Collocated Deduplication with Fault Isolation for Virtual Machine Snapshot Backup [draft]

Wei Zhang[⋆]*, Michael Agun*[⋆]*, Tao Yang*[⋆]*, and Hong Tang*[†]
[⋆] *University of California at Santa Barbara,* [†]*Alibaba Inc.*
{*wei, tyang, dagun*}*@cs.ucsb.edu, hongtang@alibaba-inc.com*

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 and signature-based deduplication of data blocks is necessary to eliminate excessive redundant blocks. Collocating a cluster-based duplicate service with other cloud services reduces network traffic; however it is resource expensive and less fault-resilient to perform a global deduplication and let a data block share by many virtual machines. 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 cross-VM deduplication under common 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 a 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 [12, 24] 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 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 competes for computing resource as well. 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, inner VM duplicate search, and controlled cross-VM comparison. 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 a 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 [12, 14]. 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) [6, 16] serves a backup store for the snapshots.

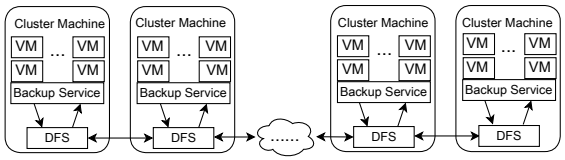


Figure 1: Collocated VM Backup System.

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 [24] 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 [21, 22]. The approximation techniques are studied in [2, 7, 23] to reduce memory requirements with the tradeoff of a reduced deduplication ratio. Additional inline deduplication techniques are studied in [10, 7, 17]. 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 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-sweep approach [7] is effective for deletion, 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 faciltate parallel execition of snaptshot operations.

- *Management of file system blocks.* The file block size in a distributed file system such as Hadoop and GFS is uniform and large (e.g. 64MB), a data chunk in a typical deduplication system is of a nonuniform size with 4KB or 8KB on average. Packaging data chunks to a file system block can create more data dependence among VMs since a file block can be shared with even more VMs. Thus we need to consider a minimum association of a fie system block 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 less memory consumption along with the fault isolation consideration discussed below.

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

- **Version-based change detection**. VM snapshots can be backed up incrementally by identifying file blocks that have changed from the previous version of the snapshot [4, 19, 18]. Such a scheme is VM-centric since deduplication is localized. We are seeking for a tradeoff since cross-VM signature comparison can deliver additional compression [7, 5, 2].

- **Stateless Data Routing**. One approach for scalable duplicate comparison is to use a content-based hash partitioning algorithm called stateless data routing [5] that divides the deduplication work with a similarity approximation. This work is similar to Extreme Binning[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.

- **Sampled Index**. One effective approach that reduces memory usage is to use a sampled index with prefetching, proposed by Guo and Efstathopoulos[7]. 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 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 VM-centric Strategies

- **VM-specific local duplicate search within similar segments.** We start with the standard dirty bit 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 changed block tracking. 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 dirtybit 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 the changed block tracking feature.

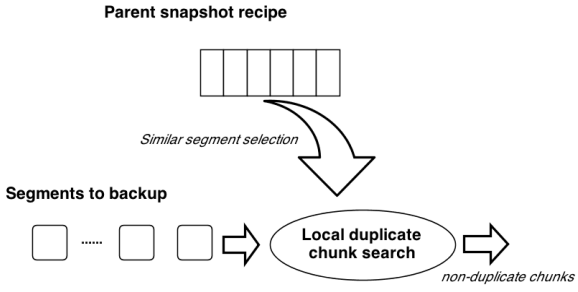


Figure 2: Min-hash based local deduplication

Since the best deduplication uses a nonuniform chunk size in the average of 4K or 8K [8], we conduct additional local inner-VM deduplication on a snapshot by comparing dirty segments' chunk fingerprints within *similar* segments from its parent snapshot. We define two segments are similar if their approximated content signature is the same. This segment signature value is defined as the minimum value of all its chunk fingerprints computed during backup and is recorded in the snapshot recipe. When we are processing a dirty segment, we can easily find similar segments from the parent snapshot recipe and load their segment recipes and also set a limit on the the number of similar segments to be searched. Then, given a set of data chunks within a dirty segment, we compare these chunk fingerprints with those in similar segments. For example, with a 2MB segment, there are about 500 fingerprints to compare. By limiting at most 10 similar segments to search, the amount of memory for maintaining those fingerprints in similar segment s is small.

- **Cross-VM deduplication with popular chunks and replication support** This step accomplishes the standard global fingerprint comparison as conducted in the previous work [23]. One key observation is that the inner-VM deduplication has removed many 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). The popularity of a chunk

is the number of data chunks from different VMs that are duplicates of this chunk after the inner VM deduplication. This number can be computed periodically (e.g., on a weekly basis). Once the popularity of all data chunks is collected, the system only maintains the top k most popular chunk signatures 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 chunks 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 an append-store scheme and discuss this in detail in Section 4. The system allows all machines conduct the backup in parallel, and each machine conducts the backup of one VM at a time, and thus only requires a write buffer for one VM.

  PDS chunks are stored in a separate append-store instance. In this way, each file block for non-PDS chunks is associated with one VM and does not contain any PDS chunks, to maintain our goal of fault isolation.

We have not adopted the sampled index [7] for popular data chunks because sampling requires the use of prefetching to be effective. For the data sets we have tested, the spatial locality is limited among popular data chunks and on average the number of consecutive data chunks is 7 among popular chunks, which is not sufficient.

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. Our previous study shows that the popularity of data chunks after inner VM deduplication follows a Zipf-like distribution[3] and its exponent α is ranged between 0.65 and 0.7. [23]. Table 1 lists parameters used in this analysis.

As summarized in Table 1, let c be the total number of data chunks. c_u be the total number of fingerprints in the global index after complete deduplication, and f_i be the frequency for the ith most popular fingerprint. By

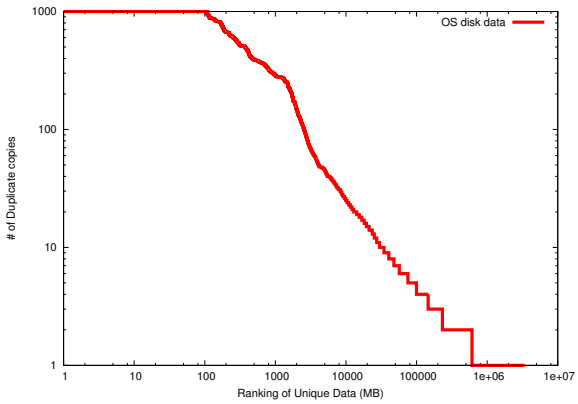


Figure 3: Duplicate frequency versus chunk ranking in a log scale after inner-VM deduplication.

Zipf-like distribution, $f_i = \frac{f_1}{i^\alpha}$. Since $\sum_{i=1}^{c_u} f_i = c(1-\delta)$,

$$f_1 \sum_{i=1}^{c_u} \frac{1}{i^\alpha} = c.$$

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

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

The k most popular fingerprints can cover the following number of chunks after inner VM 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 VC using top k popular chunks is the percentage of duplicates that can be detected:

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

We store the popular index using a distributed shared memory hashtable 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.

Ratio $\sigma^{1-\alpha}$ represents the percentage of the remaining chunks detected as duplicates in global cross-VM 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

k	the number of top most popular chunks selected for deduplication
c	the total amount of data chunks
c_u	the total amount of unique fingerprints after inner VM deduplication
f_i	the frequency for the ith most popular fingerprint
δ	the percentage of duplicates detected in inner VM deduplication
σ	$=\frac{k}{c_u}$. The percentage of unique data belong to PDS.
p	the number of machines in the cluster
V	the average number of VMs hosted on each machine
N_1	the average number of non-PDS file system blocks in a VM
N_2	the average number of PDS file system blocks in a VM
N_o	the average number of file system blocks in a VM for VO

Table 1: Modeling parameters and symbols.

test dataset. $\sigma = 2\%$ represents memory usage of approximately 100MB memory per machine for top k PDS. While the predicted value remains flat, the real coverage actually increases as the data size increases, which indicates that α increases with the data size. Thus in practice, the coverage of the PDS increases as more VMs are involved.

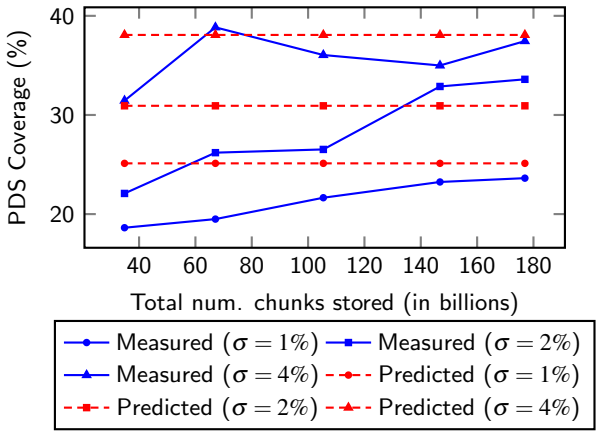
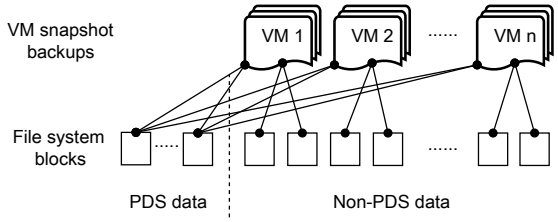
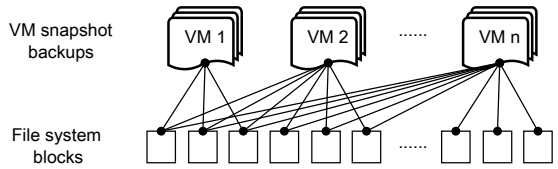


Figure 4: Predicted vs. actual PDS coverage as data size increases.



(a) Sharing of data under VM-centric dedup model



(b) Sharing of data under VM-oblivious dedup model

Figure 5: Bipartite association of VMs and file blocks under VO and VC approaches

3.3 Storage Space and Impact on Fault Tolerance

The replication degree of the backup storage is r for regular file blocks and $r = 3$ is a typical setting in distributed file systems [6, 16]. In the VC approach, a special replica degree r_c used for PDS blocks where $r_c > r$. The storage cost for VO with full deduplication is $c_u * r$ and for VC, it is $k * r_c + (c_u - k) * r$. Thus The storage cost ratio of VC over VO is

$$\sigma \frac{r_c}{r} + 1 - \sigma$$

Since *sigma* is small, term $\sigma \frac{r_c}{r}$ is small in the above expression. Thus the impact on storage increase is very small even when we choose a large $\frac{r_c}{r}$ ratio. For example, $\frac{r_c}{r} = 2$.

Next we assess the impact of losing d machines to the the VC and VO approaches. A large $\frac{r_c}{r}$ ratio can have a positive impact on full availability of VM snapshot blocks.

We use filesystem blocks rather than a deduplication data chunk as our unit of failure because the DFS keeps filesystem blocks as its base unit of storage. To compute the full availability of all snapshots of a VM, we derive the probablity of losing a snapshot file block 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 a file block to a VM if this block is used by this VM. For VC, each VM has an average number of N_1 file system blocks for non-PDS data. It also refers an average of N_2 file system blocks for PDS data. For VO, each VM has an average

of N_o file system blocks and let V_o be the average number of VMs shared by each file system block.

In VC, each non-PDS file system block is associated with one VM and PDS file system blocks are shared by an average of V_c VMs. Let s be the average number of chunks per file block. Thus,

$$V * p * N_1 * s = C_u(1-\sigma) \text{ and } V * p * N_2 * s = C_u \sigma * V_c$$

For the VO approach,

$$V * p * N_o * s = C_u V_o.$$

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

$$\frac{N_1}{N_o} = \frac{1-\sigma}{V_o} < 1.$$

Thus when there is a failure in file blocks with replication degree r and there is no failure for PDS data with more replicas, a VM in the VC approach has a lower chance to lose a snapshot than the VO approach because $N_1 < N_o$. Figure 6 shows the number of VMs sharing each file block, which shows how many VMs the failure of a single block can affect. We only show links to PDS blocks for VC because Non-PDS blocks in VC are always pointed to by exactly one VM. We can also observe that $N_1 + N_2 < N_o$ in Figure 7. This is likely because the PDS FSBs tightly pack data used by many VMs, which decreases the overall number of FSBs required to backup a VM. If the backup for multiple VMs is conducted concurrently, there would be more VMs shared by each file block on average. Therefore, even when there is a loss of a PDS block, the VC approach tolerates the fault better.

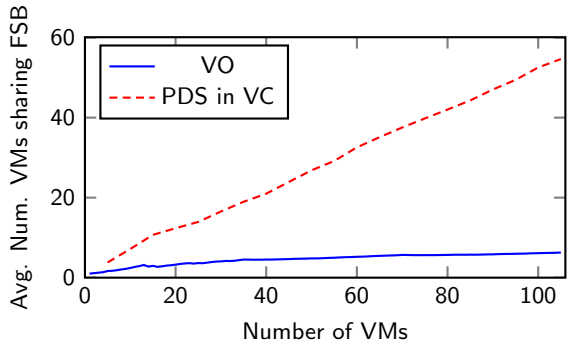


Figure 6: Measured Average number of VMs sharing a 64MB FSB with global dedup (VO), and in a 2% PDS for VC.

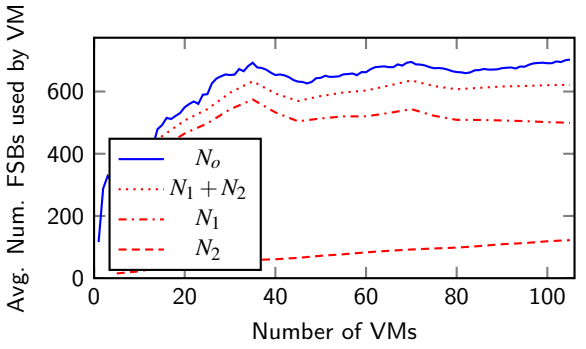


Figure 7: Measured Average number of 64MB FSBs used by a single VM. For VC both the number of PDS and Non-PDS FSBs used are shown.

The full snapshot availability of a VM is estimated as follows with parameters N_1, N_2, and N_o. With replication degree r, the availability of a file block is the probability that all of its replicas do not appear in any group of d failed machines. Namely, $1 - \binom{d}{r}/\binom{p}{r}$. The snapshot availability of a VM in the VO approach is

$$(1 - \frac{\binom{d}{r}}{\binom{p}{r}})^{N_o}. \quad (3)$$

For VC, there are two cases.

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

  $$(1 - \frac{\binom{d}{r}}{\binom{p}{r}})^{N_1}.$$

  Since $N_1 < N_o$, the VC approach has a higher availability of VM snapshots than VO.

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

  $$(1 - \frac{\binom{d}{r}}{\binom{p}{r}})^{N_1} * (1 - \frac{\binom{d}{r_c}}{\binom{p}{r_c}})^{N_2}.$$

  Figure 8 illustrates that the availability of an individual file system block with different replication degree. This demonstrates inequality $1 - \frac{\binom{d}{r}}{\binom{p}{r}} < 1 - \frac{\binom{d}{r_c}}{\binom{p}{r_c}}$. As we have also observed $N_1 + N_2 < N_o$ in our dataset, these two conditions support that VC has a higher availability.

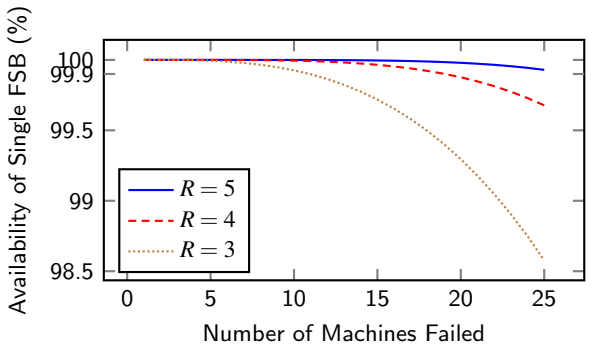


Figure 8: Availability of individual file system blocks in a 100 machine cluster with different replication degrees.

4 Architecture and Implementation Details

Our system runs on a cluster of Linux machines with Xen-based VMs. A distributed file system (DFS) manages the physical disk storage and we use an open source DFS 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[20] in Xen).

4.1 Components of a cluster node

As depicted in Figure 9, 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 index access.

We use the virtual device driver in Xen that employs a bitmap to track the changes that have been made to 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 [9], 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. [What happens with modification during snapshot backup stage.]

The representation of each snapshot has a two-level index data structure. The snapshot meta data (called snapshot 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 includes reference pointers to the actual data location to eliminate the need for additional indirection.

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 Fig.10, we explain the data structure of snapshot stores as follows.

- Data of each VM snapshot store excluding PDS is 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 copy used data chunks to another container.

  Each container is further divided into a set of chunk 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 store 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 [7, 15], 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

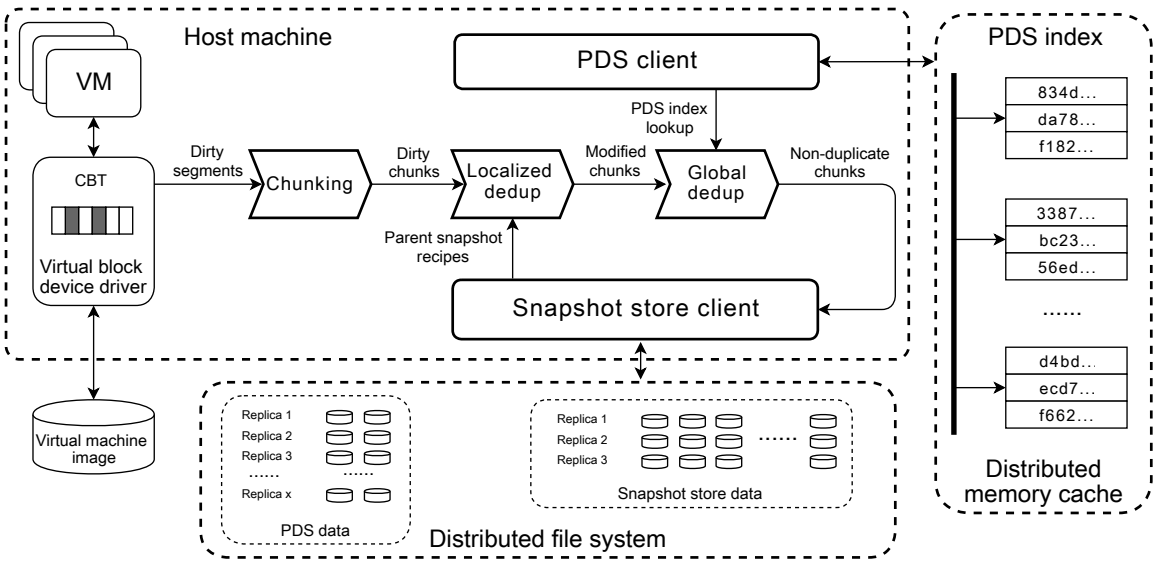


Figure 9: System Architecture

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 recipe for the snapshot will point 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 a set of commonly used data and they are stored in one PDS file. Since the total file size is relatively small, and PDS data is recalculated periodically, the PDS data file and its index are rebuilt completely. Each reference to a PDS data chunk in the PDS index is the offset within the PDS file, and the chunk size.

The snapshot store supports three API calls.

- *Put(data).* This places data 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. The writing of PDS data chunks is conducted periodically when there is a new PDS calculation.

- *Get(reference).* The fetch operation for the PDS data chunk is straightforward since each reference contains the offset and size within the PDS underlying file. 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(reference).* 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. 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 upper level 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.

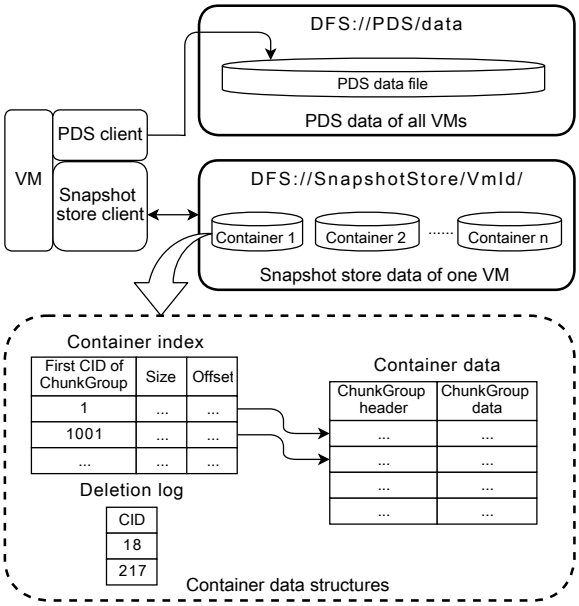


Figure 10: Data structure of a VM snapshot store.

4.3 VM-centric Approximate Snapshot Deletion with Leak Repair

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 [7], it still takes significant resources to conduct this process every time

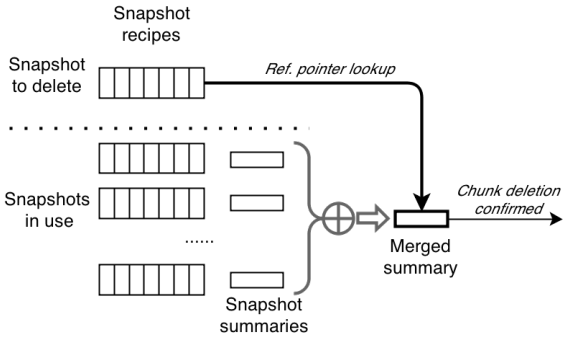


Figure 11: Approximate deletion

there is a snapshot deletion. In the case of Alibaba, snapshot backup is conducted automatically and there are about 10 snapshot stored for every user. When there is a new snapshot created every day, there will be a snapshot expired everyday to maintain a balanced storage use.

We seek a fast solution with low resource usage to delete snapshots. Our VM-centric snapshot storage design simplifies the deletion process since we can focus on unreferenced chunks within each VM. The PDS data chunks are commonly shared among all VMs and we do not consider them during snapshot deletion. The selection of PDS data chunks is updated periodically independent of snapshot deletion process. 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 summary of reference pointers 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 pointers of deleted snapshot with the merged reference Bloom-filter summary of other live snapshots. The merging of live snapshot Bloom-filter bits uses the logical 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 pointer is not found in the

merged summary vector, we are sure that this chunk is not used by any live snapshots, thus it's safe delete it. However, among all the chunks to be deleted, there is a small percentage of unused chunks which are misjudged as being in use, resulting in a 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 through the VM snapshot metadata recipe. That requires a scanning of all chunks in a VM; however it is a VM-specific procedure and thus the cost is relatively small compared to the traditional mark-sweep procedure [7] which scans snapshot chunks from all VMs. For example, consider each reference pointer consumes 8 bytes plus 1 mark bit, a VM that has 40GB backup data with about 10 million chunks will need less than 90MB 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, given a VM which keeps h snapshots in the backup storage, and it creates and deletes one snapshot everyday. Let u be the total number of chunks brought by the initial backup, 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. The probablity that a particular bit is 0 in all h summary vectors is $(1-\frac{1}{z})^{jU}$. 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. Then the misjudgment rate of being in use ε is:

$$\varepsilon = (1-(1-\frac{1}{z})^{jU})^j. \quad (4)$$

For each snapshot deletion, the number of chunks to be deleted is nearly identical to the number of newly add 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 need to execute a leak repair:

$$\frac{L}{U} = \frac{R\Delta u\varepsilon}{u+(h-1)\Delta u} > \tau \Longrightarrow R > \frac{\tau}{\varepsilon} \times \frac{u+(h-1)\Delta u}{\Delta u} \quad (5)$$

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, $u \approx 10$ millions. Then $U = 10.45$ millions. We choose $\varepsilon = 0.01$ and $\tau = 0.1$. From Equation 4, $z = 10U = 100.45$ million bits. From Equation 5, 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 VM scheduled for every 12 days. Each repairs 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, 23] and the underlying DFS uses QFS with default replication degree 3 while the PDS replication degree is 5. Our evaluation objective is to study the benefit in fault tolerance and deduplication efficiency of the VC approach 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 our VC approach with a VO approach using stateless routing with binning (SRB) based on [5, 2] SRB executes a distributed deduplication by routing a data chunk to a machine [5] 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 of this machine locally.

5.1 Settings

We have performed a trace-driven study using a production dataset [23] from Alibaba Aliyun's cloud platform with about 100 machines. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, and win2008. Each machine hosts upto 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 backup of VM snapshots is completed within a 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. All data are divided into 2 MB fix-sized segments and each segment is divided into

variable-sized content chunks [11, 13] with an average size of 4KB. The signature for variable-sized blocks is computed using their SHA-1 hash. Popularity of data blocks are collected through global counting and and top popular ones ($\delta = 1-2\%$) are kept in the PDS, as discussed in Section 4.2.

5.2 Impact on Fault Tolerance

Figure 12 shows the availability of VM snapshots when there are up to 10 storage nodes failed in a 100-node cluster. Two VC curves use $V_c = 1250$ and $V_c = 2500$ representing the average number of VMs that share a PDS file system block. $V_c = 2500$ represents the absolute worst case for 2500 VMs. The VO curve has $V_o = 150$ which is the average number of VMs that share a file system block. Our results show that the VC approach even with $V_c = 2500$ has a higher availability than VO. For example, with 10 machines failed, VO with 98.5% availability would lose snapshots of 37 VMs while VC with 99.9% loses snapshots of 2.5 VMs. The key reason is that $N_1 + N_2 < N_o$, caused by the fact that the VM-centric approach localizes deduplication and packs data blocks for one VM as much as possible. The extra replication for PDS blocks also significantly increases the snapshot availability even when a PDS file block is shared by every VM.

Figure 13 shows the snapshot availability when increasing the replication degree for PDS blocks. It can be seen that increasing the replication of the PDS blocks can have a good impact on the overall availability of the VM backups. PDS with more replication makes up a small percent of the stored data; however, the benefit to the snapshot availability is significant.

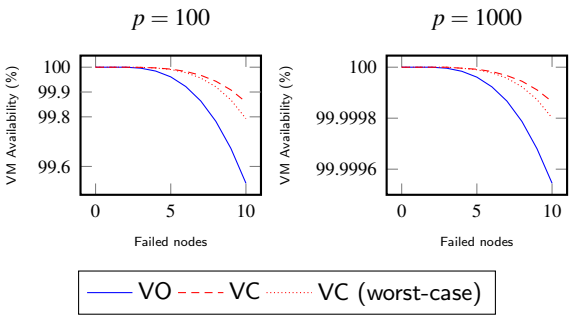


Figure 12: Availability of VM backup snapshots for VO and VC when failed machines vary from 1 to 10. Non-PDS replication is fixed at 3 and PDS replication is 4.

5.3 Deduplication Efficiency

Figure 14 shows the deduplication efficiency for SRB and VC. Deduplication efficiency is defined as the percent of duplicate chunks which are detected and deduplicated. The figure also compares several PDS sizes chosen for VC. "$\sigma = 2\%$" is defined in Table 1. With $\sigma = 2\%$, memory usage for PDS index lookup per machine is about 100MB and the deduplication efficiency can reach over 90%. When $\sigma = 4\%$, the deduplication efficiency can reach 92%. The loss of efficiency in VC is caused by the restriction of the total physical memory available in the cluster to support PDS index. The loss of efficiency in SRB is caused by the routing of data chunks which restricts the search scope for global comparison. In all cases, VC provides similar or better deduplication efficiency than SRB.

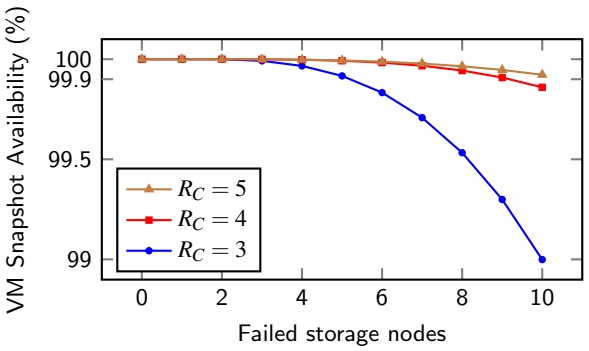


Figure 13: Availability of VM backup snapshots as nodes fail in VC with different PDS Replication degree (Non-PDS replication fixed at 3, and average PDS block links set to 1250)

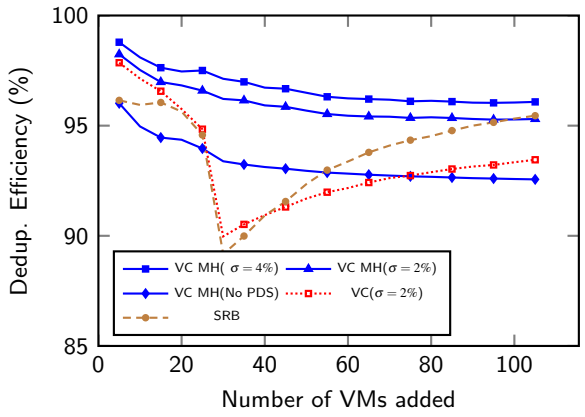


Figure 14: Deduplication efficiency of VC and SRB.

This result shows VC can remove a competitive amount of duplicates. In general, our experiments show that dirty-bit detection at the segment level can reduce the data size to about 23% of original data, which leads about 77% reduction. Local search within a segment can further reduce the data size to about 18.5% of original size, namely it delivers additional 4.5% reduction. The

PDS replication degree	Disk usage per machine
3	3065GB
4	3072GB
5	3079GB
6	3086GB
7	3093GB

Table 2: Storage space cost per machine for DFS under different PDS replication degree

Tasks	CPU	Mem (MB)	Write (MB/s)	Time (hrs)
1	19%	18.1	8.37	1.314
2	35%	31.8	8.97	1.226
4	63%	54.1	9.33	1.179
6	77%	71.9	9.46	1.162

Table 3: Resource usage of concurrent backup tasks in each node

PDS-based cross-VM detection with $\sigma = 2\%$ can reduce the size further to 8% of original size, namely it delivers additional 10.5% reduction.

5.4 Resource Usage and Processing Time

Storage cost of replication. Table 2 shows the total storage space required by VC as the PDS replication degree changes. The increase in storage cost is minimal because the PDS makes up a small percent of the total storage space, while increasing replication degree has a more significant benefit for availability as shown in Figure 13.

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 node. Table 3 gives the CPU, memory and disk write workload of backing up multiple VMs at the same time on each physical machine. We see our hybrid deduplication scheme only occupies a small amount of system resources. The local deduplication only needs to keep the parent snapshot recipe and a few similar segment recipes in memory during duplicate detection.

Base on the observed performance numbers, even with a single backup task per node and enforcing the disk read speed to 50 MB/s, we can still process raw data at near 175 MB/s. If we consider the extreme case in which each machine has 25 VMs at 40GB size, our snapshot system can finish backing up all the VMs (1TB total) in 1.58 hours.

Processing Time breakdown. Figure 15 shows the average processing time of a VM segment under VC and SRB. 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. SRB has a higher index access and fingerprint comparison because once a chunk is routed to a machine, it relies on this local machine to access its index (often on the disk, supported by a bloomer 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. SRB spends slighter more time in reading data and updates because it also updates the on-disk local meta data index. Overall, VC is faster than SRB, though data reading dominates the processing time for both algorithms.

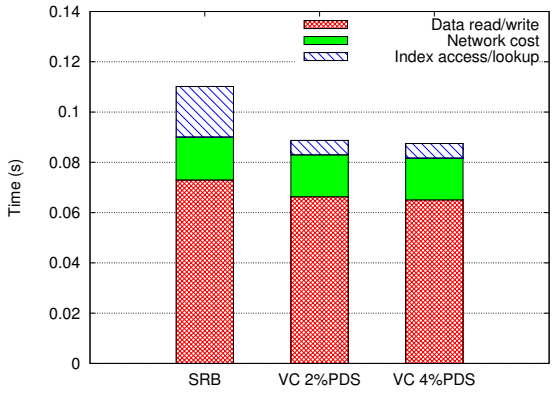
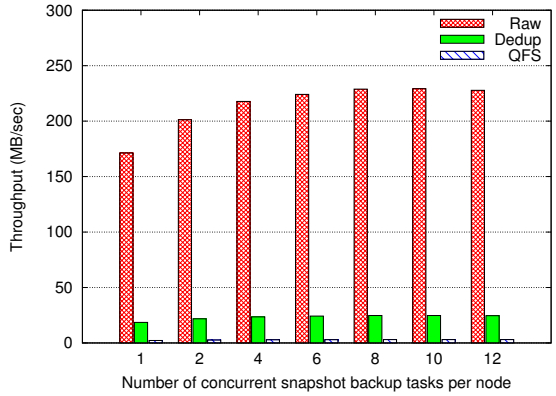


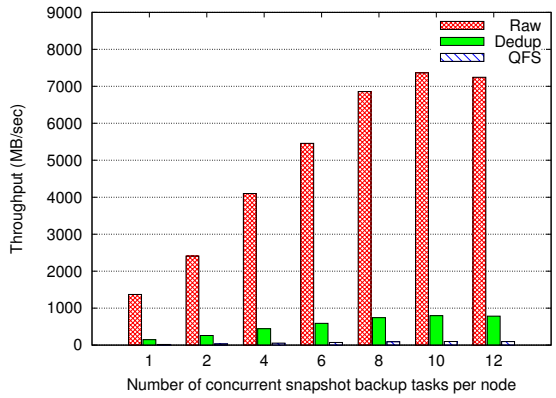
Figure 15: Time to backup a dirty VM segment under SRB and VC appraoches

Figure 15 also reports the average backup time for a VM in VC when varying the PDS size. It lists the time distribution for data reading, similarity-guided local search, cross-VM PDS lookup, and non-duplicate data writing. While data reading dominates the backup process, PDS lookup spends about a similar amount of time as local search. The change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time.

Throughput. Figure 16 shows the backup throughput per each machine when all machine nodes handle several VMs in parallel. To begin, on each node we write snapshots for 4 VMs concurrently, and gradually increase number of VMs to 12 to saturate our system capability. We observed the per-node throughput peaked at 2700 MB/s when writing 10 VM snapshots in parallel, which is far beyond our QFS file system capability. The reason behind it is our efficient deduplication architecture and compression which greatly reduce the amount of data that needs to be written to the file system. The main bottleneck here is that our QFS installation only manages one disk per node, which prevents it from fully utilizing the the benefits of parallel disk access. We expect our architecture can perform even better in production clusters, which often have ten or more disks on each node.



(a) Backup throughput per node under controlled I/O bandwidth usage



(b) Deduplication and storage system performance

Figure 16: Throughput per-node with concurrent snapshot backup tasks

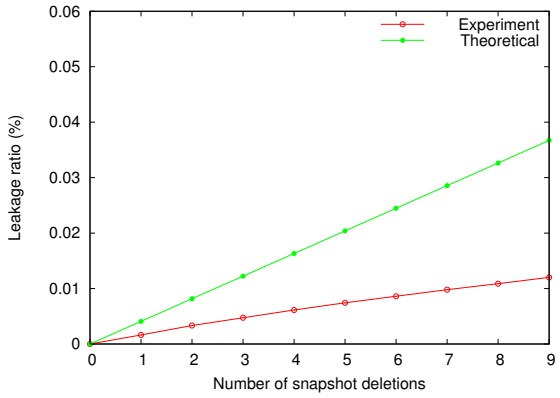


Figure 17: Accumulated storage leakage by approximate snapshot deletions

5.5 Effectiveness of Approximate Deletion

Figure 17 shows the average accumulated storage leakage in terms of percentage of storage space per VM caused by approximate deletions. The slot line is the predicted leakage using Formula **??** 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. Inner-VM deduplication localizes backup data dependency and exposes more parallelism while cross-VM deduplication with a small common data set. VM-specific file block packing also reduces inter-VM dependence and enhances fault tolerance. The design places a special consideration for low-resource usages as a collocated cloud service. Evaluation using VM backup data shows that our solution can accomplish 92% 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 dedu-

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] 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.

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

[16] 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.

[17] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *FAST'12*, 2012.

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.