Bulletin of Electrical Engineering and Informatics
Vol. 9, No. 4, August 2020, pp. 1558~1568
ISSN: 2302-9285, DOI: 10.11591/eei.v9i4.2011

The enhancement of logging system accuracy for infrastructure as a service cloud

Surapong Wiriya[1], Winai Wongthai[2], Thanathorn Phoka[3]
[1,2,3]Department of Computer Science and Information Technology, Faculty of Science, Naresuan University, Thailand
[1]Major of Computer Multimedia, Faculty of Science and Technology, Chaopraya University, Thailand
[2]Research Center for Academic Excellence in Nonlinear Analysis and Optimization, Thailand Faculty of Science, Naresuan University, Thailand

Article Info

Article history:

Received Aug 10, 2019
Revised Oct 5, 2019
Accepted Dec 4, 2019

Keywords:

Cloud security
Energy reduction
IaaS
Logging system
Smart application

ABSTRACT

We introduce the novel technical results of the enhanced logging system for customer virtual machines (VMs) in an infrastructure as a service (IaaS) cloud. The main contribution is that the enhanced system can work with a better system's accuracy and speed, with the simplicity of the design and implementation. We measure the accuracy of the unenhanced logging system, then find a quick solution to enhance the system based on the results of the measurement. To measure and enhance the unenhanced system, we increase the main memory and CPU cores of the VMs then collect the accuracy results from each increment configuration. We analyze the results and propose to use the taskset tool to enhance the accuracy of the system. Found three main findings include: firstly, the accuracy of the enhanced system is about 20% on maximum better than the unenhanced one; the enhanced system accuracy becomes 100%; lastly, the enhanced system can detect a file with the smaller file size as almost 12% smaller. The findings can be a basis to design the logging systems in an IaaS cloud, to decrease hardware and energy investment. To the best of our knowledge, the contribution and findings are not in the literature.

Corresponding Author:

Winai Wongthai,
Department of Computer Science and Information Technology,
Faculty of Science,
Naresuan University,
Phitsanulok 65000, Thailand.
Email: winaiw@nu.ac.th

1. INTRODUCTION

This paper describes a novel technical result, which is a performance evaluation of our previous system. The evaluation is obtained through appropriate measurements. Then, based on the evaluation, we introduce a solution to enhance the system. A cloud computing deploys virtualization technology to enable itself to store and process massive data [1]. The cloud computing, or for short, the cloud is increasingly important for an information technology/IT ecosystem [2, 3]. One of the factors to realize the smart applications to improve human living life is the cloud, which can store and process huge data generated from human activities. However, the security of this data is also critical. One type of cloud called an infrastructure as a service (IaaS) cloud, such as Amazon Elastic Compute Cloud [4] focuses on offering a virtual machine or VM product to an IaaS customer. The customer can rent the VM and access it via the Internet. A VM mainly composes of data storage and network. The customer can also install his/her favorite operating system in the VM. An IaaS cloud is widely deployed in many areas, for example, government, education, or medical experiment, as agreed by [5-8]. For example, to improve students living

life, [8] exploits the combination of the advantages of the cloud and the internet of things to build a simple, smart classroom.

However, the security issue of the IaaS cloud is critical and needs to be mitigated [9]. To indicate the issue, the Cloud Security Alliance or CSA has published many reports on threats to the IaaS cloud, such as [10, 11]. A logging system can log the data of incidents that happened in a customer IaaS VM, such as who has access to or what happens with a customer file in a disk of the VM [12, 13]. The logging system, which composes of a logging process and log file [12], can be one of the solutions to mitigate the risks associated with the threats above.

- Research gaps**:** previous research such as [9, 13-17] does not focus on measuring the accuracy of the logging system with a varying method, such as by increasing the number of the CPU cores and main memory sizes of a customer VM. The results of this measurement can be analyzed. Then, the result of this analysis can be a foundation for appropriately enhancing the accuracy of the logging system. Thus, this paper focuses on the design and implementation of this measurement. Then, we will enhance the accuracy of the logging system, based on the results of the measurement.
- Contribution**:** the main contribution is that the enhanced logging system can work with better system's accuracy and speed, and with the simplicity of the design and implementation. The contribution is from the three findings, as briefly described as following. Firstly, when with a varying method by increasing the CPU cores of a customer VM, the accuracy of the enhanced logging system is 20% on maximum better than the unenhanced one and more stable. Secondly, when with a varying method by increasing the main memory sizes of an IaaS customer VM, the enhanced logging system accuracy becomes 100%. Alternatively, we can say that increasing the main memory sizes of the VM does not affect the accuracy of the enhanced logging system, whereas it does with the unenhanced one. Note that, from both findings above, we deploy both varying methods (by increasing the CPU cores and the main memory sizes of a customer VM) for the aim to measure the accuracy of the logging system. Then, we use the result of the measurement to be analyzed to find a quick solution to enhance the system. We did not aim to compare both methods.

Lastly, the last finding is that the enhanced logging system can detect a sensitive file in the VM with the smaller file size as 12.19% smaller than the unenhanced one can. This also makes the enhanced system works faster or better. A logging system performance has based the accuracy of the system [18] (will be discussed in section 2.4) and the speed of the system; thus, the findings above can successfully enhance the performance of our previous logging system [18]. The findings can be a guild-line for a logging system designer to enhance his/her system. This can also enable the system to work faster or better and increase its accuracy. These findings can also help in planning to decrease hardware and energy investment in a cloud ecosystem. As a result, the guild-line from this paper can be one of the solutions to sustainably mitigate risks associated with the security issue of the IaaS cloud. To the best of our knowledge, there are no these three contributions in the literature.

2. BACKGROUND

2.1. Infrastructure as a service cloud architecture

Figure 1 illustrates an IaaS Cloud and logging system architectures. In this paper, both architectures are adapted from our previous work [12, 18]. Section 2.2 will describe the main components or the logging system architecture or all the shaded boxes in Figure 1. This section here will briefly describe the IaaS Cloud architecture as follow. All the white boxes, ellipse, and text file shape in Figure 1 are the main components of the IaaS architecture. The white boxes components include hypervisor, dom0, hw0, domU, hwU, disk0, diskU, and memU. A component name ending with '0' means that it is physically owned and managed by an IaaS provider. Whereas, ending with 'U' means that component is virtually owned and managed by a cloud customer. The box with number 2 in Figure 1 is a hypervisor, which is software that allows a physical computer to host more than one VM. The box inside the domU, the ellipse, and the text file shape in Figure 1 are the experimental purposes. Their detail will be discussed in section 2.2.

The topmost left white box in the figure is a dom0 or domain 0, which is a manager of the entire customer created VMs. At the system booting time, a dom0 is launched by the hypervisor. A dom0 is also a VM, and exclusively accesses and control hw0, and all the customer created VMs or domUs. The bottom box in Figure 1 is hw0 that is all the physical hardware managed by a dom0. The topmost right white box in Figure 1 is a domU or user domain, which is a user VM that is created by dom0. This domain runs on top of the hypervisor. A domU is an IaaS cloud product that the provider offers to an IaaS customer. HwU is physically located in hw0 and is domU's virtual hardware. HwU is physically owned and managed by a dom0 or by the provider. However, hwU is virtually owned and managed by a domU owner or IaaS

customer. A disk0 is a physical disk of a dom0, and a diskU is a virtual disk of a domU. Finally, memU is domU's virtual main memory.

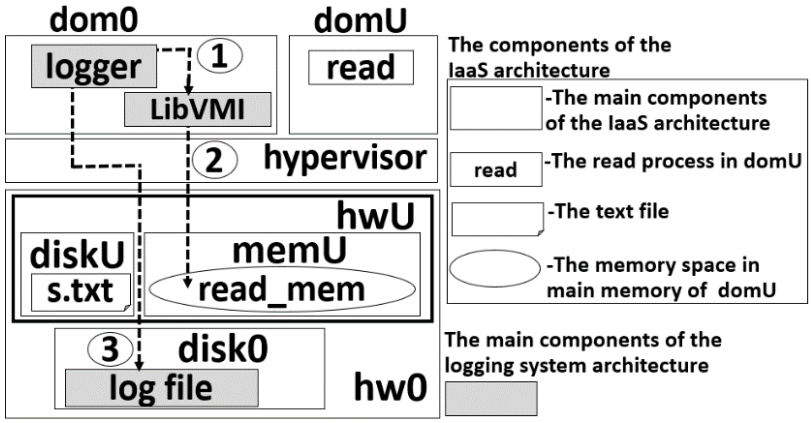

Figure 1. Infrastructure as a service cloud architecture and logging system architecture adapted from [12]

2.2. The logging system architecture

A logging system can log incidents that happened in a customer IaaS VM such as who has access to or what happens with a customer file in a disk of the VM [12, 13]. A logging system can compose of a logging process and log file [12]. This paper will call the logging process as a logger. The system architecture of the logging system is from our previous work [12, 18], and is also illustrated in Figure 1. In this paper here, the architecture will be applied to the experiment in section 3.

The box inside the domU in Figure 1 is the read process. For the purposes of the experiment, we assume that somehow, this process can be controlled by an attacker. As a result, he/she can maliciously read a sensitive file or s.txt of an IaaS customer in diskU, see the document shape inside the diskU. Read_mem in memU represents a reserve memory space for the read process provided by the OS that hosts this process. The white box in the dom0 is LibVMI, which is the new name of XenAccess [19]. It is a C library that is installed in the dom0, and then it can access read_mem in memU for detecting the malicious reactivity of the read process that is reading s.txt.

Note that we did not demonstrate the assumption above. However, the assumption is also referred to in proposing logging solutions in the cloud. In [13, 20] consider this assumption when proposing their logging solutions in the cloud. For example, [13] assumes that when a user 'Alice' creates a sensitive file (such as s.txt) and modifies this file. Then, somehow, a user 'Bob' can maliciously read the file without Alice's permission.

From Figure 1, the three working steps of the logger are represented by the circles with numbers 1 to 3. Step1, the logger in the dom0 calls LibVMI to access memU to obtain the logging data from read_mem (step2). This data includes i) a file name of s.txt or the string "s.txt", ii) the process ID of the read process. Then, LibVMI accesses memU to obtain this data in read_mem. Then, it returns the obtained data back to the logger. Finally, the logger manages the data then writes (in step3) the data into the log file.

2.3. The context switching task and taskset tool

A context switching task is a swapping process between a former executed process from the CPU and a latter process [21]. Thus, when the logger process is hosted in a dom0 with more than one core, the operating system (OS) of the dom0 may switch the logger from the current core to the new one, for example, from the current core number 1 to the new core number 2. When switching, the OS has to move the associated processing data of the logger from mem0 to caches of the new core (such as the core number 2) [22, 23]. A cache is a very high-speed memory and is a buffer between the main memory and the CPU [24]. Thus, this switching task of the OS mentioned above can consume more processing time of the logger.

Taskset is a tool that can specify a process to be run on a particular CPU core in a Linux OS [25]. In our experiment in Section 3, this tool is used to fix one CPU core for the logger. Thus, the logger can have its own core to be exclusively performed. As a result, the OS that is hosting the logger process does not need to switch the process from the current core to another one. Consequently, this non-switching task situation can decrease the processing time of the logger.

Taskset tool is used as a solution to enhance the accuracy of the logger in this paper. However, there are other solutions for enhancing the accuracy of the logger. For example, we can enhance the logger by analyzing the existing logger code to enhance the algorithm of this logger. The taskset tool has well-known performance effects in a Linux system; however, in this paper uses the tool as the solution. This is because of that; we want to initially introduce a simple, available, and quick solution for the enhancement to obtain the initial results. This is also to avoid the modification of the existing logger.'[root@localhost logger_tester_new]# taskset –c 6-7./logger' is the full command of taskset that will be used in the experiment in Section 3. This command means that to set the logger process ('./logger') to be run on the CPU core number 4 ('-c 6-7') of a dom0 that is hosting the logger.

2.4. Performance and accuracy of a logging system

The cloud architecture is complicated to understand and more abstract than a traditional client-server model [26]. As a result, this may also cause the performance measurement of logging systems in the IaaS cloud to be complicated. However, the performance measurement is essential because it can provide trust in the security of cloud user data and processing in an IaaS [9, 18]. Briefly, from our previous work [18], one of the four application system key performance indicators (KPIs) is response time. [27] states that response time is the amount of time it takes for the application when responding to a user request. This paper focuses only on the response time, as the example KPI of performance measurement to encourage researchers to concern all the KPIs. We also concern only on the accuracy of a logging system that is hosted by a dom0. The accuracy means the accuracy of the logging system in logging the logged information from memU of a target monitored domU. The accuracy is considered as one of the factors of the response time KPI, as also argued by [18]. Thus, we assume that if the accuracy of a logging system is enhanced, then the performance of the system is also enhanced. To measure the performance of a logging system, the next section fully discusses the measurement method of the accuracy of the system.

2.5. The measurement of the logger accuracy

We performed the measurement of the accuracy of the logger in our previous work [12]. The measurement can represent a real-world scenario [18]. Thus, this paper here aims for new purposes of the experiment and with the new experimental environment, apart from the previous work above. Thus, the detail below is a brief description of the measurement that will be adjusted then applied to the experiment of this paper.

The box in the domU in Figure 1 is the read process. The practical steps of the process are: 1) to open the s.txt file (the document symbol in diskU in Figure 1), 2) to read and display the content of the file 3) to close the file, finally 4) the process is terminated. For the purposes of the measurement of the accuracy of the logger, a sleeping time (from [12]) is a time value that is added to the read process steps above before the process is terminated. A sleeping time will be in a millisecond unit, for example, 60ms. This means that after the 4^{th} step activity of the read process, the process will be idled or slept for 60ms. After that, it will be terminated. We aim to decrease the value of the sleeping time. The more we can decrease this value, the accuracy of the logger will be increased. Ideally, the sleeping time is 0.

When measuring the accuracy of the logger (the shaded box in the dom0), we firstly set the sleeping time for the read process, such as 65ms. Then, we run the logger and run the read process for, for example, 100 times. Finally, this allows the logger to detect the file name of s.txt for 100 times. The accuracy of the logger is measured by a number of times that the logger captures the correct file name as the string "s.txt" from read_mem. This is called 1 'hit', otherwise is 1 'miss'. Thus, if the read process reads s.txt files 100 times, and the logger can capture the string "s.txt" for all these 100 times, this is 100 hits [9]. This means that the logger has the accuracy as 100% with when the read process has the sleeping time as 65ms. This can be translated as the logger can have 100% of its accuracy when the read process (that is used by an attacker to read s.txt) needs to be in memU/read_mem for at least 65ms.

3. THE EXPERIMENT

It is because of that before the enhancement, and from our primarily extra experiment, the logger can perform well with hw0 equipped with 8 CPU cores. Thus, the enhanced logger is fixed to run on only one CPU core of this hw0. Before measuring the accuracy of the enhanced logger, we also performed the same experiment of the enhanced logger to the unenhanced one. Then, we obtained the results of both to be compared. Note that, on the hw0 8 cores, the unenhanced logger was not fixed to run on only one CPU core. It can be switched from one to another core, depending on its OS manager at that point in time. The other details of the experiment are below. Note that the experimental methods in this paper are the same as some

parts of the methods done by our other work [9]. However, the work is for examining the accuracy of logging systems, but this paper here is for enhancing the systems.

3.1. The hardware and software of the experimental environment

We set up the experimental environment based on an Intel Xeon 3.06 GHz with CPU 64-bit 8 CPU cores. The machine has SDRAM 8 GB of main memory, and 320 GB of the secondary memory. A Fedora 16 64-bit is the OS of the machine. Then, we installed Xen 4.1.4 as the hypervisor (the box with number 2 in Figure 1) on top of the OS. This hypervisor is used to simulate an IaaS cloud on this machine or Figure 1. Then, we installed LibVMI 0.10.1 library (the box inside the dom0 in Figure 1) on the OS. This library can be called by the logger to access to memU from the dom0, as discussed in section 2.2. In the domU, we also installed a Fedora 16 64-bit as its OS and set up the read process on top of this OS.

3.2. The hardware environment configurations of the logger process

The read process in domU in Figure 1 can perform actions on s.txt or the document shape in diskU in the figure. The details of the actions, of how the logger (the white box in the dom0) preform, and of how to measure its accuracy were mainly discussed in section 2.2 (in Figure 1), and 2.5 above. The main step of the experiment in this paper is to measure the accuracy of the logger when the logger is recording logging data from some of the actions of the read process with the s.txt file. Then we will analyze the results to be ready to provide a solution to enhance the accuracy of the logger. To see the trends of the logger accuracy, the experiment will be performed on mainly different hardware configuration environment of the domU, as will be discussed in the following sections. In Figure 1 and on the different hardware configuration environment of the logger, the experiment will be performed by measuring the accuracy values of the logger after each hardware environment configuration by increasing the numbers of CPU cores of the domU, and by increasing the sizes of memU of the domU. Section 3.3 gives an overview of increasing domU CPU cores labs, illustrated in Figure 2(a). Then, Figure 2(b) illustrates the increasing sizes of memU in section 3.5.

3.3. The overview of domu cpu cores increasing labs

Section 3.4 will give the full definition of a lab. Here in this section, it is the overview of increasing domU CPU cores labs. Figure 2(a) illustrates eight labs of increasing the CPU cores of a domU from 1 to 8. In Figure 2(a), see the freeform shape that rounds both the dotted-line box and the top shaded box with labeled 'du1c', this is one lab of our experiment or a d08c-du1c lab. In Figure 2(a), this lab also represents by the first arrow from the dotted-line box to the top shaded box. Thus, the right white box or Figure 2(a) is d08c, which is with eight labs or d08c-du1c, d08c-du2c, ..., and d08c-du8c. Section 2.5 already discussed that the accuracy of the logger is measured by the number of times that the logger captures the right file name or a string "s.txt" in read_mem. Moreover, section 2.5 already discussed hit and miss, and how to calculate the actuary of the logger in a percentage. Then, we will describe what is a lab in detail below.

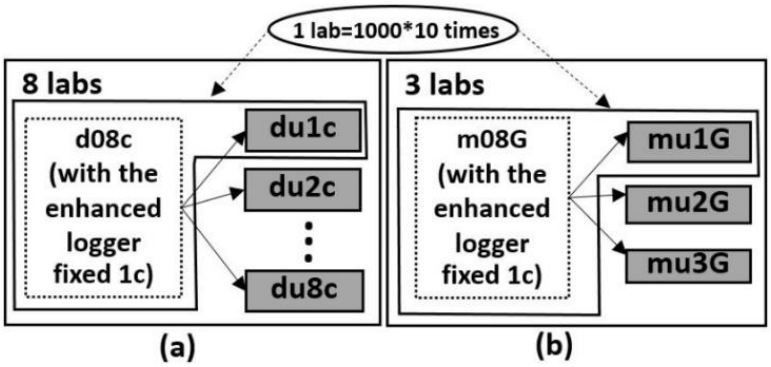

Figure 2. d08c and m08G experimental environment (with the enhanced logger fixed 1c),
(a) Eight labs of increasing the CPU cores of a domU from 1 to 8, (b) A dom0 deploying 8 GB of mem0

3.4. 'A lab' for increasing the CPU cores of domU

We modified the logger and the read process in Figure 1 for the experiment in this paper. Thus, the procedures of the logger and the read here may be slightly different from Figure 1. A lab is illustrated by the freeform shape that rounds both the dotted-line box and the top shaded box with labeled

'du1c' in Figure 2(a). The lab has three steps or st1 to st3. St1 is to run the logger in dom0. St2 is to run the read process 1000 rounds in domU. Each round is independent but is ordered from 1^{st} to 1000^{th}. Thus, when the first read process is run and finished, then the second one is run and finished. Then, this is so on until the 1000^{th} read process is run and finished. At the same time, the logger (that is already run only one time by st1, before the first round of the read process running) will capture the "s.txt" string (this is 1 hit) from each round of the read process, string from the 1^{st} to 1000^{th} rounds. Then, it will store the string into the log file. This is also one round of the read process is done. Then the logger will wait for the next read process round running. When the logger obtains the "s.txt" string of each round, each hit will be accumulated by 1 per hit.

The last step or st3 is when the accumulated number of hits of these 1000 rounds is calculated and recorded. To sum up, a lab means we performed st1 to st3. Then, we obtain the first accumulated number of hits of these first 1000 rounds of running the read process. Then, we perform st1 to st3 for the other nine times. After that, we will achieve the other nine accumulated numbers of hits. Finally, we can calculate the accuracy of the logger from these ten accumulated numbers of hits as a percentage. This is a lab and also the first lab or lab 1, which is d08c-du1c or the freeform shape in Figure 2(a). Then, we performed the other sever labs as d08c-du2c, d08c-du3c, …, d08c-du8c, as illustrated by Figure 2(a).

3.5. Increasing the sizes of memU

The objective of this section is to study the trend of the accuracy of the enhanced logger when increasing the sizes of memU. The logger has to capture the necessary logging data from domU. The data are in read_mem and as PID of the read process, the name of the read process, UID or the ID of the owner of the read process, lastly, the file name that the read process is reading, as described in section 2.1. For dom0, the dotted-line box in Figure 2(b) is m08G, which represents a dom0 that deploys 8 GB of mem0. We will explain three labs or the white box of Figure 2(b), as follows. In Figure 2(b), see the freeform shape that rounds both the dotted-line box and the top shaded box with labeled 'mu1G', this is one lab of our experiment or m08G-mu1G lab or the first thick line with the arrow from the top of Figure 2(b). Thus, see the white box of Figure 2(b), m08G is with three labs or m08G-mu1G, m08G-mu2G, and m08G-mu3G. From Section 2.5, we already discussed that the accuracy of the logger is measured by the number of times that the logger captures the right file name or a string "s.txt" in read_mem. Moreover, section 2.5 also already discussed a 'hit' and 'miss', and how to calculate the actuary of the logger in a percentage. Then, we will describe what a lab of main memory experiment is, below.

3.6. 'A lab' of memU experiment and the three labs for the main memory of dom0 8GB

We modified the logger and the read process in Figure 1 for the experiment in this paper. Thus, the procedures of the logger and the read here maybe not entirely the same as ones from Figure 1. The logger is run in our maximum and available main memory of dom0 as 8GB. We labeled this memory as m08G. See in Figure 2(b) at the freeform shape that rounds both the dotted-line box and the top shaded box with labeled 'mu1G'; this is a lab of increasing main memory. This lab has three steps or sm1 to sm3. These three steps are as the same st1 to st3, respectively, as just discussed in Section 3.4 above. Thus, a lab here means we performed sm1 to sm3, and gets the first accumulated number of hits of the first 1000 rounds of running the read process. Then, we perform sm1 to sm3 for the other nine times. Thus, we can obtain the other nine accumulated numbers of hits. Finally, we can calculate the accuracy of the logger from these ten accumulated numbers of hits as a percentage. This is a lab and also the first lab or lab 1, which is m08G-mu1G as showed by the freeform shape in Figure 2(b). For the three labs for m08G, see the white box of Figure 2(b), it is three labs. The first lab is just discussed above. Then, the two labs are outside the freeform shape. They are lab 2 or m08G-mu2G and lab 3 or m08G-mu3G. These three labs for m08G or the white box or Figure 2(b).

4. RESULT AND DISCUSSION

The conclusion is that from section 4.1, 4.2, and 4.3 below, the enhanced logger is faster or better and more stable than the unenhanced one. This can be a guild-line for a logging system designer to enhance his/her system. This enables the system to work faster or better and to increase its accuracy. The guild-line can also help in planning to decrease hardware and energy investment in a cloud ecosystem. As a result, the guild-line can be one of the solutions to truly and sustainably mitigate risks associated with the security issue of the IaaS cloud. The details of the results and discussions are below. This paper is focusing on to enhance the accuracy of the logger process. Thus, the paper did not measure or discuss the effect on aggregated system performance caused by pinning the process. This is future work.

4.1. The enhanced and unenhanced logger with increasing domU CPU cores finding

The results and discussions in this section are from the experiment descriptions in section 3.3 and 3.4. The conclusions are that the unenhanced logger has unstable accuracy values when increasing the CPU cores of the domU. However, the enhanced logger can work faster or better to capture the read process; even the read process is run in a domU with any number of its cores. This enables the accuracy of the enhanced logger to be better than the unenhanced one, and more stable. The details are below.

4.1.1. The unenhanced logger with its low and unstable accuracy when increasing domU CPU cores

From Figure 3, the x-axis is when the number of CPU cores of the domU is increasing from 1 to 8 cores (du1c, du2c, …, du8c). Also, the y-axis shows the sleeping time values of the read process. The lower sleeping time value increases the accuracy of the logger. The conclusion is that the enhanced logger can capture the read process when this process is in memU for at least 60ms. Whereas, the unenhanced logger can capture the process when the process is in memU for at least 65ms. The enhanced logger can work faster or better than the unenhanced one for 5ms or at 60ms. Importantly, when CPU cores of the domU are increasing from 1 to 8 cores, the logger accuracy values are all 100% at when the read process with 60ms.

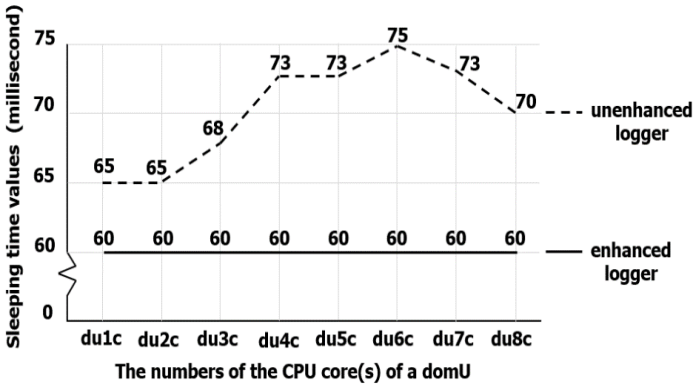

Figure 3. The enhanced and unenhanced logger with increasing the number of the domU CPU cores findings

The dotted line in Figure 3 shows the sleeping time values of the read process when CPU cores of the domU are increasing from 1 to 8 cores. There are three periods of the trend in this figure. The first one is when the CPU cores of domU are increased from 1 to 2 cores, the sleeping time values are the same as 65ms. This period has the best accuracy values of the logger. It is because there are only one and two cores of the domU. Thus the OS of the domU did not switch the read process so many times, as discussed in section 2.3. Thus, the read has more chances to appear in memU. As a result, the logger also has more chances to capture the read process, then the accuracy of the logger is better than the other periods.

The second period is when the CPU cores are increased from 2 to 6. This makes the sleeping time values are continuously increased from 65 to 75ms. This means that the logger accuracy values are roughly decreased when increasing the CPU cores of the domU from 2 to 3, 4, 5, and to 6. This is because when the domU can have 2 to 6 CPU cores, the OS of this domU may need to switch the read process from one to another core. This switching task reduces the chances of the read process to be appeared in memU, as discussed above and in section 2.3. Thus, this may also decrease the chances of the logger to capture the read process. Thus, the logger accuracy is also decreased.

Then, the last period is when CPU cores of the domU are increased from 6 to 7, and to 8. This makes the sleeping time values dropped from 75ms to 73ms, then to 70ms. This means that the accuracy values of the logger are increased when the CPU cores of the domU are increased from 6 to 8 periods. This may be because of that the domU starts to have enough CPU cores to perform the read process. Thus, this is also not many switching tasks. Thus, the read process has more chances to be appeared in the memU, compared to when the number of the CPU cores is, for example, at 6 or 5 cores. Thus, this also increases more chances for the logger to capture the read. Then, the accuracy of the logger is increased.

To sum up, the dotted-line shows the unstable accuracy values of the unenhanced logger when increasing the CPU cores of the domU, this is the host of the read process. This should be because of the switching tasks, as discussed above. After we analyze the results and discussions in this section, we then perform the experiment with taskset tool to enhance the logger. The next section (section 4.1.2) is the results and discussions of the enhanced logger.

4.1.2. The enhanced logger with its better and more stable accuracy when increasing domU CPU cores

As the accuracy values of the logger or the dotted-line are not stable, then we enhanced the logger, and the results are shown by the thick line in Figure 3. The conclusion is that the accuracy of the enhanced logger is about 7.69% to 20.00% and 14.35% on average better than the unenhanced one. This is because of that the enhanced logger can stably capture the read process when this process is in memU for at least 60ms. This time interval is constant for all domU' s configurations. Whereas the unenhanced logger can unstably capture the process when the process is in memU for at least 65ms (the minimum) and 75ms (the maximum). Thus, the enhanced logger can work faster or better than the unenhanced one for 5ms or 7.69% on the minimum case, and on 15ms or 20.00% for the maximum case, and for 10.25ms or 14.35% on the average case. Importantly, when CPU cores of domU are increasing from 1 to 8, the accuracy values of the enhanced logger are all still 100%, when the read process with 60ms. Thus, the enhanced logger is stable compared to the unenhanced one. This is because of the enhanced logger is fixed to run on only one CPU core solely. Thus, there are no switching tasks to be performed on the logger, as already discussed in section 2.3. Thus, the logger can work faster or better to capture the read process; even the read process is run in a domU with 1, 2, ..., or, 8 cores. This makes the accuracy of the enhanced logger better than the unenhanced one, and more stable. This can be seen by the thick line in Figure 3, which is a straight line with all the same sleeping time values at 60ms for all the CPU cores of the domU from 1 to 8.

To sum up, the logger can work faster or better to capture the read process; even the read process is run in a domU with 1, 2, …, or, 8 cores. This makes the accuracy of the enhanced logger to be better than the unenhanced one, and more stable.

4.1.3. Summary

Section 4. 1. 1 shows the unstable accuracy values of the unenhanced logger when increasing the CPU cores of the domU, which is the host of the read process. This should be because of the switching tasks. After the enhancement, the results and discussions in section 4. 1. 2 are that the enhanced logger can work faster or better to capture the read process, even the read process is run in a domU with any number of its cores. This enables the accuracy of the enhanced logger to be better than the unenhanced one, and more stable.

4.2. The enhanced and unenhanced logger with increasing memU findings

The results and discussions in this section are from the experiment descriptions in Section 3.5 and 3.6. The conclusion of Figure 4 is that taskset tool makes the logger accuracy to be enhanced and to be 100%. This section also describes the results and discussions from the experiment before and after the enhancement of the logger. See, the x-axis of Figure 4, to obtain the results from the experiment before and after the enhancement of the logger, the axis illustrates increasing the size of memU from 1 to 3GB (mu1G, mu2G, and mu3G). Y-axis of Figure 4 illustrates the accuracy values of the logger in percentages when increasing the sizes of memU from 1 to 3GB.

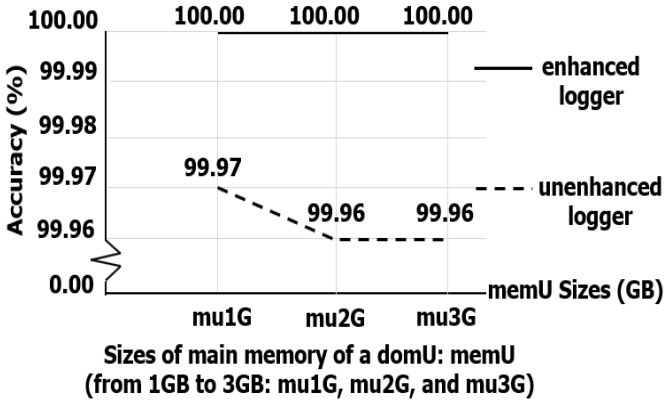

Figure 4. The enhanced and unenhanced loggers with increasing the size of memU finding

4.2.1. Increasing memU sizes affects the accuracy of unenhanced logger

The dotted line in Figure 4 shows the trend of the accuracy values of the logger before the enhancement. The logger accuracy values are decreased as 99.97%, then 99.96%, and finally also

99.96%, when the memU sizes are increased from 1, to 2, and finally to 3GB, respectively. This may be because of that the logger needs to search for the data of the read process in a winder area of memU such as 3GB, compared to 1GB. This may make the logger miss the read process, compared to when the logger searches in a smaller area of memU such as 1GB.

4.2.2. Increasing memU sizes does not affect the accuracy of the enhanced logger

Section 2.3 discussed that the logger accuracy value of the fixed core environment should be better than the switching core environment. From the experiment described in section 3, the thick line in Figure 4 shows that after the enhancement of the logger by deploying taskset tool, the accuracy values of the logger are increased from 99.97% to be 100%, 99.96% to 100%, and also 99.96% to 100%, when the memU sizes are increased from 1 to 3GB respectively. This is because of that; when fixing a CPU core for the logger to be run solely, this enables the accuracy of the logger to be enhanced. This is because when the logger is fixed with one CPU core, it will not be switched to another core, which may decrease the accuracy of the logger, as discussed in section 2.3.

Another point is that when increasing the amount of memU from 1 to 3GB, it does not affect the accuracy of the enhanced logger. All the accuracy values are still the same as 100%. This is because the enhanced logger is not needed to be switched. Thus, the logger can focus only on capturing the read process, and the logger did not miss the read process, even when memU sizes are larger. Thus, the accuracy values of the enhanced logger are all 100%.

4.2.3. The summary

It can be seen that when enhancing the accuracy of the logger by using taskset tool which is available in the Linux OS, the enhanced logger accuracy becomes 100%, even when increasing the memU sizes. Alternatively, we can say that increasing memU sizes does not affect the accuracy of the enhanced logger, whereas it does the unenhanced logger.

4.3. The enhanced logger can detect a sensitive file with the smaller file size than the unenhanced logger

Before enhancing the logger, we performed the further experiment and found that the logger can capture s.txt file with when the smallest file size of this file as about 26,952 bytes. If a file size of s.txt is smaller than 26,952 bytes, the logger will not be able to capture this file. It is because the read process can read this smaller file very fast. Thus, the logger may not be able to capture this reading action on s.txt. After the enhancement, the enhanced logger can detect s.txt with the smaller file size as 23,667 bytes, compared to 26,952 bytes with the unenhanced logger above. The difference is 3,285 bytes or 12.19% . This means that the enhanced logger may work faster or better than the unenhanced one. It is because the enhanced logger can capture the read process when it is reading s.txt file with the size as small as 23,667 bytes. This can be useful when an attacker uses the read process to maliciously quickly read s.txt with this small file size as 23,667 bytes, but the enhanced logger can still capture this speedy reading activity. Whereas, the unenhanced logger cannot do so. To sum up, the enhanced logger can detect the sensitive file with the smaller file size than the unenhanced one can.

4.4. The overall summary of the enhancement of the logger

From section 4.1, 4.2, 4.3 above, there are three primary findings after the enhancement of the logger. A logging system performance has based on the accuracy of the main component (or the logger) of the system [18] (as discussed in section 2.4) and the speed of the system; thus, these findings can successfully enhance the performance of our previous logging system in [18]. For the findings, firstly, when with a varying method by increasing the CPU cores of a domU, the enhanced logger can work faster or better to capture the read process, even the read process is run in a domU with any number of its cores. This enables the accuracy of the enhanced logger to be better than the unenhanced one, and more stable. Secondly, when with a varying method by increasing memU sizes of a domU, the enhanced logger accuracy becomes 100%. Alternatively, we can say that increasing memU sizes does not affect the accuracy of the enhanced logger, whereas it does with the unenhanced one. Lastly, the enhanced logger can detect the sensitive file with the smaller file size than the unenhanced one can.

These findings can be a basis in a situation, such as, for detecting when an attacker maliciously read a sensitive file in a domU/IaaS cloud customer VM, with any CPU core numbers and/or any memU size of the VM. For this situation, we recommend a logger designer to use taskset tool to pinpoint the logger process. Thus, this process needs to be configured according to Figure 2, to be running on a dom0 with eight cores and eight GB of mem0. As a result, the logger can work with better accuracy (and can become 100%) and better speed compared to without pinpointing.

5. CONCLUSION

An infrastructure as a service or IaaS cloud can offer virtual machines or VMs to a cloud customer. The VM can be used to store and operate his/her sensitive files for the business. However, security is a critical issue to slow down IaaS growth. For example, the customer worries whether an attacker can maliciously access the sensitive files. A logging system that composes of a logging process (logger) and the log file can help in mitigating the risks associated with the issue above. However, the accuracy of the logger needs to be measured and analyzed in various hardware configurations to introduce the solution to enhance the logger accuracy. Thus, this paper discusses the enhancement of the accuracy of the logger. The enhancement is performed by using taskset tool to fix one CPU core for the logger to run. Then, we test the enhanced and unenhanced logger with the same experimental environment to obtain the results of both to be compared and analyzed. The experimental environment is a varying method by increasing the number of CPU cores and the main memory or (memU) sizes of a customer VM or domU. We assume that a customer sensitive file is in this domU's disk, and, an attacker may access the file.

We found three main findings from this paper, in a varying method by increasing the CPU cores and memU sizes of a domU. To the best of our knowledge, there are no these three findings in the literature. Firstly, the accuracy of the enhanced logger is better than the unenhanced one and more stable. Secondly, enhanced logger accuracy becomes 100%. Also, increasing memU sizes does not affect the accuracy of the enhanced logger, whereas it does with the unenhanced one. Lastly, the enhanced logger can detect the sensitive file with the smaller file size than the unenhanced one can. Due to the performance of a logging system is based on the accuracy of the system (as discussed in section 2.4) and the speed of the system; thus, these findings successfully enhance the performance of our previous logging system. The findings can be a guild-line for a logging system designer to enhance his/her system. This enables the system to work faster or better and increases its accuracy. The findings can also help in planning to decrease hardware and energy investment in a cloud ecosystem. As a result, the guild-line can be one of the solutions to truly and sustainably mitigate risks associated with the security issue of the IaaS cloud. We also successfully applied the same logging system from this paper to a real-world IaaS cloud production/scenario or the OpenStack system environment. Thus, we believe that the measurements and the results from this paper can apply to a real-world scenario. Thus, the future work is to apply the guild-line from this paper to the OpenStack system environment.

REFERENCES

[1] A. Bhawiyuga, D. P. Kartikasari, K. Amron, O. B. Pratama, and M. W. Habibi, "Architectural Design of IoT-Cloud Computing Integration Platform," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 3, pp. 1399, June 2019.

[2] S. Alshamrani, "An Efficient Algorithm for Monitoring Virtual Machines in Clouds," *Bulletin of Electrical Engineering and Informatics*, vol. 8, no. 2, pp. 699-705, June 2019.

[3] R. Kaur and G. Kaur, "Proactive Scheduling in Cloud Computing," *Bulletin of Electrical Engineering and Informatics*, vol. 6, no.2, pp. 174-180, 2017.

[4] S. Chaisiri, R. Kaewpuang, B-S. Lee, and D. Niyato, "Cost Minimization for Provisioning Virtual Servers in Amazon Elastic Compute Cloud," *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 85–95, July 2011.

[5] K. Flanagan, S. Nakjang, J. Hallinan, C. Harwood, R.P. Hirt, M.R. Pocock, and A. Wipat, "Microbase2.0: A Generic Framework for Computationally Intensive Bioinformatics Workflows in the Cloud," *Journal of integrative bioinformatics*, vol. 9, no. 2, pp. 212, Jun. 2012.

[6] K. Flanagan, "A Grid and Cloud-Based Framework for High Throughput Bioinformatics," PhD Thesis, School of Computing Science, University of Newcastle upon Tyne, 2009.

[7] R. De Paris, "FReMI: A Middleware to Handle Molecular Docking Simulations Offully-Flexible Receptor Models in HPC Environments," Master Thesis; Faculty of Computer Science, Pontifical Catholic University of Rio Grande Do Sul, 2012.

[8] W. Runathong, W. Wongthai, and S. Panithansuwan, "A System for Classroom Environment Monitoring Using the Internet of Things and Cloud Computing," *Information Science and Applications*, vol 424, pp. 732-742, 2017.

[9] T. Auxsorn, W. Wongthai, T. Porka, and W. Jaiboon, "The Accuracy Measurement of Logging Systems on Different Hardware Environments in Infrastructure as a Service Cloud," *ICIC Express Letters, Part B: Applications, An International Journal of Research and Surveys*, vol 11, no. 5, 2020.

[10] CSA, "The treacherous 12 - cloud security alliance. Cloud security alliance," Cloud Security Alliance. 2016.

[11] CSA, "The treacherous 12: cloud computing top threats in 2016," Cloud Security Alliance. 2016.

[12] W. Wongthai, *"Systematic Support for Accountability in the Cloud,"* PhD Thesis, Newcastle University, 2014.

[13] R. K. L. Ko, P. Jagadpramana, and B. S. Lee, "Flogger: A File-Centric Logger for Monitoring File Access and Transfers within Cloud Computing Environments," *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 765 –771, Nov. 2011.

[14] H. Lee, H. Kim, C. Seo, and S. U. Shin, "Framework of Service Accountability and Policy Representation for Trustworthy Cloud Services," *Advances in Computer Science and Ubiquitous Computing*, vol 373, pp. 437-443, 2015.
[15] R. K. L. Ko and M. A. Will, "Progger: An Efficient, Tamper-Evident Kernel-Space Logger for Cloud Data Provenance Tracking," *2014 IEEE 7th International Conference on Cloud Computing*, pp. 881-889, Jun. 2014.
[16] C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. S. Lee, "S2logger: End-to-End Data Tracking Mechanism for Cloud Data Provenance," *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 594-602, Jul. 2013.
[17] T. Auxsorn, W. Wongthai, T. Phoka, W. Jaiboon, "Performance Considerations of a Logging System Simultaneously with a Customer Virtual Machine in Infrastructure as a Service Cloud," *Information Science and Applications*, vol. 621, pp 285-296, 2020.
[18] P. Chan-In, W. Wongthai, "Performance Improvement Considerations of Cloud Logging Systems," *ICIC Express Letters*, vol. 11, no. 1, pp. 37-43, 2017.
[19] B. D. Payne, M. D. P. D. A. Carbone, and W. Lee, "Secure and Flexible Monitoring of Virtual Machines," *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 385-397, Dec. 2007.
[20] S. Sundareswaran, A. Squicciarini, and D. Lin, "Ensuring Distributed Accountability for Data Sharing in the Cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 556-568, Jul. 2012.
[21] M. M. Tajwar, M. N. Pathan, L. Hussaini, A. Abubakar, "CPU Scheduling with a Round Robin Algorithm Based on an Effective Time Slice," *Journal of Information Processing System*, vol. 13, no. 4, pp. 941-950, 2017.
[22] P. Gepner, M. F. Kowalik, "Multi-Core Processors: New Way to Achieve High System Performance," *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, pp. 9-13, 2006.
[23] E. Cota-Robles, "Priority based simultaneous multi-threading," *United States Patent*, 2003.
[24] J. Handy, "The cache memory book," Morgan Kaufmann, 1998.
[25] J. -P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The Linux Scheduler: A Decade of Wasted Cores," *Proceeding of the Eleventh European Conference on Computer Systems – EuroSys'16*, no. 1, pp. 1-16, 2016.
[26] CSA, "Cloud Security Considerations," The National Security Agency, Tech. Rep, 2011.
[27] I. Molyneaux, "The Art of Application Performance Testing: From Strategy to Tools," O'Reilly Media, Inc., 2014.

BIOGRAPHIES OF AUTHORS

Surapong Wiriya received his science bachelor degree in computer science from Chaopraya University in 2007, his master degree in computer science from the National Institute of Development Administration in 2012, He is preparing his Ph.D. thesis on the cloud security, Naresuan University. E-mail: surapongw58@email.nu.ac.th

Winai Wongthai received his science bachelor degree in computer science from Naresuan University in 2000, his master degree in the computer science from Asian Institute of Technology in 2001 and master degree in the Systems Design for Internet Applications from University of Newcastle upon Tyne, UK in 2009, and his Ph.D. in the computer science from University of Newcastle upon Tyne, UK in 2014. E-mail: winaiw@nu.ac.th

Thanathorn Phoka received his engineering bachelor degree in computer engineering from Chulalongkorn University in 2002, his master degree in computer engineering from Chulalongkorn University in 2004, and his Ph.D. in the computer engineering from Chulalongkorn University in 2011. E-mail: thanathornp@nu.ac.th