Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 96 (2016) 1701 – 1709

Procedia Computer Science

20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

Increasing The Diversity of Resilient Server using Multiple Virtualization Engines

Idris Winarno[a,*], Takeshi Okamoto[b], Yoshikazu Hata[a], Yoshiteru Ishida[a]

[a]*Toyohashi University of Technology, Tempaku, Toyohashi, Aichi 441-8580, Japan*
[b]*Kanagawa Institute of Technology, 1030, Shimo-ogino, Atsugi, Kanagawa 242-0292, Japan*

Abstract

The Virtual machine monitor and container is an example of virtualization engine technologies that are currently in wide use. Both types of virtualization engines have vulnerabilities that could hamper a virtual machine. In our previous work, we created a resilient server using virtualization technologies with a single virtualization engine. To improve the resilience of the server, we use two different types of virtualization engines that monitor and respond by the SRN manager. The experiments results show that using multiple virtualization engines could cover their vulnerabilities against limited scenarios of failure.

Peer-review under responsibility of KES International

Keywords: Multiple virtualization engine, resilient server, Self-repair network

1. Introduction

Server existence becomes critical in an information system since its core functionality is to provide and process information required by users. Currently, server technologies have shifted from traditional to an era of virtual architecture as shown in Fig.1. Virtual machine (VM) is an implementation of virtual architecture. Although VM's have weaknesses and challenges that have yet to be solved[11], they are still very popular as they offer a lot of benefits (i.e. portability and energy efficiency). There are several types of VM model that will be used in this paper that will be discussed in Section 5.

* Corresponding author. Tel.: +62-81-2309-8214
E-mail address: idris@pens.ac.id

Peer-review under responsibility of KES International
doi:10.1016/j.procs.2016.08.218

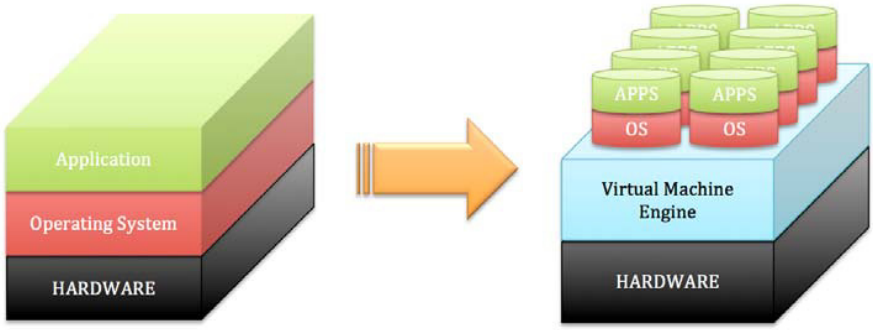

Fig. 1. Architecture of traditional (right figure) and virtual (left figure) server.

Since the server plays such an important role in an information system, we have to preserve its functionality. When the servers encounter a failure, the information system is hampered. Some scenarios that cause the server to stop functioning are hang, denial of service (DoS) attack, and computer virus. We never know when, how or how often this failure will occur. Thus, many researchers have tried to prevent and solve the problem by involving some type of mechanism. Fault injection framework[6] is used as a hang detection process used to find the hang problem. While, SecondDEP[9] is introduced to prevent a zero-day attack on a Windows operating system (OS). ReVirt[7] can conduct an analysis of intrusion through virtual-machine logging and replay.

We have simulated a resilient server using XEN[18] as a virtual machine monitor (VMM) and compare the performance using Docker as a container. Docker with container technology performs faster and is more resource efficient than VMM[20]. However, VMM provides higher diversity than container[19] since VMM is able to virtualize the hardware environment for their VM rather than sharing the kernel of the host operating system. We need to increase the diversity of the server with an aim to decrease failures that occur on the server. This paper presents a new model of a resilient server by combining the VMM and container by implementing a self-repair network (SRN) model[8].

2. Related Work

One of the mechanisms available for the server in order to avoid system failure is by the utilization of a fault-tolerant system where the key point of a fault-tolerant system is redundancy[16]. Fault-Tolerant focuses on "passive action" where the system continues functioning without major changes, while a resilient server focuses on "active action" where the system is able to adapt to environmental changes by changing fundamental methods or structures.

The idea of increasing the diversity of resilient server is inspired by the diversity of an operating system[10] where the diversity is focused on the kernel modules of the operating system. Since the current server architecture uses virtual machine technology, we can have a higher diversity of operating systems than with traditional architecture as shown in Fig. 1. Meanwhile, the recovery model of the server to maintain functionality when failures occur is motivated by an immunity-based system[8] that will be discussed in Section 4. Further, we have to consider that Internet services (e.g., DNS) are required to be secured from a cyber-attack. Our work is focused on making the server more resilient against cyber-attack by involving virtual machine technology[13] and it is inspired by the concept of biological diversity[17].

3. Resilient Server

We have created two types of resilient servers: a homogeneous and a heterogeneous resilient server. Further, we create several VM's that we refer to as nodes. Each of the resilient servers has their advantages and disadvantages. The homogeneous resilient server offers a facility to replace or copy the failed part from the other nodes. However, the homogeneous resilient server has the same vulnerability on all of the nodes, since they have the same structure. Meanwhile, the heterogeneous resilient server offers a higher diversity of node that makes the node become more resilient than the homogeneous resilient server though the heterogeneous resilient server cause higher operating cost

for setup, operation and maintain the entire server since it has the diversity of server. Therefore, we need to increase the diversity of the heterogeneous resilient server.

3.1. Homogeneous resilient server

On our previous research[18,20], we created a homogeneous resilient server using virtual machine monitor (VMM) and container technology as the virtualization engine. There are two types of VMM: native and hosted. Native VMM structure consists of three layers (hardware, VMM, and Guest OS) while hosted VMM structure consists of four layers (hardware, host OS, VMM, and guest OS). We created the homogeneous resilient server with four nodes running the same structure (e.g., operating system, disk partition, service types, etc.) using native VMM as shown in Fig. 2a. In addition, we created the homogeneous resilient server using container technology since it offers higher performance than VMM.

3.2. Heterogeneous resilient server

Similar to the homogeneous resilient server, we create the heterogeneous resilient server using VMM and container technology[19]. Further, in contrast to the homogeneous resilient server, we can create several types of heterogeneous resilient server with some combination of virtualization engine, guest OS and applications (services) as shown in Table 1. Type 1 of resilient server shows as a homogeneous resilient server since there is no different combination (X) of virtualization engine, guest OS and application on each node. Meanwhile, type 2 to type 8 shows as the heterogeneous resilient server since there is a different combination (O) on each node. Fig. 2b illustrates the structure of the heterogeneous resilient server with type 3 or type 4.

Table 1. Types of resilient server

Types of Resilient Server	Virtualization Engine	Guest OS	Application (Service)
Type 1	X	X	X
Type 2	X	X	O
Type 3	X	O	X
Type 4	X	O	O
Type 5	O	X	X
Type 6	O	X	O
Type 7	O	O	X
Type 8	O	O	O

Both of them, homogeneous and heterogeneous resilient servers are equipped with an application that monitors and responds to the failures that occur on the nodes. This application is created by implementing the self-repair network (SRN) model and called the SRN manager. Fig. 2 shows that each of physical servers is running the SRN manager by involving the virtualization feature.

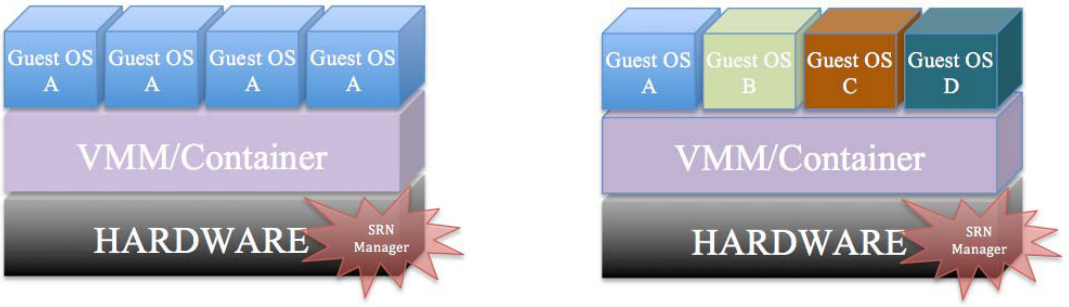

Fig. 2. Structure of (a) the homogeneous and (b) heterogeneous resilient server.

In this paper we build a resilient server with type 8, which aims to increase the diversity of the resilient server by combining two different virtualization engine technologies (VMM and container). The detail of the design is explained in Section 5.

4. The Self-Repair Network (SRN) Model On The Resilient Server

There are two main basic models of self-repair network (SRN) model (*self-repair* and *mutual-repair*)[8]. Further, these two models are extended into four models of SRN (*mixed-repair* and *switching-repair*). All of the SRN models are implemented to the SRN manager (discussed in Section 5.2), which is executed on the homogenous and heterogeneous resilient server. Details of the SRN model are as follows:

4.1. Self-repair model

When the nodes encounter a failure (e.g. hang), we implement the *self-repair* model to recover the node from its faulty state. The *self-repair* model has a characteristic to repair their part or component. Hang is one of the failures that are hard to be recovered[6]. However, through VM technology, we can realize the *self-repair* to recover the hang failure that occurs to the node by resetting the node. This solution cannot guarantee that the failure will not happen in the future, but this solution can be an alternative way for the node to keep running services while the administrator investigates the problem.

4.2. Mutual-repair model

Another failure that possibly occurs to the server is with the alteration of files on a server such as web contents. For example, attackers modify web contents in order to insert some iframe elements so that the attacker redirects user's access to their exploit kit server that attacks various vulnerabilities on a PC. The *mutual-repair* is implemented to solve this kind of problem since it has the characteristic to repair the other nodes. When the alteration is detected by a security tool such as Tripwire, the missing or modified files can be copied from the other nodes as long as they have the same structure or composition. However, we need to be aware of the "double edge sword" phenomena[8] since copying from the other nodes is not the best solution to solve the problem. In addition, mutual-repair is not a viable solution unless the vulnerability that allows the attacker to modify files on the server can be solved.

4.3. Mixed-repair model

The combination of the *self-repair* and the *mutual-repair* model is a *mixed-repair* model. This model can be implemented to repair the node from a fault occurring from such an event as a denial of service (DoS) attack. This attack often happens when a hacker tries to hamper services (e.g. web services), and we assume that the DoS attack is based on TCP. DoS attacks work by sending a lot of TCP connections to particular services in order to make those services unable to process requests from normal users. Since the *mixed-repaired* is applied, the implementation of *self-repair* is reflected when a hacker attacks a node, then the attacked node responds by adding the IP address of the attacker to the firewall rule. Further, the attacked node informs the virtualization engine where the attacked node runs to drop all of the connections that come from the hackers IP address in order to secure the other nodes. Whereby, the implementation of *mutual-repair* is reflected when the attacked node informs the IP address of the attacker to the different virtualization engines to secure their nodes.

4.4. Switching-repair model

In the heterogeneous environment, the biggest challenge for the node is how to replace the faulty part from the other nodes when they have a different structure. For a solution to this problem, we can implement a *switching-repair* model, which works by replacing the failure node with the normal node rather than fixing the faulty part. This solution is more expensive compared to the other model since we have to provide a normal node to replace the

faulty node. However this solution offers higher resilience when compared to the other model since it provides a higher diversity of nodes.

5. System Design and Implementation

It is not only the applications or the guest operating systems that may have a failure but virtualization engines also may have a risk of failure during operation. To reduce the risk of failures that can possibly occur on virtualization engines we provide multiple virtualization engines and an SRN manager. Fig. 3 shows the design of the network for the experiment where we used three physical servers for VMM, Container, and SRN Manager. While, the description of multiple virtualization engines and SRN manager is explained as follows:

5.1. Multiple virtualization engines

To actualize type 8 of the resilient server on Table 1, we need to use a distinct virtualization engine on each of the physical server machines. Virtualization engine is an environment where the VM executed. If the virtualization engine fails, then all the VM running on it will stop functioning. Therefore, we have to preserve the existence of the virtualization engine by providing an alternative VM technology to prevent failure on the virtualization engines (e.g., CVE-2016-1571[5], CVE-2014-8866[3]). There are two VM technologies that we use in this paper (VMM and container). The combination of them will give the advantage of increasing diversity at the virtualization engines level. Since the VMM is used for server operation, accordingly XEN as native VMM are chosen. We choose XEN as native VMM since it is freeware and successful (i.e., Citrix Suites). Further, XEN provides flexibility and ease of integration[14] with other applications and programming libraries to support the SRN Manager. The other virtualization engines that we use as container is LXC. We use LXC since it has simple design, which makes the LXC have less vulnerability. Further, LXC provides a facility to configure the network for the VM^2 without involving the other application (e.g., `iptables` command). Fig. 4 shows the system design of the resilient server with type 8 where the virtualization engine is different on each physical server machine. Table 2 shows the detail of the specification on each of the physical server machines. This table show the diversity of OSes although the LXC use the same kernel since its use shared kernel of the host OS.

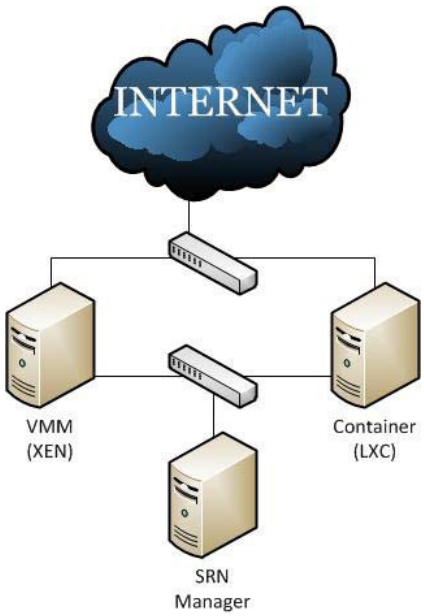

Fig. 3. Design of the network for the experiment.

Table 2. The detail of the specification on each physical server machine

Virtualization Engines	Guest OS	Build number (release)	Kernel version	Application
VMM (XEN)	Debian GNU/Linux	7.8 (Wheezy)	3.2.0-4-amd64	Apache
	Ubuntu	12.04 (Precise)	3.11.0-15-generic	Apache Tomcat
	Fedora	22	4.0.4-301.fc22.x86_64	Nginx
	Windows 7	7601	-	IIS
Container (LXC)	CentOS	6.7 (Final)	3.16.0-4-686-pae	Lighttp
	Debian GNU/Linux	8 (Jessie)	3.16.0-4-686-pae	Apache Tomcat
	Fedora	23	3.16.0-4-686-pae	Apache
	Ubuntu	16.04 (Xenial Xerus)	3.16.0-4-686-pae	Nginx

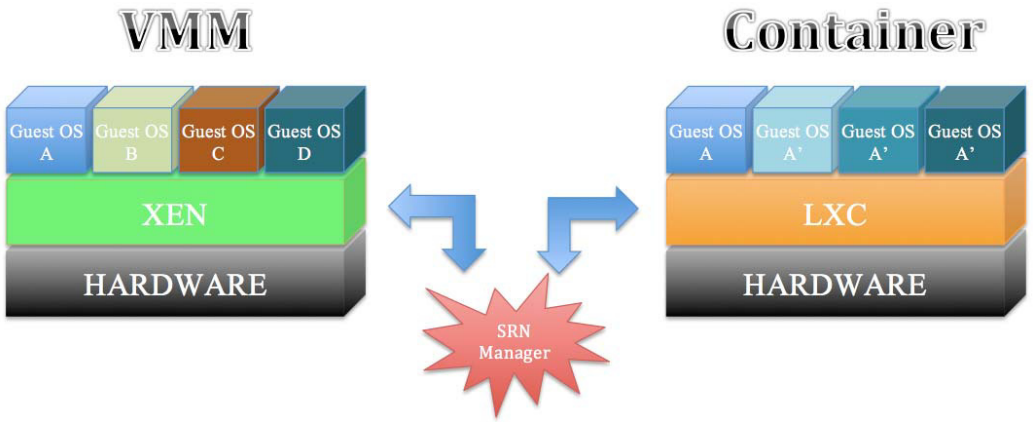

Fig. 4. Resilient server with two different virtualization engines.

5.2. SRN Manager

As described in Section 4, the SRN manager implements the SRN model to solve several failures that possibly occur on the server. Therefore, we need to assume some scenarios of failures that could be addressed by the SRN manager. In this paper, we define three scenarios of failure related to our previous works[18,19]: (1) hang, (2) DoS attack, and (3) malware. When the SRN manager detects unusual activity (e.g., DoS attack), then it will respond by utilizing the virtualization engine feature. Since we use multiple virtualization engines, we have to accommodate all of the virtualization engines features. For example, when a hang failure happens on the XEN then the SRN manager has to reset the node using a command as follows:

```
# xm reset <Domain>
```

or we also can reboot the the node by this command:

```
# xm reboot <Domain>
```

A different way to reset the node when the hang failure happens on the LXC is for the SRN manager to reset the node using the following command:

```
# lxc-stop -n <container_name>
```

and we can continue to start the node using the following command:

```
# lxc-start -n <container_name>
```

Another example of the failure is when the node is being attacked by a DoS attack. When a particular IP address is suspected to be from an attacker, the SRN manager updates the firewall rule to all of the virtualization engines in order to protect all of the nodes. Since XEN and LXC using the same host OS, we can use the same command to update the firewall rule using `iptables` to drop the forwarded packet that heads toward the other nodes[1]. Here is the `iptables` command to drop the forwarded command that lead to the other nodes:

```
# iptables –I FORWARD –s <attacker IP address> -j DROP
```

There are two options where the SRN manager can be placed. The first option, we can place the SRN manager outside of the physical server. The second option, we can place the SRN manager inside one of the physical servers. In this work we use the first option for the design since this options offer ease and simple of code, however this option weaker than the second option. Further, we use the Secure Shell (SSH) protocol to communicate between the SRN manager and the remote physical server. One example to instruct the virtualization engine on the remote physical server is by executing the following command:

```
# ssh <host target> "<command-to-reset-the–faulty-node>"
```

To optimize the communication between the SRN manager and the remote physical server, we can use public key authenticaton[12].

6. Experiment Results

Based on the design that was shown in Fig. 3, we test the response of SRN manager when the nodes encounter a failure based on the scenario that was already mentioned in Section 5.2. The goal of the experiment is to know the advantages of the diversity of the virtualization engine for the existence of the nodes by comparing the result of this work (type 8 of resilient server) with the other types of resilient server that are using the same virtualization engines (e.g., type 2 to type 4).

6.1. Hang

On type 8 of the resilient servers, when the hang failure occurs to the node then the *self-repair* model is applied to repair the node from the hang condition. Both on the VMM and container, we use the `halt` command to trigger the hang condition on each node. Each node on the VMM and container has to send an ICMP packet to the SRN manager every minute to inform that the node is in normal condition. If the SRN manager did not receive an ICMP packet from the node that is running on the VMM or container then the SRN manager will assume that the node is in halt (faulty) condition. In this case the SRN manager instructs the virtualization engine to reset the faulty node. When the SRN manager is on the same physical server as the faulty node, the SRN manager directly instructs the virtualization engine to reset it. However, if the SRN manager is on a different physical server then the SRN manager instructs the remote virtualization engine to reset the faulty node through SSH protocol.

The hang failure that not only occurs on the node but also occurs in the virtualization engine (e.g., CVE-2015-5307[4]) could cause all of the nodes inside the virtualization engine to stop functioning. When this happens in the virtualization engine, the SRN manager implements the *switching-repair* model by activating the standby node on the other virtualization engine. Meanwhile, the network administrator manually identifies and solves the problem. Compared to the previous type of resilient server (type 2 to type 4), the type 8 of resilient has an ability to avoid the hang failure that happens on a homogeneous virtualization engine.

6.2. Denial of service (DoS)

Since we assume that the DoS attack is based on the TCP connection, we use *slowhttptest*[15] to simulate the DoS attack to hamper the web services. When the DoS attack is detected on one of the nodes in type 8 of the resilient server, then it will provide the attackers IP address to the SRN manager. *Mixed-repair* model is applied by the SRN manager to protect the node by adding the attackers IP address to the firewall rule. The *mixed-repair* model implements two action models. The first model, *self-repair* is implemented by adding the IP address of the attacker to the firewall rule in the node that is being attacked. Meanwhile, the *mutual-repair* is implemented by adding the IP address of the attacker to the firewall rule in both of virtualization engines (VMM and container) using `iptables` command in `forward` table. The entire packet from the attacker will be dropped by the firewall on both of the virtualization engines. On the previous type of resilient server (type 2 to type 4), the node only protects the node itself from the DoS attack, while in the type 8, the SRN manager helps the node to protect the others.

6.3. Malware

In the malware scenario, the SRN manager applied *self-repair* and *switch-repair* model. Since we use a heterogeneous environment for the nodes, it is hard to apply the *mutual-repair* model to the SRN manager. The *mutual-repair* model can be applied when the environment is homogeneous so that the normal node can repair the abnormal node by copying the normal component to the abnormal node[18]. On the type 8 of resilient server, when the malware infects the node then the node will respond by cleaning the malware by itself. However when the malware remains present the SRN manager instructs the virtualization engine to shutdown the abnormal node and switch to the other nodes that have the same service. In this case the SRN manager has to search the equivalent node of the abnormal node on both the VMM and container.

In this paper we only use two virtualization engines, XEN as VMM and LXC as container. We have to consider that the diversity of virtualization engines could increase the resilience of the server. However, when we decide to use many different virtualization engines then the consequence is that the SRN manager has to accommodate all of the features of the virtualization engine.

7. Conclusion

We implemented the Self-repair network (SRN) model to the type 8 of resilient server where this type has a higher diversity of resilient server than in our previous work[18,19,20]. This diversity includes applications (service), guest OSes, and virtualization engines. Further, in this work we added a new diversity to the resilient server by adding multiple virtualization engines. The result of the experiment shows that the diversity virtualization engines are able to rescue the nodes (servers) from function stoppage since the other virtualization engine is able to replace the faulty virtualization engine to keep the service running. In addition, the higher diversity of the resilient server will make the server more resilient, but we have to consider that this diversity causes the SRN manager to work harder. Further, in order to increase the resilience of the entire system we need equip the SRN manager with the action, which applied the SRN model.

In the future, we will distribute the SRN manager on each physical server machine so that the SRN manager can provide higher availability than the current design. In this work, the SRN manager is only provided by one of the physical servers. When physical servers who provide the SRN manager encounter a failure, there is no SRN available to monitor and respond to the threat to the server.

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. Al-Musawi BQM. Mitigating DoS/DDoS attacks using iptables. *International Journal of Engineering & Technology* 2012;12:101-111.
2. Banerjee T. Understanding the key differences between LXC and Docker. https://www.flockport.com/lxc-vs-docker/
3. CVE-2014-8866, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8866
4. CVE-2015-5307, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-5307
5. CVE-2016-1571, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1571
6. Cotroneo D, Natella R, Russo S. Assessment and Improvement of Hang Detection in the Linux Operating System. *In Proc. of the 28th IEEE International Syposium on Reliable Distributed Systems*; 2009. p. 288-294.
7. Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. *In Proc. of ACM SIGOPS Operating Systems Review* 36; 2002.
8. Ishida Y. *Self-Repair Networks: A Mechanism Design*. Springer; 2015.
9. Okamoto T. SecondDEP: Resilient Computing that Prevents Shellcode Execution in Cyber-Attacks. *Procedia Computer Science* 2015;60: 691-699.
10. Pu C. A specialization toolkit to increase the diversity of operating system. PhD Thesis. 1996. Portland State Univeristy.
11. Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends. *Computer* 2005; 38.5:39-47.

12. Saive R. SSH Passwordless Login Using SSH Keygen in 5 Easy Steps. 2015 http://www.tecmint.com/ssh-passwordless-login-using-ssh-keygen-in-5-easy-steps/
13. Sano F, Okamoto T, Winarno I, Hata Y, Ishida Y, A cyber attack-resilient server inspired by diversity, *In Proc. Of The 21st International Symposium on Artificial Life and Robotics*; 2016. p. 31-35.
14. Schluting C. VMWare or XEN? Depends on Your Fluency in Linux. 2009, http://www.enterprisenetworkingplanet.com/linux_unix/article.php/3836936/VMWare-or-Xen--Depends-on-Your-Fluency-in-Linux.htm
15. Slowhttptest Application Layer DoS attack simulator, https://code.google.com/p/slowhttptest/ □
16. Tanenbaum AS, Steen MV. *Distributed Systems: principles and paradigms (2nd edition)*. Pearson; 2007.
17. Tarao M, Okamoto T, Toward an artificial Immune server against cyber attacks. *In Proc. of the 21st International Sysmposium on Artificial Life and Robotics*; 2016. p. 36-39.
18. Winarno I, Ishida Y. Simulating Resilient Server Using XEN Virtualization. *Procedia Computer Science* 2015:60:1745-1752.
19. Winarno I, Okamoto T, Hata Y, Ishida Y. A resilient server based on virtualization with a self-repair network model. *International Journal of Innovatives Computing, Information and Control*. 2016. (to be appeared)
20. Winarno I, Okamoto T, Hata Y, Ishida Y. Implementing SRN for Resilient Server on The Virtual Environment Using Container. Intelegent System Research Progress Workshop; 2015. p. 32-37.