Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: 0005-1144 (Print) 1848-3380 (Online) Journal homepage: www.tandfonline.com/journals/taut20

Optimization of virtual machines performance using fuzzy hashing and genetic algorithm-based memory deduplication of static pages

N. Jagadeeswari, V. Mohanraj, Y. Suresh & J. Senthilkumar

To cite this article: N. Jagadeeswari, V. Mohanraj, Y. Suresh & J. Senthilkumar (2023) Optimization of virtual machines performance using fuzzy hashing and genetic algorithm-based memory deduplication of static pages, Automatika, 64:4, 868-877, DOI: 10.1080/00051144.2023.2223479

To link to this article: https://doi.org/10.1080/00051144.2023.2223479

Published online: 27 Jun 2023.

AUTOMATIKA
2023, VOL. 64, NO. 4, 868–877
https://doi.org/10.1080/00051144.2023.2223479

Optimization of virtual machines performance using fuzzy hashing and genetic algorithm-based memory deduplication of static pages

N. Jagadeeswari[a], V. Mohanraj[b], Y. Suresh[b] and J. Senthilkumar[b]

[a]Department of CSE, Thanthai Periyar Government Institute of Technology, Vellore, India; [b]Department of IT, Sona College of Technology, Salem, India

ABSTRACT

The demand for memory capacity has increased, and cloud energy usage has soared. The performance and scalability of virtualization interfaces in cloud computing are hampered by a lack of sufficient memory. To figure out this problem, a technique defined as memory deduplication is widely used to reduce memory consumption utilizing the page-sharing method. However, this method of memory deduplication using KSM has significant drawbacks, such as overhead owing to many online comparisons, which will consume so many CPU resources. In this research, a modified approach of Memory Deduplication of Static Memory Pages (mSMD), which is based on the identification of similar applications by Fuzzy hashing and clustering them using the Hierarchical Agglomerative Clustering approach, followed by similarity detection between static memory pages based on Genetic Algorithm and details stored in Multilevel shared page table, both operations performed in offline and final memory deduplication is carried out during online, is proposed for achieving performance optimization in virtual machines by reducing memory capacity requirements. When compared to existing techniques, the simulation results indicate that the proposed approach mSMD efficaciously minimizes the memory capacity required while improving performance.

ARTICLE HISTORY
Received 21 February 2023
Accepted 5 June 2023

KEYWORDS
Memory deduplication; cloud computing; classification; genetic algorithm; fuzzy hashing; hierarchical agglomerative clustering

1. Introduction

Cloud computing, the widely accepted and most popular computational technique, is the delivery of various services over the Internet and is best suited for both academic and industrial areas, as it reduces computational-hardware management and cost of ownership while increasing dexterity and resource on-demand scalability [1]. Companies like Foursquare, Netflix, and Snapchat serve billions of cloud customers by setting up their own computer infrastructure framework and then moving their operations to cloud environments like EC2, or Amazon Elastic Computing Cloud, which can readily scale services [2,3]. Various services like Pinterest, Vivino, Kroger, Gameloft, Etsy, eBay, Twitter, and Paypal serve millions of users through cloud providers [4]. Several businesses and customers have opted for cloud computing services in recent years due to their great scalability and inexpensive cost. Due to the limitless potential for automated resource pooling and sharing and improved user interface that cloud computing offers, it has become one of the hottest topics in the study. A de facto prerequisite for offering services and scalable and effective resources is cloud computing. Software, databases, data analytics, servers, and networking are all computer services that can be offered through the Internet for more flexible resources, quicker deployment, and cost-effectiveness due to economies of scale. The economic impact of cloud computing is enormous, and it is rapidly developing. In order to provide flexible resources, quick response, and scalability, cloud services employ the Internet to deliver computer services such as servers, databases, software, networking, and data analytics.

Virtualization creates a virtual framework by using code that mimics the functionality of equipment. Virtualization paradigm permits many operating systems to contribute to share computing resources by managing numerous virtual machine instances on an individual physical server [5,6]. Each virtual machine can handle multiple individual customers, and all virtual machines share and make use of the same physical resources, lowering the cost for all users. Various firms are experimenting with virtualization technologies to reduce operational expenses by accommodating multiple clients and an individual physical server. Because the number of Virtual Machines on a real server grows, the server's memory requirements grow as well [7]. Two key limits that cause memory capacity to suffer as requirements grow: (i) development of physical memory is comparatively slower than memory demand growth, and (ii) the power consumption rate is quite

CONTACT N. Jagadeeswari jagadeeswarin.cse@outlook.com; n.jagadeeswari@gmail.com Department of CSE, Thanthai Periyar Government Institute of Technology, Vellore 632002, India

high for servers with more memory than 64 GB. This is the reason that memory capacity has become more challenging in cloud computing environments [8,9].

Memory deduplication is one of the best memory-management strategies because it uses page sharing to combine similar data into a single copy, thereby reducing memory needs. Memory deduplication, as stated by both Difference Engine and VMware, is used to save memory in virtual machines. Furthermore, according to VMware, 40 percent of RAM is saved, and it can be increased to fifty percent according to Difference Engine [10,11]. Because of its success, the memory deduplication approach has been applied in various systems for their architecture. The Linux system uses Kernel Same Page Merging (KSM) to implement memory deduplication. This way of implementation is known as content-based page sharing (CBPS), and it is used to detect duplicate content by using page granularity [12,13]. In the KSM, two comparison trees of global red-black trees are presented, the first of which is an unstable tree and the second of which is a stable tree [14,15]. The shared pages with the copy-on-write-protect policy are stored in the stable tree, while all unique pages are saved in the unstable tree. Each candidate page is compared to both unstable and stable tree pages, and the candidate pages are appropriately reorganized [16,17]. Several machine learning algorithms have been developed, and hybrid strategies have been presented [18–20] for the classification of applications and prediction and identification of similarity pages, which can be used to categorize pages processed in memory.

With the development of cloud computing, it is now possible to host a huge number of virtual machine instances on a single cloud server. Each virtual machine instance can run several programmes, and the number of virtual machines and the number of apps they can run is constantly growing. As a result, memory capacity has become more in demand, and cloud energy consumption has skyrocketed. Insufficient memory hinders the performance and scalability of virtualization interfaces in cloud computing. In order to solve this issue, the page-sharing method is frequently used in conjunction with a technique known as memory deduplication to lower memory use. KSM has a severe flaw, which is that it is part of the Linux hypervisor and is deployed as a kernel daemon thread which wakes up every 20 milliseconds to scan memory pages. When the process is operating with the crucial path-leading, the system's response time is enhanced by this frequent scanning, which causes a significant drop in system performance. The KSM performs several superfluous comparisons, which is a severe issue that boosts the system's load [21,22].

In overview, this paper contains some important contributions:

- **Reducing the number of unnecessary correlations:** This is focused on the duplicate-content demand zones' specific profile. The code segment with the highest likelihood of holding shared content is detected. Even if the stack and data segments have similar pages, their proportion is comparatively low [22]. As a result, code segment comparison is limited, and the number of comparisons required is reduced, resulting in improved speed.
- **Similar content page detection in offline mode:** A thorough examination of KSM reveals overhead profiling [23]. Several online comparisons during the execution of virtual machine instances will increase the system's response time, which is a factor of crucial performance benchmark in cloud computing. To address the problem, a modified Memory Deduplication of Static Pages (mSMD) technique is presented, which can identify duplicate content pages in an offline mode.
- **Classification based on Fuzzy Hashing and Genetic algorithm and Page comparison is limited to a single cluster of application**: Initially, all applications are clustered based on Fuzzy Hashing approach and Agglomerative hierarchical clustering, since similar applications have more likelihood of having more similar pages. Similar applications are identified using Fuzzy Hashing and clustered using Hierarchical Agglomerative Clustering. The pages of the code section of the clustered category of applications are segmented and similarity is identified between static pages which helps to reduce unwanted comparisons while maintaining the possibility of detecting pages with comparable information using Genetic algorithm approach and entered in the Multilevel shared pages' table. As a result, when pages are compared within the same category, the count of the number of several comparisons is reduced.

The remainder of the paper is structured as follows: In Section 2, the literature review and research motivations are elaborated. In Section 3, the proposed technique and various implementation levels are explained. In Section 4, the simulation framework and performance are described. Segment 5 provides the conclusion and future work of the research work.

2. Literature review

Garg A. et al. (2017) implemented a memory deduplication approach assisted by GPU in virtualized environments [24]. By finding and combining identical pages, content-based memory page sharing approaches increase the efficiency of memory in virtualized systems. Catalyst, the suggested solution, works in two phases: the initial phase, where the GPU processes pages' of virtual machines to select likely similar

pages' for sharing, and the next, where the GPU executes the actual sharing based on page-level similarity and inspects an intended group of shareable memory pages. In virtualization situations, opportunity-driven use of the GPU-generated sharing suggestions allows for quick and low-cost memory page sharing and duplication detection. The Catalyst is tested against a variety of benchmark approaches and workloads to show that it can accomplish more memory sharing in less time and at lower or equivalent compute costs than different scan rate settings of KSM.

Sioh Lee et al. (2017) implemented an advanced KSM approach for Cloud computing [25]. The authors presented a page grouping approach to decrease page comparisons in prior work, but it demands specific monitoring hardware. XLH improved memory page sharing by including information about guest virtual machine I/O operation. However, XLH's CPU overhead remains considerable, equivalent to that of the normal KSM. To CPU cycles. Based on the findings of the simulation, a KSM advanced for cloud computing (AKC) that uses fewer cycles of CPU than the default KSM was developed. Checksums based on the RB-tree structure were used to decrease the number of page comparisons. AKC also uses a hardware-accelerated CRC32 hash algorithm to reduce page checksum overhead.

Vano Garcia et al. (2018) performed a kernel randomization technique for cancelling memory deduplication [26]. The memory deduplication paradigm is rendered useless by the address space layout randomization (ASLR) safeguard method, is a key example. The authors of this research provided a detailed investigation of how the address kernel randomization technique affects the memory deduplication technique. The findings show that the memory cost for running 24 kernels increases by 534% when kernel ASLR is enabled (from 613 MiB to 3.9 GiB). In tests where the host is running 24 virtual machines, 44.89 % of the memory pages depend on kernel randomization. The stopping of memory sharing is due to these pages. The addresses of kernel components, which alter each time the computer boots, are among their contents. The memory deduplication technique is unable to share them when kernel randomization is enabled because each virtual machine's contents differ. The memory saving rate decreases by about 50% when the kernels of guest virtual machines are not randomly generated. As a consequence, the memory needed to run the kernel of the virtual machines is severely increased.

Santhosh Kumar et al. (2019), for secure memory deduplication, designed hardware-aided Copy-on-write defect discovery [27]. Side-channel attacks based on memory deduplication, on the other hand, are threats of disclosure of information and stealthy covert channel creation among virtual machines that can be deployed utilizing timing information provided by Copy-On-Write (CoW) fault management semantics. In terms of deduplication, the CoW semantic has been a necessary offensive because it not only supports guest OS visible deduplication but also provides a special channel for exploitation. They suggested CoWLight, a combination of hardware and software technologies for safely addressing CoW page faults, to lessen the noticeable access time difference between a conventional write and a write to a sharing page. In this study, they argued that rather than reducing the side effects, they should target security flaws at their root by assigning CoW fault management to the hardware itself. Additionally, it has been demonstrated that CoWLight significantly reduces access time delay disparities (by up to 30×), which is far below the noise thresholds in a moderately busy system.

Taehum Kima and Youngjoo Shin (2020) developed a way for mitigating memory-sharing-based side-channel attacks in the Binary for Cloud environment by encoding random data [28]. In this paper, they proposed a novel mitigation technique for memory-sharing-based side-channel attacks in cloud computing systems. With this technique, vulnerable apps are turned into robust, secure ones. A secret random value is explicitly inserted into an application's executable binary using binary instrumentation. The random value will make it impossible for applications from several other security domains to share the memory with security-sensitive programmes. The application-specific approach, on the other hand, offers system-wide memory deduplication, maintaining memory efficiency. They discussed the design and implementation of the recommended mitigation as well as the findings of the evaluation.

Albalawi et al. in 2021 [29] introduced a method of defence against side-channel assaults using a cloud computing feature called memory deduplication. The memory deduplication paradigm improves the efficiency with which physical memory is used by virtual machines running on the same server by maintaining only one copy of the libraries and other software required by several virtual machines in memory. However, because memory sites used by a victim's virtual machine and the associated operations are cached and can be accessed more quickly than memory sites not used by the victim, this enables the virtual machine of an attacker to learn about the memory locations and associated operations used by the victim's virtual machine. The hostile virtual machine must conduct an abnormal series of cache flushes to carry out the attack, which their novel technique detects by monitoring memory locations linked with sensitive operations like encryption and using logistic regression to identify the abnormal caching operations. By using its cache flushing feature, this strategy also disrupts the side channel, making it more challenging for the attacker to gather pertinent data. The tests, which were conducted

on physical servers running CentOS and Debian 10 and the KVM hypervisor and Ubuntu 18.04 LTS virtual machines, reveal that the technique can identify attacks with a 99 percent accuracy rate and can provide false information to an attacker with a 2%–8% CPU cost.

Divyanshu et al. (2022) performed memory deduplication for Serverless computing [30]. The resource utilization and user experience trade-offs imposed by Serverless platforms today are rigid. Operators need to make considerable resource sacrifices to obtain high performance due to the limited controls, which are supplied via switching sandboxes between cold and warm states and keep-alive. Medes, a serverless framework that the authors offer, breaks the trade-off that is rigid and enables operators to easily move through the trade-off space. Medes takes use of the high rate of memory footprint duplication in warm sandboxes running on serverless platforms. They make use of these duplicated chunks to build a new sandbox state known as a "dedup" state that is quicker to recover from than the cold state and uses less memory than the warm state. They created innovative methods to detect memory redundancy with the least amount of overhead and the smallest possible memory footprint for the "dedup" containers. Finally, they created a straightforward sandbox management strategy that exposes a constrained, user-friendly interface for operators to simultaneously regulate warm and "dedup" sandboxes to trade-off speed for memory. Extensive tests with a prototype utilizing actual serverless applications show that Medes can reduce end-to-end latencies by up to 1–2.75 times. Medes can provide a reduction in end-to-end latencies of up to 3.8 under memory pressure, which increases its advantages. Medes does this by minimizing the number of cold beginnings by 10–50% in comparison to the most recent baselines.

Andreas et al. (2022) evaluated the performance operational efficiency of Same domain memory deduplication [31]. Memory deduplication, an Operating System memory optimization method that combines similar memory pages into one Copy-on-Write (CoW) page, has been shown to be vulnerable to a number of timing side-channel attacks. These attacks all originate using the latency variations in write times to the CoW page and the normal unique page. By offering two case studies that demonstrate how an attacker can still use the side channel for deduplication to leak information, the authors of this research assessed the effectiveness of memory deduplication in same domain as mitigation. In the first case study, they looked into a client-server model in which a server must inevitably data from an unreliable source client and showed how the client may manipulate the memory's data alignment to reveal the server's confidential information. In the second case study, they look at the latest Firefox browser that has made significant ensuring that information from various origins is separated into different domains. They show that despite these efforts, a malicious webpage can still take advantage of Site isolation is only partially implemented by the browser, causing leaks of sensitive information across tabs. They concluded that same-domain memory deduplication is insufficient since it is challenging to perform correctly.

Jagadeeswari and Mohanraj [32], performed homogeneous batch memory deduplication. 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, this homogeneous approach merges more memory pages and implementation details demonstrate a significant increase in the number of pages shared and CPU consumption.

3. Modified static memory deduplication approach (mSMD)

In this research, a modified approach of Memory Deduplication of Static Memory Pages (mSMD), which is based on the identification of similar applications by Fuzzy hashing and clustering them using the Hierarchical Agglomerative Clustering approach, followed by similarity detection between static memory pages based on Genetic Algorithm and details stored in Multilevel shared page table, both operations performed in offline and final memory deduplication is carried out during online, is proposed for achieving performance optimization in virtual machines by reducing memory capacity requirements.

In Segment 3.1, the proposed model overview is discussed. In Segment 3.2, clustering of similar applications is explained. In Segment 3.2, classifications of pages statically into various categories are explained. In Segment 3.3, storing information of multilevel page table is explained. In Segment 3.4, a modified static memory deduplication technique is proposed for reducing unnecessary memory page comparisons.

3.1. Proposed model

In this section, a novel approach of modified static memory deduplication technique of static memory pages is proposed for reducing the requirement of memory capacity while response time is kept the same. It includes two main steps: (1) the overhead of response time and unwanted page comparisons are reduced; first all applications and all pages belong each cluster of applications are partitioned into various categories in accordance with the sample content. The processing power consumption is saved as the technique is performed in offline mode. The same sampled content is present in the pages that are in the same cluster. The number of total content samples is as large as

the number of categories present in the system. Since identical applications have similar content, first similar applications are categorized and code segment pages are segmented, and the similarity of pages is identified. In this step, identical pages are detected within each category. The selected candidate page is marked as a shared memory page if the page is identical and the shared memory page is linked to others. In this way, the memory pages present in the code section are analysed and the shared pages' are identified. The merging of shared pages is done after the process is scheduled online for alleviating the memory requirement and overhead response time is decreased. The modified SMD approach is demonstrated in Figure 1 which includes the above two processes.

3.2. Classification of applications

Due to a huge number of unwanted page comparisons, the KSM approach will not be efficient, since it consumes enormous CPU cycles. For this reason, all pages are classified into various categories to avoid unnecessary comparisons. The same category pages are having higher priority of being shared and other categories of different groups are not shared. Furthermore, the similarity detection of pages is limited based on their similar category. The implementation of this application classification is done in offline mode. The method of fuzzy hashing, often referred to as similarity hashing, is used to identify an application that is nearly identical to other applications but not quite. In contrast, cryptographic hash algorithms are made to produce dramatically different hashes for even the smallest variations. The following benefits of the clustering (or classification) method known as agglomerative hierarchical clustering (AHC) include: It begins with the differences among the items to be classified together. The type of dissimilarity can be appropriate for the topic being examined and the type of data being used. Algorithm 1 shows the grouping of similar applications using fuzzy hashing and clusters are formed by the Agglomerative hierarchical clustering approach.

Algorithm 1: Formation of Application Clusters.
Input: Given a dataset of applications of various virtual machines represented as.
(V1A1, V1A2,… .V1An, V2A1,V2A2… .VnAm, … ..VoA1,VoA2… .VoAx)
Output: Cluster of similar applications.
Step 1: Begin
for i 1 to p.
for j 1 to i
compute fuzzyhash (i,j)
if (% of similarity > 30)
each data point is a singleton cluster
repeat
merge two clusters having remain
End
Step 2: Return clusters.

3.3. Classification of pages of code section application clusters

Once the applications of virtual machines form a cluster, it is followed by the process of page classification based on a genetic algorithm, which is depicted in Algorithm 2. Here, the initial population is generated based on the candidate pages. The code section of applications installed in virtual machines is partitioned into pages of segment size 4k which is the default size and creates a new table for entry of the list. The fitness is evaluated based on the similarity of object dump of the code which is partitioned. Until the termination condition is over, the process is generated. Each sampled result is checked that is present in the population P and selected based on the fitness function and stored in the table as a new population.

Algorithm 2: Identification of static similar pages of applications
Input:
Step 1: P: Initial population of sampled results for the candidate page.
Step 2: T: Creation of new table, Tj: index position of j in Table T.
Output:
Step 3: Partition of code section of applications of VM into pages.
Step 4: Begin.
Select candidate pages based on Roulette wheel selection and partial selection for scanning.
Step 5: Fitness fitness evaluated based on the object dump of the two candidate pages using:
Calculate objdump of page1 as a.
Calculate objdump of page2 as b.
if diff (a,b) $==0$ then.
pages similar and select operator applied.
else.
pages dissimilar.
Step 6: Repeat until the stop condition is not met **do.**
Step 7: Return Similar pages.

3.4. Each application using multilevel shared pages table

For optimizing the systems response time, the cost is reduced for the detection online with minimum effect on the opportunities of sharing. For this reason, it is proposed the Multilevel Shared Page Table based on the page classification for each virtual machine. The multilevel shared page table is utilized for recording the memory pages of virtual machines which are having

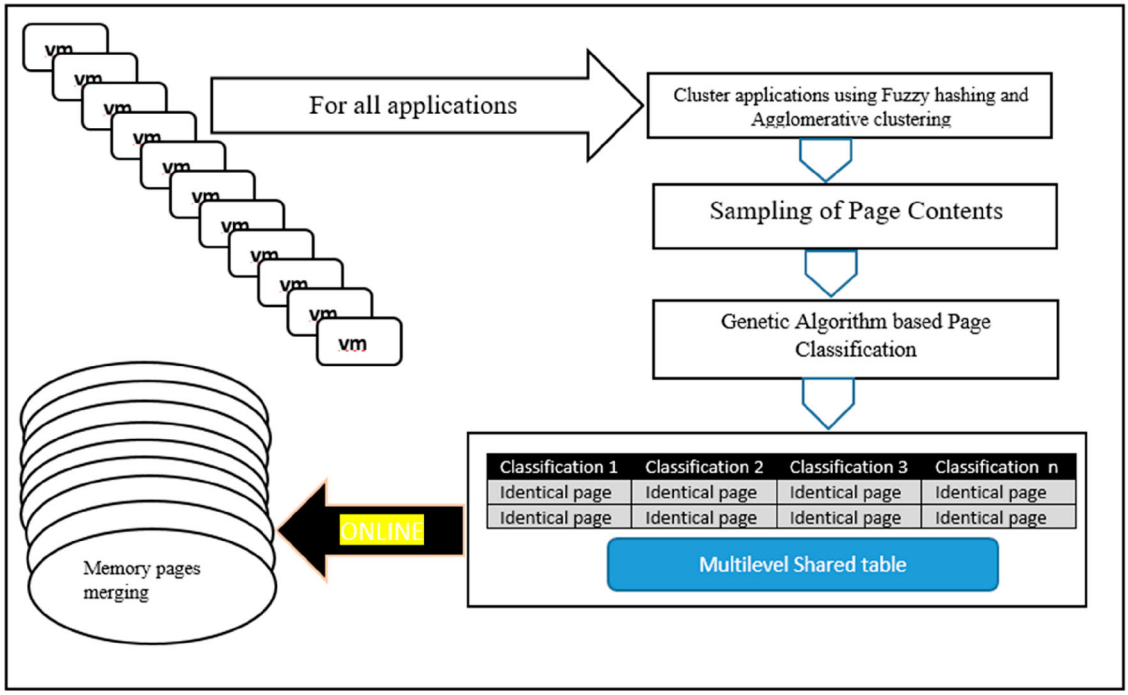


Figure 1. Framework of modified static Memory deduplication (mSMD).

similar contents to others. The size of the table is always greater than the frame size in which the shared pages' table is denoted as the variable size table, and if the shared pages of a virtual machine are higher, then the size of the table will be large or it will be small and null for no pages. Algorithm 3 shows the demonstration of the implementation by the proposed multilevel shared pages table for each cluster. In the First step, all identical code section pages are detected from the virtual machine. The one entry *i* is created in the page table which has similar content. If it has no identical pages then it will check for other pages with similar content. The process is done in offline mode and systems response time is not affected. Before scheduling, the memory pages that are shared by the applications of all virtual machines are all identified through this implementation algorithm.

Algorithm 3: Implementation Algorithm for Multilevel page table
Input:
Step 1: A: each application cluster.
Step 2: Ri: sampled result of the i pages.
Step 3: S: page table size.
Step 4: F**:** frame size in the page table.
Step 5: TL: shared pages' table.
Step 6: TLi: entry i in the table TL.
Output:
Step 7: Check S > F.
Step 8: while Ri **do** (i = 1 to n)
Classification of pages is found by PC, where Riε PC;
Detect page i = PC (i) (detection of pages with similar content)
if a page of **then**
Create a single entry in TLi
Pages with similar content stored in TLi
Continue;
else
Next page detected from Ri Continue;
end if
end of the while

3.5. Memory deduplication process

The final step is to execute memory pages' merging after the creation of the shared memory pages' table for all the virtual machine instances online, i.e. when the virtual machines are switched on. The request for memory capacity is alleviated by performing this step. The implementation pseudo-code is shown in algorithm 3. Here, all pages are checked for the VMs and each page is analysed for its presence in the shared page table that is processed for merging. Furthermore, the identical content in the pages from the list is founded and the page table entry is revised by making a page entry with other shared pages that are containing identical contents. If the entry is not found identical, then the next page of VMs is checked.
Algorithm 4: Memory Deduplication Implementation
Input:
Step 1: A:virtual machine is running.
Step 2: Ri: page i.
Step 3: TL: shared pages' table.
Step 4: TLi: entry i in the table TL.
Output:
Step 5: while Ri **do** Check Pi in TL

If Pi in TL **then**
Find pages' entry TLi in the table;
Find pages in TLi; if available in TL, process merge
else
end if Entry of Ri is revised in the table for the A;
Continue;
Continue;
end of while

4. Simulation framework and performance results

4.1. Simulation framework

Table 1 shows the settings of the experimental setup and workloads of the framework given in Table 2.

4.2. Performance of mSMD

The system performance is defined by the metric known as system throughput. The performance measure is calculated as given in the below equation.

$$Performance = \sum_{i} \frac{IPC_i^{shared}}{IPC_i^{alone}} \quad (1)$$

The instruction per cycle is defined as IPC_i of virtual machine i. Furthermore, IPC_i^{shared} denotes the same with running parallel to different VMs, and IPC_i^{alone} is running alone in the system, where i represent each virtual machine.

KSM is a thread of Linux hypervisor, which periodically wakes up every 20 ms which enhances the processing power. The normalized system performance increase for various arrangements with the proposed mSMD having 4 G and 8 G memory without deduplication carried out is shown in Figure 2. The different configurations are given on the *x*-axis and VMs running parallelly with various combinations and entire physical

Table 1. Experimental configuration setup.

Variables	Characteristics
CPU Processor	i5 Processor, Intel Core 8th Generation
Data Cache/L1 instruction	256 KB, 4 – way
Cache Memory – L2	1024 KB, 8 – way
Cach Memory – L3	8MB, 16 – way
OS & Hypervisor	Linux – Ubuntu, KVM – Kernel Based Virtual Machine QEMU – Quick Emulator, virsh
OS for VM	Ubuntu 16.0

Table 2. Simulation workloads.

Variable	Values
.Net app	Run dot net for creating web applications
Apache Server	Run the ab[] with 24 concurrent-request on Apache HTTPd Server
MySql Database	Run the MySQL database with SysBench[] in guest VMs
Genymotion	Run Genymotion (Android Emulator) installed on Virtualbox

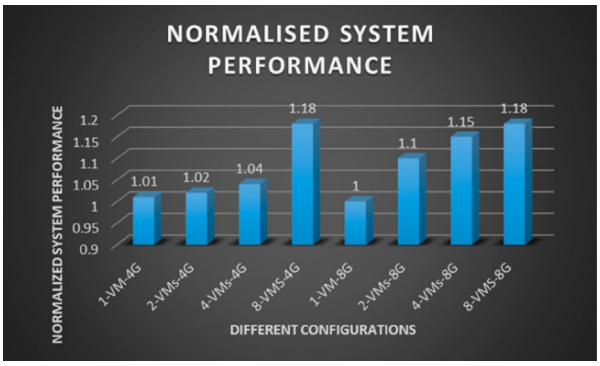


Figure 2. Performance increase of virtual machines with different Guest OS.

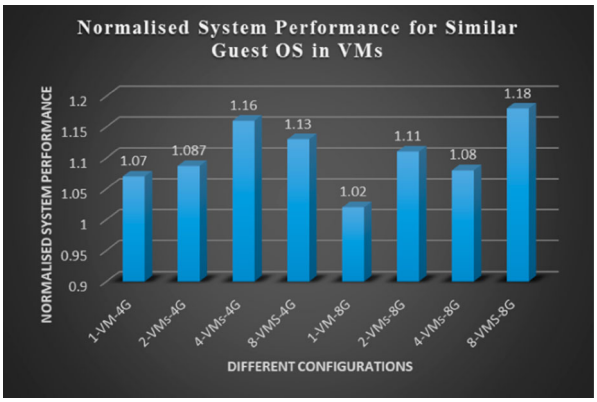


Figure 3. Performance for simultaneous VMs for same OS.

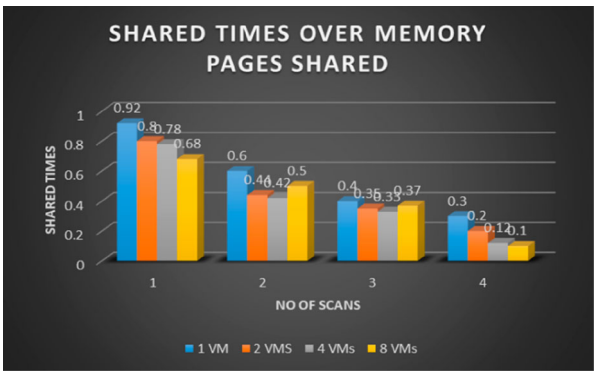


Figure 4. Physical memory pages' proportion for various times of shared.

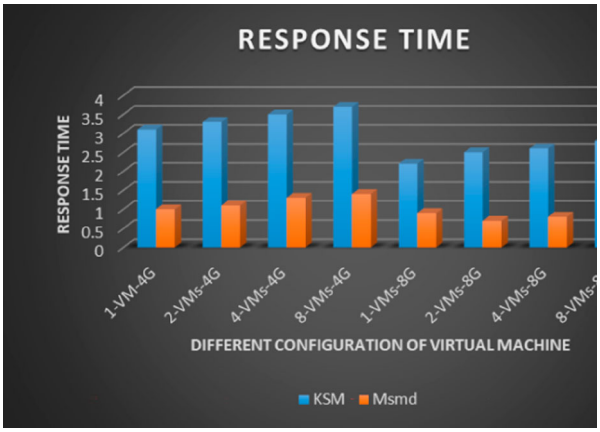


Figure 5. Response time of proposed mSMD.

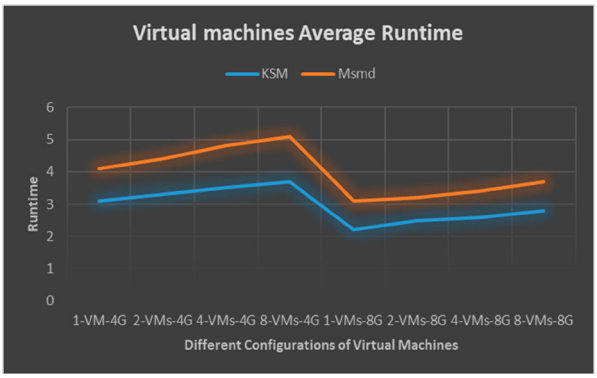
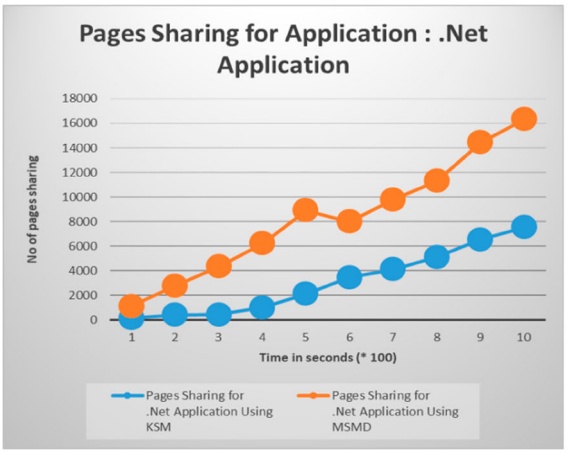


Figure 6. VMs average runtime.

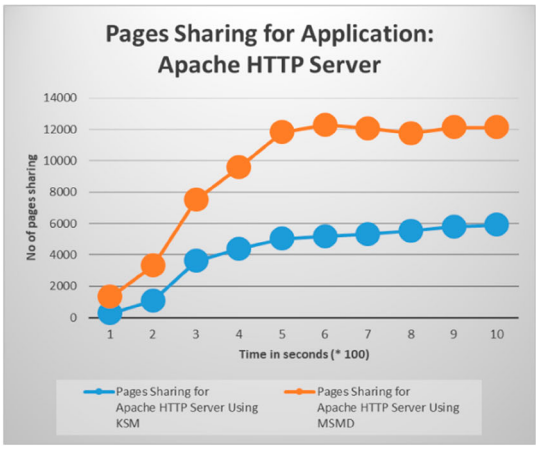
memory size. The n virtual machines are represented as n-VMs-mG that run upon a parallel server with mG memory. Different operating system is included in each VM. Better performance of the system is obtained when a similar amount of memory is given for different VMs with the proposed mSMD approach.

Figure 3 shows the performance improvement of normalization when the same OS is running simultaneously on the virtual machines. Using different OS lead to a decrease in the performance while running the same given higher. This is due to several identical contents present in various systems which lead to the opportunity for finding shared contents. The performance of the system is increased by alleviating the memory capacity request.

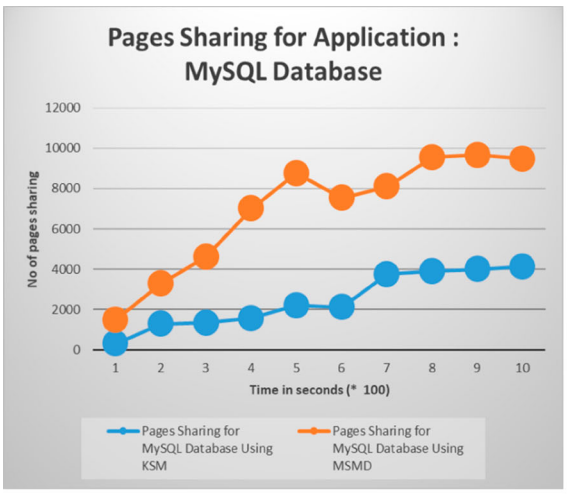
The performance shared times with regard to the percentage of physical pages of various virtual machines shown in Figure 4. For the systems' shared pages, an increase in the count of running virtual machines will lead to more pages' sharing inside the system. If pages' sharing is increased in the system, then the shared pages' can be combined for alleviating the system's memory capacity. Hence, Figure 4 shows the



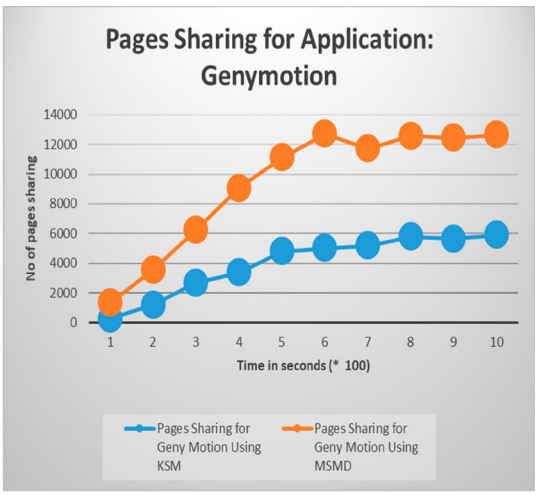
(a) : Pages' sharing: .Net Application



(b): Pages' sharing: Apache HTTP Server



(c):Pages' sharing: MySQL database



(d) :Pages' sharing: Genymotion

Figure 7. (a) Pages' sharing: .Net Application. (b) Pages' sharing: Apache HTTP Server. (c) Pages' sharing: MySQL database. (d): Pages' sharing: Genymotion.

better performance of the proposed mSMD approach for the simultaneous running of virtual machines.

4.3. Optimization of time of response mSMD

The shared memory pages' are detected by using the proposed mSMD algorithm which is performed through offline mode. The online detection using the Linux KSM is different from this offline approach. Compared to KSM the proposed mSMD approach has more improvement with regard to response time.

The system normalized average response time in comparison with other approaches like KSM shown in Figure 5. The response time results show that the mSMD gives a low response time for different configurations. The average runtime completion of virtual machines or applications is shown in Figure 6. The proposed approach shows the shortest runtime when compared with other approaches which give better performance. The KSM has higher runtime when compared with mSMD. From these results, the deduplication of memory is valuable in tweaking systems performance measures when more virtual machines or applications execute concurrently.

4.4. Unnecessary comparison and memory capacity request of mSMD

The memory page sharing opportunities in which the maximum is obtained through long – time process is shown in Figure 7(a,b,c,d). The proposed mSMD approach detects memory page-sharing opportunities in offline mode. Therefore, the proposed approach detects the pages, before running the applications or VMs. The page sharing opportunities on Dot Net applications shown in Figure 7(a), Apache Server shown in Figure 7(b), MySQL database shown in Figure 7(c), and Genymotion shown in Figure 7(d). The page detection opportunities are compared with the traditional KSM approach when running the particular application.

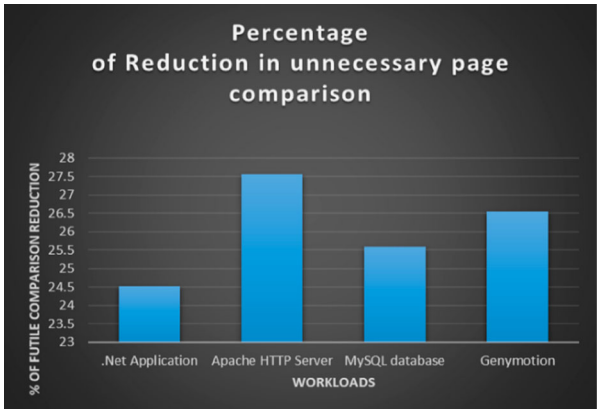


Figure 8. Rate reduction percentage for four VMs.

Figure 8 shows the proposed mSMD for the percentage of unnecessary page comparison rate reduction. The proposed work decreases about 28% of unnecessary futile comparisons. The reduction is shown for the workloads due to the page content samples of mSMD for each application.

5. Conclusion

In this research, a modified Static Memory Deduplication (mSMD) approach is proposed for reducing the requirements of memory demand for performance optimization in cloud computing. The proposed work includes major three steps: (1) Similarity between application performed using the Fuzzy hashing approach followed by Clustering of applications based on the Agglomerative Hierarchical Clustering approach; (2) The code section of various applications are segmented and similar static pages are identified using Genetic Algorithm methodology. (3) Multilevel shared page stable implementation is proposed for reducing the repeated comparisons that should affect the sharing opportunities; (3) The memory deduplication is performed after building shared pages' table for every VM online, i.e. 4e after switching on of virtual machine. This leads to alleviating the requirements of memory capacity. The major innovation of the proposed mSMD is that the performance is improved by executing the process of similar page detection of similar applications carried out offline and memory deduplication was carried out online. Through the simulation results, it is identified that the performance is improved and the requirement for memory capacity is reduced when using the proposed mSMD technique. The future work includes the grouping of virtual machines with similar operating systems and classifying static memory pages and performing a memory deduplication approach.

Acknowledgment

The author with a deep sense of gratitude would thank the Almighty for its blessings for this effort of article writing and completion.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] Xu F, Liu F, Jin H, et al. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. Proc IEEE. 2013;102(1):11–31.
[2] *Customer Success. Powered by the AWS Cloud.* [Online] Available: http://aws.amazon.com/solutions/case-studies
[3] *Amazon Elastic Compute Cloud (Amazon EC2).* [Online] Available: http://aws.amazon.com/ec2/

[4] https://customerthink.com/top-10-companies-using-cloud-and-why/
[5] Goldberg RP. Survey of virtual machine research. Computer (Long Beach Calif). 1974;7(6):34–45. doi:10.1109/MC.1974.6323581
[6] He S, Chen J, Li X, et al. Mobility and intruder prior information improving the barrier coverage of sparse sensor networks. IEEE Trans Mob Comput. 2013;13(6):1268–1282.
[7] Mutlu O, Subramanian L. Research problems and opportunities in memory systems. Supercomput Front Innov. 2015;1(3):19–55.
[8] Kong Y, Zhang M, Ye D. A belief propagation-based method for task allocation in open and dynamic cloud environments. Knowl Based Syst. 2017;115:123–132.
[9] Waldspurger CA. Memory resource management in VMware ESX server. Oper Syst Rev (ACM). 2002;36 (SI):181–194. doi:10.1145/844128.844146
[10] Gupta D, Lee S, Vrable M, et al. Difference engine: harnessing memory redundancy in virtual machines. Commun ACM. 2010;53(10):85–93.
[11] Xia Z, Wang X, Sun X, et al. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Trans Parallel Distrib Syst. 2016;27(2): 340–352. doi:10.1109/TPDS.2015.2401003
[12] Ren YJ, Shen J, Wang J, et al. Mutual verifiable provable data auditing in public cloud storage. J Internet Technol. 2015;16(2):317–323.
[13] Fu Z, Sun X, Liu Q, et al. Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing. IEICE Trans Commun. 2015;98(1):190–200.
[14] Liu Q, Cai W, Shen J, et al. A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment. Secur Commun Netw. 2016;9(17):4002–4012. doi:10.1002/sec.1582
[15] Chen L, Wei Z, Cui Z, et al. CMD: classification-based memory deduplication through page access characteristic. ACM SIGPLAN Not. 2014;49(7):65–76. doi:10.1145/2674025.2576204
[16] Jia G, Han G, Wang H, et al. Cost aware cache replacement policy in shared last-level cache for hybrid memory based fog computing. Enterp Inf Syst. 2018;12(4):435–451. doi:10.1080/17517575.2017.1295321
[17] Jia G, Han G, Rodrigues JJPC, et al. Coordinate memory deduplication and partition for improving performance in cloud computing. IEEE Trans Cloud Comput. 2019;7(2):357–368. doi:10.1109/TCC.2015.2511738
[18] McMillan C, Grechanik M, Poshyvanyk D. Detecting similar software applications. 34th International Conference on Software Engineering (ICSE); 2012. p. 364–374
[19] Reyhani Hamedani M, Shin D, et al. Androclass: an effective method to classify android applications by applying deep neural networks to comprehensive features. Wireless Commun Mobile Comput. 2018: 21, Article ID 1250359.
[20] Sarantinos N, Benzaïd C, Arabiat O, et al. Forensic Malware analysis: the value of fuzzy hashing algorithms in identifying similarities. 2016 IEEE Trustcom/BigDataSE/ISPA; 2016. p. 1782–1787.
[21] Borate M, Kumar A, Agrawal A, et al. U.S. Patent Application No.16/171,010, 2019.
[22] Jia G, Han G, Wang H, et al. Static memory deduplication for performance optimization in cloud computing. Sensors. 2017;17:968.
[23] Veni T, MarySairaBhanu S. Mdedup++: exploiting temporal and spatial page-sharing behaviors for memory deduplication enhancement. Comput J. 2016;59(3): 353–370. doi:10.1093/comjnl/bxu149
[24] Garg A, Mishra D, Kulkarani P. Catalyst: GPU-assisted rapid memory deduplication in virtualization environments. VEE '17: Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments; 2017. p. 44–59.
[25] Lee S, Kim B, Kim Y, et al. Akc: advanced KSM for cloud computing. SoCC '17: Proceedings of the 2017 Symposium on Cloud Computing; California, Santa Clara; September 2017. p 634.
[26] VañóGarcía F, Marco-Gisbert H. How kernel randomization is canceling memory deduplication in cloud computing systems. 2018 IEEE 17th international symposium on network computing and applications, Cambridge, MA, USA; November 1–3. 2018.
[27] Santhosh Kumar T, Mishra D, Panda B, et al. Cowlight: hardware-assisted copy-on-write fault handling for secure deduplication. Hasp '19: proceedings of the 8th international workshop on hardware and architectural support for security and privacy, New York, USA; Article No.: 3, June 2019. p 1–8.
[28] Kim T, Shin Y. Poster: mitigating memory sharing-based side-channel attack by embedding random values in binary for cloud environment. Asia CCS '20: proceedings of the 15th ACM Asia conference on computer and communications security; Taipei, Taiwan; October 2020. p 919–921.
[29] Albalawi A, Vassilakis V, Calinescu R, et al. Memory deduplication as a protective factor in virtualized systems. Applied Cryptography and Network Security Workshops. ACNS 2021. Lecture Notes in Computer Science; vol 12809. Springer, 2021.
[30] Saxena D, Ji T, Singhvi A, et al. Memory deduplication for serverless computing with medes. Seventeenth European conference on computer systems (EuroSys '22), RENNES, France; April 5–8, 2022; ACM, New York, NY, USA, 2022.
[31] Costi A, Johannesmeyer B, Bosman E, et al. On the effectiveness of same-domain memory deduplication. 15th European Workshop on Systems Security (EUROSEC '22), RENNES, France; April 5–8, 2022; ACM, New York, NY, USA, 2022.
[32] Jagadeeswari N, Raj VM. Homogeneous batch memory deduplication using clustering of virtual machines. Comput Syst Sci Eng. 2023;44(1):929–943. doi:10.32604/csse.2023.024945