International Journal of Innovative
Computing, Information and Control
Volume **12**, Number **4**, August **2016**

ICIC International ©2016 ISSN 1349-4198
pp. 1059–1071

A RESILIENT SERVER BASED ON VIRTUALIZATION WITH A SELF-REPAIR NETWORK MODEL

Idris Winarno[1], Takeshi Okamoto[2], Yoshikazu Hata[1]
and Yoshiteru Ishida[1]

[1]Department of Computer Science and Engineering
Toyohashi University of Technology
Tenpaku, Toyohashi 441-8580, Japan
{ idris; hata }@sys.cs.tut.ac.jp; ishida@cs.tut.ac.jp

[2]Department of Information Network and Communication
Kanagawa Institute of Technology
1030 Shimo-ogino, Atsugi, Kanagawa 243-0292, Japan
take4@nw.kanagawa-it.ac.jp

Received January 2016; revised May 2016

Abstract. *The performance of an information system depends on the reliability of the data center infrastructure. The server, one of the most critical parts of a data center, can negatively affect the overall performance of an information system in cases of system failure (i.e., hang faulty, DoS attack, and virus infection). Recently, virtualization has become a popular method to develop servers for data center infrastructure. In this paper, we introduce a new model of resilient server by implementing a Self-Repair Network model (SRN) in the virtualization environment to address failures that occur to the operation of the server. The simulations show that virtualization with SRN could provide failure recovery in limited scenarios. Moreover, the heterogeneous environment present in virtualization will reduce the risk of virus spread.*
Keywords: Fault recovery, Virtual machine, Server, Self-repair network

1. **Introduction.** Nowadays, the Internet has become an essential part of human life. Many common daily activities rely on the information that we can obtain from the Internet. The Internet is not only just for news or entertainment, but also offers a significant amount of useful and practical content for users. The growth in Internet users is increasing each year, in particular, mobile technology has had significant impact on Internet users. The number of smartphones sold and the unavailability of IPv4 addresses as a result are indicators that demonstrate this growth.

The server is one of the essential parts of the Internet since servers are responsible for providing services for Internet users. A server is a computer of high specification that runs services or applications accessed by multiple users over a network. The Domain Name System (DNS), Web server, Email, and other services are usually realized as a server. Therefore, the network administrators, who are responsible for maintaining the server, have to monitor and recover the server when failure occurs. In traditional operations, the network administrator regularly makes routine checks to the server remotely.

Since the server runs important services, we should endeavor to protect the server against threats in order to keep it running and serving information to users. There are many scenarios of failure that could affect a server's functionality such as hang faulty, virus infection, Denial of Services (DoS) attack, and numerous other threats. When a failure occurs on the server, the simplest way to overcome this condition is by resetting it.

However, if the server has a complex failure, then a reinstallation of the operating system and all necessary applications may need to be undertaken.

Virtual machine technologies (virtualization) offer features designed to maintain resources more easily and efficiently than traditional technologies where a single physical computer runs a single operating system. However, by using virtualization technology, the network administrator can create multiple virtual servers on a single physical computer. This means that a single physical computer can run multiple operating systems simultaneously. Thus, we require a model that can deal with virtual servers as a repair unit. The Self-Repair Network (SRN) is such a model, and we can use the results of the SRN to improve the resilience of the server. The SRN offers a general framework of self-action models (Section 3.2) for the server to self-recognize, self-repair, and self-replace its own failed parts.

To address the problems that have been described earlier, we were motivated to design a resilient server by combining virtualization technology and a self-repair network model [13]. The goal of this design is to create a resilient server that can recover based on virtualization with homogeneous and heterogeneous nodes (server) when failure occurs on those nodes during operation. Data center evolution and virtual machine technology implementation are reviewed in Section 2. In Section 3, we discuss the different types of virtualization technology and the self-repair network model. In Sections 4 and 5, we outline the design of the virtualization nodes (homogeneous and heterogeneous) with the self-repair network model for a number of limited scenarios. The main result is presented in Section 6 where we compare performance with a homogeneous environment [1], and that with a heterogeneous environment of virtualization, against server threats. Section 7 concludes with a discussion comparing homogeneous and heterogeneous (diversified) nodes.

2. **Related Work.** Collections of computer servers are known as Data Centers (DC). To develop a DC, there are high costs as the DC always has complex requirements for physical space, electricity, a cooling system, security, networking and other essential elements [2]. We need to provide a large area or building for the placement of physical server machines, redundancy safeguards for electricity and networking, and we have to secure the area with a high level of security. However, the DC has been evolving due to the evolution of computers, which are continuously getting faster, smaller, and more efficient [3]. The first models of DC utilized a mainframe server that offered centralization and was in wide use during the 1990s. The second model of DC, known as distributed technology, is where client-server and distributed computation mechanisms are applied. The most recent DC model that we use today is known as virtualization technology where a single physical computer machine hosts multiple virtual computer servers. Figure 1 illustrates data center evolution technology from data center version 1.0 to 3.0.

A fault-tolerance system is one of the models that are usually used in a common DC to prevent system stoppages from failures [4,5]. The key point of fault-tolerance is redundancy [6]. In DC version 3.0, redundancy is designed through a model where the system provides duplication of a critical component (the resilient server). There are several models of resilient server utilizing virtual machines in the DC, e.g., an attack-resilient server where the server is resilient against cyber-attack [7].

This paper focuses on the resilient server by increasing the diversity of the node where we build and evaluate the implementation of an SRN model that runs on a heterogeneous environment to overcome failures (i.e., hang) that occur during server operation.

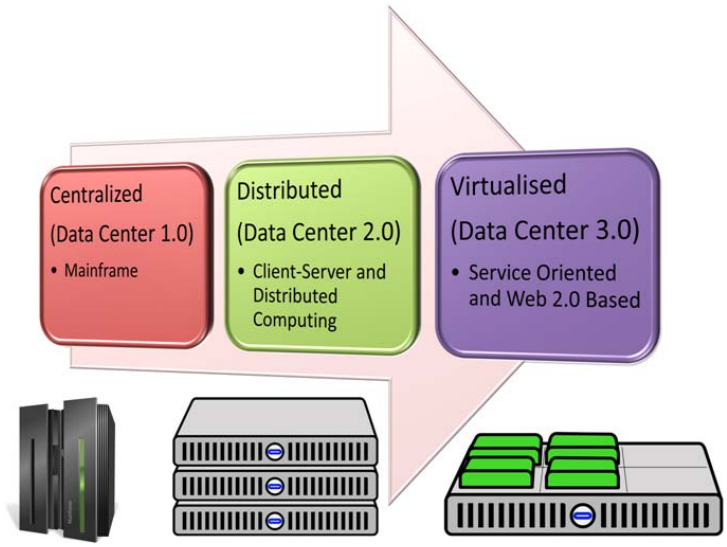
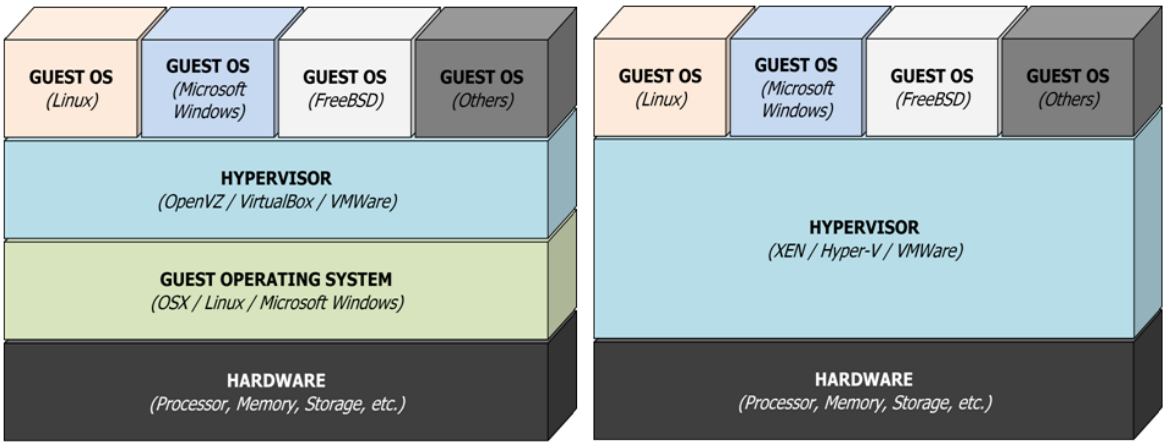


FIGURE 1. The data center evolution



(a) (b)

FIGURE 2. Comparison between (a) hosted and (b) native VMM

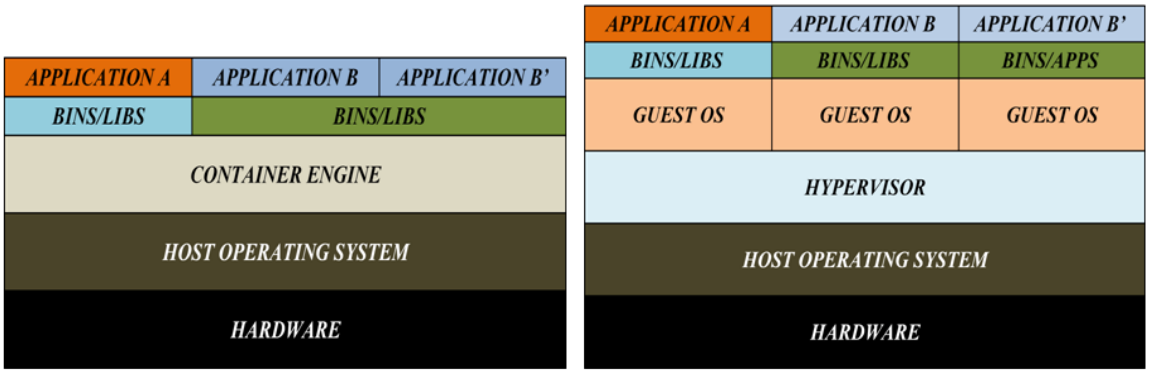
3. **Virtualization and Self-Repair Network Model.** We simulated a resilient server using virtualization technology and a self-repair network model. The detail of the virtualization and self-repair network model that is used in this paper is explained as follows.

3.1. **Virtualization.** Many technologies were used with the virtualization such as for simulation (desktop) and operational (server) purpose. In this paper, we divide the virtualization technologies into two parts. The first technology involves a Virtual Machine Monitor scheme and the second technology involves a Container scheme.

3.1.1. *Virtual machine monitor (hypervisor).* There are two types of Virtual Machine Monitor (VMM): native and hosted. The native VMM (also known as bare-metal) is a virtualization that runs directly on the computer hardware to control and manage the guest operating systems. In comparison, hosted VMM is a virtualization that runs under a common operating system (such as Microsoft Windows, Mac OS, etc.). This means that native VMM is the most suitable type to run the virtual machine as a server. Figure 2 illustrates the comparison between the two types of VMM.

3.1.2. *Container.* Container (also called a Linux Container) technology has become popular over the last two years [8]. Linux Container (LXC) offers several features and advantages over a VMM in which LXC claims to be faster and more efficient in its use of resources [9-11]. It uses a different virtualization technique than that of VMM. In particular, the LXC does not simulate the hardware environment. However, the LXC isolates its own domain to execute particular guest programs and acts as if the program is running on a separate system. Figure 3 shows the comparison between the LXC and hosted VMM schemes. OpenVZ, Docker and Wardern are the variants of the LXC. However, LXC has several weaknesses due to:

- Running only on a Linux operating system.
- Can only virtualize applications that run on the Linux OS.
- Uses a shared kernel.
- Is vulnerable to the containment environment.



(a) (b)

FIGURE 3. Comparison between (a) container and (b) hosted VMM schemes

3.2. **Self-repair network model.** A Self-Repair Network (SRN) model has been discussed in order to develop a resilient system [12]. There are two main basic models for a Self-Repair Network (SRN) model [13]. The first model, called *self-repair* (intra-node repair), in which the nodes can repair themselves by removing contaminated parts or resetting the state (Figure 4(a)). The second model, called *mutual-repair* (inter-node repair), in which the connected nodes are capable of repairing the other nodes (Figure 4(b)). We can combine the *self-repair* and the *mutual-repair* in two ways: *mixed-repair* (Figure 4(c)) and *switching-repair* (Figure 4(d)).

In the virtualization environment, we can assume that the server acts as a node. The SRN model implements to the virtualization environment to repair or recover the failure. In this paper, the *self-repair* and the *mutual-repair* are implemented to the virtualization within the homogeneous environment (Section 4.1). Meanwhile, the *mixed-repair* and the *switching-repair* are implemented to the virtualization with the heterogeneous environment (Section 4.2).

4. **Virtualization Environment Design.** Since we are using virtualization technology, we can create the virtualization in various environments. In this experiment we designed homogeneous and heterogeneous environments. Based on Section 3, there are two possible virtualization technologies that we can use to design the simulation. The first is that we can use VMM technology and the other by using Container technology. Container performs faster and is more efficient than VMM, however, the level of diversity for the

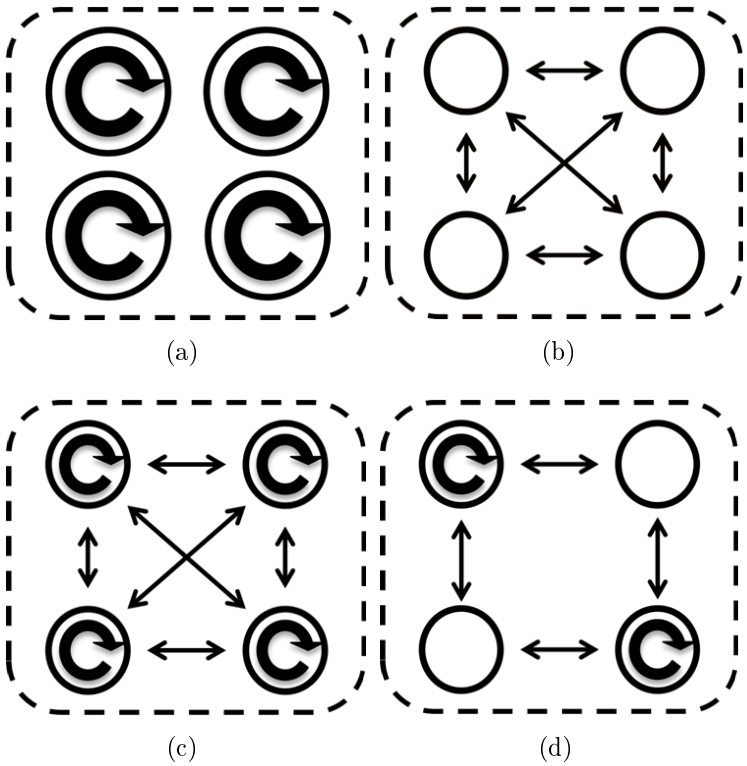


FIGURE 4. (a) Self-repair, (b) Mutual-repair, (c) Mixed-repair, (d) Switching-repair. An arrow indicates the direction from a repairing node to a node being repaired.

container is lower than VMM [20]. Consequently, we only use VMM technology to design the virtualization environment. Both of the virtualization environment designs were tested using simulation scenarios (Section 5).

4.1. **Homogeneous environment.** The definition of virtualization with a homogeneous environment means that we are running the virtualization with the same (homogeneous) application types, the guest Operating System (OS), and architecture of the virtual hardware (i.e., CPU). In this design, we implement the SRN on the virtualization environment through the simulation using XEN. XEN is a native VMM and free open source software [14]. We use XEN version 4.1 as a hypervisor that runs under Debian 7.8 GNU/Linux with an x86 CPU architecture. The hypervisor has an important function to monitor and manage the nodes. We create four nodes as a server (Figure 5) and modify the nodes giving them the same specifications such as memory and storage disk capacity since we are focusing on a homogeneous environment. We define each node by giving them storage that contains separate disk partitions. Figure 6 shows an abnormal node and normal node where an abnormal node has an infected file in one or more of the disk partitions. Moreover, we can store the disk partition as a file (Table 1).

4.2. **Heterogeneous environment.** In contrast to the homogeneous environment, in the heterogeneous environment we have to make the nodes different to each other. In this design we distinguish each node by the operating system and the application running specific services (e.g., web server or DNS server).

For example, node X is running a Linux operating system and an Apache web server, meanwhile node Y runs a Windows operating system and an IIS web server. Similar to the homogeneous environment, we also design four nodes as a server (Figure 7).

TABLE 1. Partition table of each node

Disk partition	Mount point	Size (GB)	Disk filename	Usage
1	/	1	data1.img	file system
2	/var	0.5	data2.img	variable data
3	/usr	0.5	data3.img	usr-land programs and data
4	/home	0.5	data4.img	home directory

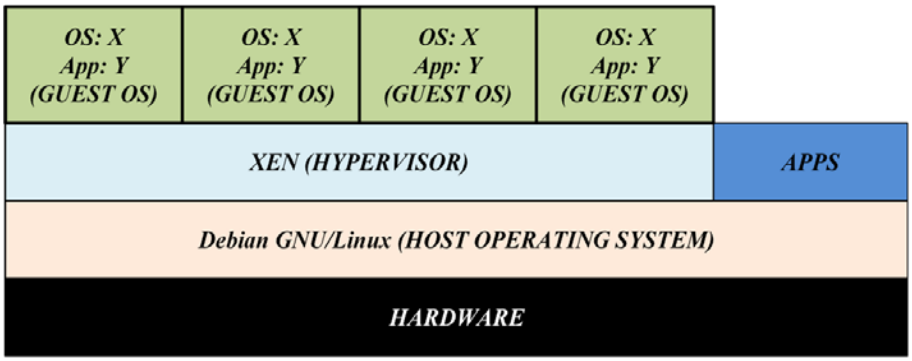
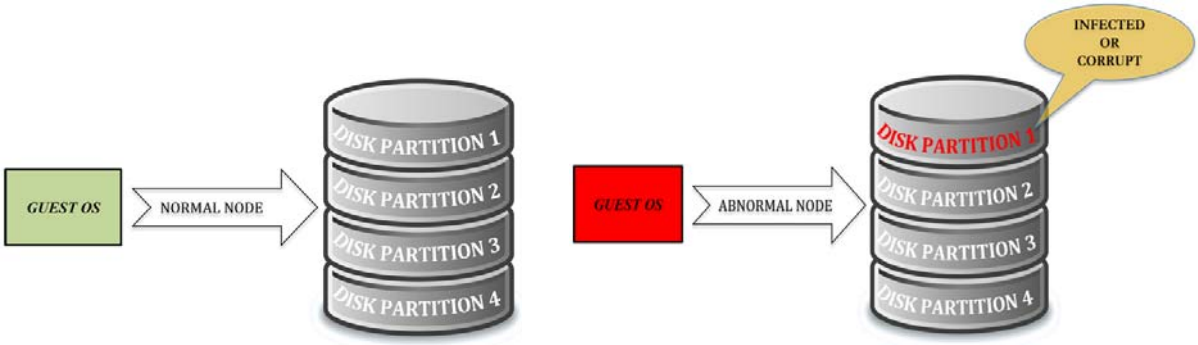


FIGURE 5. Logical design of the homogeneous environment



(a) (b)

FIGURE 6. Disk partition design on each node: (a) normal node, (b) abnormal node

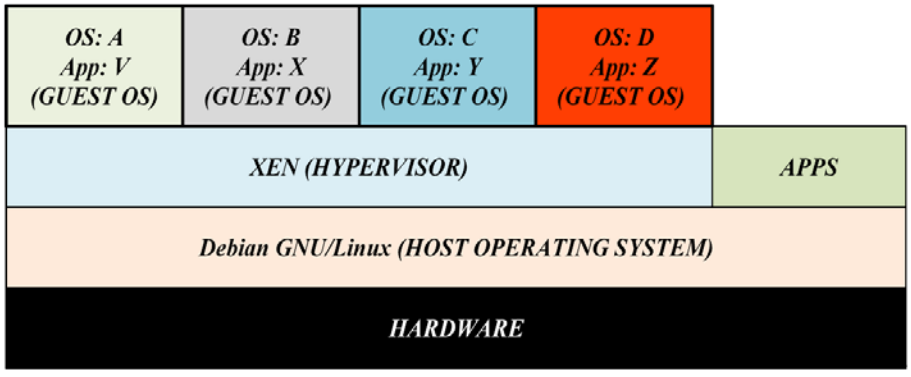


FIGURE 7. Logical design of the heterogeneous environment

5. **Simulation Scenarios.** To simulate the failures that occur on the virtualization with the homogeneous and heterogeneous environment, we define the three scenarios as follows:

a) Hang scenario: a node totally stops functioning
b) Denial of Service (DoS) scenario: a node being attacked by a DoS
c) Virus scenario: a node being infected by a virus where the contaminated node may infect other nodes

Therefore, we need to install an application or script that operates as a sensor. The sensor will detect and recognize an abnormal condition present on the nodes. These sensors are inspired by an immunity-based system [21] when the credibility values of the sensor (R) are 0 and 1. ICMP, TCP and UDP ping are examples of the network transport that we can use to create a script to detect the existence of the nodes from the host operating system or hypervisor. Further, in addition to installing the sensor on the host operating system or hypervisor, we have to install the sensor on each of the nodes (e.g., anti-virus and tripwire). When the sensors detect an abnormal activity, the hypervisor and the nodes respond to the sensor alert by initiating the SRN model (e.g., *self-repair* or *mutual-repair*) which is followed by sending a report to the network administrator. The sensors are executed by the script that runs the hang, DoS and virus scenarios (Algorithms 1, 2 and 3, respectively). We need to evaluate the node condition (N) as to whether the node is getting faulty (value = 1) or not (value = 0) by checking the credibility ($0 \leq R \leq 1$) with the threshold as shown in Equation (1). In this simulation we assume that the threshold value is 0.5. The assumption of the threshold value refers to the middle value where if the threshold value is too low then the node is easily determined as the faulty node and vice versa.

$$N = \begin{cases} 1, & R > 0.5 \\ 0, & R \leq 0.5 \end{cases} \tag{1}$$

$$R = \begin{cases} R + 0.1, & I = 0 \\ R - 0.1, & I = 1 \end{cases} \tag{2}$$

$$R = \begin{cases} R + 0.1, & I_c \geq 300 \\ R - 0.1, & I_c < 300 \end{cases} \tag{3}$$

5.1. **Hang scenario.** There are many potential activities that could trigger a hang condition [15]. However, for this simulation we created a script that runs as a sensor to check the node condition as to whether the node is in normal condition or in abnormal (hang) condition by utilizing an ICMP ping. This script is executed by the host operating system (hypervisor) as a *daemon*. Algorithm 1 shows the algorithm of the script. To simulate the hang scenario on the nodes, we use the halt command to the node to create a hang condition. When the script did not (value = 0) receive the ICMP ping (I) request from the node within a particular period, it makes the credibility (R) value increase 0.1 and vice versa (Equation (2)). The host operating system (hypervisor) will assume that the nodes are in the halt condition when the R value is above the threshold and the hypervisor resets the node by restarting the node (intra-repair). After the resetting node becomes normal, the credibility of the node will increase along with the resetting node sending the ping request to the hypervisor.

5.2. **DoS scenario.** DoS is one of the various techniques often used by attackers to interfere with specific services (e.g., web server). It is difficult to overcome this attack [16], since DoS works by flooding traffic directly to the service and causing abnormal operation. We used the *slowhttptest* script [17] to simulate a DoS attack to the web server. In this scenario, we create a script by utilizing the netstat command to check

Algorithm 1 Pseudo-code implementation of the script for hang scenario

```
R ← 0
threshold ← 0.5
repeat
    if receiveIcmpRequest = FALSE then
        if R ≥ 1 then
            if R ≥ threshold then
                resetTheHangNode()
            end if
        else
            R ← R + 0.1
        end if
    else if R ≤ 0 then
        R ← R − 0.1
    end if
until scriptIsStopped
```

Algorithm 2 Pseudo-code implementation of the script for DoS scenario

```
R ← 0
threshold ← 0.5
repeat
    if N_c ≥ I_c then
        if R ≥ 1 then
            if R ≥ threshold then
                addTheSuspectedIpToFirewall()
        end if
        else
            R ← R + 0.1
     end if
    else if R ≤ 0 then
            R ← R − 0.1
    end if
until scriptIsStopped
```

the number of connections on a particular service and whether the connections are over the limit or not (Algorithm 2). When the script detects the number of connections (N_c) from an IP address that is accessing particular services is higher than the threshold of the maximum number of connections (I_c) within a certain period, it makes the credibility (R) value increase 0.1. However, if the N_c from the same IP address is detected lower than the threshold of I_c on the next check, it will make the R value of the IP address decrease 0.1 (Equation (3)). When the R value of the suspected IP address is higher than the threshold, the script will add the IP address to the firewall list using the iptables command [16] (intra-repair).

5.3. **Virus scenario.** Since we separate the file system and other paths (Table 1) into the different partition disk (Figure 6), we could repair the contaminated partition disk in two ways: (1) recovering by antivirus, and (2) switching with the normal node. We utilized antivirus and tripwire to detect the abnormal activity that was caused by the viruses. When the antivirus or tripwire detects the virus, then the antivirus tries to

TABLE 2. The credibility value of the partition disk

Disk Partition ID#	Mount point	Credibility
1	/	0.2
2	/var	0.2
3	/usr	0.2
4	/home	0.4

Algorithm 3 Pseudo-code implementation of the script for virus scenario

```
R ← 0
threshold ← 0.5
repeat
    if detectVirus = TRUE then
        R_p ← partitionDiskInfection()
        R ← R + R_p
        if R ≥ threshold then
            isFound ← findNormalNodeWithDifferentApplication()
            if isFound = TRUE then
                switchTheAbnormalNodeWithStandbyNode()
            else
                isolateTheNode()
            end if
        end if
    else
        if R ≤ 0 then
            R ← R − 0.1
        end if
    end if
until scriptIsStopped
```

remove the virus and inform the script that is running on the hypervisor. The script will then update the credibility (R) value of the node while the antivirus is trying to remove the virus. We assume the credibility ($0 \leq R \leq 1$) of the node based on the partition disk of the nodes (R_p) (Table 2). The nodes are healthy (normal) when the R value is 0 and vice versa. After the R values have been updated by the script, then the script checks whether the antivirus has succeeded or not. If the R value is higher than the threshold, the script continues to search the normal node ($R = 0$). If there are no normal nodes then the script will isolate (turn off) the infected node in order to reduce the risk of the virus spreading to the other nodes. However, if the script found the normal node with a different application, then the abnormal node will be switched with the standby node (Algorithm 3).

6. **Main Results.** Using the scenarios that we defined, we simulated them on the homogeneous and heterogeneous environments as follows.

6.1. **Hang simulation.** In either the homogenous or heterogeneous environment, when the node does not send the ping request on the particular node to the hypervisor then the script will increase the credibility value of the node (R) and vice versa. Tables 3 and 4 show the test case result from the hang scenario. Some nodes may not send the ping

request to the hypervisor due to heavy load processes occurring inside the node for a certain period (Table 3, node id #2 and 4). However, when the nodes return to a normal state then the R value will decrease on the next check. All of the nodes in Table 3 show that the R value is higher than the threshold and consequently this condition will trigger the hypervisor to reset the node. When the node changes from an abnormal (X) to a normal (O) state after the reset process then the R value will decrease on the next check. Since the homogeneous environment has the same structure (OS), the probabilities of the hang faulty recovery are the same (Table 3). On the other hand, the heterogeneous environment has different probabilities for a hang faulty recovery (Table 4).

TABLE 3. Test cases of hang scenario on the homogeneous environment

Node ID #	OS	IP Address	Credibility	State	Action
1	M	192.168.0.1	0.6	X	Reset
2	M	192.168.0.2	0.6	X	Reset
3	M	192.168.0.3	0.6	X	Reset
4	M	192.168.0.4	0.6	X	Reset

TABLE 4. Test cases of hang scenario on the heterogeneous environment

Node ID #	OS	IP Address	Credibility	State	Action
1	D	192.168.1.1	0.0	O	–
2	R	192.168.1.2	0.1	O	–
3	M	192.168.1.3	0.6	X	Reset
4	S	192.168.1.4	0.2	O	–

6.2. **DoS simulation.** In order to simulate a DoS attack, we use *slowhttptest* as an attacking tool to the web server. The simulations are run on both the homogenous and heterogeneous environments. The first action of this simulation is to define the target IP address of the web server. When the *slowhttptest* is executed, it creates multiple concurrent connections to the web server and makes the web server busy or operationally hampered. The nodes that are being attacked are running a script (Algorithm 2) that utilizes the netstat command to detect unusual connections coming from a particular IP address. Tables 5 and 6 show that homogeneous and heterogeneous environments have the same probability to become abnormal node (X) when the Denial of Service (DoS) attacks the node. The script will add the IP address that is suspected of being an attacker to the firewall list by utilizing the iptables command if the credibility value is higher than the threshold.

TABLE 5. Test cases of DoS scenario on the homogeneous environment

Node ID #	Service	Suspected IP Address	Credibility	State	Action
1	A	192.168.0.100	0.6	X	Block
2	A	192.168.0.100	0.6	X	Block
3	A	192.168.0.100	0.6	X	Block
4	A	192.168.0.100	0.6	X	Block

TABLE 6. Test cases of DoS scenario on the heterogeneous environment

Node ID #	Service	Suspected IP Address	Credibility	State	Action
1	T	192.168.1.100	0.6	X	Block
2	N	192.168.1.100	0.6	X	Block
3	A	192.168.1.100	0.6	X	Block
4	L	192.168.1.100	0.6	X	Block

TABLE 7. Test cases of virus scenario on the homogeneous environment

Node ID #	OS	IP Address	Infection Disk Partition ID#	Credibility	State	Action
1	M	192.168.0.1	1, 4	0.6	X	Turn Off
2	M	192.168.0.2	1	0.2	X	Turn Off
3	M	192.168.1.1	–	0	O	–
4	M	192.168.1.2	–	0	O	–

TABLE 8. Test cases of virus scenario on the heterogeneous environment

Node ID #	OS	IP Address	Infection Disk Partition ID#	Credibility	State	Action
1	D	192.168.2.1	–	0	O	–
2	R	192.168.2.2	–	0	O	–
3	M	192.168.2.3	1, 4	0.6	X	Switch
4	S	192.168.2.4	–	0	S	–

6.3. **Virus simulation.** In this simulation we create a script by involving antivirus and tripwire. The script collects information from the antivirus or tripwire whether the virus or anomaly inside the node is found (X) or not (O). In the homogeneous environment, the probabilities of the virus infection recovery on each node are the same. Meanwhile, the heterogeneous environment has different probabilities for virus infection recovery. Tables 7 and 8 show the test case results of the virus infection in the homogeneous and heterogeneous environment. There are two actions that can be executed by the script on each node: (1) Turn off or isolate the infected node, and (2) switch the infected node with the standby node.

Table 7 shows that the script on node id #1 detected the virus and that the credibility value was over the threshold and the script responded by turning off (isolating) the infected node. Similar to node id #1, node id #2 is isolated from the network by turning off the node. Meanwhile, Table 8 shows that node id #3 has the same condition with node id #1 in Table 7. However, since the nodes are running in the heterogeneous environment then the script responds by switching the infected node with the standby node (state = S). While node id #4 replaces the position of node id #3, we could repair the infected node. This repair makes the application that runs vital services able to provide the service continuously and could reduce the chances of virus infection to the entire system (data center).

Container (LXC) technology performs faster and more efficiently uses resources (e.g., memory consume) than Virtual Machine Monitor (VMM) technology. LXC performs 10 times faster than VMM [18]. However, the VMM has much more types of guest operating systems than the LXC. For that reason VMM has a higher diversity than LXC to create

a heterogeneous environment. By increasing the diversity of the node we can reduce the risk of fault to the server [19].

7. **Conclusions.** The combination of virtualization technology and a Self-Repair Network (SRN) model may be used to make servers resilient against failures that occur during operation. Simulations using virtualization with a homogenous and a heterogeneous environment have been conducted. These simulations show that the server is able to recover from a failure in limited scenarios. Although Container technology offers higher performance than the Virtual Machine Monitor (VMM), VMM technology offers higher diversity. Similarly to the immune system, the diversity of nodes in the heterogeneous environment may lessen the vulnerability of the server since the virus in one infected node is unable to spread to other nodes.

In our future work, we will implement the SRN model on a larger-scale simulation of a resilient server and then evaluate the SRN's resilience and proactive capability in more realistic situations.

Acknowledgment. We would like to thank the anonymous reviewers for their valuable comments and suggestions that greatly contributed to improving the presentation. IW would like to thank DIKTI (Indonesian Government for Higher Education) for the scholarship.

REFERENCES

[1] I. Winarno and Y. Ishida, Simulating resilient server using XEN virtualization, *Procedia Computer Science*, vol.60, pp.1745-1752, 2015.
[2] C. D. Patel and A. J. Shah, *Cost Model for Planning, Development and Operation of a Data Center*, 2005.
[3] B. Ian, *Data Centre Evolution: The Role of the Network in Data Centre Transformation*, Cisco, 2008.
[4] R. S. Couto, M. E. M. Campista and L. H. M. K. Costa, A reliability analysis of datacenter topologies, *Global Communications Conference*, 2012.
[5] A. Greenberg et al., VL2: A scalable and flexible data center network, *ACM SIGCOMM Computer Communication Review*, vol.39, no.4, 2009.
[6] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*, 2nd Edition, Pearson, 2007.
[7] F. Sano, T. Okamoto, I. Winarno, Y. Hata and Y. Ishida, A cyber attack-resilient server inspired by biological diversity, *Proc. of the 21st International Symposium on Artificial Life and Robotics*, 2016.
[8] D. Merkel, Docker: Lightweight linux containers for consistent development and deployment, *Linux Journal*, vol.14, no.239, 2014.
[9] D. Rajdeep, R. A. Reddy and K. Dharmesh, Virtualization vs containerization to support PaaS, *Proc. of IEEE International Conference on Cloud Engineering*, pp.610-614, 2014.
[10] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, An updated performance comparison of virtual machines and linux containers, *IBM Research Report*, 2014.
[11] A. M. Joy, Performance comparison between linux containers and virtual machines, *Proc. of International Conference on Advances in Computer Engineering and Application*, pp.342-346, 2015.
[12] Y. Ishida, Special issue on immunity-based systems: Systems sciences for robust and resilient engineering, *International Journal of Innovative Computing, Information and Control*, vol.10, no.1, pp.315-316, 2014.
[13] Y. Ishida, *Self-Repair Network: A Mechanism Design*, Springer, 2015.
[14] D. Schlosser, M. Duelli and S. Goll, Performance comparison of hardware virtualization platforms, *The 10th International IFIP TC 6 Networking Conference, Lecture Notes in Computer Science*, vol.6640, pp.393-405, 2011.
[15] D. Cotroneo, R. Natella and S. Russo, Assessment and improvement of hang detection in the linux operating system, *Proc. of the 28th IEEE International Symposium on Reliable Distributed Systems*, pp.288-294, 2009.

[16] B. Q. M. Al-Musawi, Mitigating DoS/DDoS attacks using iptables, *International Journal of Engineering & Technology*, vol.12, no.3, 2012.
[17] *Slowhttptest Application Layer DoS Attack Simulator*, https://code.google.com/p/slowhttptest/.
[18] I. Winarno, T. Okamoto, Y. Hata and Y. Ishida, Implementing SRN for resilient server on the virtual environment using container, *Intelligent Systems Research Progress Workshop*, 2015.
[19] C. Pu, *A Specialization Toolkit to Increase the Diversity in Operating Systems*, Ph.D. Thesis, Portland State University, 1996.
[20] R. Rosen, Linux containers and future cloud, *Linux Journal*, vol.204, 2014.
[21] Y. Ishida, *Immunity-Based System: A Design Perspective*, Springer, 2004.