Journal of Parallel and Distributed Computing 139 (2020) 18–31

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

QuickDedup: Efficient VM deduplication in cloud computing environments

Shweta Saharan [a,*], Gaurav Somani [a], Gaurav Gupta [b], Robin Verma [c], Manoj Singh Gaur [d], Rajkumar Buyya [e]

[a] Central University of Rajasthan, Ajmer, India
[b] Ministry of Electronics and Information Technology, New Delhi, India
[c] Indraprastha Institute of Information Technology, Delhi, India
[d] Indian Institute of Technology, Jammu, India
[e] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 5 May 2019
Received in revised form 23 November 2019
Accepted 8 January 2020
Available online 28 January 2020

Keywords:
Deduplication
VM disk image
Storage
Hashing performance

ABSTRACT

Deduplication is one of the major storage optimisation techniques for Virtual Machines (VMs) in cloud environment. Usually, hashing of blocks helps in identifying duplicate data blocks. This paper proposes a novel deduplication approach, QuickDedup that reduces the overall deduplication time, metadata overhead and the number of hash computations, and subsequent comparisons for the VM disk images. In addition to minimising the deduplication related metadata, which is a necessary by-product useful in checking deduplication, QuickDedup, follows novel byte comparison scheme to prepare various block classes. This way, QuickDedup eliminates or minimises the need for hash calculation and subsequent comparisons. QuickDedup performs the calculation and comparisons of hashes within the respective categories only. QuickDedup saves the space required for hash storage during deduplication and makes deduplication of VM disk images much faster. We conducted a detailed evaluation of QuickDedup on various metrics with different kinds and sizes of VM images taken from publicly available datasets. The evaluation results show a substantial improvement of up to 96% in the overall deduplication time required to deduplicate VM images apart from significant savings in metadata and storage overhead.

1. Introduction

Cloud Computing adoption has grown rapidly due to several advantages and features such as multi-tenancy, higher utilisation of servers, energy efficiency and elasticity derived from on-demand utility computing services [32]. The creation and deployment of VMs are quick, added with the capability of easy scalability. The growing popularity of VMs has made ways for the VM appliances [33], which are pre-configured VM images that readily run on a hypervisor. The benefits of VM appliances over traditional software packaging include simplified deployment and enhanced isolation [11]. As Cloud Computing and Big data are becoming more prevalent, an increasing amount of data is stored in the cloud. According to Gartner, enterprise IT's biggest challenge today is double-digit data growth. In fact, data is growing in enterprise storage banks at the alarming average and will increase up to 50 times in next decade as predicated.

Deduplication technology helps in handling data growth to a large extent. It is a data compression technology to eliminate redundant/duplicate data from the storage [3,19,23]. VMs are large in size, which exacerbates the storage problem. There are multiple contributors, who have provided storage optimisation solutions utilising deduplication techniques [14,35]. The common element of many of these techniques is to calculate the hashes of small chunks of data known as blocks. A hash comparison helps in identifying common blocks. There are various ways of performing deduplication, and broadly it is categorised into three categories, i.e. based on time, level and location [13]. Deduplication can either follow a source-based or target-based approach based on the location of data storage. In source based deduplication, redundant data is eliminated at the client-side before sending the data to the server reducing the overall bandwidth cost. In target-based deduplication, the server performs the deduplication process utilising higher bandwidth as the redundant data also gets transferred over the network. Based on the level at which the similarity is exercised, the deduplication can be either at byte-level, block-level or at the file-level. Byte-level deduplication process compares bytes to eliminate redundant bytes. On the other hand, block-level deduplication eliminates redundant block and comparison takes place at the block level. On the basis of time, it can either be post-process or inline deduplication. In post-process or offline

* Corresponding author.
E-mail address: shweta.17oct@gmail.com (S. Saharan).

https://doi.org/10.1016/j.jpdc.2020.01.002
0743-7315/

deduplication, there is no computation before storing the data, ensuring better storage performance. However, duplicate data is stored for a short time which can be an issue if the storage system is near full capacity. In-line deduplication requires less storage as it does not store duplicate data. However, computation takes time which may degrade the storage performance. As cloud environment uses pay-as-you-go service model, time becomes an important factor and cloud resources are enormous therefore no issue of storage being full. Therefore, an offline algorithm is preferable for VM deduplication in the cloud. Deduplication can be source based or target based, depending upon the location of data storage. From the perspective of deduplication, VM disk images are classified as flat and sparse VM disk images [13,46]. Popular hypervisors, such as Xen and KVM support sparse format *qcow2* and virtual box supports *vdi* sparse format. Deduplication is one of the ways to optimise VM storage over the cloud [24,46]. Deduplication process helps in saving storage space to a large extent in VM disk images. There are many other areas, where deduplication is useful like in backup systems, databases, and networks [10,19].

Deduplication in a traditional file system leads to higher storage utilisation and improves the disk cache efficiency [48]. When considered from virtualization perspective, deduplication offers additional benefits like supporting multi-tenancy and reducing the effects of VM sprawl problem. For each user's VM, there exists a VM disk image that stores OS image, applications data, and other free blocks. Each VM when individually stored occupies huge space on VM storage, which can be optimised using deduplication. Deduplication can be done either on the VM image (intra-VM image deduplication) or among different VMs (inter-VM image deduplication).

Surveys by AFCOM [1] and COMMVAULT [2] shows that over 63% of data centre surveyed have seen tremendous growth in their storage costs. However, in the present scenario, the deduplication ratios are not rising proportionately to the data growth. Therefore, a deduplication technique should be efficient and fast enough to deduplicate the storage even when the deduplication ratio is low or moderate.

In this paper, we focus at the important factors contributing to the overall deduplication process and identify that a major amount of effort for deduplication is spent on identifying exactly same disk blocks. Hash calculation of disk blocks and subsequent comparison among them helps in identification of exactly same blocks. We argue the overall deduplication time can be minimised by minimising the number of hash calculations, and subsequently the eligible blocks for comparisons. Our proposed approach QuickDedup utilises a novel byte-to-byte comparison scheme to reduce the number of candidate blocks for hash calculations and subsequent comparisons. We perform a number of performance evaluation experiments to evaluate the proposed approach which show that it is much faster than the pure hash-based technique, maintaining the same deduplication ratio.

1.1. Contributions

The following are the major contributions of our work:

1. Deduplication being an important performance optimisation for cloud and VM storage, we collate a primary list of important requirements for deduplication efficiency.
2. Through extensive experiments on VM disk image datasets, we observe that the major factor in deduplication process is hash computation of each block and subsequent comparison among them to identify duplicate blocks. We design QuickDedup, where at initial stages of preprocessing, we make byte-to-byte comparisons to identify deduplicable blocks, which allows to reduce the total number of hash-based comparisons quickly.
3. We propose a set of byte comparison strategies with different aspects of their suitability with the types of disk blocks such as filled, zero blocks, and partially filled blocks.
4. QuickDedup with the help of a novel meta-data structure, Deduplication Data Tree (DDT), provides savings by minimising the number of hashes (up to 95%) and the overall deduplication time (up to 96%). These savings also helps in reducing the meta-data overhead during the deduplication process.

1.2. Organisation

The rest of the paper is organised as follows: Section 2 provides a deduplication background listing the necessary criterion for an efficient deduplication technique. Section 3 comprises of literature survey and discusses various file systems that implement deduplication for VMs and traditional systems. In Section 4, we propose the QuickDedup algorithm and its design in detail. Section 5 consists of experiments and their results to test the efficiency of our proposed technique. Section 6 discusses various cases where QuickDedup provides best and worst case performance. In Section 7, we conclude and propose future work for further improving deduplication in context of different operating systems.

2. Deduplication: Background

There are a large number of deduplication techniques proposed in the literature. However, a unified set of requirements to test and verify the suitability of a deduplication technique are missing. Before proceeding towards identifying the key requirements, we provide an overview of the deduplication ratio calculation among inter and intra-VM Images. Let us consider that size (V_i) denotes the original size of a VM and *dedup_size*(V_i) represents the size of a VM after deduplication where i = 1, 2, 3 $\cdots$ k images. Eq. (1) gives the deduplication ratio of intra-VM image deduplication of *i*th VM. Inter-VM image deduplication ratio for VM images from V_1 to V_k is given by Eq. (2).

$$\text{Deduplication\_Ratio}(V_i) = 1 - \frac{\text{dedup\_size}(V_i)}{\text{size}(V_i)} \tag{1}$$

$$\text{Deduplication\_Ratio}(V_1, V_2, \ldots, V_i) = 1 - \frac{\text{dedup\_size}(V_1 + V_2 + \cdots + V_k)}{\text{size}(V_1 + V_2 + \cdots + V_k)} \tag{2}$$

Critical requirements for an efficient deduplication technique are listed below:

1. **Least number of hashes:** The key task in a deduplication system is to determine whether two or more blocks are same or not. A preferred way to do this is by calculating and comparing hashes. A hash calculation, being an expensive task as the number of hashes grows, the metadata increases. Hash calculation also requires a considerable amount of time. Hash calculations should be minimised to save space, time and computational overhead.
2. **Least number of comparisons:** VM disk images are of multi-gigabytes in size. Before storing an incoming VM, it is determined whether any block of that VM already exists in the storage. For this, hash calculation and comparison are performed, among incoming VM blocks and the already stored blocks. Inline deduplication calculates performs hash calculation and comparison before storing the VM disk image that mostly affects the storage performance.

Even if the client side computes the hashes, their comparisons are carried out at the time of storage only. Therefore, the number of comparisons should be minimised to improve the storage performance.

3. **Minimum Metadata:** A large amount of additional (sometimes temporary) data is generated during a deduplication process. This data includes the hashes generated for each block; the copy created for the block, when a VM accesses a shared block. Since deduplication is performed to optimise the use of available storage, the overhead of additional data generated should be minimised.
4. **Optimum block size selection:** Selection of appropriate block size for deduplication is a vital factor that affects the deduplication ratio [13,22]. On increasing the block size, the deduplication ratio falls and opting for small block size leads to difficulty in managing a large number of blocks and proportionately increased hash calculations. The block size should be chosen such that, a good balance between both these factors can be achieved.
5. **Fast Retrieval:** VM can be created and deployed quickly, however deduplication may cause fragmentation in VM storage. At the time of retrieval of a VM, it becomes slower since the VM blocks do not remain sequential because the common block of different VMs gets stored at a single location. The blocks of first VM are stored sequentially, afterwards, if next VMs have deduplicate blocks then those blocks are not stored, only a reference is passed. The deduplicable and non-deduplicable blocks of a VM are stored at a separate location in the storage. For backup systems, it can be made faster using RevDedup [20].

3. Related work

There are various file systems which implement deduplication strategies to improve storage efficiency. Deduplication is applied in several system domains viz. databases, VM disk images, data files and network data, which in turn benefits both the user and the resource provider. LIQUID [46] and LiveDFS [21] are file systems, which implement deduplication for VMs. LIQUID, a VM deduplication file system, along with deduplication offers features as instant cloning, on-demand fetching, low storage consumption, P2P data transfer of data and caching with local disks. Fingerprints are calculated using MD5 and SHA-1 for all blocks. In QuickDedup, we focus on early detection of deduplicable blocks and minimise the total number of hash calculations. LiveDFS, along with deduplication comprises of prefetching of metadata, spatial locality, and journaling. The main idea here is to place metadata and actual data nearby, for reducing seek overhead. The metadata consists of fingerprints of blocks present in the block group. Fingerprint filter and fingerprint store are being introduced for speeding up the hash comparison task, whereas we minimise the number of hash comparison by categorising the blocks into various categories based on the byte read due to which inter-category hash comparisons are never required. We only perform intra-category comparisons. VMDedup [26] detects duplicacy among the memory pages by comparing the hashes of the pages. Hash is calculated for all the pages, however for disk deduplication, the number of hashes are comparatively high.

Apart from VMs, deduplication applies to different file systems and fields. ZFS and Opendedup are file systems, which support inline deduplication. However, the memory requirement of both of these filesystems is quite large as they calculate fingerprints of all blocks. MAD2 [34] and HYDRAstor [6] proposes distributed deduplication architectures. Former uses scalable secondary storage and bloom filter, which is not suitable for VM image storage and later uses hash table. Lithium [9] is a cloud-based VM image storage system that aims for fault tolerance, but it does not consider deduplication. In Chord [29] and Droplet [44], the data is distributed over virtual nodes, which is migrated when needed to balance the load. Extreme Binning [4] makes use of file similarity for performing deduplication rather than chunk locality. Singleton [27] increased the memory usage efficiency by solving the problem of double-caching i.e. pages are cached both at the hypervisor and the VM. DBLK [30], a deduplication based primary storage system used multilayer bloom filters for reducing the data structure in the memory for indexing. For improving the performance of searching hash, Zhu et al. [48] had utilised spatial locality of data. iDedup [28], an inline storage deduplication system that minimised the I/Os and seeks using spatial locality and temporal locality in the access pattern for optimisation. DeDE [5], a decentralised deduplication system that performs block-level deduplication atop an existing file system. HPDV [17] parallelise the resource-intensive chunking process, by making use of idle CPU. It divides the globally shared fingerprint into sub-index based on the OS on the VM images to parallelise I/O. These existing technique are either not suitable for VMs being large in size, or calculates the hashes for all blocks to check duplicacy.

In many cases, deduplication is also used to speed up the backup process in VMs [8,41,45]. Upadhyay et al. [31] provided a deduplication approach that makes use of "Binning" at an initial stage based on block size. KSM (Kernel Samepage Merging) [3] is a deduplication feature that periodically scans the registered areas of user memory and looks for pages of identical contents to be replaced by a single page that is write-protected. These are granularity, locality, timing, indexing, technique, and scope. Zhang et al. [42] increases the efficiency of the VM-centric backup service by making use of the VM-centric file system block management which increases the VM snapshot availability. PDedup [36] makes use of both inline and post process deduplication. It estimates the temporal locality of data streamed and allocate the cache based upon priority. In an analysis performed over 525 VM Images, Jayaram et al. [12] showed that an image is more likely to be similar to a small subset in a repository and fixed size chunking is more appropriate for the VMs. Ethan et al. [13] have studied various factors that affect deduplication in VMs showing that fixed sized chunking is better for VM disk images. The choice of operating system of the VM has a huge impact on the deduplication ratio. Another important contribution in this direction is by Bloom filters. Bloom filter is capable of efficient and fast searching, uses probabilistic approach which leads to false positive results. When considered for large VMs over the cloud, the data is enormous. If bloom filter is being used there, then chances of a false positive result will increase to a large extent. QuickDedup uses fixed size block chunking as, fixed size chunk yields best result in terms of deduplication ratio in case of VMs.

DARE [38] made use of adjacency based resemblance detection for performing deduplication for backup systems. Their work aims to find the best possible blocks for delta compression in low overhead. Li et al. [16] improve the efficiency of the backup storage system by imparting both the inline and out-of-line deduplication. As deduplication causes fragmentation that degrades the read performance, Mao et al. [18] worked to improve the read performance. Ddelta [40] provides fast chunking algorithm and greedy byte-wise scanning to find more redundancy using delta compression. Fu et al. [7] make use of application awareness for improving data deduplication efficiency. It provides a balance between saving cloud storage capacity and duplication time. P-Dedupe [39] aims to minimise the time required by a deduplication process by parallelising the hash calculation process, however hashes are calculated for all blocks. Asymmetric

Extremum [43] based on content defined chunking provides a fast chunking algorithm with low computational overhead and higher throughput efficiency. Koller and Rangaswami [14] improved the I/O performance during deduplication by exploiting content-based similarity. SiLo [37] and Zhou et al. [47] utilised both locality and similarity of the content for attaining high deduplication throughput. HANDS [35], based on access pattern, placed correlated data together so that it can be atomically accessed when needed, reducing the memory access. Seo and Lim [25] used NAND flash memory to reduce computational overhead, and depending upon the casualty between I/O. Our work focuses on improving the efficiency of the out-of-line deduplication performed on VM disk images.

As per the literature, the majority of the approaches make use of pure hash-based approach for checking similarity of the blocks. Our novel approach instead of calculating hashes of all the blocks, based on byte comparisons, categorises the blocks. Along with the reduction in deduplication time, QuickDedup also minimises the metadata generated by eliminating the need to calculate the hash of all the disk blocks. In addition, most of the existing deduplication approaches are specific to backup, network or VM. Though, we evaluate and study QuickDedup for VM images, it is equally applicable to any storage domain with deduplication needs.

4. QuickDedup: An new approach for efficient deduplication

An efficient deduplication technique for VM is the one that follows the criteria listed in Section 2. Based on these, we propose QuickDedup, which is useful for VM appliance stores and VM disk backup/snapshots. QuickDedup is an offline algorithm where at initial stage complete VM image is stored. At a later stage, the deduplication process is invoked, which checks for the presence of deduplicable blocks. Initially, we store the VM image sequentially. During the categorisation process, only the block numbers are placed in different categories, whereas the data blocks remain intact. Once QuickDedup completes the processing, the space corresponding to the blocks numbers which fall under the "deduplicable blocks" is freed. We perform hash calculation using SHA-1 which can be replaced by any other hash algorithm based on the requirements. QuickDedup is specially applicable for VMs, as there are various parameters regarding QuickDedup which we have decided based on VMs. In the case of block size selection, we selected the optimum block size which is best for VMs. We follow fixed-size chunking which is usually better for VM disk images as discussed in [13]. However, QuickDedup can be applicable to general storage or file systems after the selection of the best suitable parameter based on general storage or filesystem.

4.1. Stages of algorithm

Broadly, the QuickDedup algorithm has two stages: Preprocessing stage and Deduplication stage. Further, deduplication stage has various functions to perform the overall deduplication activity. For easy understanding of the algorithm, Fig. 1 depicts the algorithm flow using a flowchart.

4.1.1. Preprocessing stage

In this stage, QuickDedup reads the first byte of each block and based on its character value; QuickDedup places blocks into different categories. Each category is named on a single character which identifies all the blocks having first byte same as the name of the category. The categorisation process places blocks into various categories based on their first byte character. This process reduces the number of blocks eligible for hash calculations and hash comparisons. Many blocks get classified as unique blocks on the basis of a single byte read. Therefore, for those blocks, the need for deduplication (having steps such as hash computation and subsequent comparison) does not arises. With this stage, one "pass" of the QuickDedup algorithm is completed.

4.1.2. Deduplication stage

At this stage, QuickDedup further processes only the categories having more than one block. Rest of the categories, having only one block, show that those blocks are unique. We term "Pass" as a complete phase when all the blocks of a VM image are processed and categorised into a specified category based upon the value of the byte selected in that pass. For next "pass" another byte position is read depending upon the byte selection strategy explained in Section 4.2. For all the blocks of these categories, QuickDedup reads the next selected byte and depending upon the value of the selected n^{th} positioned byte in each non-unique block, we perform another categorisation. This process repeats for 'M' passes. At the end of the last pass (M^{th}), for all the categories having more than one blocks, hash calculation and comparison of those blocks within that category is made. Number of passes (M) is a crucial performance factor which we discuss later in Section 5.3.

4.2. Byte selection strategy

In the first pass, QuickDedup reads the first byte of each block and for all subsequent passes, it reads a fixed byte of each block from a position decided by the algorithm for performing comparisons. We chose single-fixed byte as "far" as possible from the last read byte, as the chances of two consecutive read byte of two different blocks, of being equal are quite high [15,37]. It is the most important reason for selecting bytes in a non-consecutive manner. For a quick categorisation, after each pass, the position of byte chosen should be in such a way that it can differentiate blocks in separate categories quickly. We do not consider hash collisions or performance of hash algorithm in general in QuickDedup. On the other hand, the byte comparison approach is collision free. We created multiple combinations of byte selection strategies. However, experiments show that out of 5 different strategies, the fixed byte selection strategy yields the best results.

1. **Sequential Byte Selection:** The byte in each pass is read in sequential order for comparison i.e. Pass 1, 1^{st} byte of the block; Pass 2, 2^{nd} byte; Pass 3, 3^{rd} byte and so on.
2. **Random Byte Selection:** In this strategy, in each pass a random number is generated between 1 to *BS* (block size). The random number generated is the next byte for comparison in that pass.
3. **Multiple of 100:** In each pass, the byte read is in multiple of 100. As in 2^{nd} pass, byte read is 200^{th} byte, in 3^{rd} pass it is 300^{th} byte and so on.
4. **Power of 2 Selection:** In this strategy, selected bytes are in a power of 2 i.e. $2^{pass\_no.}$. Byte read are in the order 2^{nd}, 4^{th}, 8^{th}, 16^{th} and so on.
5. **Fixed Byte Strategy:** In this, the byte read follows a fixed pattern. For Pass 1, 1^{st} byte; Pass 2, last byte; Pass 3, the middle byte of block size (BS); Pass 4, byte at $1/4^{th}$ position of block size (BS) and so on. Fig. 2 shows categorisation of a single block after each pass of the algorithm. In pass 1, the first byte of block 5 is read, which is having character

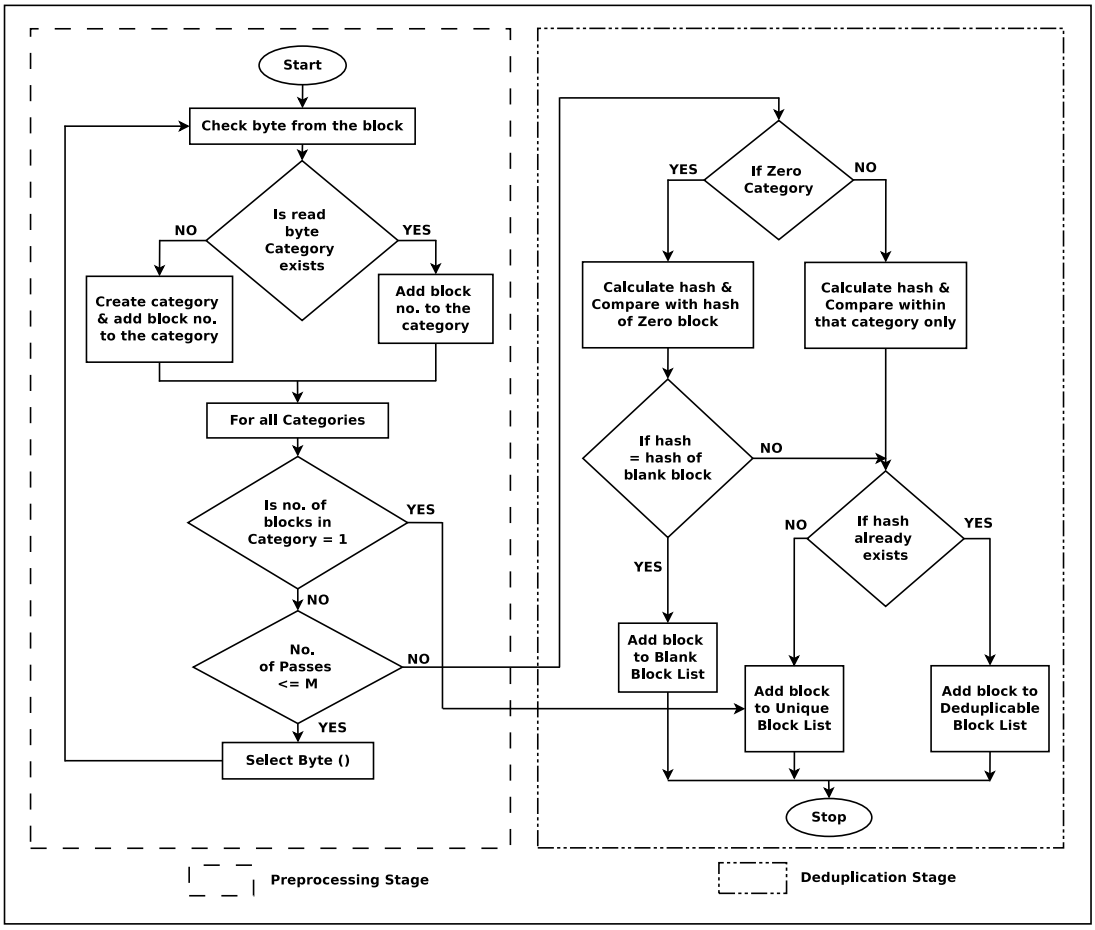

Fig. 1. Flowchart of Algorithm 1.

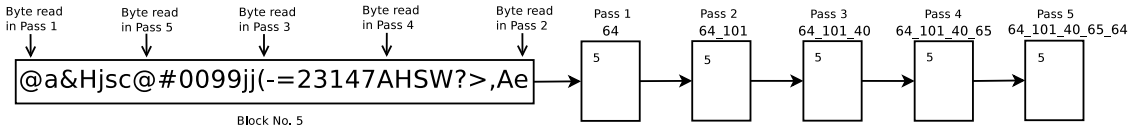

Fig. 2. Processing for Single Block: Block no. 5 getting categorised in five passes.

value ('@') and the ASCII value (64). We categorise this block under the category named '64'.

In pass 2, QuickDedup reads the last byte of the block which is 'e' having ASCII value '101', so the category formed is '64_101' and this block gets listed. This process continues till the M^{th} pass as discussed in Section 5.3. The categories are named by their appended ASCII value. These bytes of each block, for naming categories in each pass are read in a single go in the first pass when the block comes into memory. It helps in saving the access cost which incurs in accessing the same block multiple times.

Fig. 3 illustrates the categorisation among multiple blocks of the VM disk using the QuickDedup algorithm, named as Deduplication Data Tree (DDT). After each pass, the category that has only one block, is not processed further as the block of these categories are unique. As shown, block 6, after pass 2, is not categorised further. After the M^{th} pass, categories having more than one block, are processed. Hashes are only calculated for blocks inside these categories.

4.3. QuickDedup working in case of VM update

VMs once stored after deduplication can later be updated by the users. During updation, QuickDedup fetches the required blocks from the storage and update them. At the time of saving the updated block (s), QuickDedup reads all the required bytes (depending upon the byte selection strategy). These bytes depict a category name in which the block falls. If a category similar to the block exists among the DDT leaf nodes, then QuickDedup places the block that category. After that, QuickDedup calculates the hash of the block and compares it with the hashes of the already existing blocks in the category. If no such category exists, then QuickDedup creates a new category with the name of the block and placed the block in the DDT. In this case, no hash comparison is required.

4.4. QuickDedup algorithm

The basis of our proposed technique is optimising the number of comparisons during deduplication, and minimising or

eliminating the need for hash calculation as much as possible. Instead of calculating hashes of all the VM blocks, blocks can be categorised into various categories on the basis of our byte selection and comparison strategy. Performing the same process on multiple bytes will result into many distinct groups, among which need for comparison does not arise. The idea behind the proposed Algorithm 1 of QuickDedup lies in the fact that the byte comparison is quite efficient as compared to comparison of two hash values.

Algorithm 1 QuickDedup

Input: VM disk image(s), M = Number of passes, S = Byte selection strategy, BS = Block size, Hash() = Hash calculating function

Output: Deduplication ratio, list of unique blocks (U), blank blocks (B), deduplicable blocks (D)

Preprocessing Stage:

```
Divide the VM Disk image into N blocks of block size = BS
for Block 'b' = 1 to N do
    Read first byte of each block in "Read_Byte"
    if ( "Read_Byte" Category exists) then
        Add block 'b' to "Read_Byte"
    end
    else
        Create Category "Read_Byte"
        Add block 'b' to "Read_Byte"
    end
end
```

Deduplication Stage :

```
Check_Deduplication (M, S, BS, Category List)
Show Number of Blocks in Blank Block List, B
Show Number of Blocks in Unique Block List, U
Show Number of Blocks in Deduplicable Block List, D
Show Deduplication Ratio
```

$$\text{Deduplication Ratio} = 1 - \frac{\text{Number of Unique Blocks}}{\text{Total Number of Blocks in VM Image}} \quad (3)$$

```
Check_Deduplication (M, S, BS, Category List)
for pass 2 to M do
    for each Category i = 1 to X of previous pass do
        if (number of blocks in Category(i) = 1) then
            Add that Block 'b' to Unique Block List
        end
        else if (number of blocks in Category(i) >= 2) then
            for each Block 'b' of Category(i) do
                Read Select_Byte(M, S, BS)
                Repeat step 4 to 8 of QuickDedup
            end
        end
    end
end
for all Category(i) = 1 to X of 'M'th pass do
    if (Category(i) consists all 'zeros' ) then
        Zero_Category_Check (Category(i))
    end
    else if (number of blocks in Category(i) = 1) then
        Add Block 'b' to Unique Block List
    end
    else if (number of blocks in Category(i) >= 2) then
        Hash_Calculation (Category(i))
    end
end
return U, B and D
```

```
Select_Byte (M, S, BS)
if (S=1) then
    Byte selected in sequential order from 1 to BS
end
if (S=2) then
    Byte selected at a random position between 1 to BS
end
else if (S=3) then
    Byte = (M-1) * 100
end
else if (S=4) then
    Byte = 2^(M-1)
end
else if (S=5) then
    if (M = 1) then
        Byte = 1
    end
    else if (M = 2) then
        Byte = BS
    end
    else if (M is odd) then
        Byte = ⌈B/2⌉
    end
    else if (M is Even) then
        Byte = ⌈(B+BS)/2⌉
    end
end
return B (Selected Byte)
```

```
Zero_Category_Check (Category(i))
for each Block 'b' in Category(i) do
    if (Hash(Block 'b') = Hash(Blank Block)) then
        Add Block 'b' to Blank Block List
    end
    else if (hash already exists in stored hash) then
        Add Block 'b' to Deduplicable Block list
    end
    else
        Add Block 'b' to Unique Block List
    end
end
return U, B and D
```

```
Hash_Calculation (Category(i))

for each Block 'b' in Category(i) do
    if (Hash(Block 'b') already exists in stored hash) then
        Add Block 'b' to Deduplicable Block List
    end
    else
        Add Block 'b' Unique Block List
    end
end
```

5. Performance evaluation

To evaluate QuickDedup algorithm, we conduct extensive experiments to demonstrate the efficiency of QuickDedup compared to the traditional approaches on various parameters. These experiments also demonstrate the selection of optimal block size and number of passes for QuickDedup technique to achieve the

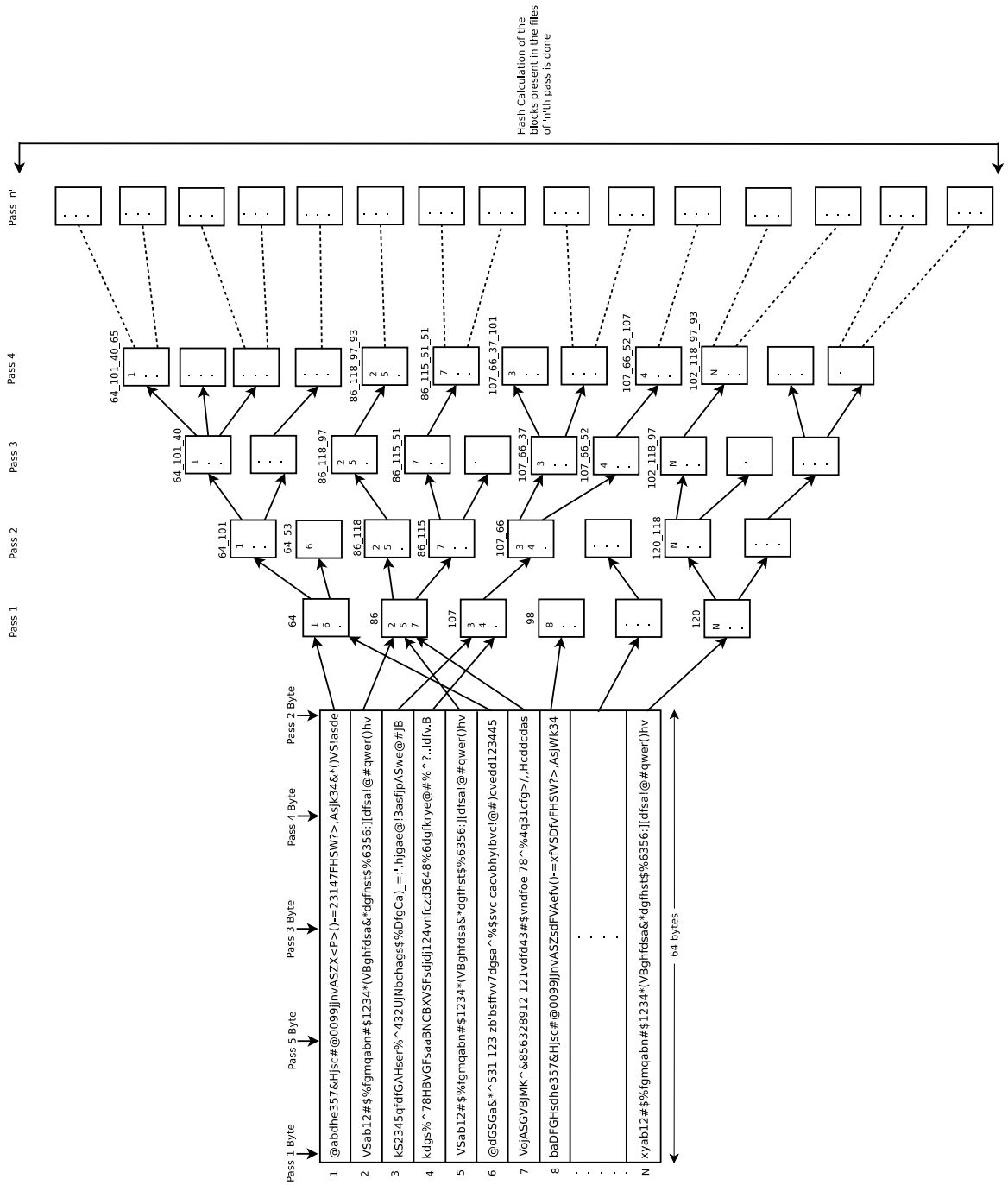

Fig. 3. Deduplication Data Tree (DDT): Categorisation of all blocks after each pass based on read byte and naming of category at each pass.

best results. The details of the experimental configuration used for the evaluation purpose, are provided in Table 1. We used SHA-1 to showcase our experimental results owing to the popular and generic nature of SHA-1 algorithm. In addition, the overall goal of our experimental evaluation was to compare a traditional pure hash-based approach to the proposed QuickDedup approach which has seldom use of a hash based comparisons. We believe that a faster hash algorithm will certainly affect and reduce the overall deduplication time, however, the total number of hash comparisons will remain same in both the approaches. We download the VM images 2,3,4,5 and 6 from [33]. These VMs are of different operating systems and are of variable sizes. The number of deduplicable blocks also varies among them. We created the VM images 1,7,8,9 and 10 in virtual box as no VMs having all deduplicable blocks or zero deduplicable blocks are available.

We performed a total of six different evaluation sets. We design each evaluation set to show important metrics related to various folds of proposed deduplication approach.

5.1. Evaluation set I: Byte and hash comparison

In the first evaluation set, we compare primitive Byte-to-Byte comparison with comparison of two hash values. This evaluation compares the time-taken by two approaches of block comparison.

Hash comparison: We read two blocks one by one from a VM image and calculate their hashes. We compare these hashes

Table 1
Experimental configuration.

Component	Specification
Hardware	Processor: Intel i5 RAM: 4 GB Harddisk: 500 GB, 2 TB
Operating system	Ubuntu 14.04 (64-bit)
Dataset	Data from [33] and additional dataset of created VM images (Table 3)
Hashing algorithm	SHA-1

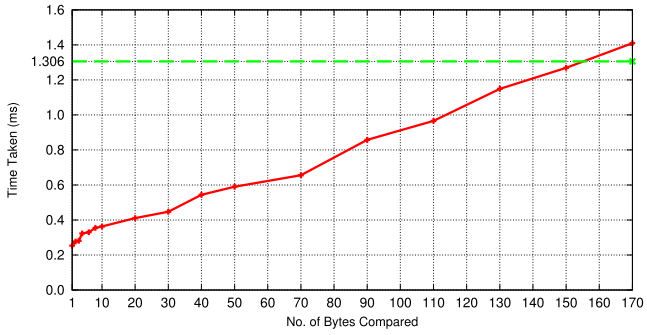

Fig. 4. Time variation on varying the byte read vs direct hash calculation and comparison. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

for similarity check. The time-taken by the complete process is 1.306 ms which comprises following components given by Eq. (4).

$$T_{Hash} = 2T_{Seek/Access} + 2T_{Block\_Read} + 2T_{Hash\_Cal} + T_{Hash\_Comp} \quad (4)$$

where
T_{Hash} = Time-taken by Hash
T_{Block_Read} = Time to read a block
T_{Hash_Cal} = Time required to calculate hash of a block
T_{Hash_Comp} = Time-taken to compare two hashes
$T_{Seek/Access}$ = Time required to access the block

Byte-to-Byte comparison: In this, we read the first byte of each block and then compare them for similarity. The time-taken by this process is 0.254 ms. This time is much lesser than hash comparison time, as in this approach only bytes are read and compared, no hash is calculated. Various components of this approach are represented by Eq. (5).

$$T_{Byte-Byte} = 2T_{Byte\_Seek} + 2T_{Byte\_Read} + T_{Byte\_Comp} \quad (5)$$

where
$T_{Byte-Byte}$ = Time-taken by Approach Byte–Byte comparison
T_{Byte_Read} = Time to read a byte from a block
T_{Byte_Comp} = Time-taken to compare one byte with other
T_{Byte_Seek} = Time taken to access one byte

$$T_{Hash} \gg T_{Byte-Byte} \quad (6)$$

To see whether the byte-byte comparison time is multiplicative with respect to the number of bytes, we perform another experiment. Fig. 4 indicates the number of bytes read from two blocks, and the time-taken for their reading and comparison. The red line in Fig. 4 shows that the amount of time taken for reading and comparing bytes as we increase the number of bytes. The green line shows the time taken by calculating hashes of two blocks and their subsequent comparison. Time-taken by byte-byte comparison is still smaller as compared to hash comparison even if we compare 150 bytes.

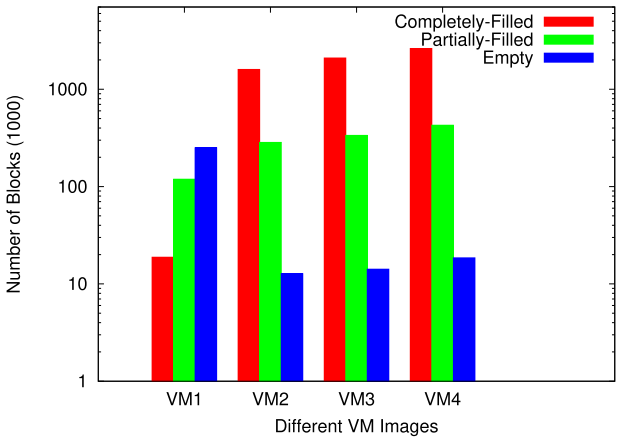

Fig. 5. Comparison of different types of blocks present in various VM Images.

Table 2
Type of blocks in various VM disk images.

S.No.	VM image size	Completely filled	Partially filled	Empty
1	285.2 MB	18 773	119 100	250 216
2	969.3 MB	1 596 017	284 341	12 737
3	1300 MB	2 091 607	335 719	14 145
4	1600 MB	2 625 016	426 545	18 530

5.2. Evaluation set II: Byte selection strategy

This experiment demonstrates the suitability of fixed byte strategy for comparison explained in Section 4.2. The fixed byte selection strategy aims to choose the comparison byte "far" or distant from a previously read byte. This idea is useful when maximum blocks of the VMs are completely-filled, as in the case of partially or empty block, the last byte will always be zero or garbage.

We conduct this experiment to determine the number of blank, partially-filled and completely-filled blocks exist in various VMs. Table 2 shows the type of blocks present in the different VM disk images. The outcome shows that most of the blocks of VMs are completely-filled. It shows that using fixed byte strategy is beneficial, as in fixed byte we are reading the last byte, in the second pass. If most of the blocks are half-filled or blank, then reading the last byte would not categorise the block well. Fig. 5 shows the presence of different types of blocks in various VM images.

5.3. Evaluation set III: Selection of number of passes (M)

One of the important factors on which the efficiency of QuickDedup algorithm depends, is the number of passes for categorisation. With increasing the number of passes, the number of categories increases as the number of bytes read increases as shown in Fig. 3. We evaluated QuickDedup on various number of passes, to determine the threshold. As shown in Fig. 6, initially as the number of passes increases, the time taken for the overall deduplication process decreases. However, after a certain threshold i.e. 7^{th} pass, the time taken starts increasing. This shows that after 7^{th} pass, the time taken in reading bytes and categorisation exceeds the time saved through categorisation. Hence, we set the threshold as 7 passes for best efficiency.

The results show that corresponding to each VM in the set, as the number of passes increases, the deduplication time decreases, but after 7th pass the deduplication time starts increasing. Even after 7th pass, the categorisation gets better and the number of hash comparisons and calculations decrease. However, the time

Table 3
Comparison of "QuickDedup" with "Pure hash-based" approach.

VM images			Pure hash-based approach			QuickDedup approach			δ[d]	ε[e] (%)
S.No.	Name	Size	α[a]	β[b]	γ[c]	α[a]	β[b]	γ[c]		
1	Blank Image	200 MB	5	17	14	5	28	22	1.00	0
2	Tiny Linux (VM1)	122 MB	2	49	30	2	18	13	0.37	0
3	DeLi Linux (VM2)	285 MB	6	1030	850	3	45	38	0.09	50
4	TinyMe (VM3)	660 MB	16	5590	4905	5	76	63	0.05	68
5	CentOS (VM4)	1304 MB	30	7048	6019	1	101	89	0.12	96
6	FluxUbuntu (VM5)	2048 MB	47	9541	8788	35	447	398	0.14	25
7	Windows (VM6)	10 GB	2121	86 413	68 266	122	58 128	23 418	0.60	94
8	Windows (VM7)	50 GB	1011	301 176	232 727	812	195 182	173 294	0.48	19
9	Ubuntu (VM8)	105 GB	2210	490 958	597 333	1128	235 881	281 183	0.22	49
10	Ubuntu (VM9)	1000 GB	20 121	3 953 667	3 413 321	12 521	2 011 813	1 825 519	0.37	37

[a] α = No. of hashes calculated (10^4).
[b] β = Time (s) (without using Hash Table).
[c] γ = Time (s) (with Hash Table).
[d] δ = Deduplication Ratio.
[e] ε = Approximate percentage of hash calculation reduction using QuickDedup.

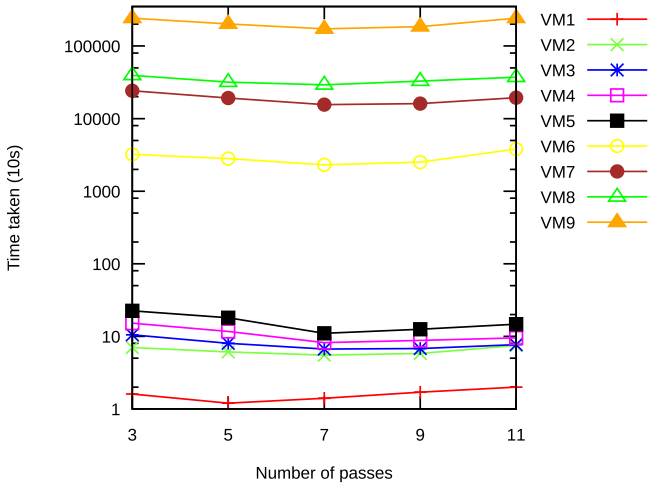

Fig. 6. Variation of time corresponding to variable passes of QuickDedup.

saved in this case is less than the extra time-taken in categorisation process as the number of passes increase. Based on the results of this experiment, we set the pass threshold (M) at 7. We perform all the other experiments considering 7 passes for best performance. For a detailed and wide performance check, we choose various block sizes between 512 bytes to 100 kB for each pass and performed the evaluation.

5.4. Evaluation set IV: Block size selection

Another factor that affects the deduplication efficiency, both in terms of deduplication time and deduplication ratio, is block size. Since block size affects the deduplication ratio and deduplication time in a different manner, a block size should be chosen in such a way that it provides a good balance between both deduplication ratio and deduplication time. We perform experiments on given VM disk images by varying block size from 512 bytes to 100 kB, keeping the number of passes constant. Based on these sizes, we compare our approach in terms of deduplication ratio and deduplication time. In the case of deduplication ratio, it falls as we increase the block size. It is obvious that the probability of finding similar blocks decreases while we go for higher block sizes. Fig. 7 shows that the decrease in deduplication ratio is higher for block sizes 4 kB.

Fig. 8 shows the variation of deduplication time corresponding to various block sizes. Initially, on increasing the block size up to 4 kB, the decrease in deduplication time is quite high. After 4 kB, even after varying the block size to a large extent, there is a small decrease in deduplication time, as compared to the deduplication time of blocks size less than 4 kB. One important reason for this relation is that the disk reads the VM disk image in blocks of 4 kB. Even when the selected block size is greater than 4 kB, the data is fetched in blocks of 4 kB from disk. Therefore, there is no significant decrease in deduplication time for block sizes more than 4 kB. Based on both comparisons, we select the optimum block size as 4 kB as it provides a good balance between both the factors.

5.5. Evaluation set V: Comparison of "QuickDedup" with "pure hash-based" approach

Most of the existing deduplication techniques [13,21,46] use hash comparison [34] to check the similarity of two blocks. Two blocks, similar in content, will have the same hash. For every incoming block, its hash is compared with all the previously stored hashes to decide whether its deduplicable or not. The hashes are stored in a hash table so that searching becomes fast. As there is no benchmark specified for the various criteria of a deduplication algorithm against which we can test the efficiency of our proposed approach. For comparison purpose, we design a benchmark that follows pure hash-based approach for deduplication. Designed benchmark calculates hashes of all blocks of a given VM.

For the experimental evaluation, few VMs are pre-made VM disk appliances, consisting of different operating systems from [33] and few are created in Virtualbox as detailed in Table 1, to perform the efficiency test. QuickDedup aims at improving the efficiency of the overall deduplication process with a focus on the aspects such as deduplication time and the metadata overhead without focusing on improving the deduplication ratio. Considering that the deduplication ratio is fixed and depends on the similarity of the data among the blocks, we evaluated our approach on a set of 10 VMs. Our results are independent of the number of VMs under consideration, as different kinds of VMs would only vary the deduplication ratio. On the other hand, size of a VM affects the results, therefore, we considered VM having size up to 1000 GB for our performance evaluations. Table 3 shows the comparison between the proposed approach and the pure hash-based approach. The first two columns of Table 3 shows the names and sizes of the VM image among which we perform the evaluation. Next 3 columns of Table 3 consist of 3 parameters of pure hash-based approach which are compared with the similar parameters of QuickDedup approach

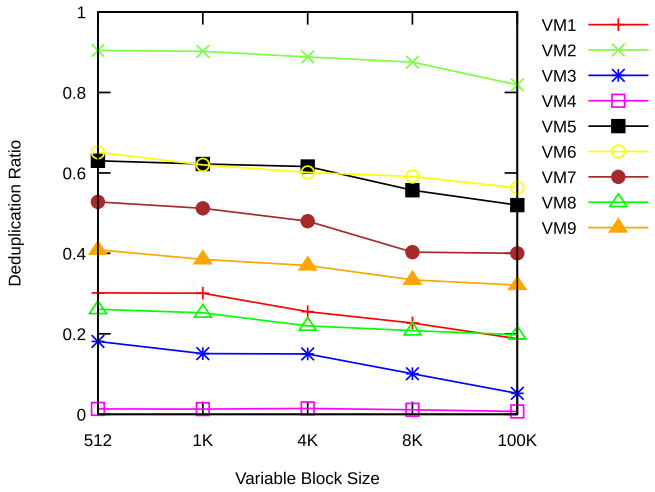

Fig. 7. Variation of deduplication ratio corresponding to variable block sizes.

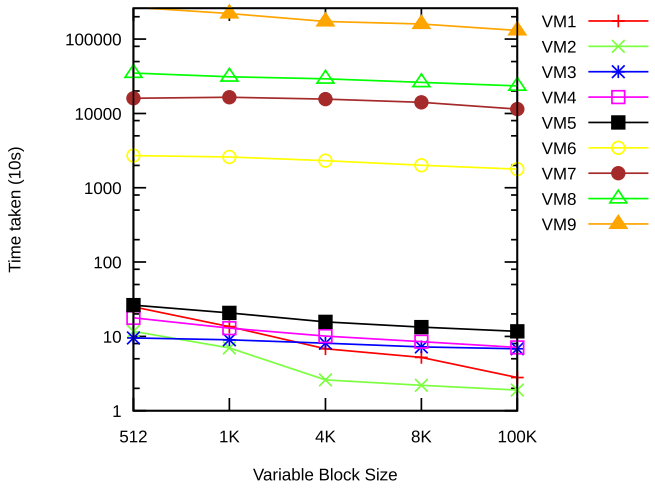

Fig. 8. Variation of deduplication time corresponding to variable block sizes.

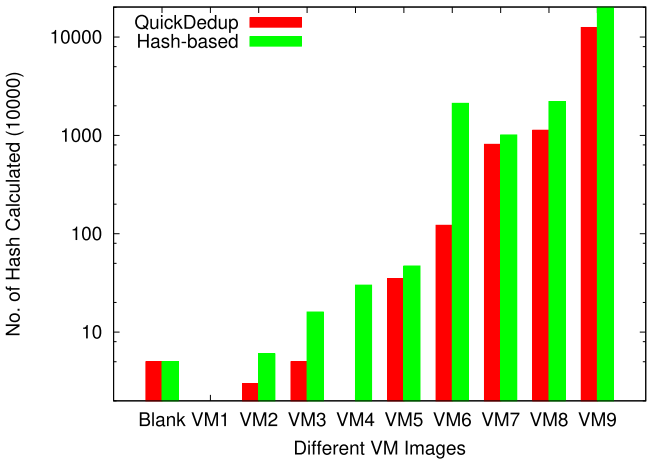

Fig. 9. Comparison of Hash calculations "QuickDedup" with "Pure hash-based" approach.

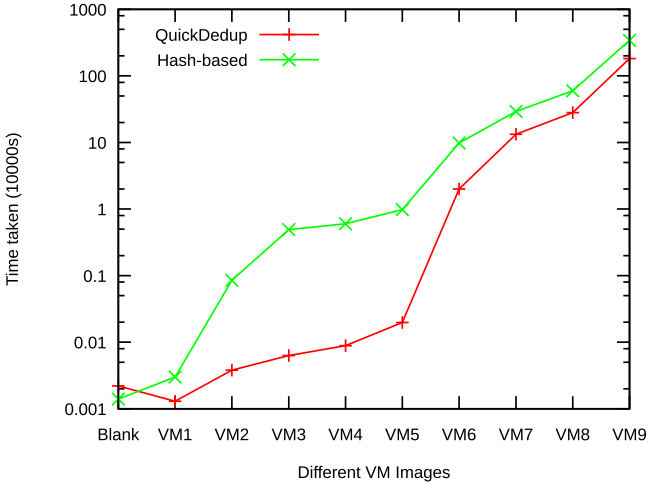

Fig. 10. Comparison of deduplication time "QuickDedup" with "Pure hash-based" approach (with hash table).

(last 3 columns of Table 3). Here, we evaluate the approaches both using hash table (for efficient search) and without making use of hash table. The comparison between the number of hashes calculated and time taken by both the approaches is shown in Fig. 9 and Fig. 10 for better comprehension of the evaluation. Fig. 9 depicts that in comparison of pure hash-based approach using hash-table, QuickDedup performs much better in terms of time-taken for deduplication. The worst case scenario is only the case of completely blank VM, when QuickDedup takes more time than pure hash-based approach.

We evaluated QuickDedup against pure hash-based approach on various parameters viz. number of hashes calculated, total time-taken, and the amount of metadata generated. We stress that the existing deduplication approaches do not show performance evaluation based on these important parameters. To support our claims, we compared our proposed approach with a pure hash-based approach. In addition, almost all the existing approaches mentioned in Section 3 focus on calculating the hashes for all the blocks. The results show that both hash calculations and hash comparisons by QuickDedup are much less as compared to pure hash-based approach. In the pure hash-based approach, there is no byte comparison, however QuickDedup performs byte comparison. In spite of byte comparison, the overall deduplication time by the QuickDedup for determining the deduplicated blocks, is much less than the pure hash-based approach. Fig. 9 shows the hash calculation comparison between both the approaches. Fig. 10 depicts that in comparison of pure hash-based approach using hash-table, QuickDedup performs much better in terms of time-taken for deduplication. The worst case scenario is only the case of completely blank VM, when QuickDedup takes more time than pure hash-based approach.

5.6. Evaluation set VI: Stage-wise time-taken evaluation

We conduct this experiment to evaluate the time taken in both the stages of QuickDedup. The experimental results shown in Table 4, depict that the "preprocessing stage" consumes less amount of time in comparison to the "deduplication stage". The only case when "preprocessing stage" takes more time than "deduplication stage", is when the deduplication ratio is very low and only few number of hashes are calculated at the end of QuickDedup approach.

6. Analysis of QuickDedup complexity

6.1. Time complexity analysis

In case of deduplication of a VM image having completely unique blocks, QuickDedup performs its best. As QuickDedup

Table 4
Time taken in the "Preprocessing" and "Deduplication" stage of QuickDedup.

VM images		Preprocessing stage	Deduplication stage	Total
Name	Size	Time (s)	Time (s)	Time (s)
Blank Image	200 MB	7	15	22
Tiny Linux (VM1)	122 MB	4	9	13
DeLi Linux (VM2)	285 MB	12	26	38
TinyMe (VM3)	660 MB	25	38	63
CentOS (VM4)	1304 MB	53	36	89
FluxUbuntu (VM5)	2048 MB	87	311	398
Windows (VM6)	10 GB	608	22 810	23 418
Windows (VM7)	50 GB	3142	170 152	173 294
Ubuntu (VM8)	105 GB	5812	275 371	281 183
Ubuntu (VM9)	1000 GB	61 132	1 764 387	1 825 519

Table 5
Size of metadata generated by "QuickDedup" and "Pure hash-based" approach.

VM images		Pure hash-based	QuickDedup
Name	Size		
Blank Image	200 MB	20 MB	22 MB
Tiny Linux (VM1)	122 MB	1.22 MB	0.98 MB
DeLi Linux (VM2)	285 MB	2.84 MB	1.32 MB
TinyMe (VM3)	660 MB	6.60 MB	2.90 MB
CentOS (VM4)	1304 MB	13.04 MB	3.90 MB
FluxUbuntu (VM5)	2048 MB	20.48 MB	14.60 MB
Windows (VM6)	10 GB	102.20 MB	69.80 MB
Windows (VM7)	50 GB	512 MB	342 MB
Ubuntu (VM8)	105 GB	1075 MB	718 MB
Ubuntu (VM9)	1000 GB	10 240 MB	7568 MB

categorises blocks based on byte comparison, all blocks being unique, fall into different categories i.e. each category will have only one block. After the last pass, when each category is checked, each category will have only one block, therefore, the need for calculating and comparing any hash, will not arise. In the best case, the number of hashes computed and compared will always be zero. As the hash table is used as a structure for hash storage, searching has $\mathcal{O}(1)$ time complexity. Even without using a hash table, QuickDedup algorithm has $\mathcal{O}(1)$ searching time complexity in this case, as each category has only single block. Therefore, maintaining a separate category, is advantageous and useful. Since in this case we do not calculate any hash, it saves the time of hash calculations. The deduplication time is represented by Eq. (7).

$$T_{Total} = T_{Byte\_Comp} + T_{Hash\_Cal} + T_{Hash\_Comp} \tag{7}$$

where
T_{Byte_Comp} = Time-taken in byte comparison
T_{Total} = Time required for deduplication
T_{Hash_Cal} Time-taken in hash calculation
T_{Hash_Comp} = Time-taken in hash comparison

Here for this case, T_{Hash_Comp} is $\mathcal{O}(1)$ and since no hash is being calculated, T_{Hash_Cal} and T_{Hash_Comp} will be zero. The total deduplication time can be represented by Eq. (8).

$$T_{Total} = T_{Byte\_Comp} \tag{8}$$

To perform deduplication, pure hash-based approach calculates hash of all the unique blocks of a VM. It compares every new incoming block based upon hash table searching, i.e. searching time complexity will be $\mathcal{O}(1)$. Eq. (9) represents the deduplication time of pure hash-based approach.

$$T_{Total} = T_{Hash\_Cal} + T_{Hash\_Comp} \tag{9}$$

The hash comparison time is same in both the cases. Eq. (8) consists of time-taken by byte comparison and Eq. (9) consists of time-taken by hash calculations of all blocks. Experiment I shows that the time-taken in reading and comparing a byte is significantly less than that of hash calculation and comparison. Approximately 150-byte comparison takes equal amount of time as of two hash calculation and their comparison. The overall time-taken by Eq. (9) is more than Eq. (8) i.e. QuickDedup performs notably better in the best case.

For QuickDedup, the worst case is when a deduplicated VM has all duplicate blocks. In this case, after each pass and byte comparisons, all blocks fall under only one category. Hence, QuickDedup calculates hash for all 'n' blocks. Hash table implementation also requires 'n' number of comparisons for a new incoming block as all blocks are same and lead to the same hash value, which is worst case searching in hash tables. QuickDedup complexity in the worst case is $\mathcal{O}(n)$. It is the only case, when the categorisation idea of QuickDedup leads to an extra overhead.

Here, pure hash-based approach using a hash table gives same complexity as $\mathcal{O}(n)$, since the hash of all the blocks will lead to the same value in the hash table, forming a chain, which is sequential. In the worst case, the number of hash calculations and comparisons are same in both approaches. The only extra overhead in QuickDedup is of byte comparisons for categorisation. Except this case, in all other cases, QuickDedup beats the pure hash-based approach in terms of hash calculated and time required for deduplication.

QuickDedup is an offline algorithm. The storage space reserved by VMs before deduplication is reusable, once it free the storage blocks which are deduplicable after QuickDedup. QuickDedup also supports deduplication among multiple VMs. QuickDedup performs byte-comparison between the blocks of the VM(s) for checking similarity. For multiple VMs, QuickDedup considers the blocks of all the VMs in one-go and performs comparisons same as in case of single VM. QuickDedup can also work online efficiently if the block size chosen for deduplication is same as that of disk read block size i.e. 4 kB. When the disk-read block size and deduplication block size differs, then after deduplication the blocks are written to the disk according to the deduplication block size. When read and write block size differs, the overhead of either disintegrating a larger block in smaller block or integrating smaller ones into larger on the fly for performing deduplication increases. Therefore, when disk-read and deduplication block size varies, the efficiency of online QuickDedup degrades.

6.2. Overhead analysis of metadata

As identified in Section 2, one of the criteria, which needs optimisation for a deduplication technique is to minimise the generated metadata. In the case of QuickDedup, the metadata is of the categories made after each pass. Each category only consists of the block numbers, based on byte comparison. After the last pass, if the need for hash calculation arises, then QuickDedup also stores hashes as metadata. Pure hash-based approach, stores the hashes of all the blocks of the VM for performing deduplication.

Example: Consider a VM of size 1 TB, undergoing deduplication process. The block size for deduplication is 4 kB. The VM will have 2.5×10^8 blocks of size 4 kB.

Table 6
Timing comparison between "Pure hash-based", "QuickDedup" and "QuickDedup (Improved Data Access)" approaches.

VM images		Pure hash-based	QuickDedup	QuickDedup (Improved)
Name	Size	Time (s)	Time (s)	Time (s)
Blank Image	200 MB	14	22	19
Tiny Linux (VM1)	122 MB	30	13	12
DeLi Linux (VM2)	285 MB	850	38	33
TinyMe (VM3)	660 MB	4905	63	56
CentOS (VM4)	1304 MB	6019	89	77
FluxUbuntu (VM5)	2048 MB	8788	398	372
Windows (VM6)	10 GB	68 266	23 418	23 228
Windows (VM7)	50 GB	232 727	173 294	172 244
Ubuntu (VM8)	105 GB	597 333	281 183	279 279
Ubuntu (VM9)	1000 GB	3 413 321	1 825 519	1 796 206

QuickDedup: Since the VM contains the blocks in multiple of 10^8, so the last block number will be at most of 10 bytes. For storing all block numbers, the maximum storage requirement is $10 \times 2.5 \times 10^8$ which is 2.5 GB. In addition to it, the hashes, if calculated are stored.

Pure hash-based Approach: This approach stores the hashes of all the blocks. If the SHA-1 function is used for hash calculation, which generates a 160 bit or 20-byte hash as an output for 1 block. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. For entire VM, the hashes generated will be of $40 \times 2.5 \times 10^8$ i.e. 10 GB.

In the best case, when all blocks are unique, QuickDedup will have the metadata only of categorisation which is 2.5 GB, since no hash is calculated, whereas pure hash-based approach will have 10 GB metadata. In worst case QuickDedup will have 2.5 GB + 10 GB i.e. 12.5 GB data. It is the only case when QuickDedup will have more metadata than the hash-based approach. In rest of the cases, it generates less metadata as compared to hash-based traditional approach. Table 5 shows the size of metadata generated by both approaches for variable size VM images.

6.3. Data access/movement improvement

To improve the data access overhead for reading bytes in every pass, we present a modified version of QuickDedup, where data access mechanism in "preprocessing stage" is improved. In improved version, QuickDedup reads all the seven bytes together, which are required for construction of DDT, in the first pass when the block gets into memory as shown in Fig. 3. This helps us in reducing the data access from disk and minimises the overall time taken in deduplication stage. Table 6 shows the details of the experiment conducted using this optimised "byte reading strategy".

7. Conclusions and future work

Deduplication process is an important storage optimisation technique for emerging virtualization based cloud storage. In this work, we first collate a number of important performance requirements for an efficient deduplication process. Based on these essential factors, we propose a novel deduplication technique, "QuickDedup", which outperforms traditional hash-based deduplication approaches on various metrics. Instead of calculating hashes for each block in the disk, QuickDedup uses a novel byte-byte comparison strategy, based on which the overall population of comparable blocks are minimised and reduced to only a handful of partially similar blocks. The categorisation process involves a meta-data structure known as Deduplication Data Tree (DDT) which helps in minimising the overall hash calculations which subsequently reduces the number of eligible comparable blocks. At a later stage of the proposed approach, QuickDedup calculates the hashes of only a smaller set of blocks which are shown to be reduced up to 95%. We conduct extensive experimental evaluations to test the efficacy of QuickDedup. Proposed approach beats hash-based approach in terms of the number of hashes calculated, number of hash comparisons made, and overall deduplication time. Except in worst case, which is rare for VM disk images, QuickDedup performs better in all specified criterion collated as important requirements. Additionally, QuickDedup saves on deduplication time up to 96% with a huge reduction on space required for hash storage. Additionally, it outperforms all other approaches, on various parameters to judge the efficiency of any deduplication approach.

There are multiple open research issues, which are related to the deduplication. For making this deduplication file system more efficient, the retrieval process needs to be quicker in addition to fixing the issues related to disk fragmentation. We also feel that factors such as VM image type, operating system type, format, and application type may also help in reducing the deduplication time further, maintaining the deduplication ratio.

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.

References

[1] AFCOM, 2013, URL http://stage.afcom.com/digital-library/pub-type/dcmmag/storage-struggles-to-keep-pace-with-data-growth/.
[2] Commvault, 2014, URL http://www.commvault.com/blogs/2014/february/ratio-schmatio-time-to-get-over-dedupe-ratios-and-other-myths.
[3] A. Arcangeli, I. Eidus, C. Wright, Increasing memory density by using KSM, Proceedings of the linux symposium (2009) 19–28, URL http://www.kernel.org/doc/ols/2009//nhttps://www.kernel.org/doc/ols/2009/ols2009-pages-19-28.pdf.
[4] D. Bhagwat, K. Eshghi, D.D.E. Long, M. Lillibridge, Extreme binning: Scalable, parallel deduplication for chunk-based file backup, in: Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS, 2009, pp. 237–245, http://dx.doi.org/10.1109/MASCOT.2009.5366623.
[5] A.T. Clements, I. Ahmad, M. Vilayannur, J. Li, Decentralized deduplication in SAN cluster file systems, 2009, p. 14, URL http://www.usenix.org/event/usenix09/tech/full_papers/clements/clements.pdf.
[6] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, M. Welnicki, {HYDRAstor}: a scalable secondary storage, in: Conference on File and Storage Technologies (FAST), 2009, pp. 197–210.
[7] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, L. Xu, Application-aware local-global source deduplication for cloud backup services of personal storage, IEEE Trans. Parallel Distrib. Syst. 25 (5) (2014) 1155–1165.
[8] B. Gerofi, Z. Vass, Y. Ishikawa, Utilizing memory content similarity for improving the performance of replicated virtual machines, in: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, IEEE, 2011, pp. 73–80, http://dx.doi.org/10.1109/UCC.2011.20.

[9] J.G. Hansen, E. Jul, Lithium: Virtual machine storage for the cloud, in: Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10, ISBN: 9781450300360, 2010, p. 15, http://dx.doi.org/10.1145/1807128.1807134, URL http://dl.acm.org/citation.cfm?id=1807128.1807134.
[10] Q. He, Z. Li, X. Zhang, Data deduplication techniques, in: Future Information Technology and Management Engineering (FITME), 2010 International Conference on, Vol. 1, IEEE, 2010, pp. 430–433.
[11] D. Hyde, A survey on the security of virtual machines, Tech. Rep, Dept. of Comp. Science, Washington Univ. in St. Louis, 2009, pp. 1–11.
[12] K. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, H. Lei, An empirical analysis of similarity in virtual machine images, in: Proceedings of the Middleware 2011 Industry Track Workshop, ACM, 2011, p. 6.
[13] K. Jin, E.L. Miller, The effectiveness of deduplication on virtual machine disk images, in: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, ACM, 2009, p. 7.
[14] R. Koller, R. Rangaswami, I/o deduplication: Utilizing content similarity to improve i/o performance, ACM Trans. Storage (TOS) 6 (3) (2010) 13.
[15] E. Kruus, C. Ungureanu, C. Dubnicki, Bimodal content defined chunking for backup streams, Fast (2010) 239–252.
[16] Y.-K. Li, M. Xu, C.-H. Ng, P.P. Lee, Efficient hybrid inline and out-of-line deduplication for backup storage, ACM Trans. Storage (TOS) 11 (1) (2015) 2.
[17] C. Lin, Q. Cao, J. Huang, J. Yao, X. Li, C. Xie, HPDV : A highly parallel deduplication cluster for virtual machine images, in: 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2018, pp. 472–481, http://dx.doi.org/10.1109/CCGRID.2018.00074.
[18] B. Mao, H. Jiang, S. Wu, Y. Fu, L. Tian, Read-performance optimization for deduplication-based storage systems in the cloud, ACM Trans. Storage (TOS) 10 (2) (2014) 6.
[19] D.T. Meyer, W.J. Bolosky, A study of practical deduplication, ACM Trans. Storage (TOS) 7 (4) (2012) 14.
[20] C.-H. Ng, P.P.C. Lee, RevDedup: A reverse deduplication storage system optimized for reads to latest backups, in: Proceedings of the 4th Asia-Pacific Workshop on Systems, ACM, 2013, p. 15.
[21] C.-H. Ng, M. Ma, T.-Y. Wong, P.P.C. Lee, J. Lui, Live deduplication storage of virtual machine images in an open-source cloud, in: Proceedings of the 12th International Middleware Conference, International Federation for Information Processing, 2011, pp. 80–99.
[22] J.a. Paulo, J. Pereira, A survey and classification of storage deduplication systems, ACM Comput. Surv. (ISSN: 03600300) 47 (1) (2014) 1–30, http://dx.doi.org/10.1145/2611778, URL http://dl.acm.org/citation.cfm?doid=2620784.2611778.
[23] S. Saharan, G. Somani, Security of cloud-based storage, in: Guide to Security Assurance for Cloud Computing, Springer, 2015, pp. 65–81.
[24] R. Schwarzkopf, M. Schmidt, M. Rüdiger, B. Freisleben, Efficient storage of virtual machine images, in: Proceedings of the 3rd workshop on Scientific Cloud Computing Date, ACM, 2012, pp. 51–60.
[25] M.-K. Seo, S.-H. Lim, Deduplication flash file system with pram for non-linear editing, IEEE Trans. Consum. Electron. 56 (3) (2010).
[26] F. Shaikh, F. Yao, I. Gupta, R.H. Campbell, VMDedup: Memory deduplication in hypervisor, in: Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014, 2014, pp. 379–384, http://dx.doi.org/10.1109/IC2E.2014.69.
[27] P. Sharma, P. Kulkarni, Singleton: system-wide page deduplication in virtual environments, in: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, ACM, 2012, pp. 15–26.
[28] K. Srinivasan, T. Bisson, G. Goodson, K. Voruganti, 2012. iDedup: latency-aware, inline data deduplication for primary storage. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies, pp. 299–312.
[29] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord, in: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '01, Vol. 31, 2001, pp. 149–160, http://dx.doi.org/10.1145/383059.383071, (4) URL http://dl.acm.org/citation.cfm?id=383059.383071.
[30] Y. Tsuchiya, T. Watanabe, Dblk: Deduplication for primary block storage, in: Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on, IEEE, 2011, pp. 1–5.
[31] A. Upadhyay, P.R. Balihalli, S. Ivaturi, S. Rao, Deduplication and compression techniques in cloud design, in: SysCon 2012 - 2012 IEEE International Systems Conference, Proceedings, ISBN: 9781467307499, 2012, pp. 362–367, http://dx.doi.org/10.1109/SysCon.2012.6189472.
[32] L.M. Vaquero, M.A.S. Netto, I.M. Llorente, U.C.D. Madrid, S.D.E. Capitani, D.I. Vimercati, P. Samarati, 2018. A Manifesto for Future Generation Cloud Computing : Research Directions for the Next Decade, 51 (5).
[33] VirtualBox, Virtualboxes free virtualbox images, 2012, URL https://virtualboxes.org/images/ [Online; accessed June-2018].
[34] J. Wei, H. Jiang, K. Zhou, D. Feng, MAD2: A scalable high-throughput exact deduplication approach for network backup services, in: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010, 2010, http://dx.doi.org/10.1109/MSST.2010.5496987.
[35] A. Wildani, E.L. Miller, O. Rodeh, Hands: A heuristically arranged non-backup in-line deduplication system, in: Data Engineering (ICDE), 2013 IEEE 29th International Conference on, IEEE, 2013, pp. 446–457.
[36] H. Wu, C. Wang, Y. Fu, S. Sakr, L. Zhu, K. Lu, HPDedup : A Hybrid Prioritized Data Deduplication Mechanism for Primary Storage in the Cloud.
[37] W. Xia, H. Jiang, D. Feng, Y. Hua, 2011. Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In: Proceedings of The 2011 USENIX Annual Technical Conference (USENIX ATC '11), 2011, pp. 26–28.
[38] W. Xia, H. Jiang, D. Feng, L. Tian, DARE: A deduplication-aware resemblance detection and elimination scheme for data reduction with low overheads, IEEE Trans. Comput. (ISSN: 0018-9340) 9340 (c) (2015) 1, http://dx.doi.org/10.1109/TC.2015.2456015, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7155488.
[39] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, Z. Wang, P-Dedupe: Exploiting parallelism in data deduplication system, in: Proceedings - 2012 IEEE 7th International Conference on Networking, Architecture and Storage, NAS 2012, ISBN: 9780769547220, 2012, pp. 338–347, http://dx.doi.org/10.1109/NAS.2012.46.
[40] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, Y. Zhou, Ddelta: A deduplication-inspired fast delta compression approach, Perform. Eval. 79 (2014) 258–272.
[41] J. Xu, W. Zhang, S. Ye, J. Wei, T. Huang, A lightweight virtual machine image deduplication backup approach in cloud environment, in: Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual, IEEE, 2014, pp. 503–508.
[42] W. Zhang, D. Agun, T. Yang, R. Wolski, H. Tang, Vm-centric snapshot deduplication for cloud data backup, in: Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on, IEEE, 2015a, pp. 1–12.
[43] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhou, AE : An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication, Infocom (2015b) 1337–1345, http://dx.doi.org/10.1109/INFOCOM.2015.7218510.
[44] Y. Zhang, Y. Wu, G. Yang, Droplet: A distributed solution of data deduplication, in: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, IEEE Computer Society, 2012, pp. 114–121.
[45] W. Zhang, T. Yang, G. Narayanasamy, H. Tang, Low-cost data deduplication for virtual machine backup in cloud storage, in: Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems, USENIX, San Jose, CA, 2013, URL https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/Wei_Zhang.
[46] X. Zhao, Y. Zhang, Y. Wu, K. Chen, J. Jiang, K. Li, Liquid: A scalable deduplication file system for virtual machine images, Parallel Distrib. Syst. IEEE Trans. (ISSN: 1045-9219) 25 (5) (2014) 1257–1266, http://dx.doi.org/10.1109/TPDS.2013.173.
[47] Y. Zhou, Y. Deng, J. Xie, Leverage similarity and locality to enhance fingerprint prefetching of data deduplication, in: Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on, IEEE, 2014, pp. 142–149.
[48] B. Zhu, K. Li, R. Patterson, Avoiding the disk bottleneck in the data domain deduplication file system., Fast 8 (2008) 1–14.

Shweta Saharan is pursing PhD in Department of Computer Science and Engineering, Malaviya National Institute of Technology Jaipur, India. She has completed her master's degree (M.Tech) in Computer Science and Engineering (Information Security) from Central University of Rajasthan, India, in 2015 and received her Bachelor of Technology (B.Tech) from Rajasthan Technical University in 2008. She has been a part of publication committee of ICISS 2016 conference and program committee member of ISEA-ISAP 2018 conference. She is an active reviewer in various conferences series viz. SIN, ISEA-ISAP, INDICON and Journal of Supercomputing. Her research interests include Cloud Computing, Information and Network Security, and Networking.

Dr Gaurav Somani is an assistant professor at Department of CSE at Central University of Rajasthan, India. Earlier, he served as a lecturer at the LNMIIT, Jaipur. He completed his PhD from MNIT, Jaipur, MTech from DAIICT, Gandhinagar, and BE from University of Rajasthan, India. His areas of research interests include distributed systems and security. He has published his research at prestigious venues such as IEEE TDSC, JPDC, ACM Computing Surveys, ComCom, ComNet, FGCS, and IEEE Cloud Computing. He is an associate editor of IEEE Access Journal and also served as a lead guest editor for

special issue of Software: Practice and Experience Journal (Wiley) on "Integration of IoT, Cloud and Big Data Analytics". He has been in TPC of various reputed conferences and served as reviewer of many IEEE, ACM, Elsevier, Springer, and Wiley journals. He served as keynote co-chair at Asia Security & Privacy Conference 2019 and keynote and tutorial chair for ICISS 2016. He has received IEI Young Engineer Award 2020. He is senior member of IEEE and a professional member of ACM.

Dr Gaurav Gupta is a Scientist E in the Ministry of Electronics and Information Technology, New Delhi, India. His research interests include digital forensics, Privacy presvering analystics and foresics, enhacing QR Codes and security and forensic aspects in emerging technologies.

Dr Robin Verma is a postdoctoral researcher at Cyber Center for Security and Analytics in University of Texas at San Antonio, Texas, USA. His research interests include cyber security, cyber forensics, digital forensics, and application of privacy preserving technologies for digital forensic investigation process.

Dr Manoj Singh Gaur is director of Indian Institute of Technology, Jammu. Prior to joining IIT Jammu he was a Professor and Head of the Department of Computer Science and Engineering at Malaviya National Institute of Technology (MNIT) Jaipur. He has obtained his PhD from University of Southampton, UK. He has supervised research in the areas of networks on chip and information security. He has published over 150 papers in peerreviewed major conferences and journals and has coordinated national and international projects in the domains of information security and networks on chip. He has been associate editors with CSI Transaction, IET Electronics and Digital Techniques, and Journal of Information Security and Assurance. He was organising chair of SPACE 2015 and was general cochair of SINCONF 2012 and ICISS 2016. He is a member of IEEE and ACM. Currently, he is director of Indian Institute of Technology Jammu, India.

Dr. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He served as a Future Fellow of the Australian Research Council during 2012-2016. He has authored over 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He also edited several books including "Cloud Computing: Principles and Paradigms" (Wiley Press, USA, Feb 2011). He is one of the highly cited authors in computer science and software engineering worldwide (h-index=132, g-index=294, 93,000+ citations). "A Scientometric Analysis of Cloud Computing Literature" by German scientists ranked Dr. Buyya as the World's Top-Cited (#1) Author and the World's Most-Productive (#1) Author in Cloud Computing. Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" for four consecutive years since 2016, a Fellow of IEEE, and Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier and recently (2019) received "Lifetime Achievement Awards" from two Indian universities for his outstanding contributions to Cloud computing and distributed systems.

Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 50 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society TCSC. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Frost & Sullivan New Product Innovation Award". Recently, Dr. Buyya received "Mahatma Gandhi Award" along with Gold Medals for his outstanding and extraordinary achievements in Information Technology field and services rendered to promote greater friendship and India-International cooperation. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established 50 years ago. For further information on Dr.Buyya, please visit his cyberhome: www.buyya.com