Collocated Snapshot Backup with VM-Centric Deduplication in the Cloud

Abstract

A cloud environment that hosts a large number of virtual machines (VMs) has a high storage demand for frequent backup of system image snapshots. Deduplication of data blocks is necessary to eliminate redundant blocks and reduce cost. Collocating a cluster-based duplicate service with other cloud services reduces network traffic; however, performing global deduplication and letting a data block be shared by many virtual machines is resource expensive and less fault-resilient. This paper proposes a VM-centric collocated backup service that localizes deduplication as much as possible within each virtual machine using similarity-guided search and associates underlying file blocks with one VM for most cases. It restricts global deduplication to popular blocks with extra replication support. Our analysis shows that this VM centric scheme can provide better fault tolerance while using a small amount of computing and storage resources. This paper describes a comparative evaluation of this scheme in accomplishing competitive deduplication efficiency while sustaining a good backup throughput.

1 Introduction

In a cluster-based cloud environment, each physical machine runs a number of virtual machines as instances of a guest operating system and their virtual hard disks are represented as virtual disk image files in the host operating system. Frequent snapshot backup of virtual disk images can increase the service reliability. For example, the Aliyun cloud, which is the largest cloud service provider by Alibaba in China, automatically conducts the backup of virtual disk images to all active users every day. The cost of supporting a large number of concurrent backup streams is high because of the huge storage demand. Using a separate backup service with full deduplication support [14, 27] can effectively identify and remove content duplicates among snapshots, but such a solution can be expensive. There is also a large amount of network traffic to transfer data from the host machines to the backup facility before duplicates are removed.

This paper seeks for a low-cost architecture option that collocates a backup service with other cloud services and uses a minimum amount of resources. We also consider the fact that after deduplication, most data chunks are shared by several to many virtual machines. Failure of a few shared data chunks can have a broad effect and many snapshots of virtual machines could be affected. The previous work in deduplication focuses on the efficiency and approximation of fingerprint comparison, and has not addressed fault tolerance issues together with deduplication. Thus we also seek deduplication options that yield better fault isolation. Another issue considered is that that deletion of old snapshots also competes for computing resources. Sharing of data chunks among by multiple VMs needs to be detected during snapshot deletion and such dependencies complicate deletion operations.

The paper studies and evaluates an integrated approach which uses multiple duplicate detection strategies based on version detection, similarity guided local deduplication, and popularity guided global deduplication. This approach is VM centric by localizing duplicate detection within each VM and by packaging only data chunks from the same VM into a file system block as much as possible. By narrowing duplicate sharing within a small percent of common data chunks and exploiting their popularity, this scheme can afford to allocate extra replicas of these shared chunks for better fault resilience while sustaining competitive deduplication efficiency. Localization also brings the benefits of greater ability to exploit parallelism so backup operations can run simultaneously without a central bottleneck. This VM-centric solution uses a small amount of memory while delivering reasonable deduplication efficiency.

We have developed a prototype system that runs a cluster of Linux machines with Xen and uses a standard

distributed file system for the backup storage.

The rest of this paper is organized as follows. Section 2 reviews the background and discusses the design options for snapshot backup with a VM-centric approach. Section 3 analyzes the tradeoffs and benefits of our approach. Section 4 describes our system architecture and implementation details. Section 5 is our experimental evaluation that compares with other approaches. Section 6 concludes this paper.

2 Background and Design Considerations

At a cloud cluster node, each instance of a guest operating system runs on a virtual machine, accessing virtual hard disks represented as virtual disk image files in the host operating system. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations take place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data. Backup systems have been developed to use content fingerprints to identify duplicate content [14, 16]. As discussed earlier, collocating a backup service on the existing cloud cluster avoids the extra cost to acquire a dedicated backup facility and reduces the network bandwidth consumption in transferring the raw data for backup. Figure 1 illustrates the cluster architecture where each physical machine runs a backup service and a distributed file system (DFS) [7, 18] serves a backup store for the snapshots.

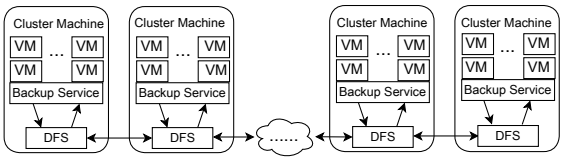


Figure 1: Collocated VM Backup System.

Since it is expensive to compare a large number of chunk signatures for deduplication, several techniques have been proposed to speedup searching of duplicate fingerprints. For example, the data domain method [27] uses an in-memory Bloom filter and a prefetching cache for data chunks which may be accessed. An improvement to this work with parallelization is in [23, 24]. The approximation techniques are studied in [2, 8, 25] to reduce memory requirements with the tradeoff of a reduced deduplication ratio. Additional inline deduplication techniques are studied in [11, 8, 19] and a parallel batch solution for cluster-based deduplication is studied in [26]. All of the above approaches have focused on optimization of deduplication efficiency, and none of them have considered the impact of deduplication on fault tolerance in the cluster-based environment that we have considered in this paper.

Our key design consideration is VM dependence minimization during deduplication and file system block management.

- *Deduplication localization.* Because a data chunk is compared with signatures collected from all VMs during the deduplication process, only one copy of duplicates is stored in the storage, this artificially creates data dependencies among different VM users. Content sharing via deduplication affects fault isolation since machine failures happen periodically in a large-scale cloud and loss of a small number of shared data chunks can cause the unavailability of snapshots for a large number of virtual machines. Localizing the impact of deduplication can increase fault isolation and resilience. Thus from the fault tolerance point of view, duplicate sharing among multiple VMs is discouraged. Another disadvantage of sharing is that it complicates snapshot deletion, which occurs frequently when snapshots expire regularly. The mark-and-sweep approach [8] is effective for deletion, but has a high cost, and its main cost is to count if a data chunk is still shared by other snapshots. Localizing deduplication can minimize data sharing and simplify deletion while sacrificing deduplication efficiency, and can facilitate parallel execution of snapshot operations.

- *Management of file system blocks.* The file system block (FSB) size in a distributed file system such as Hadoop and GFS is uniform and large (e.g. 64MB), while the data chunk in a typical deduplication system is of a non-uniform size with 4KB or 8KB on average. Packaging data chunks to a FSB can create more data dependencies among VMs since a file system block can be shared by even more VMs. Thus we need to consider a minimum association of FSBs to VMs in the packaging process.

Another consideration is the computing cost of deduplication. Because of collocation of this snapshot service with other existing cloud services, cloud providers will want the backup service to only consume small resources with a minimal impact to the existing cloud services. The key resource for signature comparison is memory for storing the fingerprints. We will consider the approximation techniques with reduced memory consumption along with the fault isolation considerations discussed below.

We call the traditional deduplication approach as VM-oblivious (VO) because they compare fingerprints of snapshots without consideration of VMs. With the above

considerations in mind, we study a VM-centric approach (called VC) for a collocated backup service with resource usage friendly to the existing applications.

In designing a VC duplication algorithm, we have considered and adopted some of the following previously-developed techniques. 1) *Version-based change detection*. VM snapshots can be backed up incrementally by identifying file segments that have changed from the previous version of the snapshot [5, 21, 20]. Such a scheme is VM-centric since deduplication is localized. We are seeking for a tradeoff since global signature comparison can deliver additional compression [8, 6, 2]. 2) *Stateless data Routing*. One approach for scalable duplicate comparison is to use a content-based hash partitioning algorithm called stateless data routing by Dong et al. [6] Stateless data routing divides the deduplication work with a similarity approximation. This work is similar to Extreme Binning by Bhagwat et al. [2] and each request is routed to a machine which holds a Bloom filter or can fetch on-disk index for additional comparison. While this approach is VM-oblivious, it motivates us to use a combined signature of a dataset to narrow VM-specific local search. 3) *Sampled Index*. One effective approach that reduces memory usage is to use a sampled index with prefetching, proposed by Guo and Efstathopoulos[8]. The algorithm is VM oblivious and it is not easy to adopt for a distributed architecture. To use a distributed memory version of the sampled index, every deduplication request may access a remote machine for index lookup and the overall overhead of access latency for all requests can be significant.

We will first discuss and analyze the integration of the VM-centric deduplication strategies with fault isolation, and then present an architecture and implementation design with deletion support.

3 VM-centric Snapshot Deduplication

3.1 Key VC Strategies

VM-specific local duplicate search within similar segments. We start with the changed block tracking approach in a coarse grain segment level. In our implementation with Xen on an Alibaba platform, the segment size is 2MB and the device driver is extended to support tracking changed segments using a dirty bitmap. Since every write for a segment will touch a dirty bit, the device driver maintains dirty bits in memory and cannot afford a small segment size. It should be noted that dirty bit tracking is supported or can be easily implemented in major virtualization solution vendors. For example, the VMWare hypervisor has an API to let external backup applications know the changed areas since last backup. The Microsoft SDK provides an API that allows external applications to monitor the VM's I/O traffic and implement such changed block tracking feature.

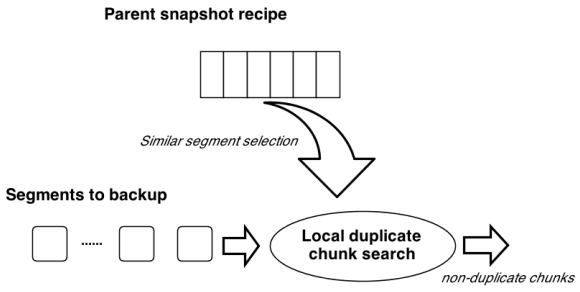


Figure 2: Similarity-guided local duplicate detection

Since the best deduplication uses a non-uniform chunk size in the average of 4KB or 8KB [9], we conduct additional local similarity guided deduplication on a snapshot by comparing chunk fingerprints of a dirty segment with those in *similar* segments from its parent snapshot. We define two segments are similar if their content signature is the same. This segment content signature value is defined as the minimum value of all its chunk fingerprints computed during backup and is recorded in the snapshot metadata (called recipe). Note that this definition of content similarity is an approximation [4]. When processing a dirty segment, its similar segments can be found easily from the parent snapshot recipe. Then recipes of the similar segments are loaded to memory, which contain chunk fingerprints to be compared. To control the time cost of search, we set a limit on the number of similar segments recipes to be fetched. For a 2MB segment, its segment recipe is roughly 19KB which contains about 500 chunk fingerprints and other chunk metadata, by limiting at most 10 similar segments to search, the amount of memory for maintaining those similar segment recipes is small. As part of our local duplicate search we also compare the current segment against the parent segment at the same offset.

Global deduplication with popular chunks and replication support. This step accomplishes the canonical global fingerprint lookup using a popular fingerprint index. Our key observation is that the local deduplication has removed most of the duplicates. There are fewer deduplication opportunities across VMs while the memory and network consumption for global comparison is more expensive. Thus our approximation is that the global fingerprint comparison only searches for the top k most popular items. This dataset is called the **PDS** (popular data set). We define chunk popularity as the number of unique copies of the chunk in the data-store, i.e., the number of copies of the chunk after local deduplication. This number can be computed periodically, e.g., on a weekly basis. Once the popularity of all data

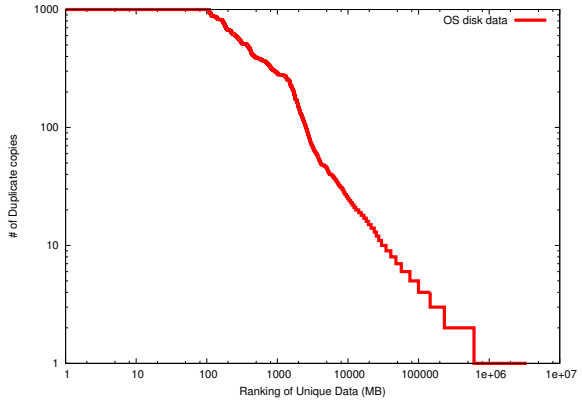


Figure 3: Duplicate frequency versus chunk ranking in a log scale after local deduplication.

chunks is collected, the system only maintains the top k most popular chunk fingerprints (called **PDS index**) in a distributed shared memory. Compared to a preliminary solution which uses data popularity [25], this paper provides a more comprehensive scheme with improved deduplication efficiency and fault tolerance, and analytic design guidance.

Since k is relatively small and these top k chunks are shared among multiple VMs, we can afford to provide extra replicas for these popular chunk data to enhance the fault resilience.

VM-centric file system block management. When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written to the file system. Since the backend file system typically uses a large block size such as 64MB, each VM will accumulate small local chunks. We manage this accumulation process using a log-structured storage scheme built on a distributed file system discussed in Section 4. Each file system block is either dedicated to non-PDS chunks, or PDS-chunks. A file system block for non-PDS chunks is associated with one VM and does not contain any PDS chunks, such that our goal of fault isolation is maintained. In addition, storing PDS chunks separately allows special replication handling for those popular shared data.

3.2 Impact on deduplication efficiency

Choosing the value k for the most popular chunks affects the deduplication efficiency. We analyze this impact based on the characteristics of the VM snapshot traces studied from application datasets. A previous study shows that the popularity of data chunks after local deduplication follows a Zipf-like distribution[3] and its exponent α is ranged between 0.65 and 0.7 [25]. Figure 3 illustrates the Zipf-like distribution of chunk popularity. The parameters we will use in our analysis below are defined in Table 1.

k	the number of top most popular chunks selected for deduplication
c	the total amount of data chunks in a cluster of VMs
c_u	the total amount of unique fingerprints after perfect deduplication
f_i	the frequency for the ith most popular fingerprint
δ	the percentage of duplicates detected in local deduplication
σ	$=\frac{k}{c_u}$ which is the percentage of unique data belonging to PDS
p	the number of machines in the cluster
E_c, E_o	deduplication efficiency of VC and VO
N_1	the average number of non-PDS FSBs in a VM for VC
N_2	the average number of PDS FSBs in a VM for VC
N_o	the average number of FSBs in a VM for VO
A(r)	the availability of an FSB with replication degree r

Table 1: Modeling parameters

By Zipf-like distribution, $f_i = f_1/i^\alpha$. The total number of chunks in our backup storage which has local duplicates excluded is $c(1-\delta)$, this can be represented as the sum of each unique fingerprint times its frequency:

$$f_1 \sum_{i=1}^{c_u} \frac{1}{i^\alpha} = c(1-\delta).$$

Given $\alpha < 1$, f_1 can be approximated with integration:

$$f_1 = \frac{c(1-\alpha)(1-\delta)}{c_u^{1-\alpha}}. \quad (1)$$

Thus putting the k most popular fingerprints into PDS index can remove the following number of chunks during global deduplication:

$$f_1 \sum_{i=1}^{k} \frac{1}{i^\alpha} \approx f_1 \int_1^k \frac{1}{x^\alpha} dx \approx f_1 \frac{k^{1-\alpha}}{1-\alpha} = c(1-\delta)\sigma^{1-\alpha}.$$

Deduplication efficiency of the VC approach using top k popular chunks is the percentage of duplicates that can be detected:

$$E_c = \frac{c\delta + c(1-\delta)\sigma^{1-\alpha}}{c - c_u}. \quad (2)$$

We store the PDS index using a distributed shared memory hash table such as Memcached and allocate a

fixed percentage of memory space per physical machine for top k popular items. As the number of physical machines (p) increases, the entire cloud cluster can host more VMs; however, ratio σ which is k/c_u remains a constant because each physical machine on average still hosts a fixed constant number of VMs. Then the overall deduplication efficiency of VC defined in Formula 2 remains constant. Thus the deduplication efficiency is stable as p increases as long as σ is a constant.

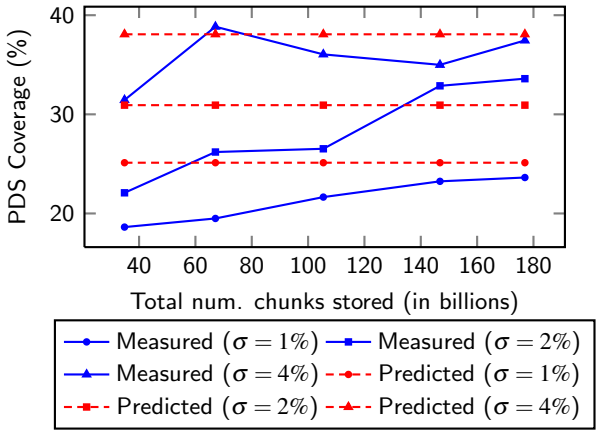


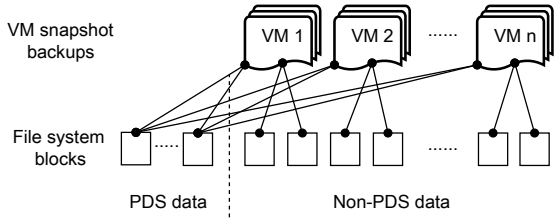
Figure 4: Predicted vs. actual PDS coverage as data size increases.

Ratio $\sigma^{1-\alpha}$ represents the percentage of the remaining chunks detected as duplicates in global deduplication due to PDS. We call this PDS coverage. Figure 4 shows predicted PDS coverage using $\sigma^{1-\alpha}$ when α is fixed at 0.65 and measured PDS coverage in our test dataset. $\sigma = 2\%$ represents memory usage of approximately 100MB memory per machine for the PDS. While the predicted value remains flat, measured PDS coverage increases as more VMs are involved. This is because the actual α value increases with the data size.

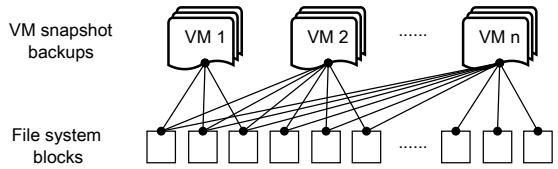
3.3 Impact on Fault Isolation

The replication degree of the backup storage is r for regular file system blocks and $r = 3$ is a typical setting in distributed file systems [7, 18]. Since σ is small (it is 2% in our experiments), the impact of replication on storage increase is very small even when we choose a large r_c/r ratio. For example, $r_c/r = 2$ or 3.

Now we assess the impact of losing d machines to the VC and VO approaches. A large r_c/r ratio can have a positive impact on full availability of VM snapshot blocks. We use FSBs rather than a deduplication data chunk as our unit of failure because the DFS keeps file system blocks as its base unit of storage. To compute the full availability of all snapshots of a VM, we derive the probability of losing a snapshot FSB of a VM by estimating the number of file system blocks per VM in each approach. As illustrated in Figure 5, we build a bipartite graph representing the association from unique file system blocks to their corresponding VMs in each approach. An association edge is drawn from an FSB to a VM if this block is used by the VM.



(a) Sharing of file system blocks under VC



(b) Sharing of file system blocks under VO

Figure 5: Bipartite association of VMs and file system blocks under (a) VC and (b) VO.

For VC, each VM has an average number of N_1 non-PDS FSBs and has an average of N_2 PDS FSBs. Each non-PDS FSB is associated with one VM and we denote that PDS FSBs are shared by an average of V_c VMs. Let s be the average number of chunks per file system block, V be the average number of VMs hosted on each machine. Then,

$$V * p * N_1 * s \approx c - E_c(c - c_u) - c_u\sigma \text{ and } V * p * N_2 * s \approx c_u\sigma V_c.$$

For VO, each VM has an average of N_o FSBs and let V_o be the average number of VMs shared by each FSB.

$$V * p * N_o * s = (c - E_o(c - c_u))V_o.$$

Since each FSB (with default size $64MB$) contains many chunks (on average 4KB), each FSB contains the hot low-level chunks shared by many VMs, and it also contains rare chunks which are not shared. Since $c >> c_u$, from the above equations:

$$\frac{N_1}{N_o} \approx \frac{1 - E_o}{(1 - E_c)V_o}.$$

When $E_c \approx E_o$ as we demonstrate in Section 5.3, N_1 would be much smaller than N_o. When there is a failure in FSBs with replication degree r and there is no failure for PDS data with more replicas, a VM in the VC approach has a much lower chance to lose a snapshot than

VO. Figure 6 shows the average number of file system blocks for each VM in VC and in VO. From data, we have observed that N_1 is much smaller than N_o. In the evaluation discussed in Section **??**, we have considered a worst case scenario that every PDS FSB is shared by all VMs in the VC approach, which leads a large N_2 value. Even with that, the availability of VC snapshots is much higher than VO and we analyze this below.

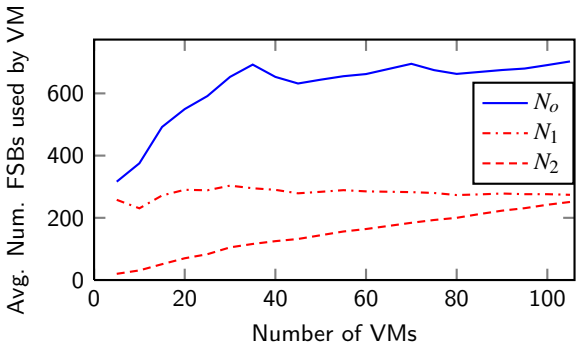


Figure 6: Measured average number of 64MB FSBs used by a single VM. For VC both the number of PDS and Non-PDS FSBs used are shown.

The full snapshot availability of a VM is estimated as follows with parameters N_1 and N_2 for VC and N_o for VO. Given normal data replication degree r, PDS data replication degree r_c, the availability of a file system block is the probability that all of its replicas do not appear in any group of d failed machines among the total of p machines. Namely, we define it as

$$A(r) = 1 - \binom{d}{r} / \binom{p}{r}.$$

Then the availability of one VM's snapshot data under VO approach is the probability that all its FSBs are unaffected during the system failure:

$$A(r)^{N_o}. \tag{3}$$

For VC, there are two cases for d failed machines.

- When $r \leq d < r_c$, there is no PDS data loss and the full snapshot availability of a VM in the VC approach is

  $$A(r)^{N_1}. \tag{4}$$

  Since N_1 is typically much smaller than N_o, the VC approach has a higher availability of VM snapshots than VO.

- When $r_c \leq d$, both non-PDS and PDS file system blocks in VC can have a loss. The full snapshot availability of a VM in the VC approach is

  $$A(r)^{N_1} * A(r_c)^{N_2} \tag{5}$$

Failures (d)	$A(r_c)$		
	$r_c = 3$	$r_c = 6$	$r_c = 9$
3	99.999381571	100	100
5	99.993815708	100	100
10	99.925788497	99.999982383	99.999999999
20	99.294990724	99.996748465	99.99999117

Table 2: $A(r_c)$ as storage nodes fail in a 100 node cluster.

Table 2 lists the $A(r)$ values with different replication degrees. This demonstrates inequality $A(r) < A(r_c)$ because of $r < r_c$. Since N_1 is much smaller than N_o as discussed previously, these two conditions support that VC has a higher availability.

4 Architecture and Implementation Details

Our system runs on a cluster of Linux machines with Xen-based VMs and an open-source package for the distributed file system called QFS [13]. All data needed for VM services, such as virtual disk images used by runtime VMs, and snapshot data for backup purposes, reside in this distributed file system. One physical node hosts tens of VMs, each of which accesses its virtual machine disk image through the virtual block device driver (called TapDisk[22] in Xen).

4.1 Components of a cluster node

As depicted in Figure 7, there are four key service components running on each cluster node for supporting backup and deduplication: 1) a virtual block device driver, 2) a snapshot deduplication component, 3) an append store client to store and access snapshot data, and 4) a PDS client to support PDS metadata access.

We use the virtual device driver in Xen that employs a bitmap to track the changes that have been made to the virtual disk. Every bit in the bitmap represents a fix-sized (2MB) region called a *segment*, indicating whether the segment has been modified since last backup. Segments are further divided into variable-sized chunks (average 4KB) using a content-based chunking algorithm [10], which brings the opportunity of fine-grained deduplication. When the VM issues a disk write, the dirty bit for the corresponding segment is set and this indicates such a segments needs to be checked during snapshot backup. After the snapshot backup is finished, the driver resets the dirty bit map to a clean state. For data modification during backup, a copy-on-write protection is set so that backup can continue to copy a specific version while new changes are still recorded.

The representation of each snapshot has a two-level index data structure. The snapshot meta data (called snap-

shot recipe) contains a list of segments, each of which contains segment metadata of its chunks (called segment recipe). In snapshot and segment recipes, the data structures include references to the actual data location to eliminate the need for additional indirection.

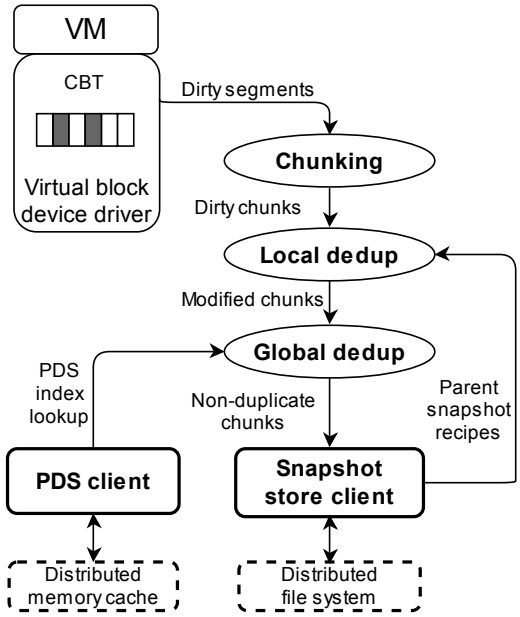


Figure 7: System architecture and data flow during snapshot write

4.2 A VM-centric snapshot store for backup data

We build the snapshot storage on the top of a distributed file system. Following the VM-centric idea for the purpose of fault isolation, each VM has its own snapshot store, containing new data chunks which are considered to be non-duplicates. There is also a special store containing all PDS chunks shared among different VMs. As shown in Figure 8, we explain the data structure of snapshot stores as follows.

Data of each VM snapshot store, excluding the PDS store, are divided into a set of containers and each container is approximately 1GB. The reason for dividing the snapshot into containers is to simplify the compaction process conducted periodically. As discussed later, data chunks are deleted from old snapshots and chunks without any reference from other snapshots can be removed by this compaction process. By limiting the size of a container, we can effectively control the length of each round of compaction. The compaction routine can work on one container at a time and move the in-use data chunks to another container.

Each container is further divided into a set of chunk

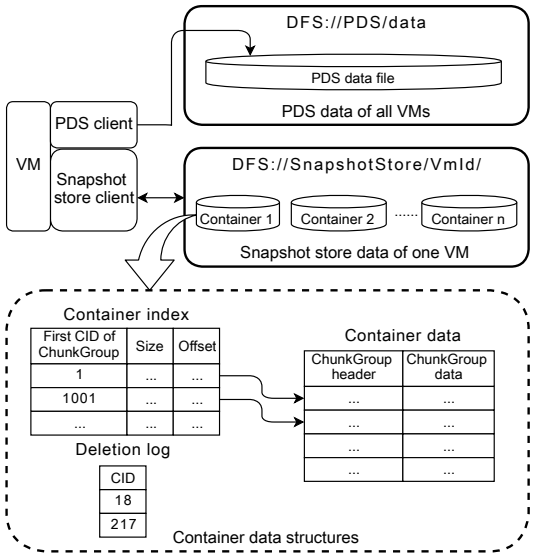


Figure 8: Data structure of a VM snapshot store.

data groups. Each chunk group is composed of a set of data chunks and is the basic unit in data access and retrieval. In writing a chunk during backup, the system accumulates data chunks and stores the entire group as a unit after a compression. When accessing a particular chunk, its chunk group is retrieved from the storage and uncompressed. Given the high spatial locality and usefulness of prefetching in snapshot chunk accessing [8, 17], retrieval of a data chunk group naturally works well with prefetching. A typical chunk group contains 100 to 1000 chunks, with an average size of 200-600 chunks.

Each data container is represented by three files in the DFS: 1) the container data file holds the actual content, 2) the container index file is responsible for translating a data reference into its location within a container, and 3) a chunk deletion log file records all the deletion requests within the container.

A data chunk reference stored in the index of snapshot recipes is composed of two parts: a container ID with 2 bytes and a local chunk ID with 6 bytes. Each container maintains a local chunk counter and assigns the current number as a chunk ID when a new chunk is added to this container. Since data chunks are always appended to a snapshot store during backup, local chunk IDs are monotonically increasing. When a snapshot chunk is to be accessed, the snapshot recipe contains a reference pointing to a data chunk in the PDS store or in a non-PDS VM snapshot store. Using a container ID, the corresponding container index file of this VM is accessed and the chunk group is identified using a simple chunk ID range search. Once the chunk group is loaded to memory, its header contains the exact offset of the corresponding chunk ID

and the content is then accessed from the memory buffer.

The PDS chunks are stored in one PDS file. Each reference to a PDS data chunk in the PDS index is the offset within the PDS file. There is an option to use a data structure similar to the non-PDS store. We opt for the simpler format because PDS chunks have less spatial locality in data access and it is not so effective to group a large number of chunks as a compression and data fetch unit.

PDS data is re-calculated periodically, but the total data size is small. When a new PDS data set is computed, in-memory PDS index is replaced, but the PDS file on the disk appends the new PDS data identified and the growth of this file is very slow. The old data is not removed because they can still be referenced by the existing snapshots. A periodic cleanup is conducted to remove unused PDS chunks (e.g. every few months).

We decide not to take the sampled index approach [8] for detecting duplicates from PDS. This is because for the data sets we have tested, we only observed very limited spatial locality among popular data chunks. On average the number of consecutive PDS index hits is lower than 7, which is not sufficient to enable effective prefetching which is required for index sampling.

The snapshot store supports three API calls:

Append(). For PDS data, the chunk is appended to the end of the PDS file and the offset is returned as the reference. For non-PDS data, this call places a chunk into the snapshot store and returns a reference to be stored in the recipe metadata of a snapshot. The write requests to append data chunks to a VM store are accumulated at the client side. When the number of write requests reaches a fixed group size, the snapshot store client compresses the accumulated chunk group, adds a chunk group index to the beginning of the group, and then appends the header and data to the corresponding VM file. A new container index entry is also created for each chunk group and is written to the corresponding container index file.

Get(). The fetch operation for the PDS data chunk is straightforward since each reference contains the file offset, and the size of a PDS chunk is available from a segment recipe. We also maintain a small data cache for the PDS data service to speedup common data fetching.

To read a non-PDS chunk using its reference with container ID and local chunk ID, the snapshot store client first loads the corresponding VM's container index file specified by the container ID, then searches the chunk groups using their chunk ID coverage. After that, it reads the identified chunk group from DFS, decompresses it, and seeks to the exact chunk data specified by the chunk ID. Finally, the client updates its internal chunk data cache with the newly loaded content to anticipate future sequential reads.

Delete(). Chunk deletion occurs when a snapshot expires or gets deleted explicitly by a user and we will discuss the snapshot deletion in detail in the following subsection. When deletion requests are issued for a specific container, those requests are simply recorded into the container's deletion log initially and thus a lazy deletion strategy is exercised. Once local chunk IDs appear in the deletion log, they will not be referenced by any future snapshot and can be safely deleted when needed. This is ensured because we only dedup against the direct parent of a snapshot, so the deleted snapshot's blocks will only be used if they also exist in other snapshots. Periodically, the snapshot store picks those containers with an excessive number of deletion requests to compact and reclaim the corresponding disk space. During compaction, the snapshot store creates a new container (with the same container ID) to replace the existing one. This is done by sequentially scanning the old container, copying all the chunks that are not found in the deletion log to the new container, and creating new chunk groups and indices. Every local chunk ID however is directly copied rather than re-generated. This process leaves holes in the CID values, but preserves the order and IDs of chunks. As a result, all data references stored in recipes are permanent and stable, and the data reading process is as efficient as before. Maintaining the stability of chunk IDs also ensures that recipes do not depend directly on physical storage locations, which simplifies data migration.

4.3 VM-centric Approximate Snapshot Deletion with Leak Repair

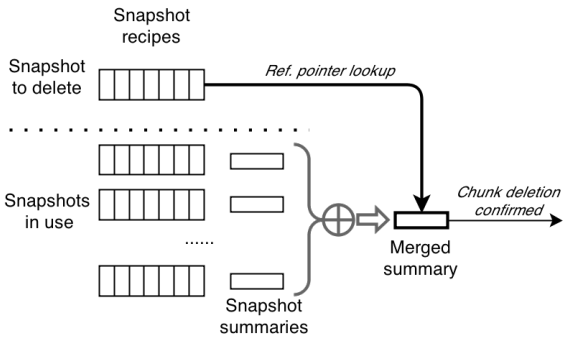


Figure 9: Approximate deletion

In a busy VM cluster, snapshot deletions can occur frequently. Deduplication complicates the deletion process because space saving relies on the sharing of data and it requires the global reference of deleted chunks to be identified before they can be safely removed. While we can use the mark-and-sweep technique [8], it takes significant resources to conduct this process every time there is a snapshot deletion. In the case of Alibaba, snap-

shot backup is conducted automatically and there are about 10 snapshot stored for every VM customer. When there is a new snapshot created every day, there will be a snapshot expired every day to maintain balanced storage use.

We seek a fast solution with low resource usage to delete snapshots and the VM-centric design simplifies the deletion process. Since PDS is small and separated, we can focus on unreferenced non-PDS chunks within each VM. Another resource-saving strategy we propose is an *approximate* deletion strategy to trade deletion accuracy for speed and resource usage. Our method sacrifices a small percent of storage leakage to efficiently identify unused chunks. The algorithm contains three aspects.

- **Computation for snapshot fingerprint summary.** Every time there is a new snapshot created, we compute a Bloom-filter with z bits as the reference summary for all non-PDS chunks used in this snapshot. Given h snapshots stored for a VM, there are h summary vectors maintained.

- **Approximate deletion with fast summary comparison.** When there is a snapshot deletion, we need to identify if chunks to be deleted from that snapshot are still used by other snapshots. This is done approximately and quickly by comparing the reference of deleted chunks with the merged Bloom-filter reference summary of other live snapshots. The merging of live snapshot Bloom-filter bits uses the bitwise OR operator and the merged vector still takes z bits. Since the number of live snapshots h is limited for each VM, the time and memory cost of this comparison is small, linear to the number of chunks to be deleted.

  If a chunk's reference is not found in the merged summary vector, we are sure that this chunk is not used by any live snapshots, thus can be deleted safely. However, among all the chunks to be deleted, there are a small percentage of unused chunks which are misjudged as being in use, resulting in storage leakage.

- **Periodic repair of leakage**. Leakage repair is conducted periodically to fix the above approximation error. This procedure compares the live chunks for each VM with what are truly used in the VM snapshot recipes. That requires a scanning of all chunks in a VM. Since it is a VM-specific procedure the space and time cost is relatively small compared to the traditional mark-sweep procedure [8] which scans snapshot chunks from all VMs. For example, consider each reference consumes 8 bytes plus 1 mark bit, a VM that has 40GB backup data with about 10 million chunks will need less than 85MB of memory to complete a VM-specific mark-sweep process.

We now estimate the size of storage leakage and how often leak repair needs to be conducted. Assume that a VM keeps h snapshots in the backup storage, creates and deletes one snapshot every day. Let u be the total number of chunks brought by the initial backup for a VM, Δu be the average number of additional unique chunks added from one snapshot to the next snapshot version. Then the total number of unique chunks used in a VM is about:

$$U = u + (h-1)\Delta u.$$

Each Bloom filter vector has z bits for each snapshot and let j be the number of hash functions used by the Bloom filter. Notice that a chunk may appear multiple times in these summary vectors; however, this should not increase the probability of being a 0 bit in all h summary vectors. Thus the probability that a particular bit is 0 in all h summary vectors is $(1-\frac{1}{z})^{jU}$. Then the misjudgment rate of being in use is:

$$\varepsilon = (1-(1-\frac{1}{z})^{jU})^{j}. \tag{6}$$

For each snapshot deletion, the number of chunks to be deleted is nearly identical to the number of newly added chunks Δu. Let R be the total number of runs of approximate deletion between two consecutive repairs. We estimate the total leakage L after R runs as:

$$L = R\Delta u\varepsilon.$$

When leakage ratio L/U exceeds a pre-defined threshold τ, we derive the condition to execute a leak repair as:

$$\frac{L}{U} = \frac{R\Delta u\varepsilon}{u+(h-1)\Delta u} > \tau \Longrightarrow R > \frac{\tau}{\varepsilon} \times \frac{u+(h-1)\Delta u}{\Delta u}. \tag{7}$$

For example in our tested dataset, each VM keeps $h = 10$ snapshots and each snapshot has about 1-5% of new data. Thus $\frac{\Delta u}{u} \leq 0.05$. For a 40GB snapshot, $u \approx 10$ million. Then $U = 10.45$ million. We choose $\varepsilon = 0.01$ and $\tau = 0.1$. From Equation 6, $z = 10U = 100.45$ million bits. From Equation 7, leak repair should be triggered once for every R=290 runs of approximate deletion. When one machine hosts 25 VMs and there is one snapshot deletion per day per VM, there would be only one full leak repair for one physical machine scheduled for every 12 days. Each repair uses at most 90MB memory on average as discussed earlier and takes a short period of time.

5 Evaluation

We have implemented and evaluated a prototype of our VC scheme on a Linux cluster of machines with 8-core 3.1Ghz AMD FX-8120 and 16 GB RAM. Our implementation is based on Alibaba cloud platform [1, 25] and the underlying DFS uses QFS with default replication degree 3 while the PDS replication degree is 6. Our evaluation objective is to study the benefit in fault tolerance and deduplication efficiency of VC and compare it with an alternative VO design. We also and assess backup throughput and resource usage for a large number of VMs.

We will compare VC with a VO approach using stateless routing with binning (SRB) based on [6, 2]. SRB executes a distributed deduplication by routing a data chunk to one of cluster machines [6] using a min-hash function discussed in [2]. Once a data chunk is routed to a machine, this chunk is compared with the fingerprint index within this machine locally.

5.1 Settings

We have performed a trace-driven study using a production dataset [25] from Alibaba Aliyun's cloud platform with about 100 machines. Each machine hosts up to 25 VMs and each VM keeps 10 automatically-generated snapshots in the storage system while a user may instruct extra snapshots to be saved. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, win2008 and win2003. The backup of VM snapshots is required to complete within few hours every night. Based on our study of production data, each VM has about 40GB of storage data on average including OS and user data disk. The signature for variable-sized blocks is computed using their SHA-1 hash [12, 15].

5.2 Impact on Fault Isolation

Table 3 shows the availability of VM snapshots when there are up to 20 machine nodes failed in a 100-node cluster and a 1000-node cluster. We assume that every VM points to every file system block in the PDS ($V_c = p * V$), which is to be expected when PDS blocks are sorted by hash, as their order is randomized. To obtain VO sharing of file blocks, we perform perfect deduplication over 105 VMs, append unique chunks to a DFS file and count the number of dependencies from a file system block to VM. Our results show that even the worst case for VC (where $v_c = p * V$) has a higher availability than VO. For example, with 5/100 machines failed and 25 VMs per machine, VO with 93.256% availability would lose data in 169 VMs while VC with 97.763% loses data for 56 VMs. The key reason is that for most data in VC, only a single VM can be affected by the loss of a single FSB. Since most FSBs contain blocks for a single VM, VMs can point to a smaller number of FSBs.

Failures (d)	VM Availability(%)			
	$p = 100$		$p = 1000$	
	VO	VC	VO	VC
3	99.304248	99.773987	99.999321	99.99978
5	93.256135	97.762659	99.993206	99.997798
10	43.251892	76.093998	99.918504	99.97358
20	0.03397	5.613855	99.228458	99.749049

Table 3: Availability of VM snapshots for VO and VC.

Although the loss of a PDS block affects many VMs, by increasing replication for those blocks we minimize the effect on VM availability. Figure 10 shows the impact of increasing PDS data replication. While the impact on storage cost is small, a replication degree of 6 has a significant improvement over 4, but the availability is about the same for $r_c = 6$ and $r_c = 9$ (beyond $r_c = 6$ improvements are minimal).

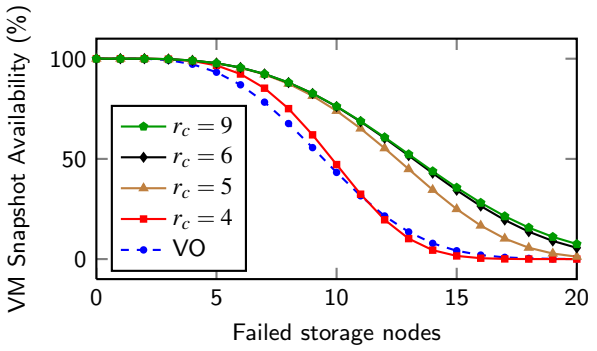


Figure 10: Availability of VM snapshots in VC with different PDS replication degrees

5.3 Deduplication Efficiency

Figure 11 shows the deduplication efficiency for SRB and VC. Deduplication efficiency is defined as the percent of duplicate chunks which are detected and deduplicated, so only global perfect deduplication can have 100% efficiency. We also compare several PDS sizes chosen for VC. With $\sigma = 2\%$, memory usage for PDS index lookup per machine is about 100MB per machine and the deduplication efficiency can reach over 96.33%. When $\sigma = 4\%$, the deduplication efficiency can reach 96.9% while space consumption increases to 200MB per machine. The loss of efficiency in VC is caused by the restriction of the physical memory available in the cluster to support fast in-memory PDS index lookup. SRB can deliver up to 97.79% deduplication efficiency which is slightly better than VC. Thus this represents a tradeoff

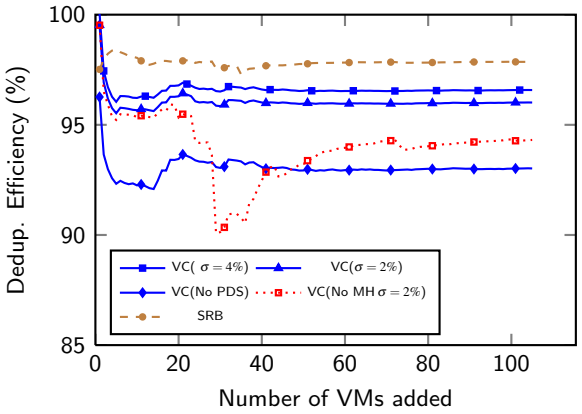


Figure 11: Deduplication efficiency of VC and SRB.

that VC provides better fault tolerance and fast approximate deletion with competitive deduplication efficiency.

Figure 11 also shows the effectiveness of the local similarity-based search in VC. Without this (i.e. when we only check the parent segment at the same offset), when an entire segment is moved to another location on disk, for example when a file is rewritten rather than modified in place, the deduplication opportunity is missed. We have found that in approximately 1/3 of the VMs in our dataset this movement happens frequently, so the similarity search becomes important for those VMs. In general, our experiments show that dirty-bit detection at the segment level can reduce the data size to about 24.14% of original data, which leads to about a 75.86% reduction. Similarity-guided local search can further reduce the data size to about 12.05% of original, namely it delivers a 50.08% reduction to the dirty segments. The popularity-guided global deduplication with $\sigma = 2\%$ can reduce the data further to 8.6% of its original size, so it provides additional 28.63% reduction to the remaining data.

5.4 Resource Usage and Processing Time

Storage cost of replication. The total storage for all VM snapshots in each physical machine takes about 3.065TB on average when the replication degree of both PDS and non-PDS data is 3. Allocating one extra copy for PDS data only adds 7GB in total per machine. Thus PDS replication degree 6 only increases the total space by 0.685% while PDS replication degree 9 adds 1.37% space overhead, which is still small.

Memory and disk bandwidth usage with multi-VM processing. We have further studied the memory and disk bandwidth usage when running concurrent VM snapshot backup on each machine with $\delta = 2\%$. Table 4 gives the resource usage when running 1 or multiple VM backup tasks at the same time on each physical machine. "CPU" column is the percentage of a single core used. "Mem" column includes 100MB memory usage for PDS index and other space cost for executing deduplication tasks such as receipt metadata and cache. "Read" column is controlled as 50MB/s bandwidth usage so that other cloud services are not impacted a lot. "Write" column is the I/O write usage of QFS and notice that each QFS write leads writes in multiple machines due to data replication.

Tasks	CPU	Mem (MB)	Read (MB/s)	Write (MB/s)	Time (hrs)
1	19%	118	50	8.4	1.31
2	35%	132	50	9.0	1.23
4	63%	154	50	9.3	1.18

Table 4: Resource usage of concurrent backup tasks at each machine

Table 4 shows that a single backup task per node can complete the backup of the entire VM cluster in about 1.58 hours. Since there are about 25 VMs per machine, we could execute more tasks in parallel at each machine. But adding more backup concurrency does not shorten the overall time a lot because of the controlled disk read time.

Processing Time breakdown. Figure 12 shows the average processing time of a VM segment under VC and SRB. VC uses $\sigma = 2\%$ and 4%. It has a breakdown of time for reading data, updating the metadata, network latency to visit a remote machine, and index access for fingerprint comparison. The change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time. SRB has higher index access and fingerprint comparison costs because all chunk signatures for whom the current machine does not control the index must be sent over the network, and then rely on the index holding machine to access its index (often on the disk, supported by a Bloom filter) and perform comparison. VC is faster for index access because it conducts in-memory local search first and then accesses the PDS on distributed shared memory, so most chunks can be dealt with locally. SRB spends slighter more time in reading data and updates because it also updates the on-disk local meta data index in addition to partition indices. Overall, VC is faster than SRB, though data reading dominates the processing time for both algorithms.

Throughput of software layers. Table 5 shows the average throughput of software layer when I/O is throttled under 50MB/second and when there is no restriction. "Backup" column is the throughput of the backup service per machine. "Snapshot store" is the write throughput of the snapshot store layer and the gap between this column and "Raw" column is caused by signifi-

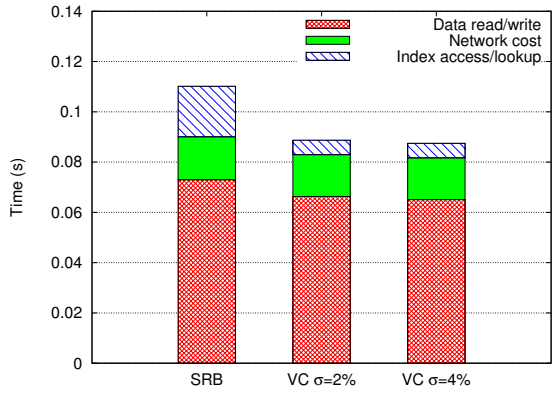


Figure 12: Average time to backup a dirty VM segment under SRB and VC

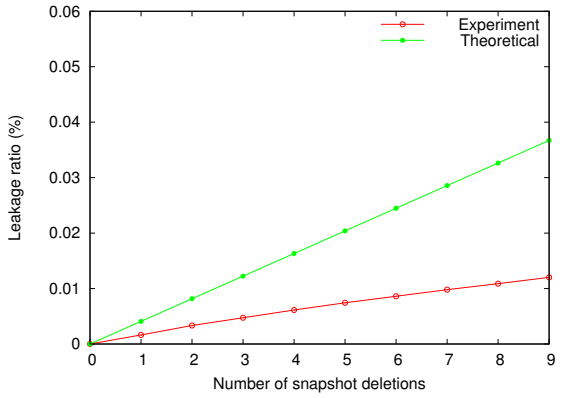


Figure 13: Accumulated storage leakage by approximate snapshot deletions

Num. of concurrent backup tasks per node	Throughput without I/O throttling (MB/s)		
	Backup	Snapshot Store	QFS
1	1369.6	148.0	35.3
2	2408.5	260.2	61.7
4	4101.8	443.3	103.1
6	5456.5	589.7	143.8

Table 5: Throughput of software layers under different concurrency

cant data reduction by dirty bit and duplicate detection. Only non-duplicate chunks trigger a snapshot store write. "QFS" column is the write request traffic to the underlying file system after compression. The underlying disk storage traffic will be three times greater because of replication. The result shows that the backup service can deliver up to 5.46GB/second without I/O restriction per machine with 6 concurrent backup tasks. With 50MB/second controlled I/O bandwidth, each machine can deliver 171MB/second with 1 backup task and can complete the backup of 25 VMs per machine in less than 1.31 hours.

5.5 Effectiveness of Approximate Deletion

Figure 13 shows the average accumulated storage leakage in terms of percentage of storage space per VM caused by approximate deletions. The solid line is the predicted leakage using Formula 7 from Section 4.3 while the dashed line is the actual leakage measured during the experiment. After 9 snapshot deletions, the actual leakage reaches 0.01% and this means that there is only 1MB space leaked for every 10GB of stored data.

6 Conclusion

In this paper we propose a collocated backup service built on the top of a cloud cluster to reduce network traffic and infrastructure. The key contribution is a VM-centric deduplication scheme to maximize fault isolation while delivering competitive deduplication efficiency. Similarity guided local search reduces cross-VM data dependency and exposes more parallelism while global deduplication with a small common data set eliminates popular duplicates. VM-specific file block packing also enhances fault tolerance by reducing data dependencies. The design places a special consideration for low-resource usages as a collocated cloud service. Evaluation using VM backup data shows that VC strikes a tradeoff and can accomplish close to 97% of what complete global deduplication can do while the availability of snapshots increases substantially with this VM-centric and a small replication overhead for popular inter-VM chunks.

References

[1] Alibaba Aliyun. http://www.aliyun.com.

[2] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In *IEEE MASCOTS '09*, pages 1–9, 2009.

[3] L. Breslau, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is*

Now (Cat. No.99CH36320), pages 126–134 vol.1. IEEE, 1999.

[4] A. Broder. On the Resemblance and Containment of Documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[5] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in san cluster file systems. In *USENIX ATC'09*, 2009.

[6] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[8] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *USENIX ATC'11*, pages 25–25, 2011.

[9] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09*, page 1, New York, New York, USA, May 2009. ACM Press.

[10] E. Kave and T. H. Khuern. A Framework for Analyzing and Improving Content-Based Chunking Algorithms. Technical Report HPL-2005-30R1, HP Laboratory, Oct. 2005.

[11] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *FAST'09*, pages 111–123, 2009.

[12] U. Manber. Finding similar files in a large file system. In *USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[13] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, J. Kelly, C. Zimmermanand, D. Adkins, T. Subramaniam, and J. Fishman. The quantcast file system. In *Proceedings of the 39th international conference on Very Large Data Bases*, VLDB'13, pages 2–2, Berkeley, CA, USA, 2013. VLDB.

[14] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *FAST '02*, pages 89–101, 2002.

[15] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, 1981.

[16] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content-addressed storage in foundation. In *USENIX ATC'08*, pages 143–156, Berkeley, CA, USA, 2008. USENIX Association.

[17] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content-addressed storage in foundation. In *USENIX ATC'08*, pages 143–156, 2008.

[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[19] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[20] Y. Tan, H. Jiang, D. Feng, L. Tian, and Z. Yan. Cabdedupe: A causality-based deduplication performance booster for cloud backup services. In *IPDPS'11*, pages 1266–1277, 2011.

[21] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *FAST'09*, pages 225–238, 2009.

[22] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. page 22, Apr. 2005.

[23] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: A scalable high-throughput exact deduplication approach for network backup services. In *IEEE MSST'10*, pages 1–14, May 2010.

[24] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan. Debar: A scalable high-performance deduplication storage system for backup and archiving. In *IEEE IPDPS*, pages 1–12, 2010.

[25] W. Zhang, H. Tang, H. Jiang, T. Yang, X. Li, and Y. Zeng. Multi-level selective deduplication for vm snapshots in cloud storage. In *IEEE CLOUD'12*, pages 550–557, 2012.

[26] W. Zhang, T. Yang, G. Narayanasamy, and H. Tang. Low-cost data deduplication for virtual machine backup in cloud storage. In *Proceedings of the 5th USENIX conference on Hot Topics in Storage and File Systems*, HotStorage'13, pages 2–2, Berkeley, CA, USA, 2013. USENIX Association.

[27] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08*, pages 1–14, 2008.