of-the-box” semantic view reconstruction⁸, which compares the VMwatch approach with the conventional one to detect the malware. Tripwire and antivirus are used as host-based anti-malware. ReVirt enables an intrusion analysis through virtual-machine logging and replay⁴, which concluded that ReVirt could determine and fix the damage the intruder inflicted by replaying the execution before, during and after the intruder compromises the system.

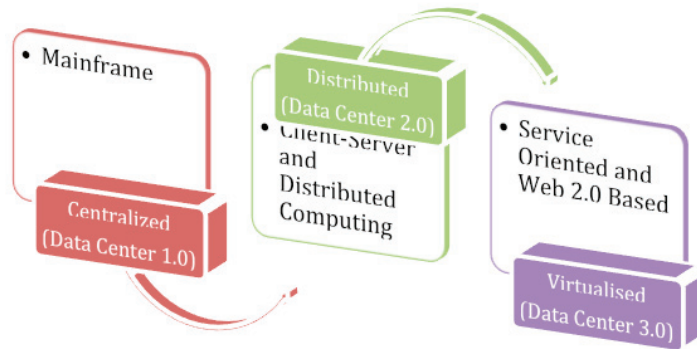


Fig. 1. Data Center and Network Evolution

3. Proposed Method

3.1. Self-repair network

The self-repair network (SRN) has been proposed as a model to design resilient systems such as a resilient server. SRN consists of autonomous nodes not only being repaired by connected nodes but also capable of repairing connected nodes (Fig. 2b). Nodes can repair themselves (i.e., nodes with a loop) as shown in Fig. 2a. Repairing may be implemented in many ways depending on target systems. One way is overwriting the contaminated contents with normal contents by copying the normal contents (called inter-node repair or *mutual-repair* in Fig. 2b). The other ways include removing contaminated parts and resetting the state (called intra-node repair or *self-repair* in Fig. 2a).



Fig. 2. (a) self-repair, (b) mutual-repair

In building a simulator, we assume the server as a node in a virtualization environment (Fig. 3a). Each node consists of separate parts of disk partitions. Since we use the virtual machine, a disk partition is stored as a file (Fig. 3b). When one of the partition having a problem, it may be repaired by overwriting the partition by the corresponding content of other nodes.

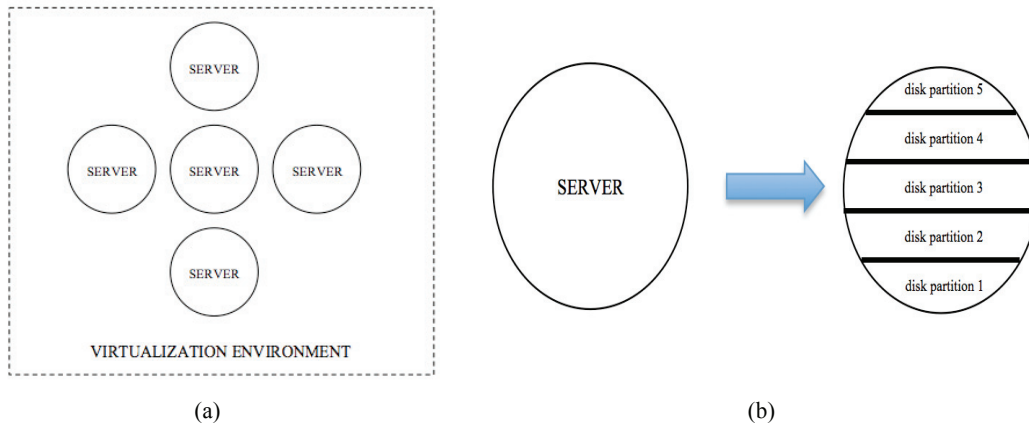


Fig. 3. (a) model of server (node) run in virtualization environment, (b) model of each node by separating disk partition

There can be many kinds of servers that are running various operating systems (e.g. Linux, Windows, BSD). Moreover, each operating system usually has many kinds of distributions (e.g. Debian GNU/Linux, CentOS). Distinct distributions allow the servers (nodes) to be heterogeneous because each operating system has the distinct structure of file system hierarchy. Therefore, to implement SRN model in this simulation, designing the nodes (servers) identical (homogeneous nodes) allows the inter-node repair to be implemented by simple copying. We used the homogeneous nodes for our preliminary test of SRN for simplicity.

3.2. Information System Failure

Failures in information systems could happen anytime and anywhere. Roughly, there are two categories of failures. The First category is a physical failure: a failure at the physical devices (hardware) of the server. The second category is a logical failure or software failure. This note focuses on the logical failures, which will be used as scenarios on a server such as Denial of Service (DoS) attack and virus spreading. We assume that failures will affect any services running on the server. Therefore, a server has to preserve resilience in the services such as the communication services as in the web server and the mail server.

3.3. Virtual Machine Monitor (VMM)

There are two types of VMM (also known as a hypervisor) that can be used for simulations or even used for an operational purpose. The first type is hosted VMM, and the second type is native (bare-metal) VMM. Fig. 4 illustrates the differences in the two types of VMM. Table 1 shows an example of the application of the VMM. We used both types of the VMM. We simulate the Native VMM under running hosted VMM using a single computer. In the simatiotios, we use XEN 4.1.4 as native VMM and VMware Fusion 7.0.0 as hosted VMM.

Table 1. Example of VMM Applications

Type	Application
Native	VMware ESX/ESXi
	KVM
	XEN
	Hyper-V
	VMware Player
Hosted	OpenVZ
	VirtualBox

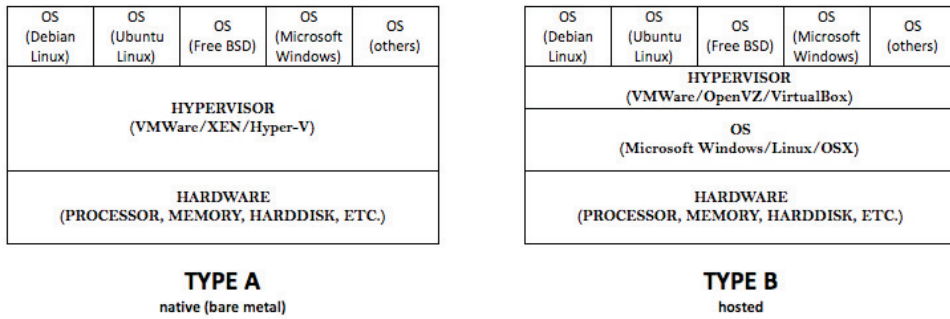


Fig. 4. Two types of VMM (hypervisor)

4. System Design and Implementation

4.1. Logical Design

We use the logical design to simulate the resilient server by implementing SRN model using both hosted and native VMM (Fig. 4). There are 4-hosted VMM: node 1 through node 4. We assume that node 1 through node 3 are normal nodes, and the node 4 is an abnormal node (Fig. 5). Each node has four partitions that are stored in separate files (virtual disk) (Fig. 5). The logical design simply aims at easier recovery of the file system or data of the node when an abnormal condition (an anomaly) is detected. Each node has to install an application (sensor) to check the condition of the node itself periodically (Fig. 6). There are three sensors that run on each node. When the sensor detects an anomaly, it instructs the node to start the recovery (the intra-node repair indicated as *self-repair* in Fig. 6). However, if the problem still remains, the node has to get the copy of the file system or data from the other nodes (the inter-node repair indicated as *mutual-repair* in Fig. 6).

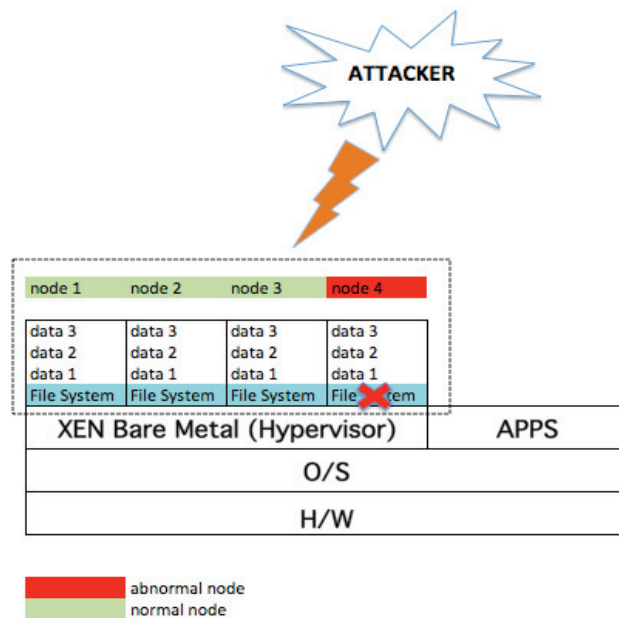


Fig. 5. Logical design using hosted and native VMM

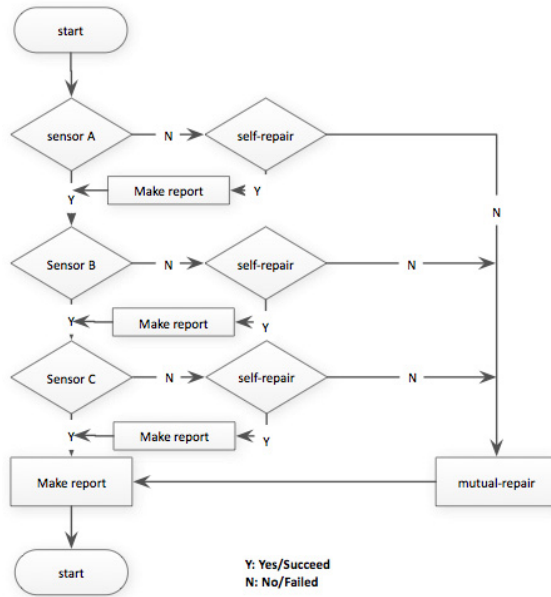


Fig. 6. Flowchart of a node to check the system periodically

4.2. Scenarios

To simulate attacks (Fig. 5), we use the following three scenarios:

- Hang scenario: node having faulty (hang),
- Denial of Service (DoS) scenario: node being attacked by DoS,
- Virus scenario: node being contaminated by a virus where contaminated nodes may infect other nodes.

For the first scenario: a node hang up, there are many possibilities that can cause the hang problem on the system². It is hard for the node to detect and solve the hang scenario. The hypervisor has been involved to solve this problem. A simple method to detect the hang problem from the network is by pinging the node. When the node is not responding to the ping, the hypervisor has to restart the node and report to the administrator.

In the second scenario: the DoS attack, the attack can be done even as distributed attacks called Distributed DoS (DDoS). DDoS attacks can be simulated with the simulator implementing SRN model with the virtualization technique. However, we use only DoS scenario to focus on the specific services (web server), which have the vulnerability. In the DoS scenario, we assume the attacker sends the DoS packet to a node. The hypervisor has to respond by checking the running services and add the IP address of the attacker to the firewall rule. If the services did not respond to the packet send by the hypervisor, the node has to restart the services by itself (intra-node repair or *self-repair*). If the problem still occurs, the hypervisor has to prepare to clean the node by copying from the normal node (inter-node repair or *mutual-repair*).

In the last scenario, we assume that a virus infects a node and spreads through the network. Each node has to check the integrity of the system whether the system changed or not. If the virus successfully changed the system and the change is detected by a sensor, the node has to copy the file system from the normal node. If the virus caused the overall system down, the hypervisor has to make a decision to isolate or disconnect the contaminated nodes from the network (when the contaminated nodes are identified).

4.3. Implementation

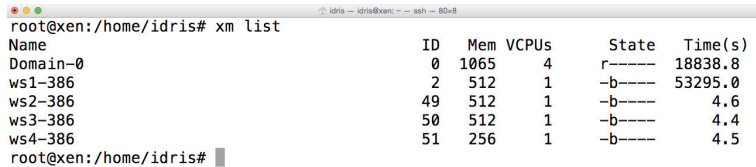
We use XEN version 4.1 as the hypervisor under Debian 7.8 GNU/Linux and created four nodes as guest Operating System with the x86 platform (Fig. 7). The hypervisor has important roles of monitoring all of the nodes and responding if there is an abnormal condition. Hence, the hypervisor must be capable of checking all of the three scenarios in section 4.2.

Moreover, we made the structure of all the nodes identical and separated the file system from the data itself as the different logical disks (Table 2). With the virtual machine, the logical disk is implemented to a file (Table 2). The identical nodes allow the inter-node repair (*mutual-repair*) to be realized by simply copying the normal component of the node to overwrite the

component of the abnormal node. To monitor the system, each node must run a program in the background periodically based on the flowchart in Fig. 6.

Table 2. Partition table of each node

Disk ID	Partition	Mount point	Size (MB)	Disk filename
1	File system	/	1024	disk1.img
2	data1	/var	512	disk2.img
3	data2	/usr	512	data3.img
4	data3	/home	512	data4.img



```

root@xen:/home/idris# xm list
Name                               ID   Mem  VCPUs   State   Time(s)
Domain-0                            0  1065    4   r----- 18838.8
ws1-386                              2   512    1   -b----- 53295.0
ws2-386                              49   512    1   -b-----  4.6
ws3-386                              50   512    1   -b-----  4.4
ws4-386                              51   256    1   -b-----  4.5
root@xen:/home/idris#

```

Fig. 7. Virtual machine with 4 nodes in x86 platform

5. Simulation Results

There are a lot more scenarios that can cause malfunction in servers, however, this preliminary simulations focus on the three scenarios: Hang (faulty), Denial of Service (DoS) and virus attacks.

5.1. Hang scenario

Through the XEN console we can manage to monitor each node and to simulate the hang condition. Since the nodes are under control of the Linux operating system, the OS allows us to simulate the hang scenario by simply sending a halt signal to the OS. When the halt signal is received, however, the node will shut down the system. Thus, we must use another way to simulate the hang scenario: by using XEN command that is "xm pause <node name>" command. We run the command for the node 4 and the hypervisor responded by restarting the node 4.

5.2. DoS scenario

We used *slowloris* script¹² to simulate the DoS scenario. Slowloris is a script that creates many connections to the service. In this scenario, we simulated an attack on the HTTP services (Apache web server). When the script of slowloris is executed to attack the node, the HTTP service ceased responding to the client requests. The hypervisor checks the services periodically and reacts to the attack by adding the attacker IP address on the firewall rules. With this simulation, DoS attacks demonstrated to hamper a certain services. Further, the other (identical) nodes also have the same vulnerability, suggesting a limitation of homogeneous nodes and a necessity of heterogeneous nodes as well. There are also possibilities to simulate the DDoS attack through this simulation assuming homogeneous nodes involving a framework such as *Metasploit* framework. Also, heterogeneous nodes must be considered to overcome this scenario since each node can have a distinct OS and service in the heterogeneous environment.

5.3. Virus scenario

In this scenario, we used ClamAV and Tripwire as sensors to detect malicious codes. We created a shell application to execute ClamAV and Tripwire to check the node condition. The shell code will execute automatically and periodically by using cron (job scheduler) (Fig. 9a). When the malicious code (virus) detected in a node, the node will try to recover or repair the infected files or systems (Fig. 9b). If the infection still remains, the node asks to copy the file system from the uninfected node (normal node) (Fig. 8). As far as the limited condition in the specific scenario is concerned, the simulation demonstrated to solve the scenario when the nodes are identical (homogeneous node).

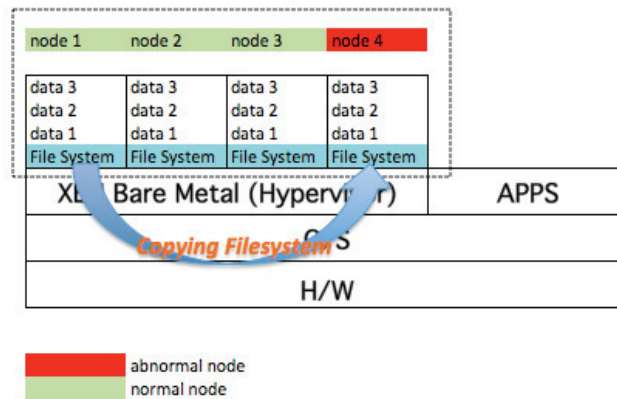


Fig. 8. Copying file system from normal node to abnormal node (mutual repair)

```

root@xen:/home/idris# xm console ws1-386
root@ws1-386:/virus# cat /etc/cron.d/rs
*/10 * * * * root /root/check
root@ws1-386:/virus# wget http://www.r57shell.net/shell/c99.txt -O sample
--2015-05-20 08:42:24-- http://www.r57shell.net/shell/c99.txt
Resolving galatea.eepis-its.edu (galatea.eepis-its.edu)... 10.252.209.208
Connecting to galatea.eepis-its.edu (galatea.eepis-its.edu)|10.252.209.208|:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 165779 (162K) [text/plain]
Saving to: `sample'

100%[=====] 165,779 40.1K/s in 4.0s

2015-05-20 08:42:28 (40.1 KB/s) - `sample' saved [165779/165779]

root@ws1-386:/virus# /root/check
repair node with hostname ws1-386...root@xen:/home/idris#
    
```

Fig. 9. (a) Application run periodically using cron, (b) virus (failure) is detected and server (node) is being repair

As far as the limited simulations are concerned, the SRN may be a sound model to involve the virtualization technique. However, homogeneous node may not be a sound assumption. For example, in the virus attack scenario, all the nodes are equally vulnerable to the attack. Moreover, it would be difficult to avoid the same infection even after the infected node is cleaned up. To overcome this problem, we can use the heterogeneous nodes (each node has a distinct operating system¹¹ or even distinct services). The heterogeneous node could be implemented by installing different distribution operating system (e.g., CentOS and Debian GNU/Linux) and services (e.g., Apache and Nginx). By implementing the heterogeneous node, it is more difficult for malicious codes to exploit the system, for they have to find the vulnerability each of the heterogeneous nodes.

6. Conclusion

Simulations revealed that the self-repair network with homogeneous nodes can be realized by involving the virtualization techniques where the self-repair is executed by copying the content of the homogeneous node. Simulations also suggested that the servers with the homogeneous environment can be made resilient against failures in limited scenarios. Although the self-repair network can deal with a specific type of failures to a limited scale of failures, the self-repair network alone cannot even recognize a large-scale failures, and recovery is also limited if we restrict ourselves to mutual repairing between homogeneous nodes. We also suggest that the diversity created by heterogeneous nodes involving a self-reconfiguration model that allows a system to replace failed nodes with heterogeneous nodes and to protect against ever growing threats involving diversity (learned from the immune system) can be a solution to keep resilience against unknown type of failures.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions that greatly contributed to improve the final version of this report. IW is also grateful to DIKTI (Indonesian Government for Higher Education) for the scholarship.

References

1. Brendan C, Geoffrey L, et al. Remus: High Availability via Asynchronous Virtual Machine Replication. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*; 2008. p. 161-174.
2. Cotroneo D, Natella R, Russo S. Assessment and Improvement of Hang Detection in the Linux Operating System. *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*; 2009. p. 288-294.
3. Cully B, Warfield A. SecondSite: Disaster Protection for the Common Server. In *HOTDEP'06: Proceedings of the 2nd conference on Hot Topics in System Dependability*; 2006.
4. Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review* 36; 2002.
5. Ian B. *Data Centre Evolution: The Role of the Network in Data Centre Transformation*. Cisco; 2008.
6. Ishida Y. A Critical Phenomenon in a Self-repair Network by Mutual Copying. *Knowledge-Based Intelligent Information and Engineering Systems. Lecture Notes in Computer Science* 2005; **3682**: 86-92.
7. Ishida Y, Tanabe K. Dynamics of Self-Repairing Networks: Transient State Analysis on Several Repair Types. *International Journal of Innovative Computing, Information and Control* 2014; **10**: 389-403.
8. Jiang X, Wang X, Xu D. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*; 2007. p. 128–138.
9. Jerne NK. The Immune System. *Sci Amer* 1973; **229**: 52-60.
10. Maffei S. Client/server term definition. In: Hemmendinger D, Ralston A, Reilly ED, editors. *Encyclopedia of Computer Science*. International Thomson Computer Publishing; 1998.
11. Pu C, Black A, Cowan C, Walpole J. A Specialization Toolkit to Increase the Diversity in Operating Systems. In: *Workshop Notes on Immunity-Based Systems. International Conference on Multiagent Systems*; 1996. p. 107-117.
12. Slowloris HTTP DoS, <http://ha.ckers.org/slowloris/>
13. Winarno I, Sani MN. Automatic Backup System for Virtualization Environment. *EMITTER International Journal Of Engineering Technology* 2014; **2**: 91-101.