ICIC Express Letters
Part B: Applications
Volume **11**, Number **5**, May **2020**

ICIC International ©2020 ISSN 2185-2766
pp. **427–437**

THE ACCURACY MEASUREMENT OF LOGGING SYSTEMS ON DIFFERENT HARDWARE ENVIRONMENTS IN INFRASTRUCTURE AS A SERVICE CLOUD

Thongrob Auxsorn[1], Winai Wongthai[1,2,*], Thanathorn Porka[1] and Wichep Jaiboon[3]

[1]Department of Computer Science and Information Technology
[2]Research Center for Academic Excellence in Nonlinear Analysis and Optimization
Faculty of Science
Naresuan University
99 Moo 9, Tambon Tha Pho, Muang, Phitsanulok 65000, Thailand
*Corresponding author: winaiw@nu.ac.th

[3]Department of Computer Science
Faculty of Business and Public Administration
Nation University
444 Vajiravudh Damnoen Road, Lampang 52000, Thailand

Received November 2019; accepted February 2020

Abstract. *Infrastructure as a Service (IaaS) cloud provides virtual computing resources such as a Virtual Machine (VM) to a customer. Many companies desire to deploy this cloud; however, data security is an issue. The issue affects the reliability of a customer and provider, such as when hackers were able to access a data or critical file of a customer. Thus, researchers introduce a logging system to mitigate the risks associated with this issue. A logging system is in a host system and can capture some incidents that appeared in a customer VM. Then, the system stores the captured data in a log file to be used as evidence to mitigate the risks. This paper focuses on examining the accuracy of our previous logging system. This is done by varying the numbers of CPU cores and the size of the main memory of both the host system and targeted VM of a customer. The contributions of this paper are the following. Firstly, we fully introduced the method of varying the sizes of main memory and the numbers of CPU cores of both a host system of the logging system and targeted VM. This method can facilitate the other types of the accuracy or performance measurement of a logging system. Secondly, we found and illustrated the inappropriate hardware configurations of the host system, the targeted customer VM, and the association of both. For example, to increase the CPU cores of the host system of the logging system will not away increase the accuracy of the logging system. Thirdly, we also found and illustrated the appropriate configurations. All these contributions can help in decreasing the cost, time, effort, and energy of a logging system development in an IaaS ecosystem. To the best of our knowledge, there are no these three contributions in the literature.*

Keywords: Cloud security, Logging system, CPU cores, Main memory, Accuracy

1. **Introduction.** The cloud exploits virtualization technology to enable itself to store and process huge data [1], and is increasingly important for an IT ecosystem [2, 3]. An Infrastructure as a Service (IaaS) cloud is renting Virtual Machine (VM) resources by a cloud customer from a cloud provider. This cloud is gaining popularity in the IT ecosystem. It has been used for many types of applications. The expense of the cloud is calculated from how much the resources that were used. The cloud is a reasonable choice for organizations to apply in the IT departments. However, sensitive data that are processed outside the enterprise can cause risks to the data [4]. The cloud causes

DOI: 10.24507/icicelb.11.05.427

many security issues, which are a significant obstacle for its marketing situation [5]. To officially indicate the issues, Cloud Security Alliance (CSA) identifies the threats of this cloud [6, 7]. A single security method may not address the issues. Thus, traditional and new technologies and strategies should be combined to deal with the issues.

Many researchers are working on mitigating the risks associated with the threats. However, this paper focuses on examining the accuracy of our previous logging system. The system records some appropriate incidents in the cloud VM. Thus, whenever there is an issue, a log file can be retrieved from the logging system for analysis and investigation to mitigate the risks. The accuracy measurement is crucial for performance measurement of the logging system [8, 9, 10]. This paper focuses on examining the accuracy of the logging system. This is done by the method of varying the size of main memory and the numbers of CPU cores of both a host system of the logging system and targeted VM. The results were obtained, and the discussions and conclusion were made. **Research Gaps:** There is no the accuracy measurement of the logging system by the method of varying the CPU core numbers and main memory of both a host system of the system and targeted VM in the literature. Thus, we aim to perform the accuracy measurement of our previous logging system on different hardware environments of the host system in the IaaS cloud.

Summary of contributions: We introduced the method of varying the sizes of main memory and the numbers of CPU cores of both a host system of the logging system and targeted VM. This method can facilitate the other types of the accuracy or performance measurement of a logging system. The example is to measure the decreed performance of targeted customer VM, while running in the host system, not only to measure the logging system in the host system. Secondly, we found and illustrated the inappropriate hardware configurations of the host system, the targeted customer VM, and the association of both. For example, to increase the CPU cores of the host system of the logging system will not away increase the accuracy of the logging system. Thirdly, we also found and illustrated the appropriate configurations. For example, when the logging system works with core numbers such as 1 or 8, this may cause the best accuracy. The last two constitutions can be useful to ensure that the designers who design a logging system like our system can avoid the inappropriate hardware configurations, and can achieve the right configurations. This can help in reducing the cost, time, effort, and energy of a logging system development in an IaaS ecosystem. To the best of our knowledge, there are no these three contributions in the literature.

2. Background.

2.1. The IaaS and logging system architectures.

Figure 1 illustrates both an IaaS cloud architecture and our existing logging system architecture. Both architectures are adapted from our previous work [10] for the experimental purposes of this paper. Firstly, this section describes components of the IaaS cloud architecture using terminologies of Xen [11]. Then, Sections 2.2 and 2.3 will describe the existing logging system architecture. In Figure 1, the components of the IaaS architecture are shown as white boxes in the figure. This includes the hypervisor, dom0, domU, hw0, hwU, disk0, diskU, and memU. The box with number 2 in the figure is a hypervisor software that can create more than one VM in one physical machine. The topmost left box in the figure is the domain 0 (dom0) that is a manager of all the created VMs. The dom0 itself is also a VM and is launched by the hypervisor at the system booting duration. Any component with '0' at the end of its name indicates that it is physically managed and owned by a cloud provider. Similarly, 'U' indicates that the component is virtually managed and owned by a cloud customer. The dom0 exclusively accesses hw0 and manages all the created VMs or domUs. A user domain (domU) is a user VM that runs on top of the hypervisor, see the topmost right box of the figure. A domU is an IaaS cloud product that the provider

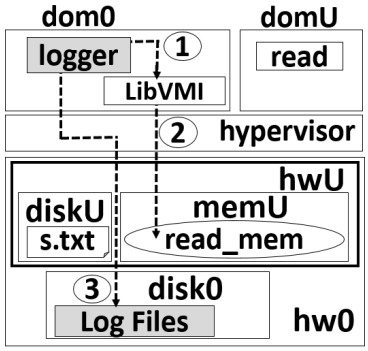

FIGURE 1. The IaaS and logging system architectures, adapted from [10]

offers to an IaaS customer. HwU is physically located in hw0 (which is owned by dom0 or by the provider and virtually owned by domU owner or by the customer), and it is domU's virtual hardware. Disk0 is a physical disk of the hw0, and diskU is a virtual disk of a domU. MemU is the virtual main memory of domU.

2.2. **The logging system types and the logger.** There are two types of logging systems. The first is an interception approach which is to intercept system calls of a domU. A system call provides the interface between a user application and kernel spaces [12]. For example, when our read process needs to read s.txt file, the read process needs to use three system calls: open, read, and close. Thus, the interception approach needs to intercept these three calls to capture desired logging data. The example of this approach is flogger [13]. It is a tool that can intercept file operations of a file that is in a customer VM. This tool can briefly check every file or folder accesses. It also can keep monitoring real-time generated system logs to save such monitored data as log files. The second approach is a memory introspection approach. The technical detail of this approach to obtain the logging data combines two factors: using memory introspection tools (such as LibVMI [14]) and the comprehension of the knowledge of Linux kernel data structures for virtual memory. In this paper, this approach is basically to access and capture logging data in the main memory of a customer targeted VM from the host VM. The example of this approach is a logging system that was introduced by [15]. They describe that this system is in the host VM of a provider. The system can also collect and store logging data about customer VM's process activities such as what process accesses which file. The logging system consists of a logging process and log file [15]. The logging process is responsible for recording necessary logging data. In our experiment in Section 3, we call this process as the 'logger'. The log file is used for storing the logging data obtained from the logger. In this paper, we focus on this logging system from [15] to monitor and capture the logging data of activities of what process accesses a critical file in a customer VM's disk.

2.3. **How the system works.** Figure 1 illustrates the system architecture of the logging system from [10]. See the box inside domU in Figure 1, it is the read process or for short 'read'. In the experiment, we assume that this process can be controlled by an attacker and maliciously read the critical file or s.txt or the document shape inside diskU. The read process in this paper refers to a process of a malicious user which he or she uses to access the critical file of the customer in diskU. When the read application is executed in domU, the operating system or OS of domU creates the read process. Then, the OS reserves some area of main memory (memU) of a domU. We call this area as read_mem, see the oval shape in memU in hwU. The read_mem stores the data of the read process such as process identification (PID), process name, user ID definition (UID), and the file name that this process is reading such as s.txt in diskU in Figure 1. In dom0, the main

components of the logging system architecture are logger or the shaded box in dom0, LibVMI or the white box in dom0, and log files or the shaded box in disk0. LibVMI is a memory introspection tool to read read_mem from memU. Figure 1 illustrates the introspection approach that uses LibVMI to obtain logging data from memU. Usually, the LibVMI performs from a dom0 to obtain logging data of the s.txt or the document shape in diskU in Figure 1. We can deploy LibVMI in the dom0 to read the memory space or read_mem in memU in Figure 1. This memory space holds all the information that we need to record, which is a log file to obtain the contents of the history of critical file s.txt using introspection listed below. The steps are the circles with numbers in Figure 1. Step1, the logger in dom0 calls LibVMI to access memU to get the information in read_mem (step2) such as a file name of s.txt. Then, the obtained information is returned to the logger. The logger manages the obtained information and then writes (in step3) the information into the log file in disk0.

2.4. **The logger and accuracy.**

2.4.1. *The logger hardware environment configurations.* The read process can have activities with s.txt or the document shape in diskU in Figure 1. The details of activities and of how the logger (the white box in the user level of the dom0) works were illustrated in Figure 1. The main objective 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 activities with s.txt file. To see the trends of the accuracy of the logger, the experiment or the measurement will be performed on different hardware configuration environments of the logger. In Figure 1 and on the different hardware configuration environments of the logger, the experiment will be performed by measuring the accuracy values of the logger after each hardware environment configuration by the method of varying the numbers of CPU cores and sizes of the main memory of both dom0 and domU. The measurement of the accuracy of the logger is also performed i) when the read process is reading s.txt in diskU, and ii) when the logger is capturing the desired logging data from the read_mem.

2.4.2. *Hit and miss.* Briefly, the accuracy of the logger or the shaded box in the dom0 is measured by a number of times that the logger captures the right file name or a string "s.txt" in read_mem. This is called 1 'hit'; otherwise it is 1 'miss'. For example, if the read process reads s.txt files 100 times, and the logger can capture the string "s.txt" in read_mem for 100 times or 100 hits, this means that the accuracy of the logger is 100% [16].

2.4.3. *The accuracy of the logger and sleeping time.* Section 2.4.2 described that the read process reads s.txt files 100 times and the logger can capture the string "s.txt" in read_mem 100 times or 100 hits, this means that the accuracy of the logger is 100%. And from our previous work [16], we measured our previous logger with a four CPU cores machine. Then, we found that the accuracy of this logger was 100% when the sleeping time is 65 ms. A sleeping time is an idle time after the read process completes opening and reading tasks of s.txt, and before it is terminated. Thus, when we state that 'the logger has 100% accuracy at 65 ms', this means that "in order to enable our previous logger to capture the right file name value as "s.txt" for every single time or we can say 100% at 65 ms, the read process needs to be in memU at least 65 ms after it finishes reading tasks but before it is terminated". Therefore, in this experiment, from the core(s) from 1 to 8, each core will be configured with 65 ms as the same as our previous work [16]. However, in the experiment in this paper, we will vary the CPU core numbers from 1 core to 8 cores. Then, we will measure the accuracy of the logger when the logger is running in a machine with 1 to 8 cores. Then, we will discuss all eight accuracy values of each logger with 1 to 8 CPU cores.

3. The Implementation.

3.1. **The hardware and software experimental environment.** To vary CPU core numbers for the experiment in this paper here, in dom0, we divide the environment into two parts. The first one is the hardware part, and we set up the experiment environment based on a Lenovo ThinkStation S20 computer machine. This machine comprises an Intel Xeon 3.06 GHz that is CPU 64-bit eight cores, SDRAM 8 GB of main memory, and 320 GB of secondary memory. The second one is the software part, we installed a Fedora 16 64-bit as the operating system or OS of the machine in the experiment, and also installed Xen 4.1.4 hypervisor 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 on the OS that can be called by the logger to access to memU of domU from dom0. In domU, we also installed a Fedora 16 64-bit as the OS and set up the read application on the OS.

3.2. **The method of varying the numbers of CPU cores of both domU and dom0.** In this experiment, a domU is fixed to 1 GB main memory (memU).

3.2.1. *Alignment of varying of CPU cores.* This section studies the trend of the accuracy values of the logger in different numbers of CPU cores configuration environment. The environment is done by varying the numbers of CPU cores of dom0 and domU. In every configuration, the logger will still perform the same routines as discussed in Figure 1 in Section 2.3. Mainly from the figure, the logger needs to capture the necessary data from memU. The data are in read_mem and included: PID of the read process, the name of the read, UID or the ID of the read process's owner, and the file name that the read process is reading, as described in Section 2.3. In the alignment of varying of CPU cores, see the top dash-lined box in Figure 2(a), it is d01c which represents a dom0 that deploys 1 CPU core. Then, the second dash-lined box from the top of Figure 2(a) is d02c which represents a dom0 that deploys CPU 2 cores, and so on, until d08c or the first dash-lined box from the bottom of Figure 2(a). Thus, this box represents a dom0 that deploys 8 CPU cores. Similarly, see the shaded box on the top of Figure 2(a), it is du1c or domU that deploys 1 CPU core. Then, the second shaded box from the top of Figure 2(a) is du2c which represents a domU that deploys 2 CPU cores and so on, until du8c which represents a domU that deploys 8 CPU cores, see the third shaded box from the top of Figure 2(a).

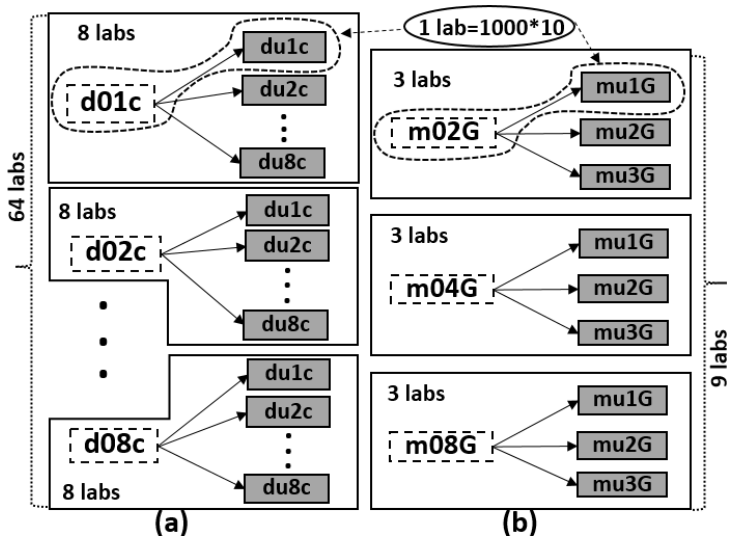

FIGURE 2. The method of varying the numbers of CPU cores and the sizes of main memory of both domU and dom0

3.2.2. *Overall CPU cores varying labs.* The full definition of a lab will be explained in Section 3.2.3. This section here explains the overall varying labs. The first eight labs, see in Figure 2(a), the freeform dotted shape that rounds both the top dash-lined box with labeled 'd01c' and the top shaded box with labeled 'du1c', this is one lab of our experiment or d01c-du1c lab. This lab is also represented by the first arrow from the top of Figure 2(a). Thus, the top white box of Figure 2(a) is d01c which is with 8 labs or d01c-du1c, d01c-du2c, ..., and d01c-du8c. The second 8 labs, see the second dash-lined box from the top of Figure 2(a), it is other 8 labs. The first lab is d02c-du1c or the fourth arrow from the top of Figure 2(a). Other seven labs are d02c-du2c, d02c-du3c, ..., and d02c-du8c. Then, the third to eighth 8 labs are aligned as the same pattern as the first two labs. Thus, there are 64 labs.

3.2.3. *A lab for varying cores.* We needed to modify the logger and the read process in Figure 1. Thus, the routines of the logger and the read here may be slightly changed from Figure 1. See in Figure 2(a) at the freeform dotted shape that rounds both the top dash-lined box with labeled 'd01c' and the top shaded box with labeled 'du1c', this is a lab. It composes of 3 steps or s1 to s3. S1, we run the logger in dom0. S2, we 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 read process is run, and finished and so on, until the 1000^{th} read process is run and finished. In the same time, the logger that is run only one time before the first round of the read process running or s1. The logger will capture the "s.txt" string (this is 1 hit) from each round of the read process starting from the 1^{st} to 1000^{th} rounds. Then, the logger will store the string into the log file. This is also one round that the read process is done, and then the logger will wait for the next read process round running. When the logger gets the "s.txt" string of each round, each hit will be accumulated by 1 per hit. S3, the accumulated number of hits of these 1000 rounds is written to text files. Thus, a lab means we performed s1 to s3, and get the first accumulated number of hits of these first 1000 rounds. Then, we perform s1 to s3 for the other nine times; thus, we will get 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 d01c-du1c or the freeform dotted shape in Figure 2(a).

3.2.4. *First 8 for d01c and 64 labs for all d01c to d08c.* See the top box of Figure 2(a), it is 8 labs. The first lab or lab 1 is the freeform dotted shape of d01c-du1c as just discussed in Section 3.2.3 above. Then, the seven labs are outside the freeform dotted shape but still in the top box of Figure 2(a). They are lab 2 or d01c-du2c or the second arrow from the top of Figure 2(a), lab 3 or d01c-du3c,..., and lab 8 or d01c-du8c or the third arrow from the top of Figure 2(a). This is 8 labs for d01c or the top white box in Figure 2(a). What is 64 labs, the first 8 labs for d01c or the top white box in Figure 2(a) is just discussed above. Then, there will be other 8 labs for d02c (the first freeform shape from the top of Figure 2(a)) and other 8 labs for d03c and so on, until the last set of 8 labs or d08c or the first freeform box from the bottom of Figure 2(a). So, the total is 64 labs or a whole of Figure 2(a).

3.3. **Varying the sizes of main memory of both dom0 and domU.**

3.3.1. *Alignment of varying of main memory.* This section studies the trend of the accuracy of the logger when varying the sizes of the main memory of dom0 and domU. The logger needs to capture the necessary logging data from domU. The data are in read_mem and included: PID of the read process, the name of the read process, UID, and the file name that the read process is reading, as described in Section 2.3. For dom0, the top dash-lined box in Figure 2(b) is m02G which represents a dom0 that deploys 2 GB of main memory (mem0). Then, the second dash-lined box from the top of Figure 2(b) is

m04G which represents a dom0 that deploys 4 GB of mem0. Lastly, the last dash-lined box from the top of Figure 2(b) is m08G which represents a dom0 that deploys 8 GB of mem0. For domU, similarly, the shaded box on the top of Figure 2(b) is mu1G which represents a domU that deploys 1 GB of memU. Mu2G represents a domU that deploys 2 GB of memU. Finally, the third shaded box from the top of Figure 2(b) is mu3G which represents a domU that deploys 3 GB of memU. We will explain the first 3 labs or the first white box on the top of Figure 2(b), as follows. In Figure 2(b), see the freeform dotted shape that rounds both the top dash-lined box with labeled 'm02G' and the top shaded box with labeled 'mu1G', this is one lab of our experiment or m02G-mu1G lab or the first arrow on the top of Figure 2(b). Thus, see the top white box of Figure 2(b), m02G is with 3 labs or m02G-mu1G, m02G-mu2G, and m02G-mu3G. We will explain the second 3 labs or the second white box on the top of Figure 2(b), as follows. See the second dash-lined box from the top of Figure 2(b), it is the other three labs. The first lab is m04G-mu1G or the fourth arrow from the top of Figure 2(b). Thus, see the second white box from the top of Figure 2(b), m04G is with three labs or m04G-mu1G, m04G-mu2G, and m04G-mu3G. We will explain the third or last three labs or the first white box from the bottom of Figure 2(b), as follows. See the third dash-lined box from the top of Figure 2(b), it is other and last three labs. The first lab is m08G-mu1G or the seventh arrow from the top of Figure 2(b). Thus, see the first white box from the bottom of Figure 2(b), m08G is with three labs or m08G-mu1G, m08G-mu2G, and m08G-mu3G. Thus, there are 9 labs.

3.3.2. *A lab of main memory experiment and the first 3 for m02G and 9 labs for m02G to m08G.* See in Figure 2(b) at the freeform dotted shape that rounds both the top dash-lined box with labeled 'm02G' and the top shaded box with labeled 'mu1G', this is a lab of varying main memory. This lab composes of 3 steps or sm1 to sm3. These three steps are as the same s1 to s3, respectively. Thus, a lab here means we performed sm1 to sm3, and get the first accumulated number of hits of these first 1000 rounds. Then, we perform sm1 to sm3 for the other nine times. Thus, we will get 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 m02G-mu1G or the freeform dotted shape in Figure 2(b). For the first 3 and 9 labs for m02G, see the top white box of Figure 2(b), it is 3 labs. The first lab or lab 1 is in the freeform dotted shape of m02G-mu1G as just discussed above. Then, the two labs are outside the freeform shape but still in the top white box of Figure 2(b). They are lab 2 or m02G-mu2G and lab 3 or m02G-mu3G. These 3 labs are for m02G. What are 9 labs, the first 3 labs for m02G or the top white box of Figure 2(b) is just discussed above. Then, there will be 3 labs for m04G (the second white box from the top of Figure 2(b)); finally, other 3 labs for m08G or the first white box from the bottom of Figure 2(b). So, the total is 9 labs or a whole of Figure 2(b).

4. **Results and Discussions.**

4.1. **The results of varying the number of CPU cores of dom0.** The results here are from the experiment of all the three dash-lined boxes on the left of Figure 2(a). Figure 3(a) shows the results or the accuracy of the logger when varying the number of CPU cores of dom0 from 1 to 8 cores for the experiment. The results show that the accuracy of the logger is 100% when dom0 deploys 1 core or d01c, and 8 cores or d08c. When dom0 deploys 2 to 7 of CPU cores or the horizontal rectangle, this decreases the accuracy of the logger. This can be seen in Figure 3(a) that the accuracies of the logger of each one from d02c to d07c are 99%, 97%, 98%, 98%, 99%, and 99%, respectively. There is no any accuracy of the logger with d02c to d07c as 100%, compared to the accuracy of d01c and d08c.

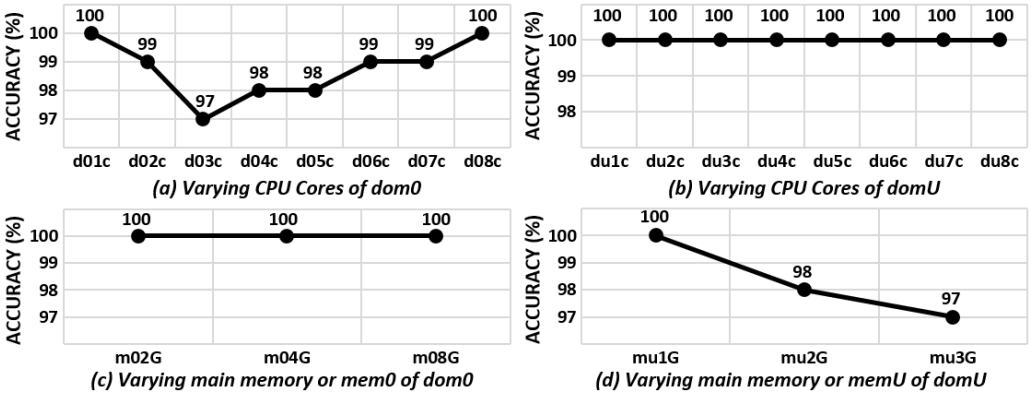

FIGURE 3. The results from varying the numbers of CPU cores and the sizes of main memory of both domU and dom0

4.2. **Discussion of varying the number of CPU cores of dom0.**

4.2.1. *General discussion of results.* For d01c, the accuracy of the logger is 100%. This is because the operating system of dom0 has only one CPU core to serve for all processes, including the logger process. Thus, the logger is never swapped from the current processing CPU core to use a new core. This may cause the accuracy of the logger to be 100%. More details of the swapping task are discussed in the next paragraph. For d02c to d07c, Section 4.1 showed that when varying the number of CPU cores of dom0 from 2 to 7 cores, this decreases the accuracy of the logger. The decreasing situation may be caused by CPU scheduling mechanisms which make the computer system have three properties: fairness, powerfulness, and rapidness, as argued by [17]. Because of these properties, whenever the CPU cores are idle, the operating system must select one of the processes from the ready queue to be executed, even the selected process is not the first one in the queue. If the logger is the selected process to be executed, and if dom0 has more than one core, then the logger process may be swapped from the current core to the new one. When swapping, from example, from the current core 1 to the new core 2, the operating system must move the processing data of the logger process from mem0 to caches of the new core 2 [18, 19]. A cache is a very high-speed memory and is a buffer between the mem0 and the CPU [20]. Thus, this swapping operation of the operating system mentioned above may consume the processing time of the logger process. This swapping task makes all the accuracy of the logger in d02c to d07c labs be lower than 100%. For d08c, the accuracy of the logger is 100%. This may be because the operating system of dom0 has enough CPU cores to serve for all processes, including the logger. Thus, the logger may have its own core to be exclusively processed. Thus, the operating system may not need to swap the logger from the current core to another core. This is no extra consuming processing time for the logger, and it can have efficient operations. Thus, this may cause the accuracy of the logger to be 100% at d08c.

4.2.2. *Trends of the accuracy from CPU cores varying.* There are two trends from the results of the varying CPU cores from the experiment. The first trend of the accuracy of the logger is for from d01c, to d02c, and finally to d03c. The trend is decreased from 100% to 99%, and finally, to 97%, respectively. This decreasing trend may be caused by the following reason. This reason is that d01c has no swapping tasks (discussed above) because it has only one core, compared to d02c, which may allow the logger to be swapped. This consumes the logger processing time. Then, this may cause the logger not be able to capture "s.txt" in time in one read process running round. Thus, to increase the cores of dom0 may decrease the accuracy of the logger in some appropriate core numbers such as 2 to 3 cores, as just discussed above. The second trend of the accuracy of the logger is

for from d03c to d08c and is increased from 97% to 100%, respectively. This increasing situation may be caused by one of the reasons. The reason is that when the logger running in dom0 that has many or enough cores such as 3 to 8, then, the operating system may not allow the logger to be swapped from one core to another. Less swapped tasks may decrease the logger's processing time. Thus, the logger has sufficient time to capture the "s.txt" in time of all the read process running rounds. Thus, to increase the cores of dom0 may increase the accuracy of the logger in some appropriate core numbers such as 3 and to 8 cores, as just discussed above. We also believe that to increase the cores for more than 8, the accuracy of the logger will still be 100%.

4.3. **Results and discussion of varying the number of CPU cores of domU.** These results and discussions are based on the experiment in Section 3.2, which already assumed that a domU is fixed to 1 GB memU. The conclusion of the results is that when we vary CPU cores of domU from 1 to 8, the accuracy of the logger has no effect by this varying task. Figure 3(b) shows all the accuracy values of the logger when increasing the number of CPU cores of domU from 1 to 8, as discussed in the experiment in Section 3.2. See, du1c to du8c in Figure 3(b), the results from the figure show that all the accuracy of the logger is all the same as 100%, when domU has the number of CPU cores for all number from 1 or du1c to 8 du8c. Thus, varying the numbers of CPU cores of domU causes no effect on the accuracy of the logger. The two appropriate reasons could be the following. Firstly, the logger performs in dom0, not domU, and the logger needs only to access memU in domU. Thus, varying by increasing domU core numbers should not affect the accuracy of the logger. This can be seen in Figure 3(b) that all the accuracy values between du1c to du8c are the same values: 100%. Secondly, the read process is small, then the process can work with both CPU 1 core or more than 1 cores of domU with no difference in processing time, as also agreed by [21]. From Figure 3(b), when the read process is running on CPU 1 core of domU, the accuracy now is 100%. And [21] states that a small process that works with 1 or many CPU cores yields no difference in the processing time of this process. Thus, from our experiment, when the logger works on 1, 2,. . ., or 8 CPU cores in domU, all the accuracies of the logger are all the same as 100% which is not different. This can be seen in Figure 3(b) that all the accuracy values between du1c to du8c are the same values: 100%. To sum up, when we vary CPU cores of domU from 1 to 8, the accuracy of the logger has no effect by this varying task.

4.4. **The results and discussions of varying the size of main memory of dom0.** Both core numbers of dom0 and domU are fixed to be 8. The conclusion is that when we vary mem0 of dom0 from 2, to 4, finally to 8, this does not affect the accuracy of the logger. Figure 3(c) illustrates all the accuracy of the logger when varying by increasing the sizes of mem0 of dom0 from 2, 4, to 8. The results from Figure 3(c) are that all the accuracy values of the logger are all the same as 100%, when dom0 has mem0 as 2, 4, and 8 GB. One of the reasons is that the logger process is small, and the logger also needs a small area of mem0 for processing. When increasing the mem0 to be larger, the logger still uses the same small area. Thus, this does not affect the logger accuracy. To sum up, when we vary mem0 of dom0 from 2, 4, to 8, this does not affect the accuracy of the logger.

4.5. **The results and discussions of varying the size of main memory of domU.** Both core numbers of dom0 and domU are fixed to be 8. The conclusion is that varying the memU of domU decreases the accuracy of the logger. See Figure 3(d), the results from the figure show that the accuracy of the logger is decreased from 100% to 98%, and finally to 97%, when the memU is increased from 1 GB to 2 GB, and finally to 3 GB, respectively. This may be because increasing the size of memU of domU will enlarge the area which is searched by the logger for the logging data. Thus, the logger will consume

more time to search the desired logging data in this enlarged area, compared to smaller memU area. We also believe that enlarging the size of memU to be more than 3 GB, this may still also decrease the accuracy of the logger.

5. **Conclusions.** This paper illustrated the accuracy measurement of a logging system or logger in different hardware configurations environments in the Infrastructure as a Service (IaaS) cloud. For the CPU cores measurement, i) when varying the numbers of CPU cores of the host machine (dom0) of the logger, there are two perspectives of the results. Firstly, there are three general results: at 1 core, from 2 to 7 cores, and at 8 cores. At 1 core, the accuracy of the logger is 100%. For 2 to 7 cores, it decreases the accuracy of the logger from 100%. Then, at 8 cores, the accuracy of the logger is still 100%. Secondly, there are two trends in the accuracy of the logger. The first trend is for from 1, to 2, and finally to 3 cores, and the accuracy is decreased from 100%. The second trend is for from 3 to 8 cores, and the accuracy is increased from 97% to 100%, respectively. Thus, to increase the cores for more than 8, the accuracy of the logger should still be 100%. ii) When varying the numbers of CPU cores of a customer virtual machine (VM) or domU, the results are that when we vary CPU cores of domU from 1 to 8, all the accuracy values are 100%. Thus, the accuracy of the logger has no effect by this varying task. For the main memory measurement, a) for dom0, the results are that when we vary main memory or mem0 of dom0 from 2, to 4, finally to 8, this does not affect the accuracy of the logger; b) for domU, the results are that varying the main memory memU of domU decreases the accuracy of the logger when the memU is increased from 1 GB to 2 GB, and finally to 3 GB. We also believe that enlarging the size of memU to be more than 3 GB, this may still also decrease the accuracy of the logger, compared to less than or equal to 3 GB. The future work is to apply parallel programming to enhancing the accuracy of the logger.

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)*, 2019.
[2] S. Alshamrani, An efficient algorithm for monitoring virtual machines in clouds, *Bulletin of Electrical Engineering and Informatics*, 2019.
[3] R. Kaur and G. Kaur, Proactive scheduling in cloud computing, *Bulletin of Electrical Engineering and Informatics*, 2017.
[4] J. Brodkin, Gartner: Seven cloud-computing security risks, *Infoworld*, 2008.
[5] W. Liu, Research on cloud computing security problem and strategy, *International Conference on Consumer Electronics, Communications and Networks*, 2012.
[6] S. Subashini and V. Kavitha, A survey on security issues in service delivery models of cloud computing, *Journal of Network and Computer Applications*, 2011.
[7] T. T. W. Group et al., The treacherous 12: Cloud computing top threats in 2016, *Cloud Security Alliance*, 2016.
[8] I. Molyneaux, *The Art of Application Performance Testing: From Strategy to Tools*, O'Reilly Media, Inc., 2014.
[9] P. Chan-In and W. Wongthai, Performance improvement considerations of cloud logging systems, *ICIC Express Letters*, vol.11, no.1, pp.37-43, 2017.
[10] W. Wongthai, *Systematic Support for Accountability in the Cloud*, Ph.D. Thesis, Newcastle University, 2014.
[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, Xen and the art of virtualization, *SIGOPS Oper. Syst. Rev.*, 2003.
[12] R. Love, *Linux Kernel Development*, 3rd Edition, Addison-Wesley Professional, 2010.
[13] R. K. Ko, P. Jagadpramana and B. S. Lee, Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments, *International Conference on Trust, Security and Privacy in Computing and Communications*, 2011.
[14] B. Payne, *About the VMI Tools Project, Google Project Hosting*, 2013.

[15] W. Wongthai, F. L. Rocha and A. van Moorsel, A generic logging template for infrastructure as a service cloud, *International Conference on Advanced Information Networking and Applications Workshops*, 2013.
[16] W. Wongthai and A. van Moorsel, Performance measurement of logging systems in infrastructure as a service cloud, *ICIC Express Letters*, vol.10, no.2, pp.347-354, 2016.
[17] N. Ishkov, *A Complete Guide to Linux Process Scheduling*, Master Thesis, Tampere University, 2015.
[18] P. Gepner and M. F. Kowalik, Multi-core processors: New way to achieve high system performance, *International Symposium on Parallel Computing in Electrical Engineering*, 2006.
[19] E. Cota-Robles, *Priority Based Simultaneous Multi-Threading*, United States Patent, 2003.
[20] J. Handy, *The Cache Memory Book*, Morgan Kaufmann, 1998.
[21] M. Rouse, *Definition: Multi-Core Processor*, TechTarget, vol.6, 2013.