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 chunks with extra replication support. Our analysis shows that this VM centric scheme can strike a balance for better fault isolation with competitive deduplication efficiency while using a small amount of computing resources. This paper describes a comparative evaluation of this scheme to assess its deduplication efficiency and 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 [**?**, **?**] 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 tradeoff and benefits of our approach. Section 4 describes our system architecture and implementation details. Section **??** is our experimental evaluation that compares with other approaches. Section **??** 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 [**?**, **?**]. 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) [**?**, **?**] serves a backup store for the snapshots.

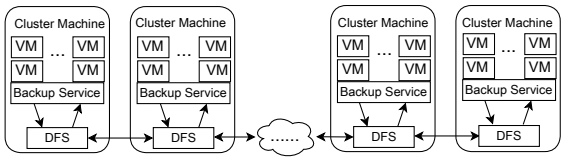


Figure 1: Collocated VM Backup System.

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 [**?**] 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 [**?**, **?**]. The approximation techniques are studied in [**?**, **?**, **?**] to reduce memory requirements with the tradeoff of a reduced deduplication ratio. Additional inline deduplication techniques are studied in [**?**, **?**, **?**] and a parallel batch solution for cluster-based deduplication is studied in [**?**]. 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 [**?**] 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 an 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 [**?**, **?**, **?**]. Such a scheme is VM-centric since deduplication is localized. We are seeking for a tradeoff since global signature comparison can deliver additional compression [**?**, **?**, **?**]. 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. [**?**] Stateless data routing divides the deduplication work with a similarity approximation. This work is similar to Extreme Binning by Bhagwat et al. [**?**] 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[**?**]. 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.

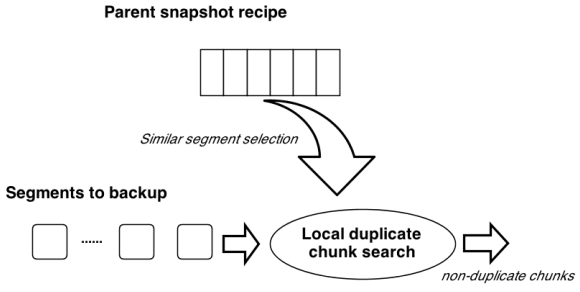


Figure 2: Similarity-guided local duplicate detection

Since the best deduplication uses a non-uniform chunk size in the average of 4KB or 8KB [**?**], 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 [**?**]. 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

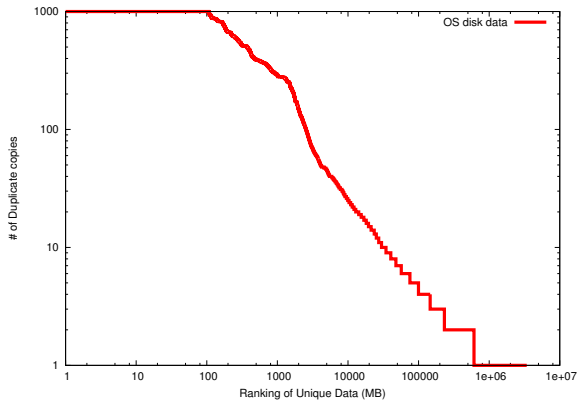


Figure 3: Duplicate frequency versus chunk ranking in a log scale after local deduplication.

chunks is collected, the system only maintains the top k most popular chunk fingerprints (called **PDS index**) in a distributed shared memory.

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[**?**] and its exponent α is ranged between 0.65 and 0.7 [**?**]. Figure 3 illustrates the Zipf-like distribution of chunk popularity. The parameters we will use in our analysis below are defined in Table 1.

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

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 blocks 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

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.

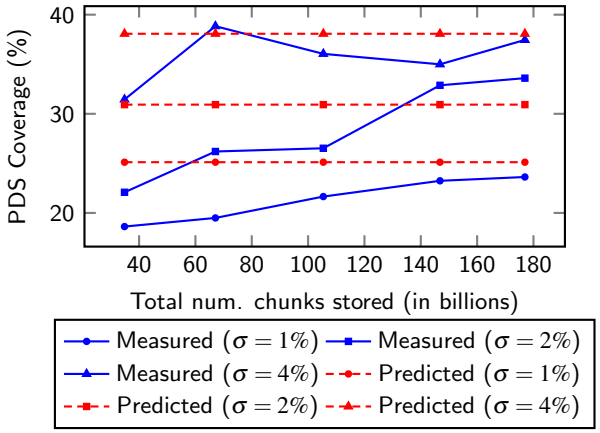


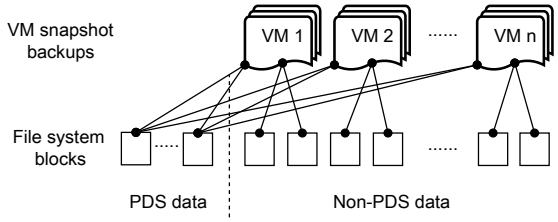
Figure 4: Predicted vs. actual PDS coverage as data size increases.

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.

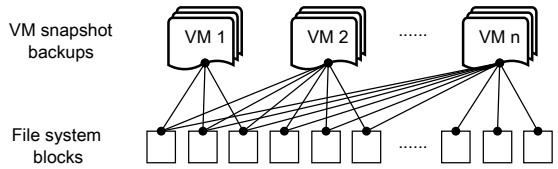
3.3 Impact on Fault Isolation

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 [**?**, **?**]. Since σ is small (e.g. 2% in our experiments), the impact of replication on storage increase is very small even when choosing r_c/r ratio as 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 an FSB 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 N_1 of 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,

$$VpN_1s \approx c - E_c(c - c_u) - c_u\sigma \text{ and } VpN_2s \approx c_u\sigma V_c.$$

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.

$$VpN_os = (c - E_o(c - c_u))V_o.$$

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 is close to E_o as demonstrated in Section **??**, N_1 is much smaller than N_o. Figure 6 shows the average number of file system blocks for each VM in VC and in VO and N_1 is indeed much smaller than N_o in our tested dataset.

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

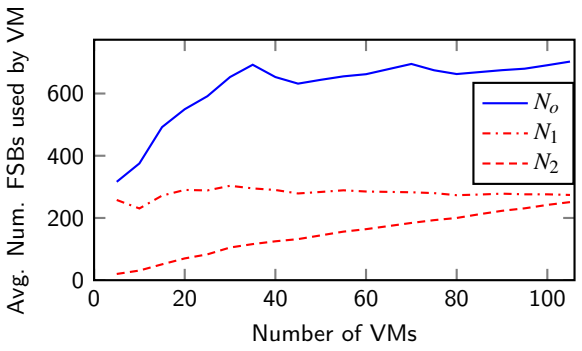


Figure 6: Measured average number of 64MB FSBs used by a single VM. For VC both the number of PDS and Non-PDS FSBs used are shown.

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. 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.

- 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

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

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 to a large N_2 value. Even with that, the availability of VC snapshots is still much higher than VO and there are two reasons. 1) N_1 is much smaller than N_o as discussed previously. 2) $A(r) < A(r_c)$ because of $r < r_c$. Table 2 lists the $A(r)$ values with different replication degrees, to demonstrate the gap between $A(r)$ and $A(r_c)$.

Failures (d)	$A(r_c) \times 100\%$		
	$r_c = 3$	$r_c = 6$	$r_c = 9$
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.

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 [**?**]. 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[**?**] 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 fixed-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 [**?**], 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, 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.

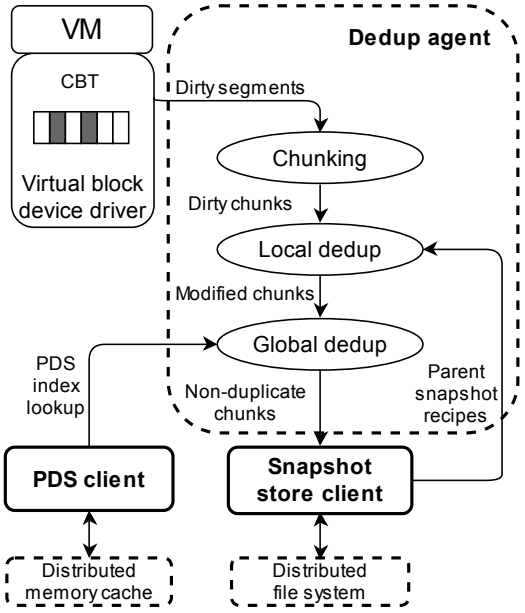


Figure 7: System architecture and data flow during snapshot write

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 the 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

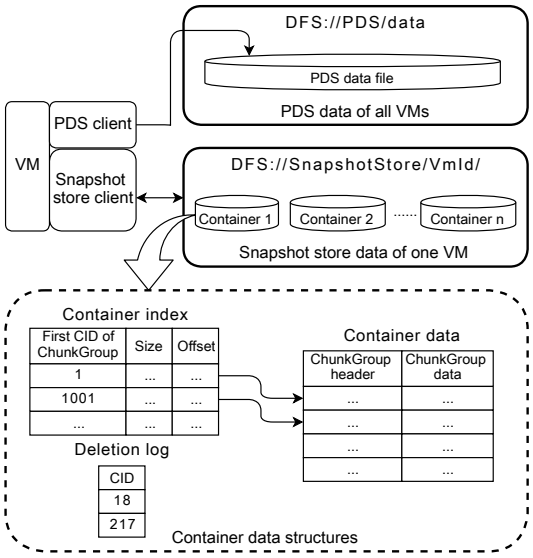


Figure 8: Data structure of a VM snapshot store.

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 compression. When accessing a particular chunk, its chunk group is retrieved from the storage and decompressed. Given the high spatial locality and usefulness of prefetching in snapshot chunk accessing [**?**, **?**], 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.