Future Generation Computer Systems 137 (2022) 219–233

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Cloud-BlackBox: Toward practical recording and tracking of VM swarms for multifaceted cloud inspection

Sang-Hoon Choi [a], Ki-Woong Park [b,*]

[a] *SysCore Lab, Sejong University, Seoul 05006, South Korea*
[b] *Department of Information Security, and Convergence Engineering for Intelligent Drone, Sejong University, Seoul 05006, South Korea*

ARTICLE INFO

Article history:
Received 31 December 2021
Received in revised form 20 May 2022
Accepted 10 July 2022
Available online 14 July 2022

Keywords:
Cloud computing
Hypervisor
Virtual machine
VM swarms
System inspection
Memory snapshot

ABSTRACT

Given the widening scope of the utilization and application of cloud computing services from general to mission-critical systems such as strategic military, financial, and the information systems of governmental agencies, the need for the development of improved methods to ensure the stability and security of cloud data and services is being increasingly emphasized. Various approaches have been developed to improve the security and stability of cloud infrastructure. In particular, the continuous inspection of the memory of Virtual Machine (VM) instances in the cloud platform has been an important factor in identifying the causes of security incidents related to zero-day vulnerabilities and critical system faults in cloud infrastructure. However, despite numerous studies in the field of continuous memory inspection, it is difficult to find a practical solution that is deployable in commercial-off-the-shelf cloud platforms. For instance, continuous memory snapshots generally cause various problems such as increased VM downtime occurrences, user-obstructive latency for memory snapshots, VM performance degradation, and massive data generation. To alleviate these limitations, we propose *Cloud-BlackBox*, which enables the recording of the memory of VM swarms running on cloud platforms that require a very high level of stability and security, and facilitates the flexible analysis of the recorded memory on a large-scale. A VM swarm refers to an environment in which multiple VMs are run in parallel. The proposed *Cloud-BlackBox* method provides the following benefits. First, by clustering VM swarm kernel memory, the amount of computation required to capture memory snapshots and the size of the generated snapshot images are minimized. Further, we propose a mechanism to merge kernel memory by rapidly identifying the homogeneity of the memory layout through analysis of the underlying base image and introspection of the running VM. The application of the proposed mechanism led to a storage reduction by a factor of 12.85. Second, a cognitive-scale bitmap was designed to track changes in the memory of VM swarms. The cognitive-scale bitmap is a mechanism that can dynamically manage the tracking of memory change information by recognizing the memory usage patterns of component VMs. With the designed cognitive-scale bitmap, the time required for the collection of a memory snapshot was reduced by more than 14.85 times, and the VM input/output (I/O) performance degradation was reduced by 50%. Third, a synchronized accessible memory interchange (SAMI) mechanism is proposed to facilitate the agile in-depth analysis of large-scale memory resources. *Cloud-BlackBox* tracks and records memory change information. Therefore, a procedure for restoring the recorded memory to a raw-memory analyzable form is required to analyze the recorded memory. The SAMI mechanism assists the analyst in ensuring consistent memory restoration performance when arbitrarily selecting recorded memory. Furthermore, SAMI is useful for reducing the scope of analysis without memory restoration simply by analyzing recorded metadata. Consequently, the revised schemes inside *Cloud-BlackBox* have several applications in various fields, such as advanced detection of malicious activities, service error recovery, malware analysis, and antivirus functions. In addition, the proposed approach has been implemented on a campus-wide cloud computing service called *SysCore-Cloud*.

1. Introduction

In recent years, cloud computing infrastructure has developed through a growth phase into a mature implementation

* Corresponding author.
E-mail addresses: csh0052@gmail.com (S.-H. Choi), woongbak@sejong.ac.kr (K.-W. Park).

https://doi.org/10.1016/j.future.2022.07.002
0167-739X/

phase [1–3]. The evidence for such development and use of cloud computing can be found in the Joint Enterprise Defense Infrastructure of the United States Department of Defense, as well as in other mission-critical infrastructure such as financial and information technology systems used by government agencies [4–6]. These types of mission-critical infrastructure are highly vulnerable to loss or theft of data and associated aftereffects in the event of cyber incidents such as service errors or malicious attacks, Thus, preparation of adequate countermeasures against such cyber incidents is imperative [7–9].

Various approaches are under development to improve the stability and security of cloud services. Specifically, considerable attention has been focused on the investigation of methods to utilize the system memory of VMs as a means to identify and address the causes of security accidents and service errors in cloud platforms. VM introspection provides a deep dive view of the Operating System (OS) that monitors transparently factors such as process, kernel driver, active network, registry, and user activities [10,11]. In the case of cloud infrastructure, memory introspection of all running services can be conducted at the host OS level without having to install an agent in each system. Owing to this benefit, memory inspection technology in virtualization environments has been applied in the field of forensics, malware analysis, antivirus, and service error recovery [12–15]. In particular, advanced persistent threats (APTs) such as Operation Cobalt Kitty, Ammyy Admin, and advanced attacks such as Ramnit, Emotet, and Ursnif are difficult to detect unless system memory is monitored [16–20]. To defend against such advanced attacks, there has been attempts to utilize memory analysis on cloud platforms as well. For instance, Microsoft Azure supports fileless attack detection on Windows systems; this feature scans the memory of all processes for evidence of fileless toolkits, techniques, and behaviors [21]. Memory forensics, to enable responses to the latest malicious code injections and advanced attacks, is gaining attention from both academia and commercial users, and steadily evolving [22–24]. However, short-term memory analysis based solutions have a limitation in that only known attacks can be detected. Responding to various security issues is with short-term memory analysis alone is challenging because system memory has the characteristics of being volatile and constantly changing [25,26]. Therefore, it is difficult to have the timing of the memory analysis coincide with security threat incidents and find meaningful data with only short-term memory analysis [27,28]. Consequently, it is necessary to respond to potential threats through long-term memory archives and analysis, rather than at the level of real-time memory scans.

Unfortunately, long-term memory snapshots incur computational overheads that lead to deterioration of VM performance and occurrence of VM downtime from the standpoint of the service user. To adopt continuous memory acquisition/analysis techniques in a cloud computing infrastructure running a large-scale of VMs, the following three problems must be addressed and resolved. First, the storage space required for memory acquisition must be efficiently managed. Although system memory is a smaller than non-volatile memory, in the case of the cumulative storage of memory over an extended period, a massive amount of storage space is required. For example, when 4 GB of memory has been cumulatively acquired for 1 h at 1-second intervals, approximately 14 TB of data is generated. This means long-term memory collection in VM swarms generates massive amounts of data. Second, VM downtime and compromised VM system performance due to memory acquisition overhead must be reduced. On cloud computing platforms, VM memory snapshots incur substantial computational overhead. According to the experimental results obtained in this study, a time of 9.3 s was required to collect 4 GB of memory snapshot in a VM swarm. VM downtime ensued for approximately 9 s as well. Thus, when memory is acquired from VM swarms concurrently using extant methods, the computational overhead causes system operation inability. Third, an explosive increase in the number of analysis targets presents further issues. Even if snapshots of the cumulatively acquired memory of VM swarms are successfully taken, it is difficult to directly use the acquired memory for identifying the cause of security incidents. This is because meaningful information can only be identified through memory analysis. Accordingly, archiving methods specialized for large-scale memory-set analysis are required.

In this study, we propose an approach called *Cloud-BlackBox* that enables computationally efficient recording, cognitive memory tracking, and analysis of the memory of VMs in large-scale cloud computing environments. A conceptual diagram of the proposed method is shown in Fig. 1. This study presents the following solutions upon an analysis of the three problems described above that may occur for memory recording on a cloud platform.

- **Minimization of wasted storage space in VM swarms:** First, the kernel memory regions are grouped into shared memory regions by using the image information of running VMs. Collection data are minimized by exploiting the fact that similar base images are used in a cloud environment [29]. With this characteristic of cloud platforms, we propose a shared memory merging mechanism for VMs that reduces the target memory size by grouping VMs using the same OS images and managing redundant memory regions as shared pages. In addition, the proposed shared memory merging mechanism has a feature that quickly scans the memory of VM swarms based on the similarity of the memory layout for each OS. Our evaluation results show that the memory merging mechanism reduces the storage space generated during memory recording by a factor of over 12.85.
- **Minimization of memory acquisition overhead through a cognitive-scale bitmap:** Second, cognitive memory tracking enables efficient tracking of the memory information of VM swarms running on cloud platforms. We found that memory acquisition at 1000 ms intervals for a single native VM with 4 GB RAM allocated results in approximately 96.7% redundant data. This means only a part of the memory region is changing on running VMs. Based on these characteristics, we introduce a memory delta tracking mechanism for VMs, which tracks the changing page information in all memory regions used by VMs. Only the changing memory is tracked and cached in the proposed approach, and the cached data are copied to disk, thereby minimizing disk I/O. Our evaluation results show that memory acquisition time caused by blocking is reduced by 14.85 times, and the VM downtime problem during memory acquisition occurs only initially when the service starts, ensuring seamless operation thereafter.
- **Maximizing large-scale memory inspection efficiency via a SAMI mechanism:** Third, the target memory is encoded for efficient analysis and stored on relatively durable media using the tracked memory information and shared memory merger information. In a cloud environment, because many VM memory images are recorded to improve security and stability, the amount of memory to be analyzed increases exponentially. In our research, we propose a memory archival mechanism called the synchronized accessible memory interchange (SAMI), which is able to efficiently analyze large-scale VM memory in consideration of the intended goals, purpose, and characteristics of the memory

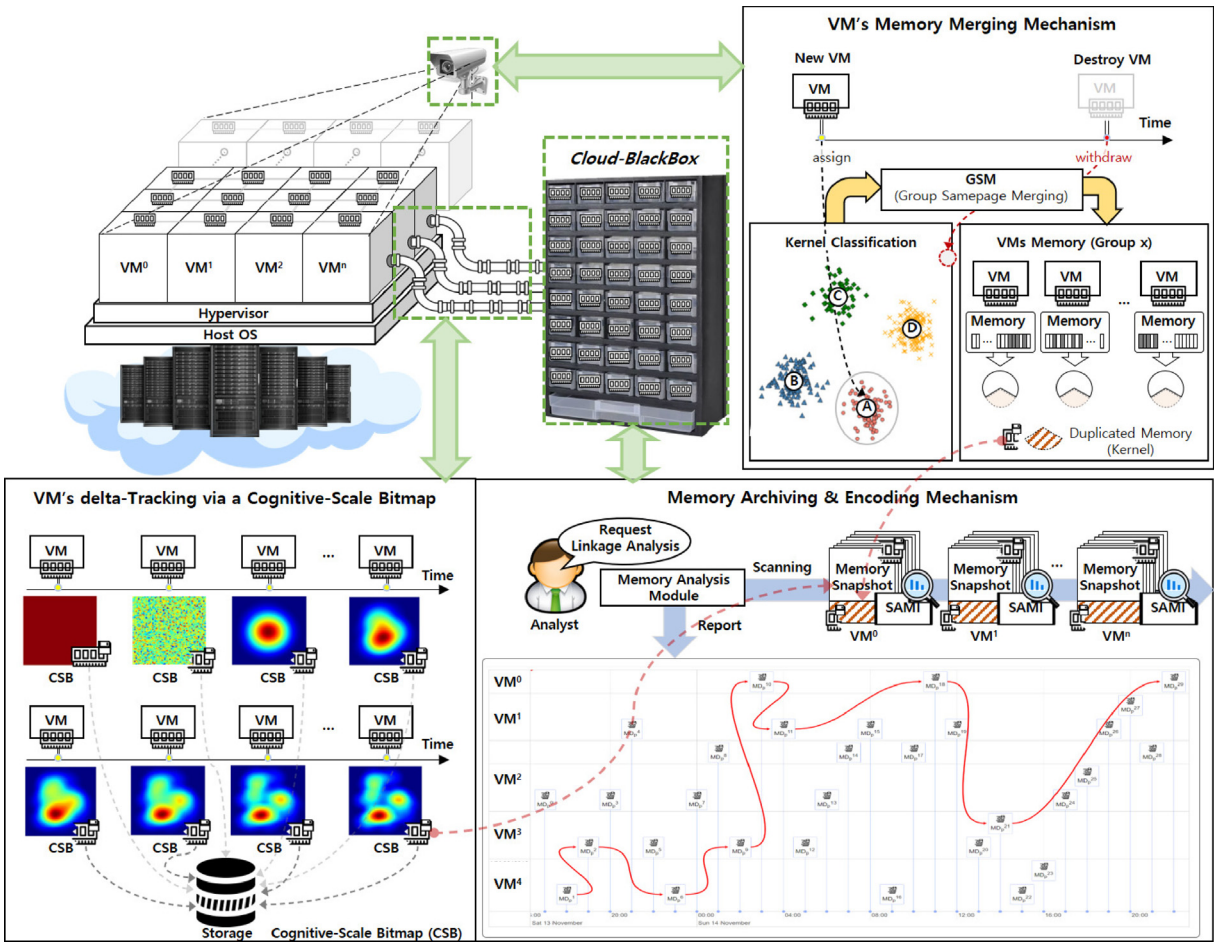

Fig. 1. Concept overview of *Cloud-BlackBox*.

analyzer. The SAMI mechanism utilizes data created through memory tracking for memory-affecting information such as processes, libraries, and networks, for correlation analysis of the target VM swarm. SAMI metadata provides a memory analyst with a macroscopic view of memory changes. The memory analyst can narrow the scope of the memory analysis subject by analyzing metadata without a memory restoration procedure. The mechanism uses checkpoints and quickly restores the recorded partial memory data to a complete memory state suitable for analysis. In addition, memory recorded through the SAMI mechanism provides scalability that can be analyzed using the existing memory forensic frameworks [30].

Contributions. In summary, this study makes the following contributions:

- A shared memory merging mechanism is proposed through analysis of the memory layout similarity of target VM swam.
- VM memory delta-tracking via a cognitive-scale bitmap is presented.
- A specialized memory archival and encoding design is also presented.

The remainder of this paper is organized as follows. Section 2 discusses the motivation for proposing *Cloud-BlackBox* and presents related works. Section 3 describes the three mechanisms proposed for VM memory recording in a cloud environment. In Section 4, the performance of the proposed *Cloud-BlackBox* approach is evaluated. Finally, Section 5 presents the conclusions of the present work and its implications for future research.

2. Related works

This section describes the research motivation for focusing on memory recording and analysis as a means to improve the level of security and safety of cloud services and overcome the limitations of related prior research.

2.1. Improving system security with memory introspection

System memory is a space dedicated to storage of the active data used in an OS. It is well-known that information systems security researchers have long been interested in OS memory. There are two main reasons for their interest.

First, memory is the most important factor in detecting malicious activities because most attack techniques target memory corruption as an attack point. That is, the mechanism for most malicious user system-attacks is a process of corrupting or loading malicious code into memory. Thus, if system memory can be monitored well, most of attack techniques can be blocked or detected. In particular, in virtual environments such as the cloud, there are no restrictions on access to memory, and active researches on improving security using memory have been conducted. The following prior studies exemplify this. VM introspection (VMI), as transparent introspection of VM memory in a virtual environment, was first proposed by Garfinkel and Rosenblum in 2003 [31]. With VMI, transparent introspection of the state, events, and memory of a VM is possible without having to install an agent inside the VM. Methods proposed in research utilizing the concept of VMI include Libvmi [32],

Ether [12], DRAKVUF [15], and Modchecker [33]. Libvmi is a tool with a library for monitoring the states of VMs in hypervisor environments such as Xen and kernel-based virtual machine (KVM). With Libvmi, transparent introspection is possible on the files, process lists, system call tables, DLL lists, and memory buffers of running VMs. LiveWire is a VMI-based intrusion detection system, that detects malicious activities inside a VM based on this policy. LiveWire detects malicious activities by using signature scanning or integrity scanning of executable files. Ether uses a hardware-assisted virtualization method in the field of memory inspection, and methods have been proposed for the transparent monitoring of VMs in a host OS. In particular, it is specialized in extracting runtime activity information of obfuscated malicious code in a virtual environment. DRAKVUF is a product of Ether-based research that can extract the function call information of an OS. With DRAKVUF, parallel VM introspection is possible by minimizing the resource requirements for performing VMI. In addition, through kernel function tracing, it can be used to detect rootkits hiding malicious payloads, as well as the conventional system call tracking Modchecker, which is a VMI-based solution for malicious activity detection designed for use on actual cloud platforms, It can detect code modification attacks such as inline hooking and DLL hooking by comparing the kernel modules of VM swarms. Mishra et al. proposed VMshiled as a method to protect virtual domains on cloud platforms. VMshiled uses VM introspection as a means to detect covert activities taking place with malicious code [24]. Furthermore, in recent years, several studies have been conducted with the objective of improving the security of a virtual environment by combining VMI with machine learning. Win et al. [34]. reported in their research that malicious codes and rootkits can be identified in Linux VMs via support vector machines. VMGuard [35]. uses random forest to perform training on known malicious code and proposed a solution able to detect known malicious codes loaded into VMs. Bozkir et al. proposed a method to convert dumped memory into visualized images to detect the latest malware that applied obfuscation or encryption [23]. As can be seen from these examples of previous research, owing to the versatile applicability of memory, research has been continuously conducted to enhance the security and stability of computational systems by exploiting memory.

Second, important evidence remains in the memory of any abnormal event. Traditional system forensics have focused on the analysis of nonvolatile data, such as system-logging and storage forensics [36,37]. This is because non-volatile data are easier to collect than volatile data. However, the important evidence in the investigation of the causes of cyber accidents or incidents is often stored in volatile memory storage. Nevertheless, memory can be easily corrupted by a malicious user or cleared through a reboot, rendering the data challenging to obtain useful evidence from. Despite these challenges, because memory acquisition is easier in virtual environments than in general systems, various studies on memory forensics have been conducted [10,38,39]. As a representative tool for collecting memory in cloud environments, LIME is a module that is loaded into the Linux kernel to enable the acquisition of full memory locally or remotely via TCP. In hypervisor environments where the host OS is Linux-based, such as QEMU and KVM, LIME can be used to acquire VM memory. Representative studies on memory forensics include Volatility and Rekall. Volatility is a memory forensic framework presented at the 2007 BlackHat DC Convention. Volatility can be used in Linux, Windows, and OS X, and provides various plug-ins. In addition, it allows the capturing of snapshots of the process memory through its own plugin. Rekall is another type of memory forensic framework developed by Google based on Volatility. Rekall is characterized by providing various memory forensics functions, such as collection, analysis, and reporting. Various studies on security using the above memory forensic tools are still being published [22,40]. Furthermore, studies that incorporate artificial intelligence (AI) technology into memory analysis research are also being published. Recently, research on DeepMem was also published; it allows quick and efficient analysis of the kernel structure of dumped memory through graph neural network model training [41].

2.2. Approach to accelerate memory snapshots

The acceleration and optimization of memory snapshot operations are imperative for efficient memory recording. The most relevant research on memory snapshot acceleration concerns VM snapshot. In the VM snapshot process, the states of the VM at a specific point in time are copied and saved in storage. The internal components of a VM snapshot include disk, memory, CPU, and devices. Various approaches have been used to accelerate VM snapshots in the area of service migration. Methods of recording VM snapshots can be broadly divided into stop-copy and pre-copy mechanisms [42,43]. Stop-copy mechanisms save the state of VMs while the operation of the VM is suspended and resumed thereafter. This method is easier to implement, but VM downtime occurs during the process, and it thus cannot be used in a real cloud service.

The pre-copy mechanism is used in live migrations, and is designed to minimize the downtime that occurs in the stop-copy mechanism [44,45]. In the process of the pre-copy mechanism, the changed page information, called dirty pages, is repeatedly transmitted while maintaining the operational state of the VM. When the memory falls below a certain threshold, the operation of the VM is temporarily suspended, and the remaining memory information is transmitted. That is, large memory resources are divided for transmission, and only part of the memory is finally transmitted, minimizing the VM downtime. However, depending on the workload, downtime can be prolonged, and in the worst case, migration can fail. There have been various approaches to address this challenge, such as memory compression, memory redundancy removal, and zero-page removal [46,47]. In addition, studies have been recently conducted on continuous snapshots [48]. Klemperer used a shared memory snapshot driver and page tracking mechanism to accelerate the memory snapshot process, enabling concurrent memory snapshots. In iConsnap, a memory snapshot operation is accelerated using a page tracking method, and an asynchronous buffer was used to minimize the I/O overhead generated during the memory snapshot process.

2.3. Limitations of memory introspection in cloud environment

The above-mentioned results have been obtained through the dedication and interest of several researchers in various fields to improve information security using memory access trace analysis. However, in commercial cloud environments, these various tools have not been utilized. This may be attributed to three key factors.

First, there are possibilities of system performance compromising problems owing to memory acquisition overhead. A memory acquisition operation requires iterative memory read/write operations, which inevitably affect VM performance. However, for in-depth memory inspection, memory acquisition must be performed. The experimental results of this study showed that it takes approximately 9.3 s to acquire 4 GB of memory using Libvirt (open-source platform virtualization management tool) in a KVM-based virtual environment. In addition, the memory snapshot operation suspends the VM's operations until the memory snapshot process is completed to maintain the consistency of the memory; thus, for each memory acquisition, a running VM must

be suspended for over 9 s. That is, the use of memory acquisition operations potentially entails the violation of users' service-level agreements, making it practically challenging to apply in commercially available commercial-off-the-shelf product platforms. To resolve this limitation, Klemperer's iConsnap research minimized VM downtime and memory snapshot duration by using the dirty-page tracking technique. However, the write permission of all the memory regions of the VM is forcibly disabled for the dirty-page tracking process, resulting in compromised memory write operation performance.

The second reason involves the problem of storage space taken up by memory acquisition. A single memory acquisition takes up a relatively small storage space. However, in terms of cumulative memory recording over a prolonged period, a massive amount of storage space is required. For instance, approximately 14 TB of data is generated when 4 GB of memory is acquired for 1 h at 1-s intervals. That is, memory recording in a cloud platform with running VM swarms implies the generation of considerable data. The management of these data is challenging, and storage costs increase exponentially over time. However, the acquisition of meaningful data from memory analysis in a cloud platform requires recording memory over long periods. In the iConsnap research, storage costs were minimized by using compression and deleting past data. Paradoxically, minimizing storage cost through compression actually increases the time required for memory inspection, while deleting past data carries the risk of potentially losing useful data. Avoidance of data loss is considered crucial in recording and managing memory. Memory acquired over a prolonged period without loss can be used as decisive evidence for detecting advanced attack techniques such as APTs or investigating the cause after the occurrence of an abnormal event. By recording information on the activities of malicious users exploiting cloud services for malicious purposes, memory recordings can be used as legal evidence when violations occur. In addition, memory recording can be used as a generic technology in various areas, such as malicious code analysis and antivirus functions. The delta snapshot technique is used in various virtualization-related products as a means of minimizing storage space wasted by snapshots [49–51]. However, in order to perform the delta snapshot, operations on a virtual machine need to be stopped, or the performance of the virtual machine is severely degraded while the snapshot is performed. These problems occur because delta snapshots are designed for efficient snapshot management and data integrity. In other words, the delta snapshot technique can minimize the storage wasted by snapshots but is limited in that, while the operation is performed, it is difficult to ensure the availability of VMs on cloud platforms.

The third reason is the sharp increase in the analysis target owing to the continuous memory snapshots. Widely used memory inspection frameworks such as Volatility and Rekall are designed with a focus on analyzing a single memory image. However, to use memory inspection to improve stability and security in a cloud computing environment, the memory of all running VMs must be collectively and periodically acquired. That is, recording the memory of VM swarms indicates that the memory to be analyzed increases exponentially. Therefore, the use of current memory frameworks remains inefficient for the analysis of large amounts of recorded memory. Moreover, it is challenging to derive meaningful analytical results when memory is repeatedly acquired without considering the intention, purpose, and characteristics of memory inspection. Therefore, to utilize memory as an element for improving security and stability in an environment with an operational VM swarm, it is necessary to implement an archiving and analysis mechanism specialized for large-scale memory inspection.

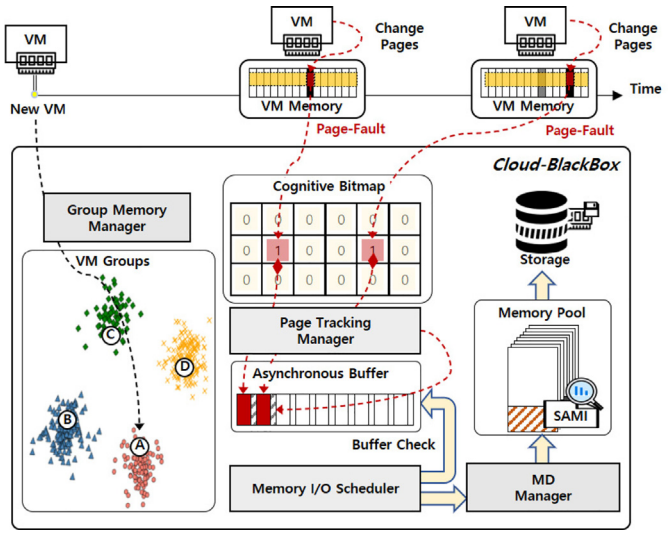

Fig. 2. Structure and flow of operations in Cloud-BlackBox.

3. *Cloud-BlackBox* internals

This section describes *Cloud-BlackBox*, which enables computationally efficient recording and analysis of memory in a cloud computing environment. *Cloud-BlackBox* is capable of time-efficient recording of VM swarms' memory and minimizes the storage costs generated by memory acquisition. In addition, it is characterized by an efficient in-depth analysis of acquired large-scale memory datasets.

3.1. Overview

As shown in Fig. 2, *Cloud-BlackBox* consists of four major components: group memory manager, page tracking manager, memory I/O scheduler, and Memory Dump (MD) manager.

The group memory manager performs cross-validation of the memory layout of VMs running on a physical machine, groups VMs using the same or similar kernels, and merges duplicate kernel memory regions. The group memory manager module is designed to regroup and merge kernel memory when a VM is created or removed from a physical machine.

The page tracking manager is a mechanism for tracking and managing the changing memory of VMs. This module tracks the memory change information of the shared page merged through the group memory manager and private page, which is an unshared, personal memory region of a VM. We propose a cognitive-scale bitmap as a means to efficiently track memory change information in VMs by actively utilizing the temporal/spatial locality properties of memory organization. The cognitive-scale bitmap consists of a micro bitmap, which tracks VM memory from a micro perspective, and a macro bitmap, which tracks VM memory from a macro perspective. In addition, the page tracking manager is executed at a specific interval specified by a user or request event.

The memory I/O scheduler is a mechanism for the efficient storage of volatile memory in a non-volatile region. The actual data of the memory page tracked through the page tracking manager are copied to the asynchronous access memory buffer, and the copied memory data are flushed to the storage by a background thread.

The MD manager is a mechanism for efficient management and analysis of the acquired memory. In this study, an acquired memory file is referred to as an MD. Because *Cloud-BlackBox*

tracks and records only memory that changes, rather than the full memory, a memory restoration procedure is necessary to acquire meaningful data through a memory inspection tool. The MD manager ensures random access memory restoration by utilizing checkpoints, and tracked memory information is utilized by the reach frame to create metadata, enabling the efficient analysis of large-scale memory. In addition, it provides compression of frames before checkpoints without loss according to user requirements for storage cost reduction.

3.2. Shared memory merge for efficient memory collection

The factor incurring the largest overhead in VM memory snapshots is the operation of repeatedly cloning the memory. The computational complexity of cloned memory is proportional to the allocated memory size. In general, VM memory of a given size in GB units is allocated, which requires numerous memory exploration and recursive cloning operations in memory snapshots. In the case of a large-scale cloud platform running VM swarms, the memory snapshot operation generates even more overhead.

The key feature of the proposed *Cloud-BlackBox* is memory merging of VM swarms to minimize the range of memory for snapshots and optimize the computational overhead generated by memory snapshots. Our experimental results indicate that over 30% of memory is redundant with VM swarms using the same kernel. This feature can be best exploited in a cloud platform environment with a VM swarm running. Most cloud platforms provide base images to users, therefore the types of kernels that can run in cloud platform environments are limited.

According to the data published by Amazon Web Services (AWS), it was observed that 80% of Amazon Elastic Compute Cloud (EC2) users primarily run Linux-based OSs [29]. This means if the memory of VMs using the same kernel is grouped and merged to share memory, the memory range for exploration and snapshot can be reduced. However, when a user's private data are uploaded to the shared memory region to be merged, it may cause a security problem in that the memory of other VMs can be accessed. Therefore, the memory dedicated to merging must be in the kernel memory region that cannot be accessed at the user level and a space where private data cannot be uploaded. This type of safe memory region can be specified through memory inspection and repeated workload processing for each kernel that is run in a cloud environment. The kernel/user memory regions are not distinguished during the process of merging the memory, but the kernel memory region can be predicted by analyzing the merged memory. Thus, because immediate detection is possible in the kernel memory region in the case of significant changes, this approach has the advantage of being able to detect kernel corruption attacks. In summary, the group memory manager is a mechanism designed to minimize redundant memory exploration and cloning operations through memory merging. We perform memory merging on a page-by-page basis to increase the effectiveness of memory merging of VMs. Various studies have demonstrated that merging memory at the page level is very effective [52,53]. However, as a downside, a page-level redundancy check requires a significant amount of time. The global memory manager provides two methods that may shorten the page-level redundancy check time. We reduced the scope of redundancy check through the method of grouping VMs with similar kernels and shortened the page-level redundancy check time through the method of hashing adjacent pages with high merge rates to verify them all at once.

The group memory manager follows four procedural steps, as shown in Fig. 3. The first step is VM kernel checking. The kernel version used by the VM can be identified by checking the specific kernel memory of the VM. In addition, the base image

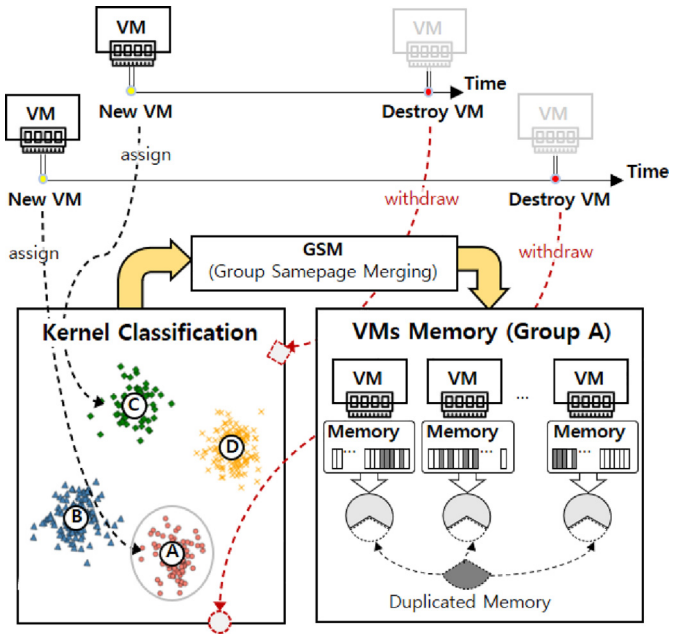

Fig. 3. Process of classifying a virtual machine's memory into shared and private pages.

of the VM can be checked using the virtual machine manager (VMM). The second step is VM grouping. VMs using the same kernel have the same kernel memory specifications. Therefore, when such VMs are grouped together, stability is improved when memory is merged, and exploration time is minimized. The third step consists of shared memory exploration. The initial page redundancy check is performed on a page-by-page basis within the group. The redundancy check also includes pages filled with zeros. The reason we include zero-pages is that only a few areas of the memory are used in a VM due to the temporal/spatial locality characteristics of memory organization. In the case of VMs performing memory-intensive workloads, zero-pages may not exist. We separately select pages that are more likely to be merged as the memory merge process progresses. In this study, rather than considering the method of efficiently selecting pages with a high probability of being merged, we assume that pages that are cloned more than five times in different VMs are likely to be merged. We hash two or more adjacent pages among the selected pages and manage them as a table. The reason for grouping and hashing only adjacent pages is that these pages are deemed to be correlated. Subsequently, a VM with no page redundancy check performed within the group or a hash table created with a new VM is inspected preferentially. We use the Rabin–Karp algorithm as a method to verify the redundancy of the hashed pages. Since the Rabin–Karp algorithm is specialized in the hash multi-pattern search, it is very effective when verifying the hash value of memory that reflects spatial locality characteristics [54,55]. When pages with a high merge rate are verified in hash units, the merge rate is higher relative to the verification computation, and the effect of verifying large-scale pages at once can be achieved. Memory merging is the final step. Shared memory merging utilizes some of the features of the Kernel Samepage Merging (KSM) function provided by Linux kernel 2.6.32 [52]. Specifically, we implement Group Samepage Merging (GSM) by modifying KSM. The GSM function merges memory by pointing the pages selected in the page redundancy check to a single page, similar to KSM. Subsequently, the merged page is managed through copy-on-write (COW) to map the VM page of an event subject to a new physical page as soon as the

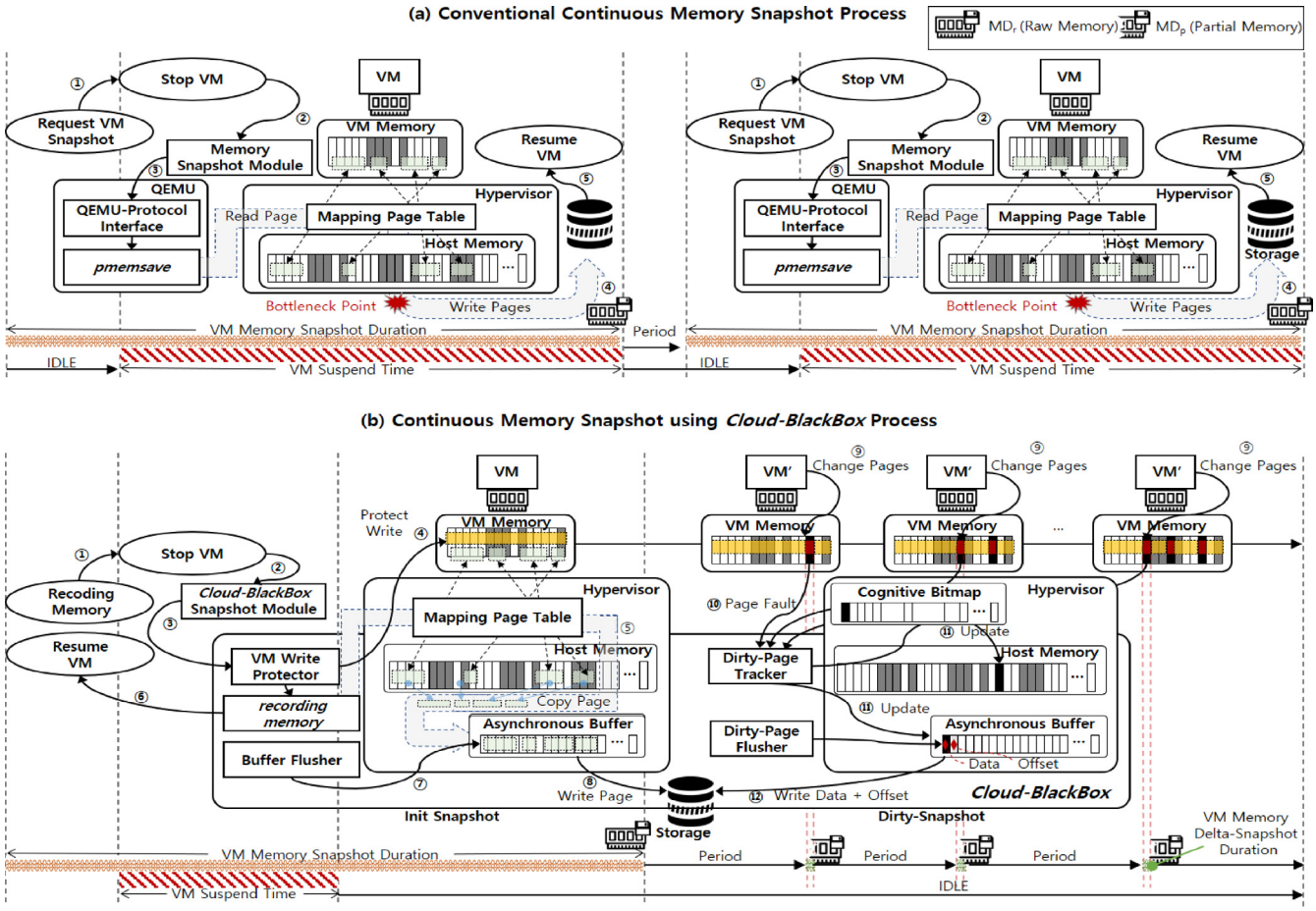

Fig. 4. Workflow of conventional continuous memory dump and memory recording.

write operation occurs. Additionally, among various functions of the KSM function, GSM utilizes a system function that can specify the range of memory merging and the red–black tree used for memory redundancy checks. The GSM we have designed captures snapshots of the shared page by calling the memory recorder and the memory I/O scheduler when memory corruption is detected in a user-specified period or a shared memory region. The reason for this design is to shorten the memory restoration time when converting partially recorded memory into the memory required for analysis.

3.3. Memory tracking mechanism for memory recording

VM memory recording requires that memory snapshot operations are performed iteratively. As described above, *Cloud-BlackBox* performs the first optimization by merging the memory to be explored during the memory snapshot operation through the MD manager. However, VM memory snapshots still generate a large computational overhead. The most pronounced limitation of the VM memory snapshot operation is that the running of the VM is temporarily suspended during the snapshot process, and the duration of the memory snapshot is substantial. From Fig. 4, it is clear that the operation of the VM is suspended for the memory snapshot, and it resumes after the memory snapshots are recorded. That is, the VM suspension time is proportional to the memory snapshot duration. According to our experimental results, it takes approximately 9.3 s to acquire 4 GB of VM memory by QEMU. Thus, cloud service for users is disrupted for 9.3 s by a single memory snapshot. This situation is even worse when performing continuous iterative memory snapshot operations. Moreover, because the memory snapshot operations share the resources of the host OS with the VM, competition occurs in the use of resources. This compromises the performance of other VMs running on the same host machine. In other words, memory snapshot operations are considered inapplicable to cloud platforms because they require the violation of service-level agreements.

Cloud-BlackBox performs an in-depth analysis of the bottleneck points that cause delays in the existing memory snapshot process to address the above limitation. The process of performing a memory snapshot is shown in Fig. 4(a). ① When a VM snapshot request is received, the VM is stopped. ② When the VM is stopped, it calls the memory snapshot module. ③ The memory snapshot module calls `pmemsave` of QEMU-Protocol. ④–⑤ The memory snapshot module then calls `pmemsave` of QEMU-Protocol to copy the VM's memory to the host non-volatile storage by referring to the mapping table. When the copy process is complete, the virtual machine is resumed.

The memory acquisition operation is delayed owing to the following three factors. The process of exploring the VM page is the first reason. To create the VM memory in the storage as a file, the physical memory region mapped to the allocated page needs to be explored. *Cloud-BlackBox* reduces the size of the memory that needs to be explored through the MD manager. The second reason for the delay is the page-read operation. To write the data stored in the memory to the storage, a read operation for the memory offset is required. However, because data are copied from memory to memory, there is no substantial performance delay. The third reason for the delay is the process of copying memory to storage. According to our experimental results, the file I/O operation accounts for more than 93% of the entire memory

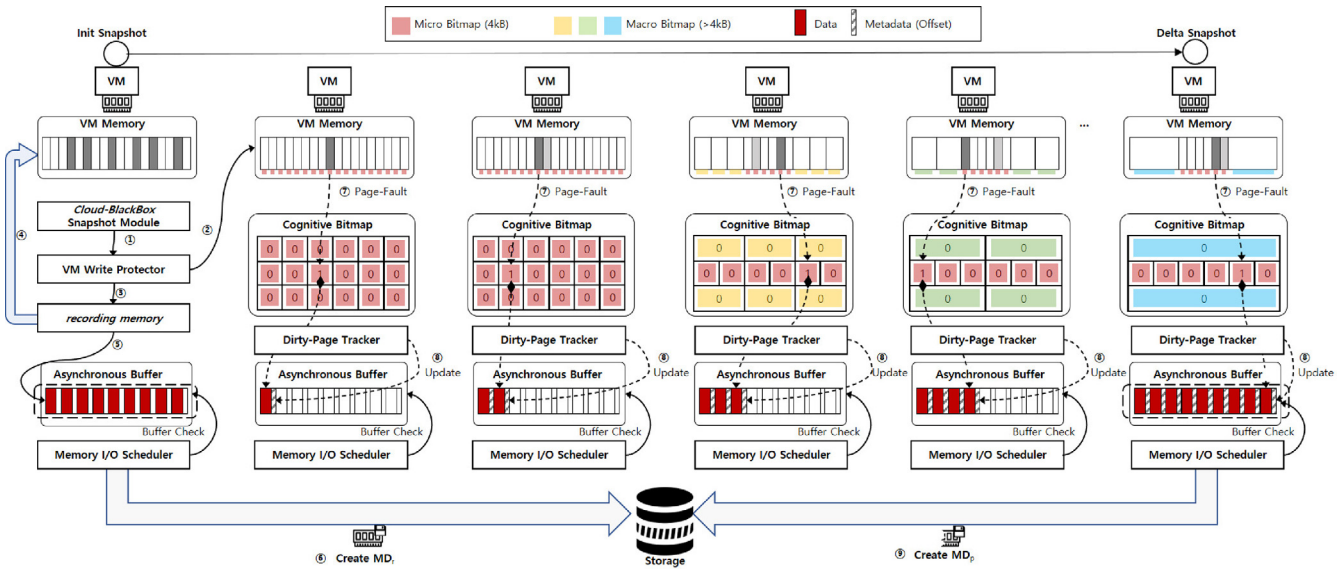

Fig. 5. Proposed process to efficiently track and record the VM memory.

acquisition process. *Cloud-BlackBox* resolves the file I/O bottleneck points through page tracking using a cognitive-scale bitmap and an in-memory-based asynchronous buffer. Additionally, the asynchronous buffer is designed to address the potential issue of concurrent memory operations in the process of tracking and recording VM memory. We designed the asynchronous buffer in the circular queue method to sequentially record memory change information, event processing time, and memory offset information to resolve the memory integrity problems that can arise from memory concurrency. Such a design can minimize bottlenecks by asynchronously handling dirty-page events taking place in parallel while addressing memory integrity problems that may occur during memory restoration by recording the event time/offset information together.

The memory recording process of *Cloud-BlackBox* is illustrated at point (b) in Fig. 4. In contrast to existing memory snapshot methods, the snapshot process is as follows.

When a VM snapshot request is received, the VM is stopped. When the VM is stopped, it calls the *Cloud-BlackBox* Snapshot module. The *Cloud-BlackBox* Snapshot module then calls the VM Write Protector. ① When a VM snapshot request is received, the VM is stopped. ② When the VM is stopped, it calls the *Cloud-BlackBox* snapshot module. ③ The *Cloud-BlackBox* Snapshot module then calls the VM Write Protector. ④ The VM Write Protector protects the `write` operation of the VM memory. ⑤ Referring to the mapping table, `memcpy` copies the VM's memory to the Asynchronous Buffer. ⑥ The VM is then resumed. ⑦ - ⑧ In the background, the asynchronous buffer is flushed to the host's non-volatile and the contents of the page written in the storage. ⑨–⑩ If a memory write operation is performed in the VM after the first snapshot, a Page Fault is triggered. ⑪ When a Page Fault occurs, the Dirty-Page Tracker marks the Cognitive Bitmap and `memcpy` the changed memory information in the Asynchronous Buffer. ⑫ According to user definition, the asynchronous buffer is flushed to the host's non-volatile storage in the background. A detailed look at the process follows below.

Cloud-BlackBox causes the VM suspension operation to run on the first occasion the VM operation starts. This is called an init dump. The init dump process includes a memory exploration process and a read memory process, as in the existing snapshot methods. However, it adds the target memory to be cloned for storage in an in-memory-based asynchronous buffer. This process serves as a means to resolve storage and memory bandwidth issues. In init dump, a VM suspend time only occurs during the memcpy operation. This design may reduce the VM suspend time but additionally requires a free memory space identical to the target VM memory size. Various live-migration techniques such as pre-copy can be considered as an alternative to address the problem of securing additional memory space for init dump and minimizing the VM suspend time. However, because these techniques are designed for a complete migration, snapshots can be delayed indefinitely or fail depending on the VM workload. Since *Cloud-BlackBox* cannot track memory until the snapshot of the frozen memory of the VM is complete, it is important to perform the snapshot of the frozen memory accurately and quickly without failure.

A macro bitmap filled with zeros is created once the init dump is complete. The macro bitmap manages VM pages in large groups and performs writing when a specific memory is altered. When the macro bitmap is created, operation of the VM is resumed. In the background thread, the memory inserted into the asynchronous buffer is created as a file during storage. We subsequently refer to the first memory snapshot file as MD_r. *Cloud-BlackBox* can produce the same result as performing continuous snapshots simply by tracking and recording only the memory that has changed since the init dump. The VM suspension does not occur from the second memory snapshot, and the memory snapshot operation is simplified. The page-fault handler we use to track the dirty-page of a VM is problematic in that it indirectly degrades the performance of the VM. We have conducted VM memory forensics studies over the past few years and found that the VM memory region exhibits strong spatial locality characteristics. Based on these characteristics, we designed a cognitive-scale bitmap that can efficiently deploy the page-fault handler according to the memory usage pattern of VMs. The cognitive-scale bitmap consists of a micro bitmap, which tracks VM memory from a microscopic perspective, and a macro bitmap, which tracks VM memory from a macroscopic perspective. The micro bitmap provides write protection at the page level to track memory change information of VMs in detail. The macro bitmap provides write protection by grouping more than N pages for regions with few memory changes. The macro bitmap does not exist at the initial time of tracking the memory of a VM, but is dynamically created as the memory usage pattern of the VM is reflected over time. As a result, the macro bitmap reflects the memory usage pattern of VMs, allowing the frequency of

page-fault events to be standardized and effectively reducing the number of page-fault handlers. The performance comparison between the bitmap generally used for dirty-page tracking in live migration and our proposed cognitive-scale bitmap can be found in Section 4.2.

The process of simplifying the snapshot operation using the cognitive-scale bitmap is as follows. Through the local alteration monitoring of the macro bitmap, a micro bitmap is created for regions with frequent changes, and regions with little change are grouped into macro bitmaps that can be managed on a large-scale. When a memory snapshot operation request is received from the user or another module, the write permission of the macro/micro bitmap region is removed. When a memory write request is received in the VM, a page fault occurs, and a dirty bit is marked on the macro/micro bitmap. After the duration specified by the user, the write protect permission is restored, and the contents of the page written in the macro/micro bitmap are inserted into the asynchronous buffer. In the background, the memory inserted into the asynchronous buffer is created as a file in the storage, along with metadata containing offset information. The reason why we store tracking time and offset information together when recording the memory is to ensure the integrity of memory data in the process of memory restoration. The file created through the macro/micro bitmap is called MD_p. The macro bitmap has the disadvantage that all pages of the corresponding region are stored even when only one page is updated. However, this disadvantage of large-scale page storage of the micro bitmap can be offset since it is optimized based on the memory usage pattern of the VM. Then, through iterative processes, the micro bitmap performs increasingly more detailed operations, whereas the macro bitmap performs operations for large areas, which leads to a gradual decrease in the number of bitmaps, thereby optimizing performance.

The detailed process for generating MD_ps is shown in Fig. 5. The MD_ps process is as follows. ①–⑤ Protects the write-protection operation of the virtual machine while performing Init Snapshot. ⑥ During the init snapshot process, MD_r is created by the Memory I/O Scheduler. ⑦–⑧ Whenever a Page-Fault is triggered, it is marked on the Cognitive Bitmap, and data and location information are updated in the asynchronous buffer. ⑨ When the MD_ps generation condition is satisfied, the asynchronous buffer is flushed to the host non-volatile by the memory I/O scheduler.

The detailed process is as follows. In the init dump process, the entire memory region of the VM is collected using a memory I/O scheduler. Then, the memory snapshot uses a macro bitmap and a micro bitmap to generate MD_ps. The macro bitmap is a space for simple writing by introspection of the memory change information in a large range, and VM snapshot performance is hardly compromised owing to the write protection in this case. The micro bitmap writes one bitmap by dividing it into page units. With a micro bitmap, write protection is required for detailed memory tracking, which may result in some degradation of performance. However, this type of bitmap has a clear advantage in that the precise memory state at the time of the snapshot request can be preserved. The generation of MD_p is possible by overlaying the micro bitmap onto the macro bitmap. When the macro bitmap tracks pages on a large-scale and the micro bitmap through which fine tracking of memory is possible in the unit of pages are overlaid, the state of the VM at the time of the snapshot request can be written as a complete bitmap. The final bitmap created through the overlay process is processed by the asynchronous buffer using a memory I/O scheduler to create a file in the storage. The advantages of *Cloud-BlackBox* are as follows. (1) Almost no suspension in the running of VMs occurs apart from the first generation of MD_r and (2) storage costs are minimized. Write-Protect events that are generated repeatedly during memory snapshots cause compromised VM performance, The *Cloud-BlackBox* proposed in this study minimizes the call of the write-protect event by using a cognitive-scale bitmap. The process of generating MD_r and MD_p is outlined in Algorithm 1.

Algorithm 1 Memory tracking mechanism for memory recording Algorithm

Input: Request to Memory Snapshot
VM Suspend ← *False*
Init_Flag ← *False*
if *Init_Flag* = *False* **then**
  VM Suspend ← *True*
  Create a *Init Dump* (MD_r)
  VM Suspend ← *False*
  Macro Bitmap ← *Clear*
  Init_Flag ← *True*
  Create a *Macro Bitmap*
end if
while *Macro Bitmap Monitoring* **do**
  Create a *Micro Bitmap*
end while
for *iteration* = 1, 2, . . . **do**
  if *Snapshot* **then**
    Micro Bitmap ← *Clear*
    Macro Bitmap , *Micro Bitmap* ← *Page Write Protect* : *Enable*
  end if
  if *ΔPage* **then**
    Macro Bitmap , *Micro Bitmap* ← *Dirty bit Mark*
  end if
  if *Process Time* > *Duration Time* **then**
    for *MacroBitmap* . . . , *L* **do**
      Asynchronous Buffer ← *Page*[*L*], *Metadata*[*L*]
    end for
    for *MicroBitmap* . . . , *T* **do**
      Asynchronous Buffer ← *Page*[*T*], *Metadata*[*T*]
    end for
    Macro Bitmap , *Micro Bitmap* ← *Page Write Protect* : *Disable*
    Macro, *Micro Bitmap* ← *Clear*
    Create a MD_p
  end if
end for
Output: $MD_r \| MD_p$

Algorithm 2 Snapshot Recovery using Checkpoint Algorithm

Input: Select *N*
MD_c ← *Nearest Completed Snapshot Before N*
MD_r ← *Snapshot in Complete State*
MD_p ← *Snapshot in Incomplete Stat*
Metadata ← *Offset of Snapshot Data*
S ← *N After* MD_c
for $MD_p = 1, 2, \ldots, N$ **do**
  if *N* = *S* **then**
    $MD_r \leftarrow MD_c$
  end if
  if *S* < *N* **then**
    for all *Metadata in Offset* **do**
      $MD_r \leftarrow MD_c[Offset]\ \delta\ MD_p[Offset]$
    end for
  end if
end for
Output: *New* MD_r

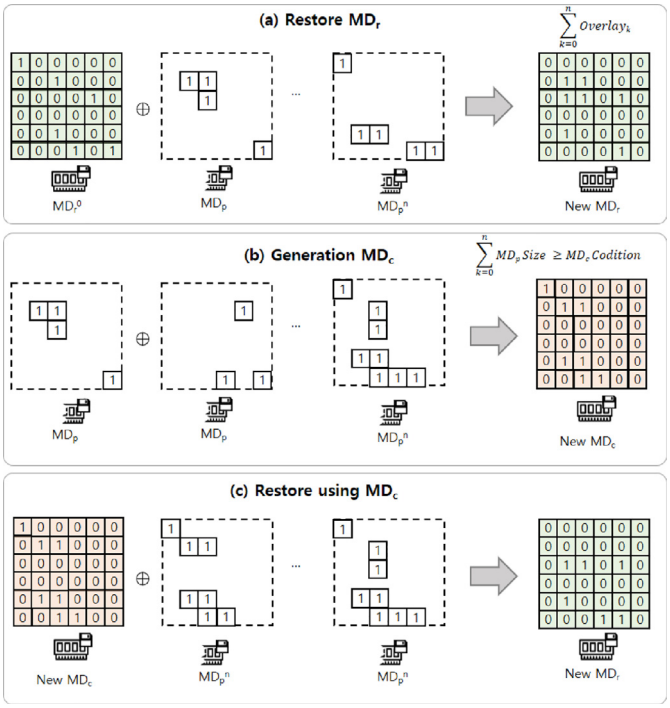

Fig. 6. Checkpoints to ensure memory resiliency.

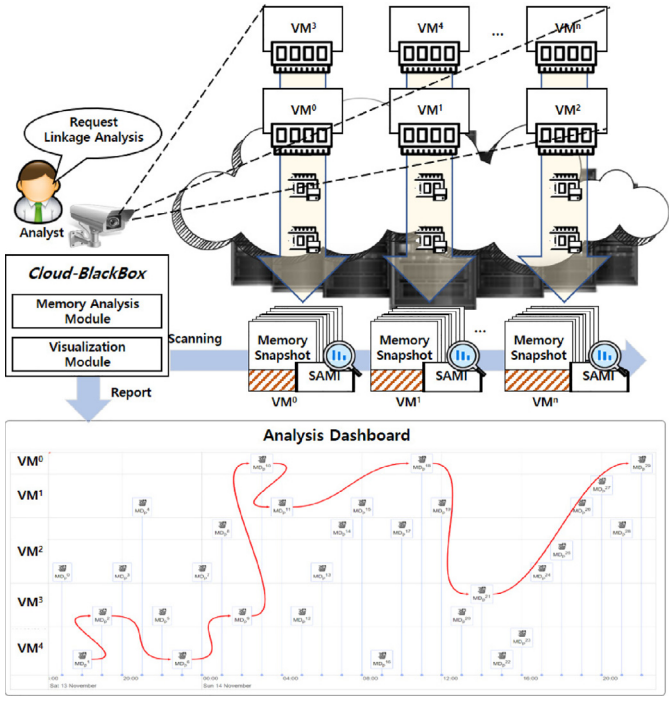

Fig. 7. SAMI metadata for efficient large-scale memory inspection.

3.4. Memory management mechanism to optimize random access restoration performance and storage cost

The memory recorded using *Cloud-BlackBox* consists of one MD_r and multiple MD_ps. For this type of structure, a memory restoration process is required for the analysis of the VM memory at a specific point in time. The memory restoration process is illustrated in Fig. 6. The memory restoration process is performed through the XOR operation of the initially created MD_r and cumulative MD_p. However, with the accumulation of MD_ps, more time is required for memory restoration. That is, the performance of random access restore operations cannot be ensured. To address the problem of ensuring random access memory restoration performance, *Cloud-BlackBox* is designed to create a checkpoint file called MD_c, which is created when the cumulative size of the accumulated MD_ps exceeds the size designated by the user. The checkpoint file MD_c is generated by the XOR operation of the previous MD_r or MD_ps created after MD_c. With MD_c, random access performance can be ensured during memory restoration. MD_c generation is controlled by the MD manager. Because the MD manager is allocated independent resources through cgroups and runs in the background, it does not lead to compromised VM and memory snapshot performance. The details of an experiment conducted on memory restoration performance according to the MD_c generation rule are described in the performance evaluation section. The process of generating MD_r and MD_c is outlined in Algorithm 2.

3.5. Memory structure specialized for large-scale memory inspection

Cloud-BlackBox creates a metadata called synchronized accessible memory interchange, as shown in Fig. 7, for the efficient analysis of large-scale memory. The offset information we record when tracking memory provides analysts with a macroscopic perspective. The memory analyst may narrow the scope of the memory analysis subject by analyzing metadata without a memory restoration procedure. Simple application examples of SAMI are as follows. SAMI can be used in memory inspection frameworks such as Volatility and Rekall, and in generating VM memory change information as a timeline. For the analysis of a large memory sets in parallel, the first offset capture needs to be performed once. The offset capture for plug-ins was performed using MD_r data analysis. For memory inspection plug-ins, pslists, dlllists, and sockets supported by Volatility and Rekall can be used, or customized plug-ins can be created for use. Based on the plug-in information entered in the memory inspection plugin table, the memory offset for each VM is captured. The captured offset information can be mapped by checking the synchronized accessible memory interchange data generated when the MD_p is created. For example, for fast analysis of the pslist change information, the offset information recorded in the SAMI can be used to analyze the MD_p that has accessed or modified the first offset. *Cloud-BlackBox* provides MD analysis results through a dashboard.

4. Evaluation

In this section, the memory acquisition performance of the proposed approach is evaluated. The evaluation of *Cloud-BlackBox* was conducted through five experiments using different types of workloads. The workload types are presented in Table 1.

Table 1
Workload types utilized in the experiments.

Workload	Description
Native	State of no activity
Apache	Web process-related workload
Flexible I/O	I/O-focused workload
Weka	Machine learning of data through Random Forest; iris

The first performance evaluation comparatively measured the dump time for memory recording through the conventional memory snapshot method and using *Cloud-BlackBox*. In the second experiment, the VM performance degradation caused by the

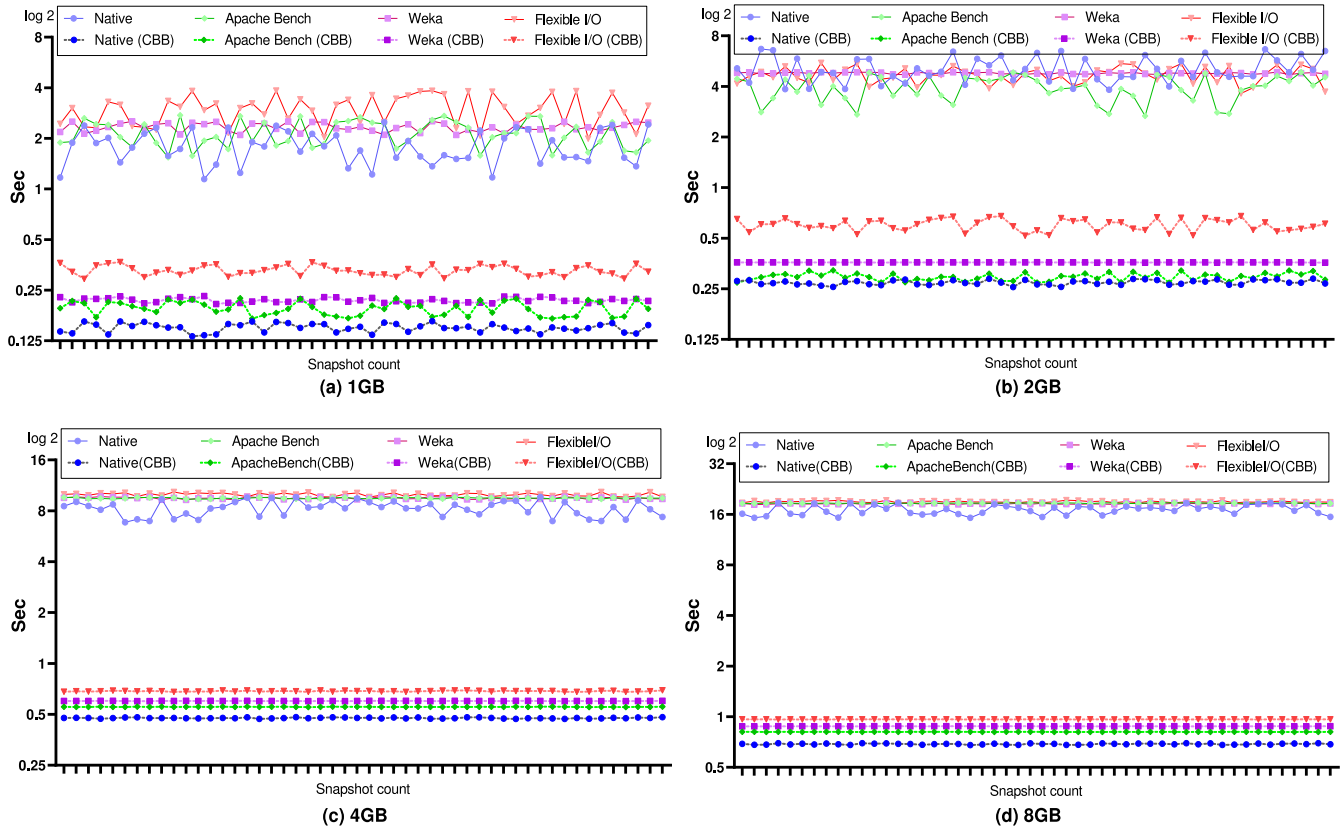

Fig. 8. Comparison of dump time by workload and VM memory size.

memory snapshot was measured and compared in terms of throughput. The third performance evaluation compared the event calls made through *Cloud-BlackBox* with calls by the conventional memory snapshot method. The fourth performance evaluation compared the size of the memory snapshot file generated through *Cloud-BlackBox* with that generated by the conventional memory snapshot method. The final performance evaluation measured the memory random access restoration time through memory checkpoints.

The experimental environment of this study was as follows. A system with an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20 GHz – 20 Core processor was used, with 384 MB of DDR RAM. One TB of SSD was used for storage. The Ubuntu 18.04 OS with kernel version 5.3.0–59 was used as the host, and Ver. 4.2.0 of QEMU-KVM was used.

4.1. Memory snapshot performance comparison by workload

The use of *Cloud-BlackBox* reduced the computational overhead and storage overhead generated during memory snapshots through the GSM and cognitive-scale bitmap. In this section, the dump time of the general memory snapshot operation according to the QEMU machine protocol and dump time of the memory snapshot operation using the proposed *Cloud-BlackBox* are comparatively evaluated. When the VM memory size increased from 1 GB to 8 GB, the time required from the point of requesting the memory snapshot to the completion of the snapshot while processing a workload was measured. The experimental results are shown in Fig. 8. The *Cloud-BlackBox* method proposed in this study requires an init dump operation, so the memory snapshot dump time for the initial operation is the same as the time taken with the conventional method. However, from the subsequent snapshot operation, the memory snapshot dump time decreased sharply. For example, when performing a Weka workload in a VM environment with a memory size of 2 GB, a memory snapshot with a conventional method took about 5.3 s on average, whereas approximately 0.61 s was required by *Cloud-BlackBox*. As a result, *Cloud-BlackBox* showed a snapshot performance improvement with a factor of approximately 8.83 compared to the time required by the conventional memory snapshot method in a 2 GB VM environment. According to the experimental results, as the size of the memory allocated to the VM increased, the memory snapshot performance of *Cloud-BlackBox* was maximized. According to the experimental results, the average snapshot performance was improved by approximately 14.85 times in terms of dump time in the 4 GB environment.

4.2. Measurement of VM performance overhead due to memory recording

In this section, we describe the experiment conducted regarding the performance degradation caused by VM memory snapshots. *Cloud-BlackBox* minimizes performance degradation during the memory snapshot process using the Cognitive-Scale Bitmap. This experiment compares the performance of the conventional bitmap used in the live-migration mechanism to that of the cognitive-scale bitmap of *Cloud-BlackBox*. Sysbench was used to the performance degradation in terms of CPU, memory, and storage. The experimental results are shown in Fig. 9. First, the performance in terms of the number of events that the CPU could process per second showed a throughput decrease of 19.82% when the bitmap method proposed by the conventional memory snapshot acceleration studies was used; when the cognitive-scale bitmap proposed in this study was used, the throughput decreased by 12.79%. With regard to performance degradation in terms of memory, with the conventional bitmap, the performance

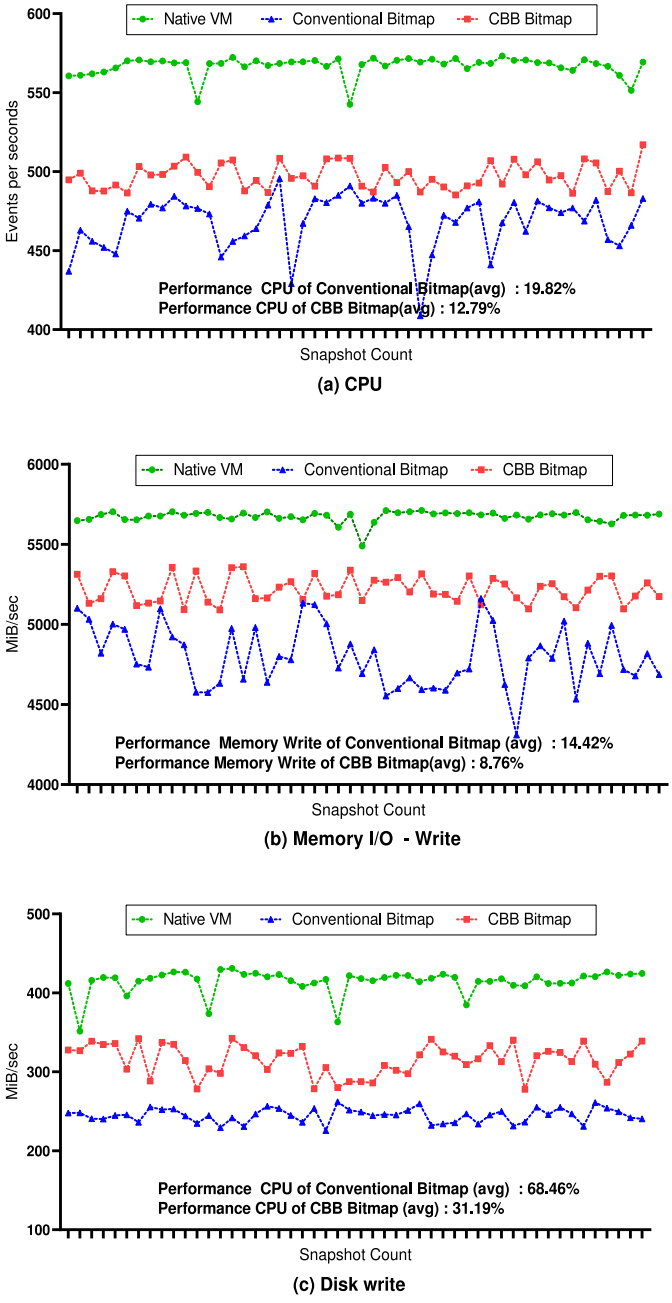

Fig. 9. Measuring VM performance degradation according to page fault handling method.

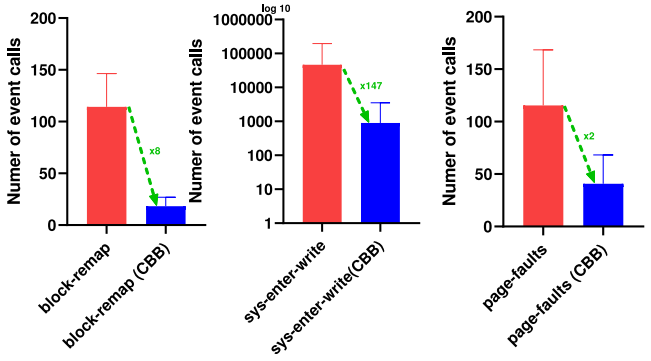

Fig. 10. Comparison of host operating system event calls according to memory snapshot method.

decreased by approximately 14.42%, and with the cognitive-scale bitmap, the decrease was 8.76%. Finally, for performance degradation in terms of storage, memory snapshots with the conventional bitmap showed a decrease of approximately 68.46%, and with the proposed *Cloud-BlackBox*, a decrease of 31.19% was measured. In summary, by using the *Cloud-BlackBox* method proposed in this study, it was confirmed that the decrease in VM performance was markedly reduced compared to the method proposed by previous studies on memory snapshot acceleration.

4.3. Measurement of VM event calls due to memory recording

In this section, information on the number of calls of major events accompanying a VM memory snapshot is presented. This experiment proves that the proposed *Cloud-BlackBox* approach is able to minimize major event calls with associated processing overhead. This evaluation compares the event call count information of the snapshot method using a conventional bitmap and the method of dumping after memory merge through the GSM. In this experiment, perf was used to measure the number of event calls [56]. In our previous research results, we identified disk I/O-related functions as the main cause of performance overhead. Therefore, we selected `block:block_remap`,`syscalls:sys_enter_write`, and `page_faults` which are most related to I/O, as trace point events. In the experiment, a virtual machine performing no specific action was operated and measured. The experimental results are presented in Fig. 10.

First, the `block:block_remap` event calls were reduced by approximately a factor of six. The `block:block_remap` event is a technique designed to prevent redundant data writing and was called when metadata changes occurred. First, when a snapshot is taken using a conventional bitmap while a single virtual machine is running, the block_remap event is called an average of 114 times. On the other hand, when the memory is merged through the GSM, the `block_remap` event is called an average of 18.1 times. In other words, the experimental results show that the proposed GSM is able to reduce the number of `block_remap` event calls by six times.

Second, syscalls: `sys_enter_write` event calls were reduced by approximately a factor of 59. The syscalls: `sys_enter_write` event fires when a file write operation is performed. In this experimental environment, the syscalls: `sys_enter_write` event is called once, and a 4kB file is created. First, for a regular snapshot, it is called 524k times because it creates the VMs total allocated memory of 2 GB to a file. In our proposed system, it is called 8.7k times on average. This is a reduction if approximately 147 times. The reason for this significant reduction is that memory consolidation reduces the amount of memory that needs to be taken from the snapshot, and the amount of change in memory that changes as the snapshot was created for the underlying VM is small. Therefore, in an operating environment where memory changes frequently, events of 8.7k and above are called.

Third, the page fault event calls are reduced by approximately two times. A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory or RAM. First, the average of 96 page fault events was called for each snapshot in the usual way. An average of 47 event calls occurred when memory was recorded using *Cloud-BlackBox*. The frequency of occurrence of `page_faults` events may vary depending on the memory technique of the host operating system. In our experimental environment, event calls were made up to 278 times in the case of a typical memory snapshot, but reduced to 35 times owing to the host operating system memory

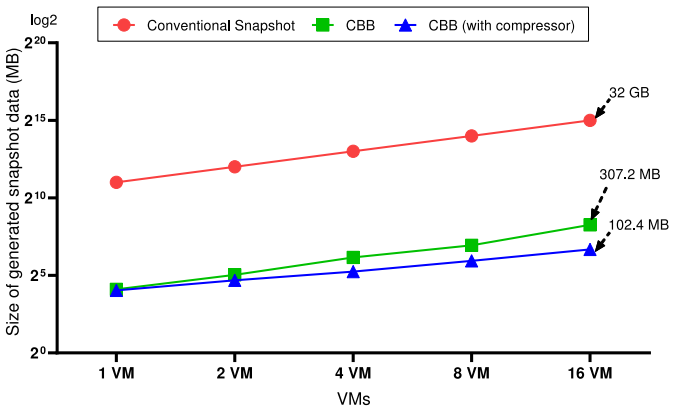

Fig. 11. Memory dump file size according to the number of parallel VM operations.

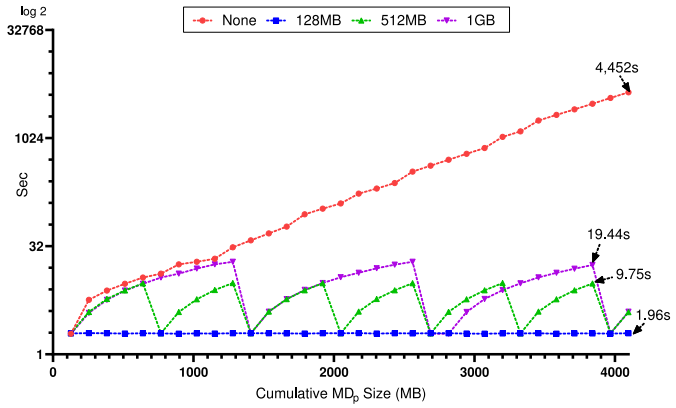

Fig. 12. Comparison of memory restore times based on MD_c generation conditions.

technique. In the case of *Cloud-BlackBox*, event calls occurred up to 146 times, but there were cases where the `page_faults` event was never called. As a result, the experimental results demonstrate that the proposed *Cloud-BlackBox* is able to reduce the `page_faults` events occurring in the host operating system owing to the memory snapshot.

4.4. Measurement of cost of storage consumed by memory recording

In this experiment, comparative measurements were taken of the storage space saved through the MD manager and memory tracking method of the proposed *Cloud-BlackBox*. In the experiment, 2 GB of memory was allocated to the native VM, Ubuntu 18.04 was installed in parallel, and the size of storage used when a memory snapshot was taken was measured. The experimental results are shown in Fig. 11. As can be seen, when memory was acquired in parallel in a cloud platform environment where 16 VMs were running, a memory snapshot file of 32 GB was created for a conventional memory snapshot, whereas a snapshot file of size 2.49 GB was created using *Cloud-BlackBox*. When compression was further activated, a file of approximately 1.75 GB in size was created. That is, we experimentally verified that when VM memory snapshots are taken using *Cloud-BlackBox*, a storage-saving effect over 12.85 times is achieved in an environment with 16 VMs running in parallel.

4.5. Memory restore random access performance evaluation

For the proposed *Cloud-BlackBox*, a memory restoration process is required for the acquisition of complete memory states. The memory restoration process is performed through the XOR operation of the MD_r and MD_p. In this process, the restoration time gradually increases with cumulative memory, and to ensure the restoration performance of memory of random selection, a checkpoint called MD_c was designed. Fig. 12 shows the experimental results in terms of memory restoration performance. As shown in the first figure, without MD_c, the restoration time gradually increased according to the accumulated memory size; when the MD_p was accumulated to 4000 MB, it took approximately 4500 s to perform memory restoration. However, with the creation of MD_c, when MD_p was accumulated over 128 MB, it took up to 2.0 s to perform the memory restoration, 9.59 s for a cumulative MD_p size over 512 MB, and 19.86 s maximum for 1024 MB. These results demonstrate experimentally that the checkpoint designed in this study ensures the minimum random access performance for memory restoration.

5. Conclusion

This study proposed *Cloud-BlackBox*, which enables the efficient recording of active memory without loss in a real-world cloud operating service environment. The proposed *Cloud- BlackBox* method has the following features. First, *Cloud-BlackBox* effectively reduces the storage space by more than a factor of 12.85 through kernel memory merging as shared memory by analyzing memory similarities of component VMs in a VM swarm. Second, *Cloud-BlackBox* reduces the memory acquisition time by a factor of up to 14.85 compared to existing memory snapshot technologies by using a cognitive-scale bitmap and memory I/O scheduler. In addition, the problem of ensuring random access performance in the delta dump method, which was designed to reduce storage costs, is addressed by introducing an MD manager. The experimental results demonstrated that a minimized random access performance is achieved by using the MDc memory restoration technique. The last feature proposed is the SAMI mechanism that analyzes large-scale recorded memory in parallel. The SAMI mechanism utilizes data created through memory tracking for memory change information on processes, libraries, and network traffic according to temporal characteristics, and is designed to perform a correlation analysis on VM swarms using to spatial characteristics.

Finally, the proposed approach was implemented on a real cloud computing service called *SysCore-Cloud*. The experimental results show that *Cloud-BlackBox* has wide applicability not only in advanced malware analysis, but also in various fields of system security, such as antivirus and forensic analysis. The mechanisms proposed in this study are limited to cloud environments using virtualization technology. Our future research goal is to enable *Cloud-BlackBox* application in multi-cloud environments, including edge clouds and federated clouds. Also, we plan to investigate self-recovery measures to address the case of intrusion and service errors in cloud environments based on the memory recording method proposed in this study.

CRediT authorship contribution statement

Sang-Hoon Choi: Conceptualization, Methodology/Study design, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Ki-Woong Park:** Conceptualization, Methodology/Study design, Validation, Formal analysis, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) (Project No. RS-2022-00165794, Development of a Multi-Faceted Collection-Analysis-Response Platform for Proactive Response to Ransomware Incidents, 30%, and Project No. 2019-0-00426, 10%), the ICT R&D Program of MSIT/IITP (Project No. 2021-0-01816, A Research on Core Technology of Autonomous Twins for Metaverse, 10%), and a National Research Foundation of Korea (NRF) grant funded by the Korean government (Project No. NRF-2020R1A2C4002737, 50%).

References

[1] N. Gruschka, M. Jensen, Attack surfaces: A taxonomy for attacks on cloud services, in: 2010 IEEE 3rd International Conference on Cloud Computing, IEEE, 2010, pp. 276–279.
[2] A. Singh, M. Shrivastava, Overview of attacks on cloud computing, Int. J. Eng. Innov. Technol. 1 (4) (2012) 321–323.
[3] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, R.B. Lee, Catalyst: Defeating last-level cache side channel attacks in cloud computing, in: 2016 IEEE International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2016, pp. 406–418.
[4] Q. Yan, F.R. Yu, Q. Gong, J. Li, Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges, IEEE Commun. Surv. Tutor. 18 (1) (2015) 602–622.
[5] B. Joshi, A.S. Vijayan, B.K. Joshi, Securing cloud computing environment against DDoS attacks, in: 2012 International Conference on Computer Communication and Informatics, IEEE, 2012, pp. 1–5.
[6] M.A. Bamiah, S.N. Brohi, Seven deadly threats and vulnerabilities in cloud computing, Int. J. Adv. Eng. Sci. Technol. 9 (1) (2011) 87–90.
[7] S. Zawoad, R. Hasan, Cloud forensics: A meta-study of challenges, approaches, and open problems, 2013, arXiv preprint arXiv:1302.6312.
[8] S. Simou, C. Kalloniatis, E. Kavakli, S. Gritzalis, Cloud forensics: Identifying the major issues and challenges, in: International Conference on Advanced Information Systems Engineering, Springer, 2014, pp. 271–284.
[9] D. Birk, C. Wegener, Technical issues of forensic investigations in cloud computing environments, in: 2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, IEEE, 2011, pp. 1–10.
[10] Y. Cheng, X. Fu, X. Du, B. Luo, M. Guizani, A lightweight live memory forensic approach based on hardware virtualization, Inform. Sci. 379 (2017) 23–41.
[11] S.T. Jones, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, VMM-based hidden process detection and identification using Lycosid, in: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2008, pp. 91–100.
[12] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: Malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 51–62.
[13] M.I. Sharif, W. Lee, W. Cui, A. Lanzi, Secure in-vm monitoring using hardware virtualization, in: Proceedings of the 16th ACM Conference on Computer and Communications Security, 2009, pp. 477–487.
[14] M. Graziano, A. Lanzi, D. Balzarotti, Hypervisor memory forensics, in: International Workshop on Recent Advances in Intrusion Detection, Springer, 2013, pp. 21–40.
[15] T.K. Lengyel, S. Maresca, B.D. Payne, G.D. Webster, S. Vogl, A. Kiayias, Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system, in: Proceedings of the 30th Annual Computer Security Applications Conference, 2014, pp. 386–395.
[16] A. Dahan, Operation Cobalt Kitty: A Large-Scale APT in Asia Carried Out by the Oceanlotus Group, Cyber Reason, 2017.
[17] T.P. Setia, N. Widiyasono, A.P. Aldya, Analysis malware flawed ammyy RAT dengan metode reverse engineering, J. Inform. 3 (03) (2018).
[18] C. Albanesius, Ramnit computer worm compromises 45K Facebook logins, PC Mag. 2817 (2398432) (2012) 00, Source: http://www.pcmag.com/article2/0.
[19] C. Patsakis, A. Chrysanthou, Analysing the fall 2020 Emotet campaign, 2020, arXiv preprint arXiv:2011.06479.
[20] E. Ramos, Analysis: Ursnif-spying on your data since 2007, 2016.
[21] C. Kaufman, R. Venkatapathy, Windows azure™ security overview, 2010, Published Aug 24.
[22] N.R. Mistry, M.S. Dahiya, Signature based volatile memory forensics: A detection based approach for analyzing sophisticated cyber attacks, Int. J. Inf. Technol. 11 (3) (2019) 583–589.
[23] A.S. Bozkir, E. Tahillioglu, M. Aydos, I. Kara, Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision, Comput. Secur. 103 (2021) 102166.
[24] P. Mishra, P. Aggarwal, A. Vidyarthi, P. Singh, B. Khan, H.H. Alhelou, P. Siano, VMShield: Memory introspection-based malware detection to secure cloud-based services against stealthy attacks, IEEE Trans. Ind. Inf. 17 (10) (2021) 6754–6764.
[25] S. Gaur, R. Chhikara, Memory forensics: Tools and techniques, Indian J. Sci. Technol. 9 (48) (2016) 1–12.
[26] A. Case, G.G. Richard III, Memory forensics: The path forward, Digit. Investig. 20 (2017) 23–33.
[27] F. Pagani, D. Balzarotti, Back to the whiteboard: A principled approach for the assessment and design of memory forensic techniques, in: 28th USENIX Security Symposium, USENIX Security 19, 2019, pp. 1751–1768.
[28] F. Pagani, O. Fedorov, D. Balzarotti, Introducing the temporal dimension to memory forensics, ACM Trans. Priv. Secur. 22 (2) (2019) 1–21.
[29] Cloud market, http://thecloudmarket.com/stats/.
[30] A. Walters, Volatility, 2006, https://www.volatilityfoundation.org/.
[31] T. Garfinkel, M. Rosenblum, et al., A virtual machine introspection based architecture for intrusion detection, in: Ndss, vol. 3, Citeseer, 2003, pp. 191–206.
[32] H. Xiong, Z. Liu, W. Xu, S. Jiao, Libvmi: A library for bridging the semantic gap between guest OS and VMM, in: 2012 IEEE 12th International Conference on Computer and Information Technology, IEEE, 2012, pp. 549–556.
[33] I. Ahmed, A. Zoranic, S. Javaid, G.G. Richard III, Modchecker: Kernel module integrity checking in the cloud environment, in: 2012 41st International Conference on Parallel Processing Workshops, IEEE, 2012, pp. 306–313.
[34] T.Y. Win, H. Tianfield, Q. Mair, Detection of malware and kernel-level rootkits in cloud computing environments, in: 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, IEEE, 2015, pp. 295–300.
[35] P. Mishra, V. Varadharajan, E.S. Pilli, U. Tupakula, VMGuard: A VMI-based security architecture for intrusion detection in cloud environment, IEEE Trans. Cloud Comput. 8 (3) (2018) 957–971.
[36] B. Martini, K.-K.R. Choo, Cloud storage forensics: OwnCloud as a case study, Digit. Investig. 10 (4) (2013) 287–299.
[37] S. Khan, A. Gani, A.W.A. Wahab, M.A. Bagiwa, M. Shiraz, S.U. Khan, R. Buyya, A.Y. Zomaya, Cloud log forensics: Foundations, state of the art, and future directions, ACM Comput. Surv. 49 (1) (2016) 1–42.
[38] A. Cohen, N. Nissim, Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory, Expert Syst. Appl. 102 (2018) 158–178.
[39] S.A. Ali, S. Memon, F. Sahito, Challenges and solutions in cloud forensics, in: Proceedings of the 2018 2nd International Conference on Cloud and Big Data Computing, 2018, pp. 6–10.
[40] D. Paul Joseph, J. Norman, A review and analysis of ransomware using memory forensics and its tools, Smart Intell. Comput. Appl. (2020) 505–514.
[41] W. Song, H. Yin, C. Liu, D. Song, DeepMem: Learning graph neural network models for fast and robust memory forensic analysis, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 606–618.
[42] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, vol. 2, 2005, pp. 273–286.
[43] M.R. Hines, U. Deshpande, K. Gopalan, Post-copy live migration of virtual machines, Oper. Syst. Rev. 43 (3) (2009) 14–26.
[44] E. Park, B. Egger, J. Lee, Fast and space-efficient virtual machine checkpointing, ACM SIGPLAN Not. 46 (7) (2011) 75–86.
[45] S. Akiyama, T. Hirofuchi, R. Takano, S. Honiden, Fast wide area live migration with a low overhead through page cache teleportation, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE, 2013, pp. 78–82.
[46] L. Wang, Z. Kalbarczyk, R.K. Iyer, A. Iyengar, VM-μcheckpoint: Design, modeling, and assessment of lightweight in-memory VM checkpointing, IEEE Trans. Dependable Secure Comput. 12 (2) (2014) 243–255.
[47] P.F. Klemperer, H.Y. Jeon, B.D. Payne, J.C. Hoe, High-performance memory snapshotting for real-time, consistent, hypervisor-based monitors, IEEE Trans. Dependable Secure Comput. 17 (3) (2018) 518–535.

[48] Z. Hao, W. Wang, L. Cui, X. Yun, Z. Ding, iConSnap: An incremental continuous snapshots system for virtual machines, IEEE Trans. Serv. Comput. (2019).
[49] J. Watson, Virtualbox: Bits and bytes masquerading as machines, Linux J. 2008 (166) (2008) 1.
[50] E. VMware, Timekeeping in VMware virtual machines, 2008.
[51] A.J. Mashtizadeh, E. Celebi, T. Garfinkel, M. Cai, et al., The design and evolution of live storage migration in VMware ESX, in: USENIX Annual Technical Conference, 2011, pp. 187–200.
[52] A. Arcangeli, I. Eidus, C. Wright, Increasing memory density by using KSM, in: Proceedings of the Linux Symposium, Citeseer, 2009, pp. 19–28.
[53] S. Rachamalla, D. Mishra, P. Kulkarni, Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems, in: 20th Annual International Conference on High Performance Computing, IEEE, 2013, pp. 59–68.
[54] C.-R. Chang, J.-J. Wu, P. Liu, An empirical study on memory sharing of virtual machines for server consolidation, in: 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, IEEE, 2011, pp. 244–249.
[55] S. Barker, T. Wood, P. Shenoy, R. Sitaraman, An empirical study of memory sharing in virtual machines, in: 2012 USENIX Annual Technical Conference, USENIX ATC 12, 2012, pp. 273–284.
[56] A.C. De Melo, The new linux 'perf' tools, in: Slides from Linux Kongress, vol. 18, 2010, pp. 1–42.

Sang-Hoon Choi received the B.S. and M.S degree in computer and information security from Daejeon University, South Korea, in 2016. He is currently working toward the Ph.D. degree in the Department of Computer and Information Security, University of Sejong, South Korea. His research interests include cloud computing, virtualization, system memory and machine learning.

Ki-Woong Park received the B.S. degree in computer science from Yonsei University, South Korea, in 2005, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007, and the Ph.D. degree in electrical engineering from KAIST in 2012. He received a 2009–2010 Microsoft Graduate Research Fellowship. He worked for National Security Research Institute as a senior researcher. He has been a professor in the department of computer and information security at Sejong University. His research interests include security issues for cloud and mobile computing systems as well as the actual system implementation and subsequent evaluation in a real computing system.