VM-Centric Snapshot Deduplication for Converged Cloud Architectures

Wei Zhang*, Daniel Agun*, Tao Yang*, Rich Wolski*, and Hong Tang†

*University of California at Santa Barbara †Alibaba Inc.

Abstract

Data deduplication is important for snapshot backup of virtual machines (VMs) because of excessive redundant content. Fingerprint search for source-side duplicate detection is resource intensive when the backup service for VMs is co-located with other cloud services. This paper proposes a low-profile VM-centric backup service with a tradeoff for a competitive deduplication efficiency while using small computing resources, suitable for running on a converged cloud architecture that cohosts many other services. The key techniques in the proposed scheme are to localize deduplication as much as possible within each VM guided by similarity search, restrict global deduplication under popular chunks with extra replication support, and use an approximate method for fast and simplified snapshot deletion. This VM-centric design also improves the snapshot availability during machine failures by separating popular chunks and associating file system blocks with one VM for non-popular chunks. This paper describes an evaluation of this VM-centric scheme to assess its deduplication efficiency, resource usage, and fault tolerance.

Categories and Subject Descriptors [*Information Systems*]: Information Storage Systems; [*Distributed Architectures*]: Cloud Computing

General Terms Design, Experimentation, Performance

Keywords Virtual machine images, snapshot backup and deletion, source-side deduplication, cluster-based architectures

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Submission to SoCC '14, October, 2014, Seattle, WA, USA.

1. Introduction

Commercial "Infrastructure as a Service" clouds (i.e. *public clouds*) often make use of commodity data center components to achieve the best possible economies of scale. In particular, large-scale e-commerce cloud providers such as Google and Alibaba deploy "converged" components that co-locate computing and storage in each hardware module (as opposed to having separate computing and storage "tiers.") The advantage of such an approach is that all infrastructure components are used to support paying customers – there are no resources specifically dedicated to cloud services. In particular, these providers use software to aggregate multiple direct attached low-cost disks together across servers as a way of avoiding the relatively high cost of network attached storage [9, 15, 20]. In such an environment, each physical machine runs a number of virtual machines (VMs) and their virtual disks are stored as disk image files. Frequent snapshot backup of virtual machine images can increase the service reliability by allowing VMs to restart from their latest snapshot in the event of a server failure. Snapshots contain highly redundant data chunks and deduplication of redundant content [17, 27] is necessary to substantially reduce the storage demand.

Source-side deduplication (e.g. [24]) eliminates duplicates before backup data is transmitted to a secondary storage, which saves network bandwidth significantly; however its resource usage can impact other co-located cloud services. It is memory-intensive to compare a large number of fingerprints and identify duplicates, even optimization or approximation techniques are developed [2, 7, 10]. When deleting unused snapshots, a grouped mark-and-sweep approach [10] is effective while it still carries a significant cost, especially in a distributed setting. Another side effect of deduplication is the possible loss of failure resilience [3]. If a shared block is lost, all files that share that block are affected. A cloud may offload backup workload from production server hosts to dedicated backup proxy servers (e.g. EMC Data Domain) or backup services (e.g. Amazon S3). This approach simplifies the cluster architecture and avoids potential performance degradation to production applications when backups are in progress. However, sending out raw and undeduplicated backup data wastes a huge amount of network bandwidth that would otherwise be available to user VMs.

This paper considers a backup service for converged cloud architectures that is co-located with user VMs, sharing the cloud compute, network, and storage resource with them. The main contribution of this paper is the development and comparative study of a VM-centric approach that simplifies deduplication and snapshot deletion and improves snapshot availability. The key advantage of this approach is that while maintaining a competitive deduplication efficiency, the scheme requires little resource usage to minimize the impact on other collocated cloud services. We term our approach *VM-centric* because the deduplication algorithm considers VM boundaries in making its decisions as opposed to treating all blocks as being equivalent within the storage system. Our work focuses on a tradeoff that allows the sharing of only "popular" data blocks across virtual machines while using localized deduplication within each VM to achieve both storage savings and fault isolation.

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 presents our scheme for snapshot deduplication and deletion. Section 4 describes a system implementation that evaluates the proposed techniques. Section 5 is our experimental evaluation that compares with other approaches. Section 6 concludes this paper. The appendix describes an analysis on the snapshot availability of our scheme during storage failures.

2. Background and Design Considerations

Figure 1 illustrates a converged IaaS cloud architecture where each commodity server hosts a number of virtual machines and storage of these servers is clustered using a distributed file system [9, 20]. Each physical machine hosts multiple virtual machines. Every virtual machine runs its own guest operating system and accesses virtual hard disks stored as image files maintained by the operating system running on the physical host. 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. In a converged setting with source-side deduplication, the resources that are used to implement snapshot backup and deduplication are the same resources that must support cloud-hosted VMs. Thus the backup service collocated with other cloud services has to minimize its resource impact.

File backup systems have been developed to use fingerprints generated for data "chunks" to identify duplicate content [17, 19]. In a simple implementation, it is expensive to compare a large number of fingerprints so a number of techniques have been proposed to improve duplicate identification. For example, the data domain method [27] uses

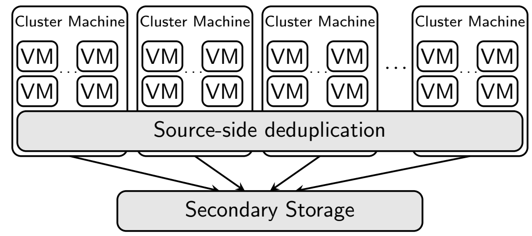


Figure 1. VM snapshot backup running on a converged cloud cluster.

an in-memory Bloom filter to identify non-duplicates and a prefetching cache for potential duplicates that might hit in the future. Additional inline deduplication and approximation techniques are studied in [2, 13, 21, 25].

Even with these optimization and approximation techniques, resource usage such as memory for deduplication is still extensive for a shared cloud cluster. For example, in the experiments discussed in Section 5.1, the raw snapshot data has a size of 1,000TB on 100 physical machines that host VMs, cross-machine fingerprint comparison using Bloom filter and approximated routing [7, 27] still needs several gigabytes of memory per machine. This can impact other primary cloud services sharing the same computing resource. Our objective is to have an approximation scheme which uses no more than a few hundred megabytes of memory during normal operation. Thus the desired ratio of raw data to memory ratio is from 100K:1 to 30K:1. Our proposed scheme achieves 85K:1 and this ratio is about the same as the number of machines increases.

Index sampling with a prefetch cache is proposed in [10] for efficient single-machine deduplication. This scheme uses 25GB memory per 500TB of raw data and thus the raw data to memory ratio is 20K:1. The scheme proposed in this paper uses three times less memory. We have not incorporated this index sampling technique because it is difficult to extend 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 is significant.

While deduplication reduces storage cost, it creates an artificial data dependence among VMs: when a shared data chunk is lost in the storage, all VMs that refer such chunks after deduplication face a data loss. The work in [3] identifies such an issue in file systems and uses more replication for data shared by more files, and we will adopt this idea. When the backup storage is implemented using a distributed file system such as Hadoop and Google file system [9], the file system block size is often chosen to be large (e.g. 64MB). On the other hand, deduplication implementations [2, 7, 10, 11, 27] typically use smaller chunk sizes (e.g. 4K bytes). Thus we need to address the size gap between file system blocks and chunks.

Snapshot deletion occurs frequently because snapshots expire regularly and reference counting is required to identify chunks that are no loner used. Grouped mark-and-sweep [10] is effective for deletion by tracking the dependencies from data containers to metadata groups, but still it carries a significant cost in a cloud cluster. This is because any data chunk can still be used by a large number of VMs, and there are multiple rounds of extensive I/O involved to scan through the meta data of snapshots to check if a chunk is referenced or not. As the number of machines increases, the I/O cost becomes more expensive. Perfect hashing [4] provides a memory efficient way to represent a large set of hashes in a compact form, however, it still requires the scanning of all data hashes before any garbage collection could begin, which limits its application to an offline batch process. In addition, it requires a centralized location to store the hashing result which is still too big to be held in a converged architecture.

Our previous work on parallel deduplication [25, 26] is not a comprehensive VM-centric solution and does not address snapshot deletion and file system block packaging for fault tolerance. The work in [26] a low-cost solution, but requires all backups for each day to occur at the same time and is less flexible than the approach presented here.

3. VM-centric Snapshot Deduplication

With the considerations discussed in the previous section, we propose a VM-centric approach (called VC) for a co-located backup service that has a resource usage profile suitable for use with converged cloud architectures. This compares the traditional deduplication approach *VM-oblivious* (VO) which manages duplicate data chunks without consideration of VMs. Section 3.1 discusses our key ideas and design for duplicate detection. Section 3.2 presents a simplified snapshot deletion with the VM-centric approach.

3.1 VM centric strategies

We describe our overall deduplication approach as consisting of three complementary strategies: VM-local duplicate search, global deduplication using popular chunks, and VM-centric file system block management.

- **Cross-VM global deduplication using popular chunks** – We separate the deduplication within a VM and cross VMs and simplify the cross-VM deduplication while maximizing the inter-VM deduplication efficiency as much as possible. This is because global deduplication that detects the appearance of a chunk in any VM requires a substantial resource for fingerprint comparison. To simplify cross-VM deduplication, we restrict the search scope of deduplication within the top k most popular items. This popular data set is called the **PDS**.

  What motivates us to use PDS is that for those chunks that appear in different VMs, the top k most popular items dominate the appearance distribution [25]. Figure 2

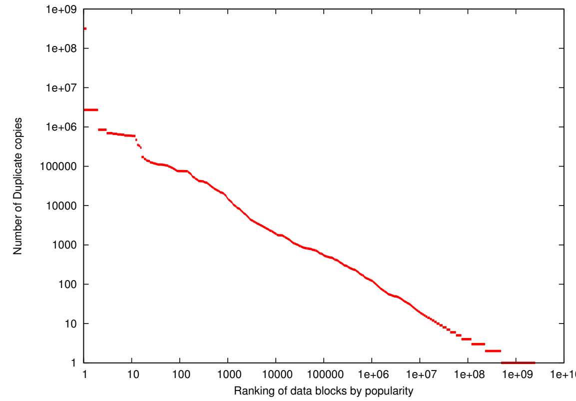


Figure 2. Duplicate frequency versus chunk ranking in a log scale after local deduplication.

  shows the distribution of chunk popularity for a data trace from Alibaba with 2500 VMs discussed in Section 5. 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. The distribution from this figure is Zipf-like. Let σ denote as the percentage of unique data chunks belonging to PDS and from the evaluation in Section 5, we find that σ with about 2% can deliver a fairly competitive deduplication efficiency.

  PDS can be computed in an offline manner periodically, e.g., on a monthly basis. Compared to [27], the fingerprint index size is reduced by $100 - \sigma$ percentage (e.g. 98% with $\sigma = 2\%$) and the searchable fingerprint space becomes very small under the popularity constraint. The fingerprint-guided distributed mapping in [2, 7] narrows the search scope of each data chunk, but it does not reduce th total amount of searchable fingerprints used for the entire deduplication.

- **VM-specific duplicate search optimization** – While cross-VM deduplication is simplified, we intend to optimize the VM-specific deduplication under a reasonable memory consumption to make up the loss of deduplication opportunities due to the cross-VM popularity constraint. We start with the standard version-based detection [6, 22] to identify changed content with dirty bits in a coarse grain segment level. The reason to choose a coarse grain segment level is that 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.

  Since the previous work typically uses a non-uniform chunk size with an average of 4KB or 8KB for the best deduplication effectiveness [2, 7, 10, 27], we conduct additional local similarity guided deduplication by compar-

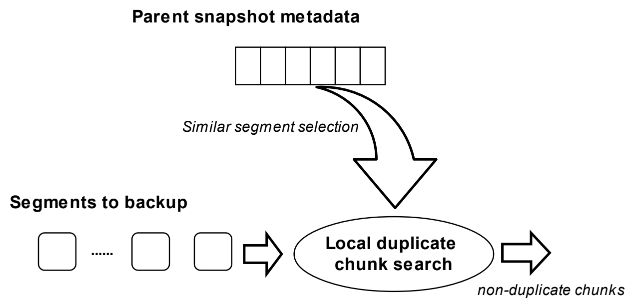


Figure 3. Similarity-guided local duplicate detection

ing chunk fingerprints of a dirty segment in a snapshot with those in *potentially similar* segments from its parent snapshot. We consider a parent segment is potentially similar to the current dirty segment if 1) the parent segment is at the same offset as the current segment. 2) the signature of these segments is the same. The signature of a segment is defined as the minimum value of all its chunk fingerprints computed during backup and is recorded in the snapshot metadata (called recipe). This signature definition is motivated by the previous work for similarity-based routing in parallel deduplication [2, 7] and near duplicate detection [5].

When processing a dirty segment, its similar segments can be found easily from the parent snapshot metadata. Then metadata of the similar segments is loaded to memory, containing chunk fingerprints to be compared. To control the time cost of search, we set a limit on the number of similar segment recipes to be fetched. For example, assume that a segment is of size 2MB, 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 190K, which is insignificant.

- **VM-centric file system block management** – When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written into a file system block. In addition to the fact that a distributed file system block [9, 20] is often configured to be much larger than a chunk, a number of chunks can be combined together and compressed further using a standard compression method such as LZO. Thus a backend file system block contains a large number of compressed chunks. We set the following two constraints in composing chunks for a file system block: 1) Each file system block is either dedicated to non-PDS chunks, or to PDS-chunks. 2) A non-PDS file system block is only associated with one VM.

  Restricting the association with one VM improves fault isolation when some file blocks are lost during storage failures. In addition, storing PDS chunks separately from non-PDS chunks allows special replication handling for those popular shared data. If we do not separate the popular chunks from the less-popular, the popular chunks are dispersed across all of the filesystem blocks in the storage system and we would have to add extra replications for *all* file blocks in order to follow the popularity-driven replication idea from [3]. That reduces the storage efficiency.

The underlying distributed file system has a fixed replication degree for all file system blocks [9, 20]. We add extra replicas for file blocks containing PDS chunks and this brings additional failure protection for these chunks shared by many VMs. Appendix A provides a detailed analysis of snapshot availability under this VC scheme and compares it with VO.

3.2 VM-Centric Snapshot Deletion with Periodic Leak Repair

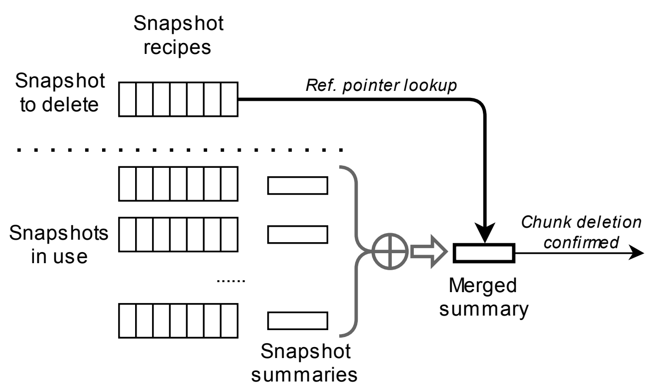


Figure 4. Fast approximate deletion

Snapshot deletions can occur frequently since old snapshots become less useful. Both space accounting and reclamation are difficult because deduplication greatly complicates the identification of unused data when deletion occurs. To filter out un-referenced data from shared duplicates, it would require to either maintain expensive live schemas or conduct global reference counting [10]. For example, the experiment in Section 5.3 shows that the grouped mark-and-sweep process takes 43 hours for a 2500 VM cluster using upto 3GB memory per machine.

One of the key benefits of our VM-centric design is that it allows for faster and simpler deletion. We propose the following strategies to speedup the deletion process and reduce its resource usage.

- First, reference counting is focused on non-PDS chunks which are not shared among VMs, and the mark-and-sweep process is applied within each VM. Therefore, the deletion can be conducted independently among VMs.
- Second, we propose an *approximate* deletion strategy to trade deletion accuracy for speed and resource usage. Since the VM size is highly skewed in practice, a large VM may still require a substantial amount of

memory to facilitate the mark-and-sweep process of data chunks used within all snapshots of this VM. Our approximate deletion method sacrifices a small percent of storage leakage to efficiently identify unused chunks using a per-snapshot bloom-filter.

The simplified deletion algorithm contains three aspects.

- **Computation for snapshot reference summary.** Every time there is a new snapshot created, we compute a Bloom-filter with z bits as the reference summary vector for all non-PDS chunks used in this snapshot. The items we put into the summary vector are all the references appearing in the metadata of the snapshot. For each VM we preset the vector size according to estimated VM image size; given h snapshots stored for a VM, there are h summary vectors maintained. We adjust the summary vector size and recompute the vectors if the VM size changes substantially over time. This can be done during periodic leakage repair described below.
- **Fast approximate deletion with summary comparison.** When there is a snapshot deletion, we identify if chunks to be deleted from that snapshot are still referenced by other snapshots. This is done approximately and quickly by comparing the reference of deleted chunks with the merged reference summary vectors of other live snapshots. The merging of live snapshot Bloom-filter vectors uses the bit-wise 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, this chunk is not used by any live snapshots. Thus it 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.

  One advantage of the above fast method is that it can finish and free storage usage immediately, while other offline methods (e.g. [4, 10]) can't. That is important for storage accounting as users pay for used storage and delayed deletion affects the accounting.
- **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. A mark-and-sweep process requires a scan of all the metadata for a snapshot store. Since it is a VM-specific procedure, the space cost is proportional to the number of chunks within each VM. This is much less expensive than the VM-oblivious mark-and-sweep which scans snapshot chunks from all VMs, even with optimization [10].

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

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

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:

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

When leakage ratio L/U exceeds a pre-defined threshold τ, we trigger a leak repair. Namely,

$$\frac{L}{U} = \frac{R\Delta u\varepsilon}{u+(h-1)\Delta u} > \tau \Longrightarrow R > \frac{\tau}{\varepsilon} \times \frac{u+(h-1)\Delta u}{\Delta u}. \tag{2}$$

For example for our test data in Section 5, $h = 10$ and each snapshot adds about 0.1-5% of new data. Thus $\Delta u/u \approx 0.025$. For a 40GB snapshot, $u \approx 10$ million. Then $U = 12.25$ million. We choose $\varepsilon = 0.01$ and $\tau = 0.05$. From Equation 1, each summary vector requires $z = 10U = 122.5$ million bits or 15MB. From Equation 2, leak repair should be triggered once for every R=245 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 9.8 days. If $\tau = 0.1$ then leakage repair would occur every 19.6 days.

4. Prototype Implementation

Our prototype system runs on a cluster of Linux machines with Xen-based VMs and the QFS [16] distributed file system. All data needed for the backup service including snapshot data and metadata resides 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[23] in Xen).

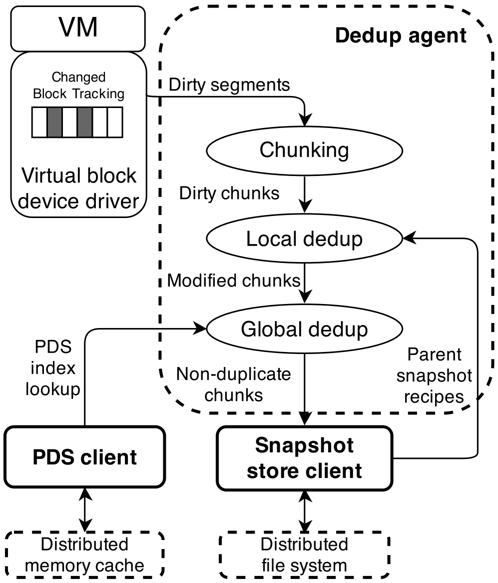


Figure 5. Data flow during snapshot backup

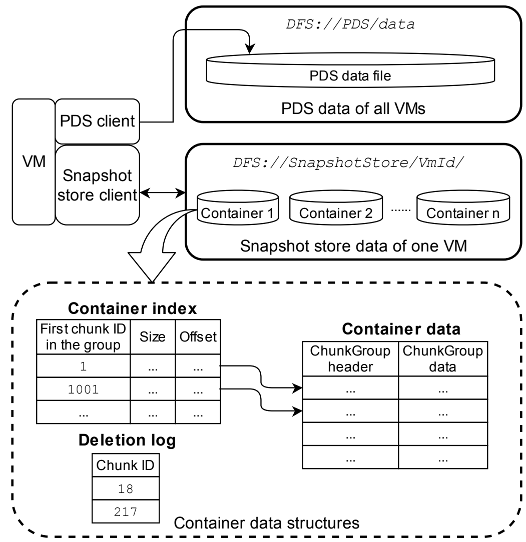


Figure 6. VM snapshot store data structures

4.1 Per Node Software Components

As depicted in Figure 5, 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 agent, 3) a snapshot 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) 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 [12], 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 recorded.

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 include references to the actual data location to eliminate the need for additional indirection.

4.2 A VM-centric Snapshot Store

We use the QFS distributed file system to hold snapshot backups. 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. As shown in Figure 6, we explain the data structure of the snapshot stores as follows. There is an independent store containing all PDS chunks shared among different VMs as a single file. Each reference to a PDS data chunk in the PDS index is the offset within the PDS file. Additional compression is not applied because for the data sets we have tested, we only observed limited spatial locality among popular data chunks. On average the number of consecutive PDS index hits is lower than 7, thus it is not very effective to group a large number of chunks as a compression and data fetch unit. For the same reason, we decide not to take the sampled index approach [10] for detecting duplicates from PDS as limited spatial locality is not sufficient to enable effective prefetching for sampled indexing.

PDS data are re-calculated periodically in an offline manner, but the total data size is small. When a new PDS data set is computed, the 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 are 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).

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

Each non-PDS 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 stores the entire group as a unit after compression. This compression can reduce data by several times in our tested data. 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 [10, 19], retrieval of a data chunk group naturally works well with prefetching. A typical chunk group contains 1000 chunks in our experiment.

Each non-PDS data container is represented by three files: 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 non-PDS 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 chunk is to be accessed, the segment recipe contains a reference pointing to a data chunk, which is used to lookup up the chunk data as described shortly.

Three API calls are supported for data backup:

- **Append()**. For PDS data, the chunk is appended to the end of the PDS file and the offset is returned as the reference. This operation may only be used during PDS recalculation. For non-PDS data, this call places a chunk into the snapshot store and returns a reference to be stored in the recipe of a snapshot. The write requests to append chunks to a VM store are accumulated at the client side. When the number of write requests reaches a fixed group size, the 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()**. Fetching PDS data 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 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. 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 identifies 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 chunk ID 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.

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 the 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 deduplication efficiency and resource usage of VC, and assess its processing time and backup throughput, and the benefit in fault tolerance. We will compare VC with a VO approach using stateless routing with binning (SRB) based on [2, 7]. SRB executes distributed deduplication by routing data chunks to cluster machines [7] using a min-hash function discussed in [2]. Once a data chunk is routed to a machine, the chunk is compared with the fingerprint index within this machine locally based on [2].

We have performed a trace-driven study based on a production dataset from Alibaba Aliyun's cloud platform [1] with about 2500 VMs, running on 100 physical machines. Each machine in the production environment is more powerful than our evaluation cluster and has 16 cores with 48GB memory. 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.

Each VM has about 40GB of storage data on average including OS and user data disk while the size distribution is highly skewed and the maximum to the average ratio is close to 40. Each physical machine deals with about 10TB of snapshot data and each 1TB data represents one snapshot version of 25 VMs. Thus the total amount of raw snapshot shot in 100 physical machines is about 1,000TB. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, win2008 and win2003. The daily snapshot change rate is about 2-3% on average. The fingerprint for variable-sized chunks is computed using their SHA-1 hash [14, 18].

5.1 Deduplication Efficiency and Memory Usage

	VC			VO
	2%No PDS	2%PDS	2%PDS no-simi	SRB
Dedup Efficiency	93.02%	96.01%	94.31%	97.86%
Memory per-machine	10MB	118MB	108MB	2.4GB

Table 1. Deduplication efficiency and per-machine memory usage for different VC settings and SRB.

Table 1 shows the deduplication efficiency and per-machine memory usage for SRB and VC with different settings. Deduplication efficiency is defined as the percent of duplicate chunks removed, comparing to a perfect scheme which detects and removes all duplicates. Notice σ is the percentage of unique chunks selected in PDS. With $\sigma = 2\%$, Column 3 shows its deduplication efficiency can reach over 96%. The loss of efficiency in VC is caused by the restriction of the physical memory available in the cluster for fast in-memory PDS index lookup. Memory usage per machine is low because each machine only hosts $1/p$ of index for PDS plus some buffer space where p is the number of physical machines. SRB in Column 5 can deliver up to 97.86% deduplication efficiency, which is slightly better than VC. Thus this represents a tradeoff as VC uses much less resource and faster deletion. Memory usage per machine in SRB includes the Bloom filter space to access the on-disk index and cache for frequently or recently accessed chunk index.

Column 2 of Table 1 shows deduplication efficiency of VC without using PDS and it achieves 93.02% and missed duplicates take up-to 3.56% of the original space, which is about 356GB per physical machine. Column 4 of Table 1 shows deduplication efficiency of VC which only uses the same offset in the parent segment to look for duplicates but does not look for other potentially similar segments. We explain why similarity-guided local search provides such an improvement as follows. There are VMs in which data segments are moved to another location on disk, for example when a file is rewritten rather than modified in place, and a dirty-bit or offset-only based detection would not be able to detect such a movement. We have found that in approximately 1/3 of the VMs in our dataset this movement happens frequently. In general, adding local similarity-guided search increases deduplication efficiency from 94% to over 96%. That is one significant improvement compared to the work in [25] which uses the parent segment at the same offset to detect duplicates instead of similarity-guided search.

In summary, our experiments show that segment-level version detection reduces the data size to about 24.14% of original data, which is a 75.86% reduction. Namely 10TB snapshot data per machine is reduced to 2.414TB. Similarity-guided local search can further reduce the data size to about 1.205T, which is 12.05% of original. Thus it delivers a 50.08% reduction after the version-based deduplication. The popularity-guided global deduplication with $\sigma = 2\%$ reduces the data further to 860GB, namely 8.6% of the original size. So it provides additional 28.63% reduction. The overall memory usage of VC for each physical machine is very small, which is insignificant to other hosted cloud services. In comparison, the 2.4GB per-machine memory usage of SRB is very visible to other services.

Comparing the experiment results of the sampled index method reported in [10], our scheme achieves a 85K:1 ratio between raw snapshot data and memory usage with 96% deduplication efficiency while the sampled index method achieves 20K:1 (25GB memory per 500TB raw data) with 97% deduplication efficiency. Thus our scheme is more memory efficient with a good tradeoff for deduplication efficiency. If we change $\sigma = 4\%$, the deduplication efficiency of VC increases to 96.58% while memory usage per machine increases to about 220MB. Notice also that the sampled index method is designed for single-machine deduplication.

5.2 More on Resource Usage and Processing Speed

We show additional experimental results on resource usage, processing time and throughput of VC.

Storage cost of replication. When the replication degree of both PDS and non-PDS data is 3, the total storage for all VM snapshots in each physical machine takes about 3.065TB on average before compression and 0.75TB after compression. 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 usage with multi-VM processing and disk bandwidth with I/O throttling. We have further studied the memory and disk bandwidth usage when running concurrent VM snapshot backup on each machine with $\sigma = 2\%$. Table 2 gives the resource usage when running 1 or multiple VM backup tasks at the same time on each physical machine. Each task handles the backup of one VM snapshot including deduplication. "CPU" column is the percentage of a single core used. "Mem" column includes ~100MB memory usage

Tasks	CPU	Mem (MB)	Read (MB/s)	Write (MB/s)	Backup Time (hrs)
1	19%	118	50	16.4	1.31
2	35%	132	50	17.6	1.23
4	63%	154	50	18.3	1.18
6	77%	171.9	50	18.8	1.162

Table 2. Resource usage of concurrent backup tasks at each physical machine with $\sigma = 2\%$ and I/O throttling.

for PDS index and other space cost for executing deduplication tasks such as recipe metadata and cache. The "Read" column is controlled to 50MB/s local disk bandwidth usage with I/O throttling so that other cloud services are not significantly impacted. The peak raw storage read performance is about 300MB/s and we only use 16.7% with this collocation consideration. "Write" is the I/O write usage of QFS; note that each QFS write triggers disk writes in multiple machines due to data replication. 50MB/s dirty segment read speed triggers about 16.4MB/s disk write for non duplicates with one backup task.

Table 2 shows that when each machine conducts backup one VM at a time, the entire cluster backup completes in 1.31 hours. Since there are about 25 VMs per physical machine, we could execute more tasks in parallel at each machine. But adding more backup concurrency does not shorten the overall time significantly in this case because of the controlled disk read bandwidth usage. When I/O throttling is not used, we will show below that processing multiple VMs concurrently does improve the throughput of backup.

Processing time breakdown without I/O throttling. Table 3 shows the average time breakdown for processing a dirty VM segment in milliseconds under VC and SRB. VC uses $\sigma = 2\%$. The overall processing latency of SRB is about 23.9% slower than VC. For VC, the change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time. It has a breakdown of processing time. "Read/Write" includes snapshot reading and writing from disk, and updating of the metadata. "Network" includes the cost of transferring raw and meta data from one machine to another during snapshot read and write. "Index Lookup" is the disk, network and CPU time during fingerprint comparison. This includes PDS data lookup for VC and index lookup from disk in SRB. The network transfer time for VC and SRB is about the same, because the amount of raw data they transfer is comparable. SRB spends slightly more time for snapshot read/write because during each snapshot backup, SRB involves many small bins, while VC only involves few containers with a bigger size. Thus, there are more opportunities for I/O aggregation in VC to reduce seek time. SRB also has a higher cost for index access and fingerprint comparison because most chunk fingerprints are routed to remote machines for comparison while VC handles most chunk fingerprints locally.

Algorithm	Time Spent in Task (ms)		
	Read/Write	Network	Index Lookup
SRB	73	17.078	20.098
VC $\sigma = 2\%$	66.328	16.626	5.784

Table 3. Average time in milliseconds to backup a dirty 2MB VM segment under SRB and VC with I/O throttling.

Concurrent backup tasks per machine	Throughput without I/O throttling (MB/s)		
	Backup	Snapshot Store (write)	QFS (write)
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 4. Throughput of software layers per machine under different concurrency and without I/O throttling.

Throughput of software layers without I/O throttling. Table 4 shows the average throughput of software layers when I/O throttling is not applied to control usage. The "Backup" column is the throughput of the backup service per machine. "Snapshot store" is the write throughput of the snapshot store layer and the significant reduction from this column to "Backup" column is caused by deduplication. Only non-duplicate chunks trigger a snapshot store write. Column "QFS" is the write request traffic to the underlying file system after compression. For example, with 148 MB/s write traffic to the snapshot store, QFS write traffic is about 35.3 MB/s after compression. However, the underlying disk storage traffic will be three times greater with replication. The result shows that the backup service can deliver up to 5.46 GB/s per machine without I/O restriction under 6 concurrent backup tasks. For our dataset, each version of total snapshots has abut 1TB per machine for 25 VMs and thus each machine would complete the backup in about 3.05 minutes. With a higher disk storage bandwidth available, the above backup throughput would be higher. In comparison, the sampled index method in [10] achieves about 5.5GB per second with 16 concurrent backup tasks and 6GB per second with more tasks. Thus the per-machine throughput of our scheme is reasonable.

5.3 Effectiveness of Approximate Deletion

Table 5 lists a comparison of processing time and memory usage using the four deletion methods when the number of physical machines $p = 100$ and $p = 50$. These four methods are 1) the standard mark-and-sweep method. 2) Grouped mark-and-sweep [10]. 3) VC without using summary vectors (Row 4). 4) VC with summary vectors. Last row of Table 5 is the cost of the leakage repair for VC with summary vectors. The mark-and-sweep process requires all machines read snapshot metadata for usage comparison. The

	Time p=50 (hours)	Time p=100 (hours)	Memory (GB)
Mark&sweep	35.9	84.3	1.2–3
Grouped mark&sweep	18.6	43.6	1.2–3
VC w/o sum	0.7	0.82	0.05 – 1.96
VC	0.012	0.014	0.015
VC Repair	0.7	0.82	0.05 – 1.96

Table 5. Processing time and per-machine memory usage of four deletion methods

I/O read speed for the backend distributed file system is about 50MB/second and there is some throughout contention when all machines read data simultaneously: the speed drops to about 30MB/second when $p = 50$ and 25MB/s with $p = 100$. We explain the results for $p = 100$ below. The explanation for $p = 50$ is similar and the result difference for $p = 100$ and $p = 50$ shows our deletion method scales well when the number of machines increases.

For the mark-and-sweep method (Row 2 of Table 5) on $p = 100$, we conduct p phases of the mark-and-sweep process. At each phase, a physical machine reads $1/p$ of the non-deduplicated chunk metadata and keeps a reference table in the memory. Then all machines read the meta data of snapshots in parallel and mark the used chunks in the above reference table. The above phase is repeated 100 times (one for for each physical machine). The memory allocated at each physical machine is for the chunk reference table at each phase. The average size is 1.2GB and the maximum is 3GB due to data skewness. There is a tradeoff between memory usage of a reference table in terms of the size and the total processing time. If we reduce the size of the reference table at each phase, then there are more phases to mark all data and the whole process will take more time. For the grouped mark-and-sweep (Row 3), about 50% of snapshot metadata reading can be avoided by actively tracking the reference usage of non-duplicate chunks. Thus it takes 50% less time, which is about 43 hours, but the memory requirement does not decrease.

For VC without summary vectors (Row 4 of Table 5), all physical machines conduct the mark-and-sweep process in parallel, but each machine only handles one VM at time and the scope of meta data comparison is controlled within the single VM. Popular chunks are excluded. The average memory usage is the index size of non-deduplicated VM chunks, which is about 50MB on average and the largest size is 1.96GB. For VC with summary vectors (Row 5), each physical machine loads the VM snapshots and only needs to compare with the summary vectors. The memory usage is controlled around 15MB for hosting the summary vectors and small buffers. The deletion time is reduced to less than 1 minute. The periodic leakage repair (Row 6) still takes about 0.83 hours while using an average of 50MB memory.

Del. step	1	3	5	7	9
Estimated	.02%	.06%	.10%	.14%	.18%
Measured	.01%	.055%	.09%	.12%	.15%

Table 6. Accumulated storage leakage by approximate snapshot deletions ($\Delta u/u = 0.025$)

For few big VMs due to data skewness, the repair uses upto 1.9GB memory. Such a repair does not occur often (e.g. every 19.6 days as discussed in Section 3.2).

Table 6 shows the average accumulated storage leakage in terms of percentage of storage space per VM caused by approximate deletions. In this experiment, we select 105 VMs and let all the VMs accumulate 10 snapshots, then start to delete those snapshots one by one in reverse order. As we know the actual storage needs after each snapshot creation, the storage leakage can be detected by comparing the size of remaining data in use after deletion to the correct number. Row 1 in Table 6 is the deletion step. Entry value 3 in this row means that Version 3 is deleted. Row 2 is the predicted leakage using Formula 2 from Section 3.2 given $\Delta u/u = 0.025$, while Row 3 lists the actual average leakage measured during the experiment for all the VMs. The Bloom filter setting is based on $\Delta u/u = 0.025$. After 9 snapshot deletions, the actual leakage ratio reaches 0.0015 and this means that there is only 1.5MB space leaked for every 1GB of stored data. The actual leakage can reach 4.1% after 245 deletions. This experiment shows that the leakage of our approximate snapshot deletion is very small, below the estimated number.

5.4 Snapshot Availability

We demonstrate the snapshot availability of VC and SRB-based VO when there are failed machines. Appendix A provides a detailed analysis of snapshot availability and the estimated result depends on how frequent a duplicate chunk is shared among snapshots of VMs. The trace driven execution allows us to estimate the number of file blocks shared by each VM in the VO and VC approaches and then calculate the average availability of VM snapshots.

Table 7 shows the estimated availability of VM snapshots when there are up to 20 machines failed in a cluster. The case for a 1000-node cluster is also listed, assuming file block sharing patterns among VMs remain the same from $p = 100$ setting to $p = 1000$. Each machine hosts about 25 VMs in both cases and with different p values, the availability varies when the number of failures in the cluster changes.

Table 7 shows that VC has a significantly higher availability than VO. With $p = 100$ and 3 failures, VO with 69.5% availability could lose data in 763 VMs while VC with 99.7% only loses data for 8 VMs out of 2500 VMs. The key reason is that for most data in VC, only a single VM can be affected by the loss of a single file block while in VO, the loss of a single block tends to affect many VMs. When the number of physical machines failed reaches 20, the availability of VO reaches 0% and VC also drops to 1.13%. This is because even though availability for PDS data remains good, VC still loses many of the non-PDS blocks given the replication degree for non-PDS is 3. When $p = 1000$, the percentage of failures is then 2% and VC delivers a meaningful availability with 99.62%, outperforming VO significantly.

Figure 7 shows the impact of increasing PDS data replication degree. While the impact on storage cost is small (because we have separated out only the most popular blocks), a replication degree of 6 has a significant improvement over 4. The availability does not increase much when increasing r_c from 6 to 9 and the benefit is minimal after $r_c > 9$. That is because when the number of failed machines increases beyond 6, the non-PDS data availability starts to deteriorate significantly given its replication degree r is set to 3. Thus when $r = 3$, the reasonable choice for r_c would be a number between 6 and 9.

Failures (d)	VM Snapshot Availability(%)			
	$p = 100$		$p = 1000$	
	VO	VC	VO	VC
3	69.548605	99.660194	99.964668	99.999669
5	2.647527	96.653343	99.647243	99.996688
10	0	66.246404	95.848082	99.96026
20	0	1.132713	66.840855	99.623054

Table 7. Availability of VM snapshots for VO ($r = 3$) and VC ($r_c = 6$, $r = 3$).

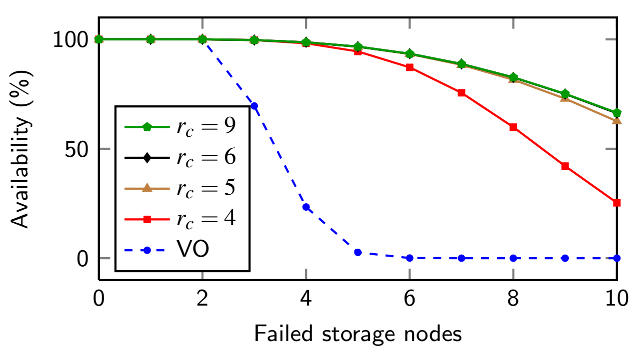


Figure 7. Availability of VM snapshots in VC with different PDS replication degrees on a $p = 100$-node cluster.

6. Conclusions

The main contribution of this paper is the development and comparative study of a low-profile and VM-centric deduplication scheme that uses a small amount of system resource in both snapshot deduplication and deletion while delivering competitive deduplication efficiency. The VM-centric design also allows an optimization for better fault isolation and tolerance. Our evaluation has the following results.

- *Competitive deduplication efficiency with a tradeoff*. The proposed VC scheme can accomplish over 96% of what complete global deduplication can do. In comparison, SRB [2, 7] can accomplish 97.86% while the sampled index [10] can deliver 97% for a different test dataset.
- *Lower resource usage in deduplication*. The VC scheme achieves a 85K:1 ratio between raw snapshot data and memory usage and is much more memory-efficient than SRB with 4K:1 ratio and sampled index with 20K:1. VC with 100 machines takes 1.31 hours to complete the backup of 2,500 VMs using 19% of one core and 16.7% of IO bandwidth per machine. Processing time of VC is 23.9% faster than SRB in our tested cases and the per-machine throughput of VC is reasonable based on the result from [10]. Noted that both VC and SRB are designed for a distributed cluster architecture while the sampled index method is for a single-machine architecture.
- *Lower cost in snapshot deletion*. The deletion in VC with summary vectors takes less than a minute for snapshot deletion using 15MB memory per machine and all machines can run this operation in parallel without data dependence. The grouped mark-sweep method can take 43 hours using 1.2 to 3GB memory per machine in our tested cases. The leak repair in VC takes 0.82 hours using 50MB on average while processing some VMs may use upto 1.96GB. Such a repair is conducted infrequently (e.g. every 19.6 days)
- *Higher availability*. The snapshot availability of VC is 99.66% with 3 failed machines in a 100-machine cluster while it is 69.54% for SRB or a VM-oblivious approach. The analysis shows that the replication degree for the popular data set between 6 and 9 is good enough when the replication degree for other data blocks is 3, and adds only a small cost to storage.

The offline PDS recomputation does require some modest I/O and memory resource and since the recomputing frequency is relatively low (e.g. on a monthly basis), we expect such resource consumption is acceptable in practice.

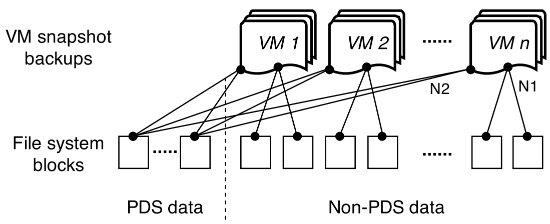
A. Fault Tolerance and Snapshot Availability

We analyze the impact of losing d physical machines to the VM centric and oblivious approaches. There are two impacts to VC. 1) Some PDS fingerprint lookup services do not respond. As a result, some duplicates are not detected and deduplication efficiency suffers, but the overall systems can still function well and fault tolerance is not affected. As discussed in Section 4, PDS index is distributed among all machines in our implementation and thus a percentage of failed nodes causes an increase in missed duplicates proportionally. 2) Some storage nodes do not respond and file blocks on those machines are lost. The availability of VM snapshots is affected and we analyze this impact as follows.

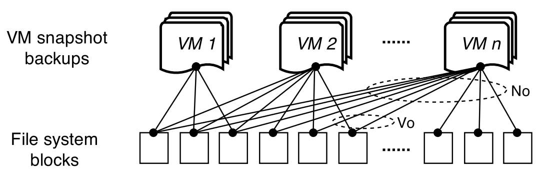
To compute the full availability of all snapshots of a VM, we estimate the number of file system blocks per VM and the probability of losing a snapshot file system block of a VM in each approach as follows. Parameters used in our analysis below are defined in Table 8.

r, r_c	replication degree of non-PDS and PDS file blocks in VC. r is also replication degree in VO.
n, p	no. of virtual and physical machines in the cluster
N_1, N_2	the average no. of non-PDS and PDS file blocks in a VM in VC
N_o, V_o	the average no. of file blocks in a VM and the average no. of VMs shared by a file system block in VO
A(r)	availability of a file block with r replicas and d failed physical machines

Table 8. Modeling parameters



(a) Sharing of file system blocks under VC



(b) Sharing of file system blocks under VO

Figure 8. Bipartite association of VMs and file blocks.

As illustrated in Figure 8, we build a bipartite graph representing the association from unique file system blocks to their corresponding VMs in two approaches. For VC, each VM has an average number N_1 of non-PDS file system blocks and has an average of N_2 PDS file system blocks. Each non-PDS block is associated with only one VM. By counting outgoing edges from VMs in Figure 8(a), we get: $n * N_1 =$ Number of non-PDS file system blocks in VC.

For VO, by counting outgoing edges from VMs in Figure 8(b) with parameters defined in Table 8, we have $n * N_o = V_o *$ Number of file system blocks in VO.

Since we choose 2-4% of unique chunks for PDS and Section 5.1 shows that the deduplication efficiency of VC is very close to that of VO, the number of non-PDS file blocks in VC is fairly close to the number of file blocks in VO. Then

$$\frac{N_o}{N_1} \approx V_o.$$

Figure 9 shows N_1, N_2, and N_o values of 105 VMs from a test dataset discussed in Section 5 when increasing the number of VMs. N_1 is much smaller than N_o as the formula shows above.

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 9. $A(r_c)$ values in a cluster with p=100 nodes.

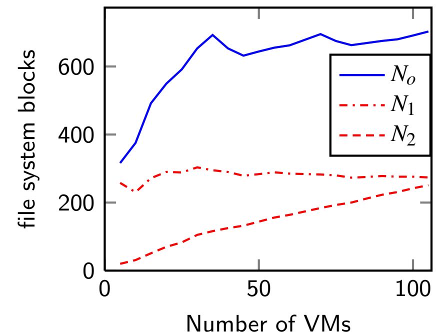


Figure 9. Measured average number of 64MB file system blocks used by a single VM in VC and VO.

Given d failed machines and r replicas for each file block, the availability of a file block is the probability that all of its replicas do not appear in any group of d failed machines among p nodes [8]. Namely, $A(r) = 1 - \binom{d}{r} / \binom{p}{r}$. Then the availability of one VM's snapshot data under VO approach is the probability that all its file blocks are unaffected during the system failure:

$$A(r)^{N_o}.$$

For VC, there are two cases: $r \leq d < r_c$ and $r_c \leq d$.
$r \leq d < r_c$: In this case there is no PDS data loss and we need to look at the non-PDS data loss. The full snapshot availability of a VM is:

$$A(r)^{N_1}.$$

Since N_1 is typically much smaller than N_o, the VC approach has a higher availability of VM snapshots than VO in this case.
$r_c \leq d$: 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

$$A(r)^{N_1} * A(r_c)^{N_2}.$$

That is still smaller than that of V_O based on the observations of our data. There are two reasons for this: 1) N_1 is much smaller than N_o and we are observing that $N_1 + N_2 < N_o$. 2) $A(r) < A(r_c)$ because $r < r_c$. Table 9 lists the $A(r)$ values with different replication degrees, to demonstrate the gap between $A(r)$ and $A(r_c)$.

References

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

[3] D. Bhagwat, K. Pollack, D. D. E. Long, T. Schwarz, E. L. Miller, and J.-F. Paris. Providing High Reliability in a Minimum Redundancy Archival Storage System. In *MASCOTS '06*, pages 413–421. IEEE, 2006. ISBN 0-7695-2573-3. .

[4] F. C. Botelho, P. Shilane, N. Garg, and W. Hsu. Memory efficient sanitization of a deduplicated storage system. In *FAST'13*, pages 81–94. USENIX. ISBN 978-1-931971-82-9. URL `http://dl.acm.org/citation.cfm?id=1960475.1960477`.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997. ISSN 0169-7552. .

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

[7] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *FAST'11*, pages 2–2. USENIX. ISBN 978-1-931971-82-9. URL `http://dl.acm.org/citation.cfm?id=1960475.1960477`.

[8] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop: Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, 4(9):575–585, June 2011. ISSN 2150-8097. . URL `http://dx.doi.org/10.14778/2002938.2002943`.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP*, pages 29–43. ACM, 2003.

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

[11] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *SYSTOR'09*, page 1. ACM. ISBN 9781605586236. . URL `http://dl.acm.org/citation.cfm?id=1534530.1534540`.

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

[13] 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. USENIX.

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

[15] Nutanix. Nutanix Complete Cluster, A Technical Whitepaper , 2013.

[16] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly. The quantcast file system. *Proc. VLDB Endow.*, 6 (11):1092–1101, Aug. 2013. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=2536222.2536234`.

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

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

[19] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content-addressed storage in foundation. In *ATC'08*, pages 143–156. USENIX. URL `http://dl.acm.org/citation.cfm?id=1404014.1404025`.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *MSST'10*, pages 1–10. .

[21] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *FAST'12*, pages 24–24. USENIX. URL `http://dl.acm.org/citation.cfm?id=2208461.2208485`.

[22] M. Vrable, S. Savage, and G. M. Voelker. Cumulus: Filesystem backup to the cloud. In *FAST'09*, pages 225–238. USENIX. ISBN 978-1-931971-66-9.

[23] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. *ATC'05*, page 22. URL `http://dl.acm.org/citation.cfm?id=1247360.1247382`.

[24] J. Wendt. A Candid Examination of Data Deduplication. White Paper, Symantec Corporation , 2009.

[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.

[26] W. Zhang, T. Yang, G. Narayanasamy, and H. Tang. Low-cost data deduplication for virtual machine backup in cloud storage. In *5th USENIX Workshop on Hot Topics in Storage and File Systems*, Berkeley, CA, 2013. USENIX.

[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. USENIX.