Computer Systems Science & Engineering
DOI: 10.32604/csse.2023.024945
Article

Homogeneous Batch Memory Deduplication Using Clustering of Virtual Machines

N. Jagadeeswari[1,*] and V. Mohan Raj[2]

[1]Department of CSE, Thanthai Periyar Government Institute of Technology, Vellore, 632002, India
[2]Department of IT, Sona College of Technology, Salem, 636005, India
*Corresponding Author: N. Jagadeeswari. Email: jagadeeswarinj@gmail.com
Received: 05 November 2021; Accepted: 24 January 2022

Abstract: Virtualization is the backbone of cloud computing, which is a developing and widely used paradigm. By finding and merging identical memory pages, memory deduplication improves memory efficiency in virtualized systems. Kernel Same Page Merging (KSM) is a Linux service for memory pages sharing in virtualized environments. Memory deduplication is vulnerable to a memory disclosure attack, which uses covert channel establishment to reveal the contents of other colocated virtual machines. To avoid a memory disclosure attack, sharing of identical pages within a single user's virtual machine is permitted, but sharing of contents between different users is forbidden. In our proposed approach, virtual machines with similar operating systems of active domains in a node are recognised and organised into a homogenous batch, with memory deduplication performed inside that batch, to improve the memory pages sharing efficiency. When compared to memory deduplication applied to the entire host, implementation details demonstrate a significant increase in the number of pages shared when memory deduplication applied batch-wise and CPU (Central processing unit) consumption also increased.

Keywords: Kernel same page merging; memory deduplication; virtual machine sharing; content-based sharing

1 Introduction

Cloud computing is becoming increasingly popular in a variety of sectors. Virtual machines (VMs) have resurfaced, presenting a huge opportunity for cluster, parallel, cloud, grid, and distributed computing. Virtualization technology serves the majority of IT (Information Technology) and computer-related sectors by allowing users to share expensive hardware resources by executing VMs on the same set of server hosts. Virtualization is a computer architecture concept that allows the execution of many virtual machines (VMs) on the same host machine. The concept of virtual reality dates back to the 1960s. The goal of a virtual machine (VM) is to facilitate resource sharing among multiple users while also increasing computer efficiency and performance in terms of resource consumption and application flexibility. In various layers of cloud architecture, hardware resources such as Central Processing Unit, memory, and Input -Output devices, as well as software resources such as operating systems and software libraries, can be virtualized. Cloud computing is increasingly based on a virtualized architecture, in which massive programmes run inside virtual servers that are assigned to each physical server. Virtual machines are installed on top of virtual machine monitors, which are in charge of allocating physical resources such as CPU, memory, and other resources to each virtual machine separately. The number of pages shared is high if the guest operating system of virtual machines contains similar applications of data. If these sharing options are appropriately exploited, the virtual machine monitor can supply significantly more memory to the virtual machines, resulting in a higher level of server consolidation [1].

Memory deduplication was established primarily in Disco [2] using the transparent page sharing methodology, which involved recognising identical copies of memory pages, merging them, and mapping them to the same physical page in order to decrease the memory footprint of redundant data. However, this para-virtualization paradigm necessitates some changes to both the guest and the host. Due to their licence agreements and source code changes, it is impossible to modify guests. The VMware ESX (Elastic Sky X) server employs content-based page sharing, which determines whether a page is available in many virtual machines by calculating the hash value. According to research, this method can help to reclaim 10% to 20% of system RAM [3]. The Xen virtual machine monitor now enables page compression and page patching, in addition to memory deduplication [4]. However, in all circumstances, the time to merge, or the time it takes to merge identical copies of memory, is often long. The scan rate is set to slow, but it may be tweaked.

Despite the fact that memory deduplication reduces memory footprints, existing techniques lack isolation and trustworthiness in addition to their efficiency [5]. Some tenants may be reluctant to share sensitive or confidential information with others. Covert Channel [6] is a system that uses CPU loads to provide a illegal channel for transmitting confidential information between virtual machines. Furthermore, the virtual machine monitor placed on the host performs a global memory deduplication procedure. It is not possible to set the appropriate sharing rate based on the virtual machine's workload [7].

The rest of the paper is organized as below. In Section 2, background and motivation of this research is presented and Section 3 contains Literature review, In Section 4, the approach and implementation has been illustrated. Experimental results are discussed in Section 5. The paper is concluded in Section 6.

2 Background and Motivation

The background and motivation of this research is given below.

2.1 Types of Sharing of Memory Pages

Intra-virtual machine sharing, inter-virtual machine sharing, homogeneous sharing, and heterogeneous sharing are the several types of memory page sharing. Intra-Virtual machine sharing refers to the sharing of identical memory pages within the same virtual machine. Inter-Virtual machine sharing refers to the sharing of identical pages between multiple distinct virtual machines. Homogeneous virtual machine sharing refers to the sharing of memory pages across identical operating systems. Heterogeneous sharing refers to the sharing of virtual machines across multiple operating system platforms [8]. Kernel Same Page (KSM) merging shared pages on Windows VMs is more effective than on Linux VMs.

2.2 Kernel Same Page Merging

Kernel Same Page Merging is a memory deduplication implementation that originally appeared in the Linux Kernel version 2.6.32. Kernel daemon, ksmd, searches the user memory for pages that can be shared among users. It scans only the prospective candidates and develops signatures for them, rather than scanning the full region of memory, which is time consuming and CPU intensive. These signatures are kept in the deduplication table. When two or more pages are verified to determine if they are in the same signatures, KSM scans at a 20 millisecond interval and at a rate of 25% of possible memory pages at a time. KSM looks at three different sorts of memory pages. 1. Volatile pages, or pages that change frequently and are therefore unsuitable for memory sharing, 2. Unshared pages, also known as deduplication candidate pages, are the locations where madvise() instructs ksmd to merge., 3. Pages shared by processes or users that have been deduplicated (shared pages) [9].

For candidate pages of deduplication, KSM uses two Red-Black trees: a stable tree and an unstable tree. The RB (Red Black) tree's efficiency is O(log n) per tree, and its height is never greater than 2log(n+1), where n is the number of nodes. In a round robin technique, KSM searches each memory location one by one. If the page is accessed, KSM first looks at the stable RB tree and merges with it if, it is identical. Otherwise, it checks the unstable tree for a match, and if one is found, it removes the page from the unstable tree and adds it to the stable tree [10].

Before starting memory sharing, allocated memory must be registered as being potentially shared by KSM. The stable tree comprises all of KSM's shared and write-protected pages. Unstable trees are those that have the potential to be shared and whose contents haven't changed in a long time. The contents of memory pages are used to index the nodes of both trees. Nodes in the stable tree point to memory pages that are shared, whereas nodes in the unstable tree indicate pages that are ideal candidates for sharing but are not shared. Both trees are initially empty. Scanned pages are examined for matches in the unstable tree as long as the shared tree is empty. A page is added to the unstable tree if there is no match in the unstable tree [11].

2.3 Kernel Virtual Machine

KVM (Kernel Virtual Machine) is a complete virtualization framework that enables hardware virtualization on x86 CPUs (Intel VT or AMD-V). It is made up of two primary parts: A group of kernel modules (kvm.ko, kvm-intel.ko, and kvm-amd.ko) that offer the underlying fundamental virtualization framework and processor specific drivers, as well as a userspace programme (qemu-kvm) that provides virtual device emulation and management mechanisms (virtual machines). The word KVM refers to the virtualization functionality at the kernel level, but it is more generally used to refer to the userspace component. Libvirt-based and QEMU-based tools could be used to manage VM Guests (virtual machines), virtual storage, and networks. libvirt is a library that provides an API (Application programming interface) for maintaining VM Guests utilising various virtualization solutions, including KVM and Xen. It has a graphical user interface and a command line program also. The QEMU (Quick Emulator) tools are specific to KVM/QEMU and are only available through the use of the command line [12].

2.4 QEMU (Quick Emulator)

QEMU is a cross-platform, fast Open - sourced machine emulation approach that can simulate a broad range of hardware architectures. QEMU allows users to run a fully functional operating system (VM Guest) on top of the current system (VM Host Server). QEMU is composed of several components: a processor emulator, emulated devices, generic devices for communicating the emulated devices to the related host devices, debugger, and a user interface for interacting with the emulator. QEMU can be used in conjunction with the KVM kernel module to provide a virtualization solution. QEMU can take use of KVM acceleration if indeed the VM Guest hardware architecture is the same as the VM Host Server's architecture. Tools based on libvirt, such as virt-manager and vm-install, provide simple interfaces for creating and managing virtual machines [13].

2.5 Libvirt & Virsh

Libvirt is a virtualization platform management toolkit that is accessible from C, Python, Perl, and GO, among many other languages, and is licenced under many standard open sources. It supports KVM, QEMU (Quick EMUlator), Virtuozzo, VMware ESX (Elastic Sky X), LXC (Linux Containers), BHyve (BSD hypervisor), and other virtualization technologies. It is destined for use with Linux, FreeBSD, Windows, and MacOS. Virsh is a shell wrapper in Libvirt that includes access to libvirt functionality on platforms that support virtualization. Virsh is a command-line and batch scriptable tool for managing all libvirt-managed domains, networks, and storage. This is included with the libvirt core distribution. libvirt-host is a libvirt module that provides several APIs. It has several macros for getting and setting various memory parameters of the virtual machine, such as virNodeGetInfo, virNodeGetMemoryParameters, virNodeGetMemoryStats, and virNodeSetMemoryParameters [14].

2.6 Attacks Based on Covert Channels Using Deduplication of Memory

A single physical server can collocate several virtual machines being used by many users in a cloud computing environment where multi-tenants are used. The public cloud environment uses sharing of identical memory pages among different users to maximise resource utilisation, which can lead to a memory disclosure attack. On deduplicated pages that are re-created by Copy-on-write, malicious users can take advantage of the time difference. Because of the Copy-on-write method, more time is spent accessing the page than if it were accessed normally. To protect against memory disclosure attacks, sharing can be enabled within single-user virtual machines but disabled for other users.

2.7 Hierarchical Agglomerative Clustering

For data clustering, hierarchical clustering is a widely used unsupervised machine learning technique. Agglomerative clustering and divisive clustering were the two broad classifications. The following are the steps involved: 1. Each data point in the dataset was treated as a separate cluster at first. 2. To create a cluster, connect the data points that are closest to each other. 3. Connect nearby clusters to form new clusters. 4. Dendrograms are used to split a large cluster into several smaller ones. The following are unique features: 1. the number of clusters does not need to be specified. 2. Dendrograms make it easy to understand how data has been grouped.

3 Literature Review

Gu et al. [15] discussed virtual machine guest OS finger printing. The upgraded OS-Sommelier+, a multiaspect, memory-exclusive methodology, reduced the virtual machine's physical memory utilisation while maintaining precision and usability. This study encourages us to conduct a thorough review of code hashing and data signature. Guest OS administration, kernel dump analysis, memory forensics, penetration testing, and virtual memory introspection are all aided by this approach.

Jia et al. [16], developed Loc-K, a new memory deduplication algorithm that uses logical addresses of separate pages to provide greater continuity, resulting in improved spatial locality. This approach enhances the prediction ratio by predicting k probable duplication places for page scanning. The prediction opportunity increased to 97.8%, with a 96.5 percent prediction hit ratio.

Wang et al. [17], implemented and evaluated Covert Inpector, a virtual machine monitor based approach to identify and eliminate covert timing channels to build on memory being shared. The test finds and throttles a covert timing channel based on shared memory. This test finds and eliminates covert channels that have a significant impact on the performance of the guest virtual machine.

Elghamrawy et al. [18], investigated the wide discrepancy between existing prediction mechanisms and actual behaviour was investigated by. They investigated memory page behaviour using page flags provided through the Linux kernel's proc file system and used the framework to anticipate memory pages that are expected to be generally stable, as well as memory deduplication and virtual machine live migration.

Garoa et al. [19], studied the impact of ASLR (Address Space Layout Randomization) over memory deduplication. They looked at how memory deduplication affects kernel randomization. When kernel ASLR is enabled, the memory cost of running approximately 24 kernels rise by 534 percent (from 613 MiB to 3.9 GiB).

Patel et al. [20], used a machine learning approach to classify virtual machines into labelled clusters for server consolidation. To group similar virtual machines, they adopted neural networks, which come under the category of supervised learning. In terms of the number of virtual samples properly identified, their work outperforms. This study motivates us to classify virtual machines according to their guest Operating Systems installed inside it.

Zhu et al. [21], proposed a new memory deduplication method called Page Correlation Aggregation (PCA). It almost effectively reduces the number of covert channel operations. Since pages with comparable features have a higher possibility of sharing, this strategy entails separating the virtual machine's pages into several sets. Pages are then classified into several classes based on their access permissions within each category. As a result, for sharing purposes, page comparisons are limited to the same classifications. PCA seems to be a strategy to minimise copy-on-write latency while using a covert channel.

Lindermann et al. [22], suggested a timing side channel attack to detect software versions operating in co-located virtual machines, and executed an intrusion to assess whether pages are unique to a certain software version in co-located virtual machines. The tests are carried out in a fair amount of time, and viable countermeasures against the described side channel attack are also examined.

Lindermann et al. [23], devised a memory deduplication side-channel intrusion to reveal applications of other virtual machines that were co-located. This entails verifying memory page availability in co-located virtual machines that are unique across all versions of the software.

You et al. [24], investigated lightweight memory deduplication. This work involves individual memory pages being divided into various segments, and the joined strings of the hash values of these segments are used as indexed keys in trie data structures, reducing the time required to search for identical twin pages and the number of memory page comparisons. As a result, CPU consumption will be reduced by 44.9 percent, and memory bandwidth usage will be reduced by 31.6 percent.

Shiba [25], discussed how the cost of finding mergeable pages is considerably high, and how pages are distributed in address spaces. MashitoShiba, classifies pages by the state of consecutive memory pages, measures the ratio of pages in each state, and shows the measuring distribution that can be used to evaluate the likelihood of merging.

The Cgroups mechanism was used by Goa et al. [26], to enable operating system containerization. Cgroups mechanisms separate processes into hierarchical groups and controllers in order to keep system resources like the CPU, memory, and I/O (Input/Outout) in check. For the establishment of resource control, newly generated child processes immediately copy Cgroups attributes from their parent processes. But the inherited Cgroups incarceration via process formation does not assure a steady and good accounting of resources. By separating processes from their original process groups, they establish a set of exploitation techniques for producing out-of-band workloads. They examined five case studies using Docker containers to see how they may overcome Cgroup's resource dominance in real-world circumstances. An adversarial container can substantially increase the amount of consumed resources in a multitenant environment by exploiting Cgroups, which appears to slow down other containers on the same host and gives it unfair benefits in the system. This study motivated us to allocate resources to various processes.

Garcia et al. [27], developed a novel kernel randomization technique adaptive with memory deduplication that detects and mitigates threats at the kernel level, as well as the memory pages shared by each area. They introduced KASLR-M+ (Kernel Address Space Layout Randomization), the first efficient and practical Kernel ASLR memory security that maximises memory deduplication savings without losing security, based on their findings.

Garcia et al. [28], investigated the impact of function granular Kernel Randomization on memory deduplication and why it is incapable of providing the utmost memory protection and sharability in their research. They proposed a method that forces guest kernels belonging to the same tenant to use the same random memory layout of memory regions, which has a significant influence on deduplication.

4 Approach and Implementation

This approach provides a mechanism that supports multiple deduplication threads, each of which is dedicated to a batch that has a set of similar operating systems. The grouping of each batch is based on similar OS. When the Kernel Virtual Machine (KVM) instantiates a new virtual machine, it registers the memory region of each virtual machine to the memory regions of KSM. Once KSM gets started, a global ksmd daemon process automatically starts and performs memory deduplication. In this approach, the global ksmd daemon is split into similar threads, each performing memory deduplication of each homogeneous batch. Cgroups, of Linux, is utilised to allocate CPU/memory resources of each user process. Overview architecture of homogeneous batch memory deduplication using clustering of virtual machines is shown in Fig. 1.

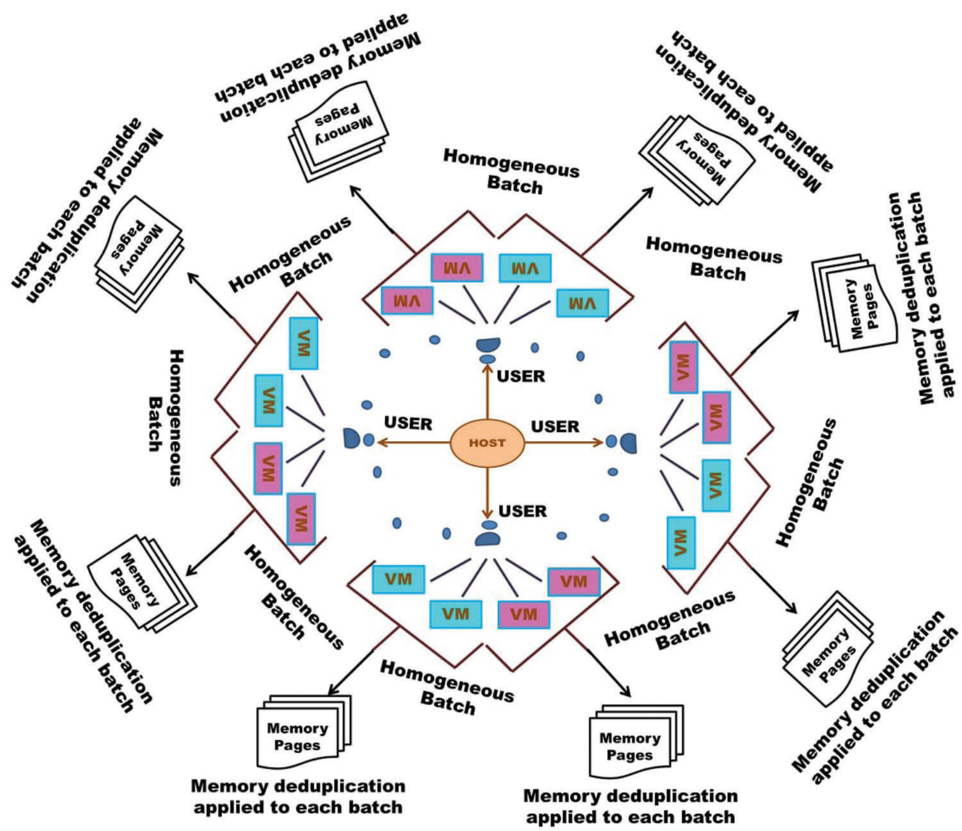

Figure 1: Overview architecture of homogeneous batch memory deduplication using clustering of virtual machines

Implementation: The aforementioned work was divided into two components for implementation.

Module1: Using Hierarchical Agglomerative Clustering, virtual machines are clustered depending on the guest operating system.

Module 2: Memory Deduplication is applied separately to each homogeneous batch.

4.1 Module 1

Module 1 entails clustering virtual machines based on the guest operating system deployed, which entails the actions below. Tab. 1 shows the information log of virtual machines created. Information about the virtual machine, such as its domain name, Universal Unique Identifier (UUID), (Operating System) OS type, OS variant, and user of the virtual machine, is entered into the information repository as soon as it is formed. When a new virtual machine is generated, the information log is updated, and the related record is deleted when a virtual machine is removed from the host. A Python code is executed to identify the list of active and inactive domains on the host, tabulated in Tab. 2. Information of all active and inactive domains with their domain id, domain name, UUID of that domain and the current status of domains either running or shutoff, listed in Tab. 2. Based on the output, another Python program is run, which uses Hierarchical Agglomerative Clustering to group only the active domains into different clusters based on their guest operating systems. The virtual machines with Domain ID 2 (Identifier) and 5 clustered on Windows, whereas virtual machines with Domain ID 3 and 4 clustered on Linux, and all four virtual machines are running domains, as shown in Fig. 2.

Table 1: Information log of virtual machines created

User	Domain name	UUID	OS type	OS variant
User 1	centos7.0	e9a601b8-f56a-44ee-943f-6ddf640ce28f	Linux	Cent OS
User 1	win7	9986cc97-5cae-4767-a003-9b1337558b06	Windows	Windows 7
User 1	Ubu1	aacd3461-4e67-7890-bc02-7b1337558b10	Linux	Ubuntu14
User 2	generic1	ea7bcc59-e22d-4a7c-a5b1-028e17cf45b4	Linux	Red Hat
User 2	generic2	d43fc653-7b2d-4edf-98f3-96ebb78544b3	Linux	Fedora
User 3	Winx	38972079-3bc4-4dff-a4b0-ed33da19b9ee	Windows	Windows XP
User 3	Win7a	5678098ea-4bcc-5bbf-a134-de54ad20acff	Windows	Windows 7

Table 2: List of active and inactive domains

Domain Id	Domain name	UUID	Status
1	centos7.0	e9a601b8-f56a-44ee-943f-6ddf640ce28f	Running
2	win7	9986cc97-5cae-4767-a003-9b1337558b06	Shut Off
3	generic1	ea7bcc59-e22d-4a7c-a5b1-028e17cf45b4	Running
4	generic2	d43fc653-7b2d-4edf-98f3-96ebb78544b3	Running
5	Ubu1	aacd3461-4e67-7890-bc02-7b1337558b10	Running
6	Winx	38972079-3bc4-4dff-a4b0-ed33da19b9ee	Running
7	Win7a	5678098ea-4bcc-5bbf-a134-de54ad20acff	Running

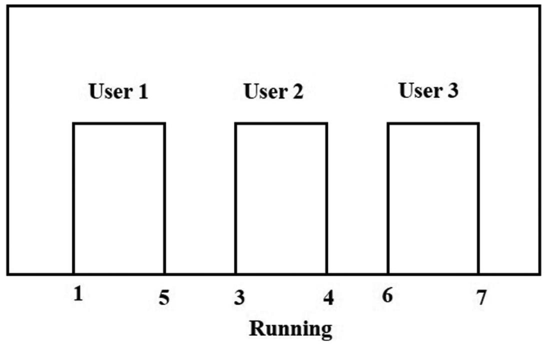

Figure 2: Snapshot generated after clustering of active domains

Once the active domains are classified a dendrogram, shown as snapshot, representing the clusters of various active domains. In Fig. 2, domains with id's 1 and 5 are clustered into a batch of User1, domains with id's 3 and 4 are clustered into a batch of User2 and domains with id's of 6 and 7 are clustered in another batch of User3 and all are currently active at the moment.

4.2 Module 2

After clustering of the active domains, it's time to move on to the next phase. Deduplication threads are applied to each batch, and memory deduplication activities are performed inside the memory given to the batch. Various scan rates are assigned to each batch. The scan rate of a batch with CPU-intensive activities, such as games, can be set as low. Each batch has a KSM daemon thread, seperate "Stable tree" and "unstable tree" are two data structures used by KSM. In a batch, scan a new page, KSM checks the new page against the stable tree and merges it with the page if a match is found. If no match was detected, a search of the unstable tree was conducted. If a match was found in the unstable tree, the page was moved from the unstable tree to the Stable tree. If no match was found, a new page entry was generated in the unstable tree, and a new page was searched for. Flow chart of homogeneous batch memory deduplication was shown in Fig. 3. The details of information log created and information of active and inactive domains are also shown by the side of the first two steps of flowchart. Next to that, step by step implementation of homogeneous batch memory deduplication was given.

5 Experiments and Evaluation

The experimental results are shown below:

5.1 Experimental Setup

The following Tab. 3 shows experimental setup:

5.2 Trial Versions

Three trial versions for three users are performed and the following Tab. 4 shows virtual machine set up assigned for each user. In each trial, minimum of 5 virtual machines executed for each user and virtual machines of Linux guest operating system are grouped in Batch I and Windows virtual machines are grouped in Batch II.

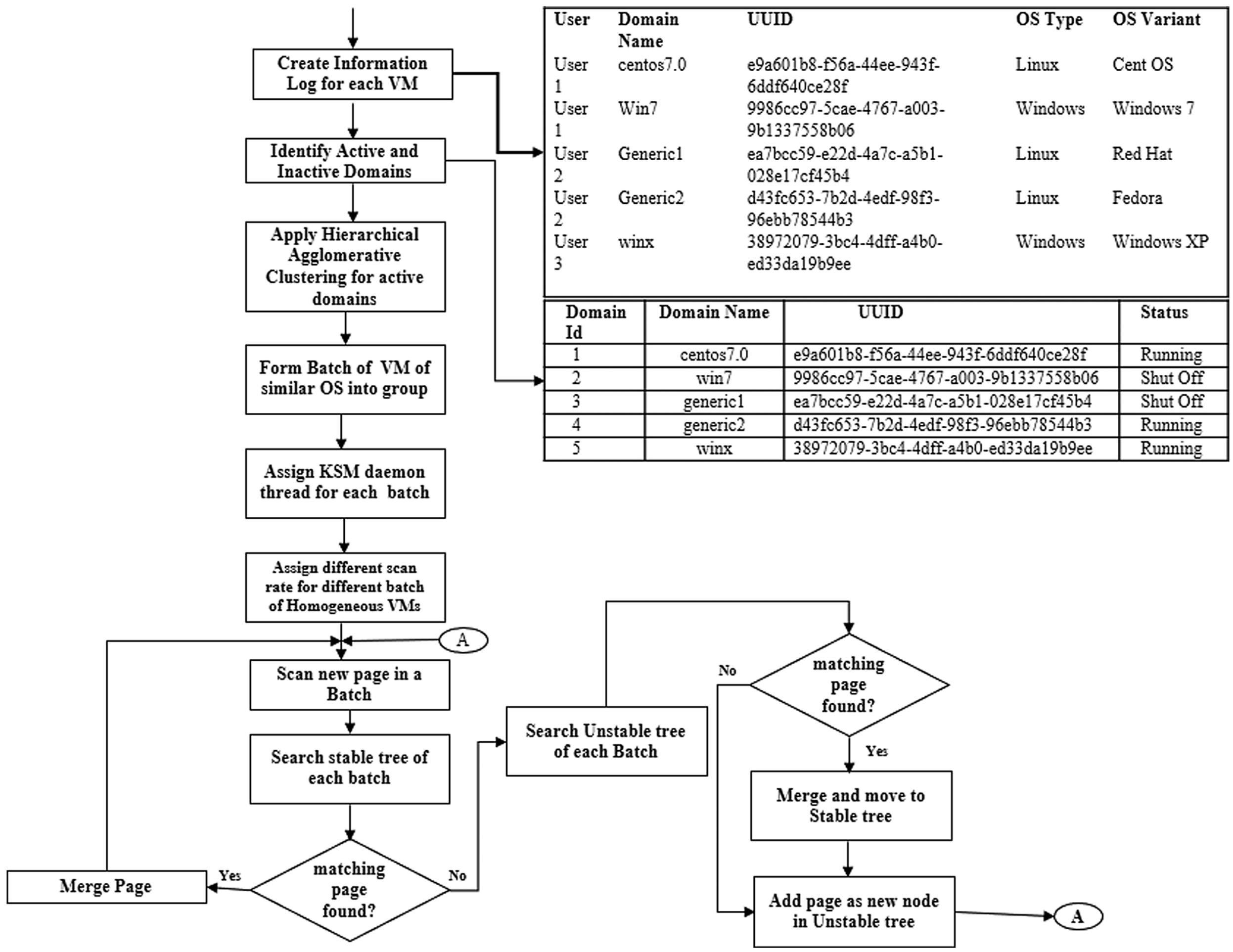

Figure 3: Flow chart of homogeneous batch memory deduplication

Table 3: Experimental setup

Variables	Characteristics
CPU Processor	Intel Core i5 Processor, 8th generation
RAM (Random-access memory)	4 GB
Hypervisor	KVM with QEMU
OS for VMs	Fedora, Ubuntu, Cent OS, Win 7, Win XP, Red Hat
API	Libvirt (virsh)

In Fig. 4, the first part of the figure show the memory deduplication applied to the entire host. The next lane shows memory pages shared for each trial of each users for each number of scans. In Fig. 5, the merge rate is comparatively high, when merging performed in various trials of users than deduplication applied to the entire host. Fig. 6 shows the memory saving rate, which is calculated from total number of pages shared per user, for each scan of three different users. is found that it is increasing rapidly at the initial stage and gradually afterwards. Fig. 7, shows the percentage of Memory Pages shared for each scan. The utilization of CPU is high when compared with memory deduplication applied to the whole host, shown in Fig. 8. Fig. 9 shows the Statistics of Memory Deduplication in which the following information are noted: No. of Pages sharing- indicates how many pages that virtual machines create, No. of Pages shared- indicates how many pages actually in use and being shared, No. of Pages unshared - indicates number of pages that are unshared, No. of Pages volatile - indicates pages that change often and too fast be inserted in RB tree, No. of scans - how many times all merge able areas have been scanned. From the information, it is found that the ratio of pages sharing to pages shared is high, which infers good sharing opportunities and the number of memory pages shared for each user in each trial is tabled in Tab. 5.

Table 4: User trial details

	User 1	User 2	User 3
Trial 1:			
Batch 1	VM1: RedHat VM2: RedHat VM3: Red Hat	VM1: RedHat VM2: RedHat VM3: Cent OS	VM1: CentOS VM2: CentOS VM3: Ubuntu 14
Batch 2	VM1: Win7 VM2: Win 7	VM1: WinXP VM2: Win 7	VM1: Win7 VM2: Win 10
Trial 2:			
Batch 1	VM1: Fedora VM2: RedHat VM3: Cent OS	VM1: CentOS VM2: Ubuntu14 VM3: Red Hat	VM1: RedHat VM2: Red Hat
Batch 2	VM1: Win10 VM2: Win 7	VM1: WinXP VM2: Win 7	VM1: WinXP VM2: Win7 VM3: Win 10
Trial 3:			
Batch 1	VM1: RedHat, VM2: RedHat, VM3: Fedora	VM1: RedHat VM2: RedHat VM3: Ubuntu 14	VM1: CentOS VM2: CentOS VM3: Fedora
Batch 2	VM1: Win7 VM2: Win 10	VM1: WinXp VM2: Win 7	VM1: Win7 VM2: Win XP

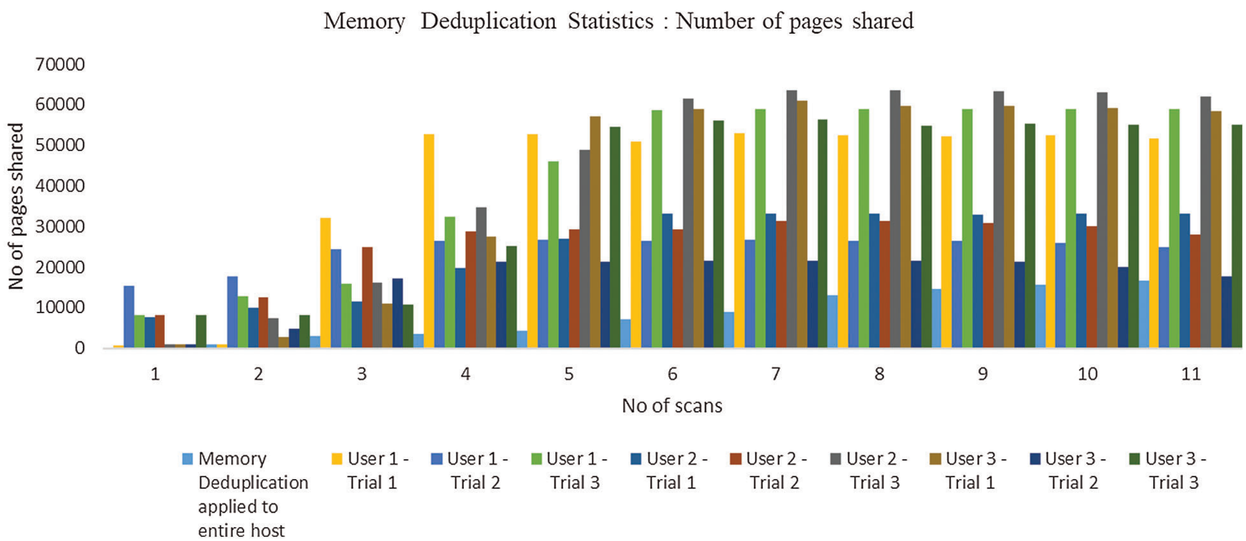

Figure 4: Statistics of homogeneous memory deduplication and KSM

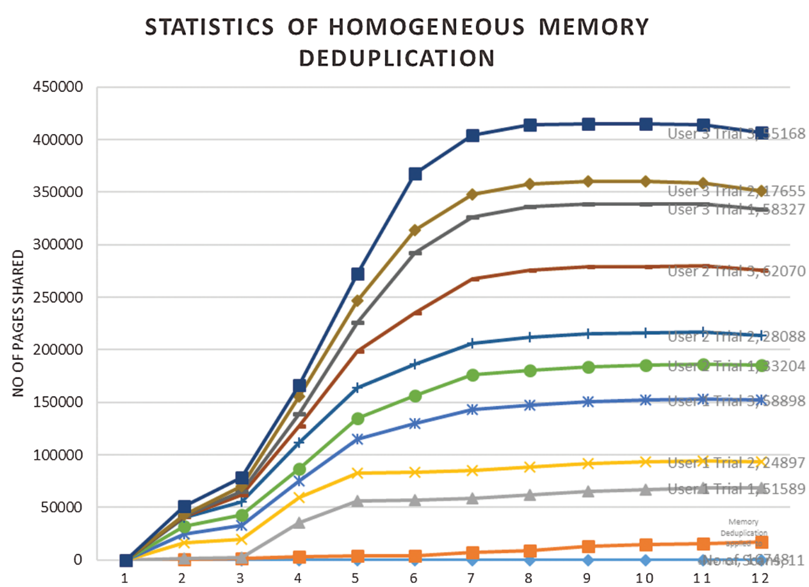

Figure 5: Number of pages shared in each trial of user

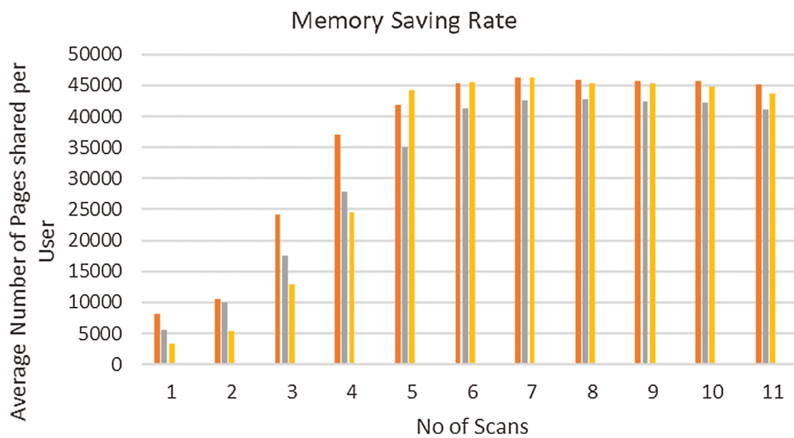

Figure 6: Memory saving rate

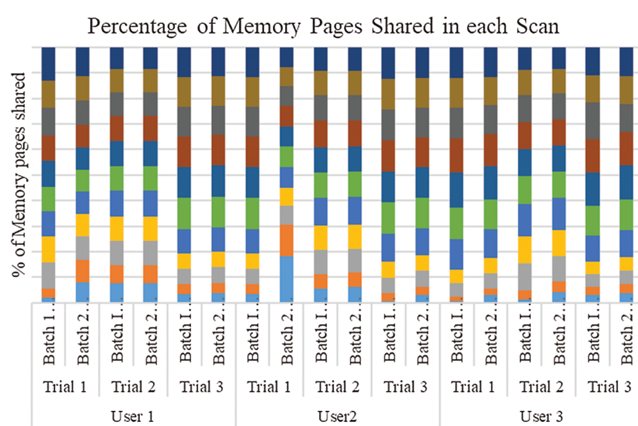

Figure 7: Percentage of memory pages shared for each scan

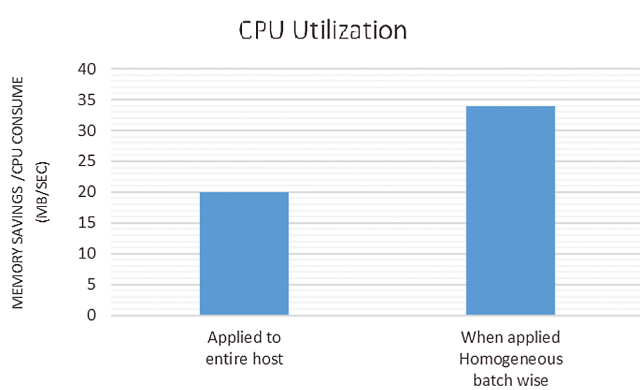

Figure 8: CPU utilization

Memory Deduplication Statistics																			
		User 1						User 2						User 3					
		Trial 1		Trial 2		Trial 3		Trial 1		Trial 2		Trial 3		Trial 1		Trial 2		Trial 3	
No.of Scans	Details	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)	Batch 1 (Linux)	Batch 2 (Windows)
	No of Pages Shared	122	643	7447	7980	3778	4326	3593	4029	3805	4361	136	688	124	707	216	815	3766	4358
	No of Pages Sharing	2310	2831	127974	128507	61382	61930	57018	57454	76839	77395	10247	10799	10112	10695	19912	20511	61247	61839
	No of Pages Unshared	2345	2866	67216	67749	24008	24556	15984	4029	76255	76811	33047	33599	32033	32616	61668	62267	22994	23586
1	No of Pages Volatile	17800	18321	65932	66465	107000	107548	119537	119973	112967	113523	154035	154587	129352	129935	98310	98909	82317	82909
	No of Pages Shared	199	720	8636	9169	6069	6617	6030	4029	5920	6476	3353	3905	1100	1683	2146	2745	3816	4408
	No of Pages Sharing	2267	2788	157249	157782	85982	86530	85781	86217	127784	128340	56517	57069	33017	33600	65680	66279	62482	63074
	No of Pages Unshared	14035	14556	52903	53436	31181	31729	31114	4029	48742	49298	27020	27572	20025	20608	16432	17031	24186	24778
2	No of Pages Volatile	841899	842420	203200	203733	323911	324459	239968	240404	121847	122403	242558	243110	236853	237436	82958	83557	318206	318798
	No of Pages Shared	15811	16332	11920	12453	7607	8155	7441	4029	12105	12661	7792	8344	5210	5793	8304	8903	5025	5617
	No of Pages Sharing	64907	65428	174299	174832	113056	113604	114862	115298	166024	166580	104781	105333	65768	66351	103962	104561	74043	74635
	No of Pages Unshared	70589	71110	96045	96578	48777	49325	58617	4029	80927	81483	33659	34211	25166	25749	37446	38045	40284	40876
3	No of Pages Volatile	166133	166654	435132	435665	248545	249093	266732	267168	573868	574424	387281	387833	250119	250702	499478	500077	111383	111975
	No of Pages Shared	26059	26580	12969	13502	15891	16439	15783	4029	14157	14713	17079	17631	13415	13998	10302	10901	12227	12819
	No of Pages Sharing	103009	103530	178040	178573	145210	145758	146085	146521	171702	172258	138872	139424	123624	124207	109526	110125	129962	130554
	No of Pages Unshared	124960	125481	139706	140239	100694	101242	86422	4029	129216	129772	90204	90756	99393	99976	88932	89531	109883	110475
4	No of Pages Volatile	114750	115271	352957	353490	231926	232474	151031	151467	512724	513280	391693	392245	373896	374479	502124	502723	214129	214721
	No of Pages Shared	26157	26678	13028	13561	22784	23332	22866	4029	14416	14972	24172	24724	28318	28901	10348	10947	26930	27522
	No of Pages Sharing	105684	106205	167266	167799	128804	129352	129237	129673	150644	151200	112182	112734	119001	119584	88926	89525	135623	136215
	No of Pages Unshared	107694	108215	141643	142176	106320	106868	120180	4029	159761	160317	124438	124990	137810	138393	109346	109945	119692	120284
5	No of Pages Volatile	126142	126663	276269	276802	186159	186707	201335	201771	297101	297657	206991	207543	213615	214198	201518	202117	192783	193375
	No of Pages Shared	25228	25749	12994	13527	29083	29631	29127	4029	14418	14974	30507	31059	29152	29735	10532	11131	27728	28320
	No of Pages Sharing	104751	105272	177317	177850	132030	132578	131583	132019	171245	171801	125958	126510	118116	118699	109080	109679	124188	124780
	No of Pages Unshared	123028	123549	121211	121744	109444	109992	133530	4029	148826	149382	137059	137611	135641	136224	101970	102569	108026	108618
6	No of Pages Volatile	95777	96298	316918	317451	228032	228580	204305	204741	289300	289856	200414	200966	529036	529619	190126	190725	556654	557246
	No of Pages Shared	26279	26800	13075	13608	29150	29698	29129	4029	15393	15949	31468	32020	30152	30735	10488	11087	27834	28426
	No of Pages Sharing	105362	105883	175889	176422	131251	131799	134538	134974	175961	176517	131323	131875	111360	111943	110364	110963	111288	111880
	No of Pages Unshared	118121	118642	134753	135286	141992	142540	111443	4029	119613	120169	126852	127404	147188	147771	91348	91947	162328	162920
7	No of Pages Volatile	116493	117014	292900	293433	196930	197478	212005	212441	305230	305786	209260	209812	231749	232332	200190	200789	219419	220011
	No of Pages Shared	25939	26460	12994	13527	29125	29673	29228	4029	15428	15984	31559	32111	29552	30135	10506	11105	27118	27710
	No of Pages Sharing	105699	106220	173251	173784	133841	134389	133930	134366	172350	172906	132940	133492	109694	110277	111840	112439	110595	111187
	No of Pages Unshared	106359	106880	141433	141966	125967	126515	152748	4029	160362	160918	144896	145448	128325	128908	100340	100939	109396	109988
8	No of Pages Volatile	6744	7265	332171	332704	372142	372690	314145	314581	275765	276321	315736	316288	217333	217916	189704	190303	273739	274331
	No of Pages Shared	25799	26320	12950	13483	29120	29668	28980	4029	15163	15719	31333	31885	29506	30089	10344	10943	27293	27885
	No of Pages Sharing	93854	94375	181182	181715	137803	138351	136599	137035	174869	175425	131490	132042	126107	126690	108640	109239	132420	133012
	No of Pages Unshared	102467	102988	114473	115006	100869	101417	100869	4029	121779	122335	108175	108727	125674	126257	88590	89189	118368	118960
9	No of Pages Volatile	123712	124233	237505	238038	30060	30608	130060	130496	332091	332647	124646	125198	214913	215496	203918	204517	120327	120919
	No of Pages Shared	25997	26518	12715	13248	29154	29702	29200	4029	14808	15364	31247	31799	29341	29924	9730	10329	27248	27840
	No of Pages Sharing	98123	98644	186446	186979	137975	138523	137975	138411	179607	180163	131136	131688	126503	127086	109312	109911	133342	133934
	No of Pages Unshared	119957	120478	132010	132543	123116	123664	126131	4029	140268	140824	131374	131926	120129	120712	94874	95473	111871	112463
10	No of Pages Volatile	112585	113106	689378	689911	192668	193216	189300	189736	1E+06	1055691	558425	558977	583334	583917	978440	979039	217577	218169
	No of Pages Shared	25534	26055	12182	12715	29175	29723	29175	4029	13766	14322	30759	31311	28872	29455	8528	9127	27288	27880
	No of Pages Sharing	79385	79906	185385	185918	138772	139320	138543	138979	173166	173722	126553	127105	119322	119905	108784	109383	131541	132133
	No of Pages Unshared	110687	111208	101173	101706	99854	100402	111795	4029	121149	121705	119830	120382	124317	124900	87456	88055	104341	104933
11	No of Pages Volatile	138822	139343	324399	324932	239939	240487	216496	216932	308803	309359	224343	224895	531457	532040	215100	215699	547053	547645

Figure 9: Amount of pages shared in each user's trial, memory deduplication statistics for user's trial

Table 5: Number of memory pages shared for each user in each trial

Memory deduplication statistics : Number of pages shared										
No of scans	Memory deduplication applied to entire host	User 1			User 2			User 3		
		Trial 1	Trial 2	Trial 3	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2	Trial 3
1	190	765	15427	8104	7622	8166	824	831	1031	8124
2	1054	919	17805	12686	10059	12396	7258	2783	4891	8224
3	2909	32143	24373	15762	11470	24766	16136	11003	17207	10642
4	3591	52639	26471	32330	19812	28870	34710	27413	21203	25046
5	4244	52835	26589	46116	26895	29388	48896	57219	21295	54452
6	7227	50977	26521	58714	33156	29392	61566	58887	21663	56048
7	8993	53079	26683	58848	33158	31342	63488	60887	21575	56260
8	12932	52399	26521	58798	33257	31412	63670	59687	21611	54828
9	14703	52119	26433	58788	33009	30882	63218	59595	21287	55178
10	15700	52515	25963	58856	33229	30172	63046	59265	20059	55088
11	16748	51589	24897	58898	33204	28088	62070	58327	17655	55168

6 Conclusion

Memory deduplication is particularly vulnerable to memory disclosure attacks. Memory deduplication can be used on virtual machines belonging to a single user group to prevent this attack. When compared to memory deduplication applied to the entire host, user virtual machines are grouped suitably and memory deduplication is performed to homogenous batch wise and the proportion of sharing of memory pages is increased and CPU utilization is high. In future work, the identical applications running in virtual machines of homogeneous batch are categorized further and memory deduplication to be performed.

Acknowledgement: The author with a deep sense of gratitude would thank the supervisor for his guidance and constant support rendered during this research.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. Hwang, J. Dongarra and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Waltham, MA, USA, China Machine Press, 2013. [online]. Available: https://books.google.com/books.

[2] E. Bugnion, S. Devine, K. Govil and M. Rosenblum, "DISCO: Running commodity operating systems onscalable multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 4, pp. 412–447, 1997.

[3] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. Sym. on Operating Systems Design and Implementation*, New York, NY, USA, pp. 181–194, 2002.

[4] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren *et al.,* "Difference engine: Harnessing memory redundancy in virtual machines," *Communications of the ACM*, vol. 53, no. 10, pp. 85–93, 2008.

[5] K. Suzaki, K. Iijima, T. Yagi and C. Artho, "Memory deduplication as a threat to the guest OS," in *Proc. European Workshop on System Security*, Salzburg, Austria, pp. 1–3, 2011.

[6] K. Okamura and Y. Oyama, "Load-based covert channels between Xen virtual machines," in *Proc. ACM Sym. on Applied Computing*, Sierre, Switzerland, pp. 173–180, 2010.

[7] Y. Deng, C. Hu, T. Wo, B. Li and L. Cui, "A memory deduplication approach based on in virtualized environments state key laboratory of software development environment," in *Proc. IEEE Seventh Int. Sym. on Service-Oriented System Engineering*, San Francisco, CA, USA, pp. 367–372, 2012.

[8] S. Barker, T. Wood, P. Shenoy and R. Sitaraman, "An empirical study of memory sharing in virtual machines," in *Proc. Annual Technical Conf.*, Boston, MA, pp. 273–284, 2012.

[9] W. C. Lin, C. H. Tu, C. W. Yeh and S. H. Hung, "GPU acceleration for kernel samepage merging," in *Proc. Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE*, Busan, Korea, pp. 1–6, 2017.

[10] A. Arcangeli, I. Eidus and C. Wright, "Increasing memory density by using KSM," in *Proc. Linux Sym.*, USA, pp. 19–28, 2009.

[11] K. V. M. Contributors, *Kernel Samepage Merging KSM*, Canada, KVM, Red Hat Open Shift, 2015. [online]. Available: https://www.linux-kvm.org/page/KSM.

[12] K. Ivanov, *KVM Virtualization Cookbook - Learn how to effectively use KVM in production*, Mumbai, India, Packt Publishing, 2017. [online]. Available: https://books.google.com/books.

[13] H. D. Chirammal, P. Mukhedkar and A. Vettathu, "Mastering KVM Virtualization" in *Packt Publishing*, 1st ed., vol. 1. New York, Kindle Edition, pp. 468, 2004.

[14] K. Suzaki, K. Iijima, T. Yagi and C. Artho, “Implementation of a memory disclosure attack on memory deduplication of virtual machines,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 96, no. 1, pp. 215–224, 2013.

[15] Y. Gu, Y. Fu, A. Prakash, Z. Lin and H. Yin, “OS-sommelier: Memory-only operating system fingerprinting in the cloud,” in *Proc. Third ACM Sym. on Cloud Computing*, New York, USA, pp. 1–13, 2012.

[16] S. Jia, C. Wu and J. Li, “Loc-K: A spatial locality-based memory deduplication scheme with prediction on k-step locations,” in *Proc. IEEE 23rd Int. Conf. on Parallel and Distributed Systems (ICPADS)*, Shenzhen, China, pp. 310–317, 2017.

[17] S. Wang, W. Qiang, H. Jin and J. Yuan, “Covert Inspector: Identification of shared memory covert timingchannel in multi-tenanted cloud,” *International Journal of Parallel Programming*, vol. 45, no. 1, pp. 142–156, 2017.

[18] K. Elghamrawy, D. Franklin and F. T. Chong, “Predicting memory page stability and its application to memory deduplication and live migration,” in *Proc. 2017 IEEE Int. Sym. on Performance Analysis of Systems and Software (ISPASS)*, Santa Rosa, CA, USA, pp. 125–126, 2017.

[19] F. V. Garcia and H. M. Gisbert, “How kernel randomization is canceling memory deduplication in cloud computing systems,” in *Proc. 2018 IEEE 17th Int. Sym. on Network Computing and Applications (NCA)*, Cambridge, MA, USA, pp. 1–4, 2018.

[20] E. Patel, A. Mohan and D. S. Kushwaha, “Neural network based classification of virtual machines in IaaS,” in *Proc. 5th IEEE Uttar Pradesh Section Int. Conf. on Electrical, Electronics and Computer Engineering (UPCON)*, Gorakhpur, India, pp. 1–8, 2018.

[21] M. Zhu, K. Zhang and B. Tu, “PCA: Page correlation aggregation for memory deduplication in virtualized environments,” in *Proc. Int. Conf. on Information and Communications Security*, Lille, France, pp. 566–583, 2018.

[22] J. Lindeman and M. Fischer, “Efficient identification of applications in co-resident vms via a memory side-channel,” in *Proc. IFIP Int. Conf. on ICT Systems Security and Privacy Protection*, Poznan, Poland, pp. 245–259, 2018.

[23] J. Lindemann and M. Fischer, “On the detection of applications in co-resident virtual machines via a memorydeduplication side-channel,” *ACM SIGAPP Applied Computing Review*, vol. 18, no. 4, pp. 31–46, 2019.

[24] L. You, Y. Li, F. Guo, Y. Xu, J. Chen *et al.,* “Leveraging array mapped tries in ksm for lightweight memory deduplication,” in *Proc. IEEE Int. Conf. on Networking, Architecture and Storage (NAS)*, EnShi, China, pp. 1–8, 2019.

[25] M. Shiba, “An examination method of mergeable memory page distribution for memory deduplication,” in *Proc. 2019 IEEE 8th Global Conf. on Consumer Electronics (GCCE)*, Osaka, Japan, pp. 1027–1031, 2019.

[26] X. Gao, Z. Gu, Z. Li, H. Jamjoom and C. Wang, “Houdini’s escape: Breaking the resource rein of linux control groups,” in *Proc. ACM SIGSAC Conf. on Computer and Communications Security*, London, United Kingdom, pp. 1073–1086, 2019.

[27] F. V. Garcia and H. M. Gisbert, “KASLR-MT: Kernel address space layout randomization for multi-tenant cloud systems,” *Journal of Parallel and Distributed Computing*, vol. 137, pp. 77–90, 2020.

[28] F. V. Garcia and H. M. Gisbert, “An info-leak resistant kernel randomization for virtualized systems,” *IEEE Access*, vol. 8, pp. 161612–161629, 2020.