VM-Centric Deduplication with Fault Isolation for Snapshot Backup in the Cloud

Abstract

Data deduplication has been widely used for cloud data backup because of excessive redundant content blocks. Common techniques perform fingerprint comparison to remove duplicates across virtual machines. However, letting a duplicate data block be shared by many virtual machines creates data dependence and is less fault-resilient. This paper proposes a VM-centric backup service on a converged storage cluster architecture and strikes a tradeoff for better fault isolation with competitive deduplication efficiency. It localizes deduplication as much as possible within each virtual machine, guided by similarity search, and it restricts global deduplication under popular chunks with extra replication support. It associates underlying file blocks with one VM for most cases and proposes an approximate method for fast and simplified snapshot deletion. This VM-centric scheme also has an advantage of using less computing resources, suitable for a collocated backup service on a non-dedicated cluster architecture. This paper describes an evaluation of this scheme to assess its deduplication efficiency and fault resilience.

1 Introduction

One of emerging architectures for building cloud services is a converged architecture that leverages commodity servers with software clustered storage, without requiring network storage (e.g. [**?**]). Leading cloud platform companies such as Google, Amazon, Alibaba, Microsoft Azure have built a converged compute and storage infrastructure and used a distributed file system such as Google file system [7, 18] to glue a large number of commodity servers. In such an environment, each physical machine runs a number of virtual machines and their virtual 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 and deduplication of redundant content blocks [14, 25] is necessary to substantially reduce the storage demand.

While version-based detection has been used to identify file blocks that have not changed from the previous version of the snapshot [5, 21, 20], a popular technique for deduplicatioon is to conduct fingerprint comparison and identify duplicates that exist among all files [8, 6, 2]. Because of highly repetitive content in snapshots from different VMs, many data chunks are shared by virtual machines. Failure of a few shared data chunks can have a broad effect and snapshots of these 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 seek a method that strikes a balance between fault isolation and deduplication efficiency.

The main contribution of this work is to evaluate the fault tolerance impact of popular data blocks shared by many virtual machines, and propose a VM-centric approach that localizes deduplication as much as possible and restricts global deduplication only to a limited set of most popular blocks. Local deduplication also uses similarity-guided elimination to improve the deduplication coverage. Since the file system block size is normally bigger than the average data chunk size used for deduplication, we package data chunks from the same VM into a file system block as much as possible to improve fault isolation. Because data sharing is restricted, this VM-centric approach reduces the overall resource usage significantly during backup and simplifies the snapshot deletion process. This low-resource design is suitable when the backup service with deduplication is collocated with other services running on a shared compute and storage cluster. We have evaluated this VM-centric approach using a prototype system.

The rest of this paper is organized as follows. Section 2 reviews the background and discusses the design options for snapshot backup with a VM-centric approach. Section 3 analyzes the tradeoff and benefits of the VM-centric approach. Section 5 describes a system implementation that evaluates the proposed techniques. Section 6 is our experimental evaluation that compares with other approaches. Section 7 concludes this paper.

2 Background and Design Considerations

Figure 1 illustrates a converged cloud architecture where each commodity server hosts a number of virtual machines and storage of these servers is clustered using a distributed file system [7, 18]. Each physical machine 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. A backup sevice may collocate with other cloud services on these commodity servers and stores deduplicated data in the storage in this cluster or in a seperate storage cluster.

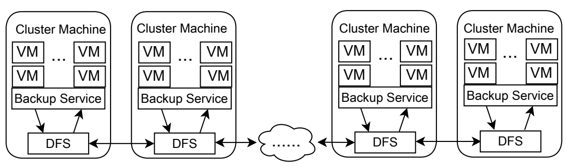


Figure 1: VM snapshot backup running on a converged cloud cluster.

File backup systems have been developed to use content fingerprints to identify duplicate content [14, 16]. It is expensive to compare a large number of chunk fingerprints and several techniques have been proposed to speedup. For example, the data domain method [25] uses an in-memory Bloom filter and a prefetching cache for data chunks which may be accessed. The approximation techniques are studied in [2, 8, 23] to reduce memory requirements with the tradeoff of a reduced deduplication ratio. Additional inline deduplication techniques are studied in [11, 8, 19] and a parallel batch solution for cluster-based deduplication is studied in [24]. 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 a cloud cluster environment considered in this paper. Our consideration is discussed below.

- *Deduplication localization.* Because a data chunk is compared with fingerprints 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. Additionally, by localizing duplicate search, the resource requirements for deduplication can be reduced, because fewer chunks must be compared.

  Another disadvantage of sharing is that it complicates snapshot deletion, which occurs frequently when snapshots expire regularly. The mark-and-sweep approach [8, 3] is effective for deletion, but still carries a significant cost to count if a data chunk is still shared by other snapshots. Localizing deduplication can minimize data sharing and simplify deletion while sacrificing deduplication efficiency, and can facilitate parallel execution of snapshot operations.

- *Management of file system blocks.* The file system block (FSB) size in a distributed file system such as Hadoop and Google file system is uniform and large (e.g. 64MB), while the data chunk in a typical deduplication system is of a non-uniform size with 4KB or 8KB on average. Packaging data chunks to an FSB can create more data dependencies among VMs since a file system block can be shared by even more VMs. Thus we need to consider a minimum association of FSBs to VMs in the packaging process. By minimizing this association we can improve fault tolerance by reducing the number of VMs affected when storage nodes are lost.

- *Low resource usage in a nondedicated cloud cluster.* When a backup service collocates with other cloud services as illustrated in Figure 1, low resource usage of deduplication is desired so that there is a minimal impact to the other existing cloud services. The key resource for global comparison is memory for storing the fingerprints. During snapshot deltion, reference counting for all unique blocks used in all VMs requires a significant resource usage too. Thus we seek for the low-profile techniques with approximation and low resource consumption.

We call the traditional deduplication approach as VM-oblivious (VO) because they compare fingerprints of snapshots without consideration of VMs. With the above considerations in mind, we study a VM-centric approach (called VC) for a collocated backup service with resource usage friendly to the existing applications in a converged clustr architecture.

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

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

3 VM-centric Snapshot Deduplication

3.1 VM centric strategies

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

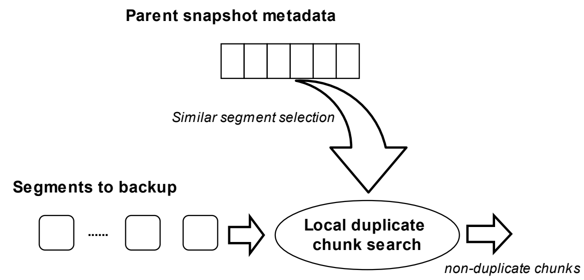


Figure 2: Similarity-guided local duplicate detection

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

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

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

Figure 3 illustrates the distribution of chunk popularity based on a data trace from Alibaba with over 2500 VMs. The distribution is zipf like and popular chunks are dominating the large portion of data chunks. We denote σ as the percentage of unique data chunks belonging to PDS and from the evaluation we find that σ with a range of 1 to 4% can deliver a fairly competitive deduplication efficiency.

VM-centric file system block management. When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written to the file system. Since the backend file system typically uses a large block size such as 64MB, each VM will accumulate small local chunks. We manage this accumulation process using a log-structured storage scheme built on a distributed file system discussed in Section 5. Each file system block is either dedicated to non-PDS chunks, or PDS-chunks. A file system block for non-PDS chunks is associated with one VM and does not contain any PDS chunks, such that our goal of fault isolation is maintained. In addition, storing PDS chunks separately allows special replication handling for those popular shared data.

It should not be noted that one could consider to rely on filesystem features to provide more replication to popular blocks, without the extra complexity of separating popular and non-popular data. The main problem with this approach is that without separating the popular chunks from the less-popular, the popular chunks are dispersed across all of the filesystem blocks in the storage system. We would have to add extra replication to each file block as long as it contains a popular chunk.

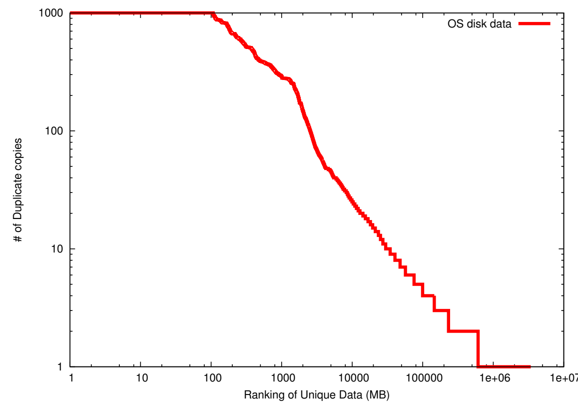


Figure 3: Duplicate frequency versus chunk ranking in a log scale after local deduplication.

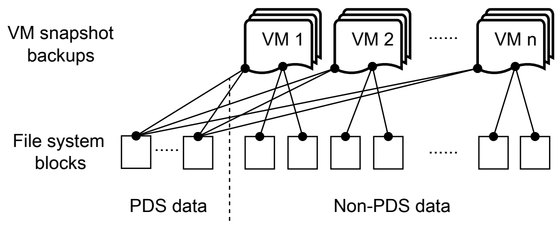
3.2 Impact on Fault Isolation

We analyze the impact of the above VM centric design on fault isolation and compare it with a VM oblivious design. Let r be the replication degree of standard file blocks in the underlying backup storage. $r = 3$ is a typical setting in distributed file systems [7, 18]. Let r_c be the replication degree of file blocks for PDS. Additional parameters used in our analysis below are defined in Table 1.

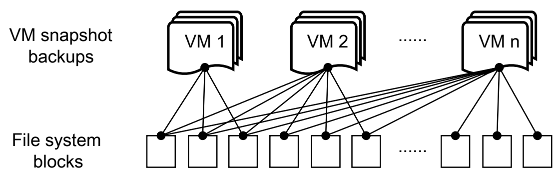
Now we assess the impact of losing d machines to the VM centric and oblivious approaches. A large r_c/r ratio can have a positive impact on full availability of VM snapshot blocks. To compute the full availability of all snapshots of a VM, we estimate the number of file sys-

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

Table 1: Modeling parameters



(a) Sharing of file system blocks under VC



(b) Sharing of file system blocks under VO

Figure 4: Bipartite association of VMs and file system blocks under (a) VC and (b) VO.

tem blocks per VM and the probability of losing a snapshot file system block of a VM in each approach as follows.

As illustrated in Figure 4, we build a bipartite graph representing the association from unique file system blocks to their corresponding VMs in two approaches. We use a file system block rather than data chunk as our unit of failure because the underlying file system keeps file system blocks as its base unit of storage, and in the case of 64MB file system blocks and 4KB chunks, there are 16K chunks per file block. For VM centric, we assign extra replication to the PDS blocks, while for VM oblivious, the replication degree is fixed.

For VC, each VM has an average number N_1 of non-PDS FSBs and has an average of N_2 PDS FSBs. Each non-PDS FSB is associated with one VM and we denote that PDS FSBs are shared by an average of V_c VMs. Then,

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

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

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

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

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

When E_c is close to E_o, N_1 is much smaller than N_o. Figure 5 shows the average number of file system blocks for each VM in VC and in VO and N_1 is indeed much smaller than N_o in our tested dataset.

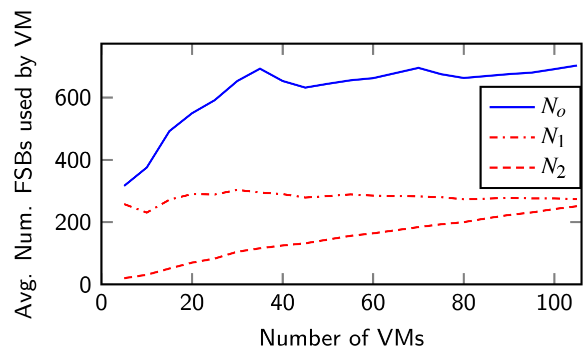


Figure 5: Measured average number of 64MB FSBs used by a single VM. For VC both the number of PDS and Non-PDS FSBs used are shown.

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

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

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

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

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

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

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

  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.

- When $r_c \le d$, 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

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

We have considered a worst case scenario that every PDS FSB is shared by all VMs in the VC approach, which

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

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

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

4 VM-centric Approximate Snapshot Deletion with Leak Repair

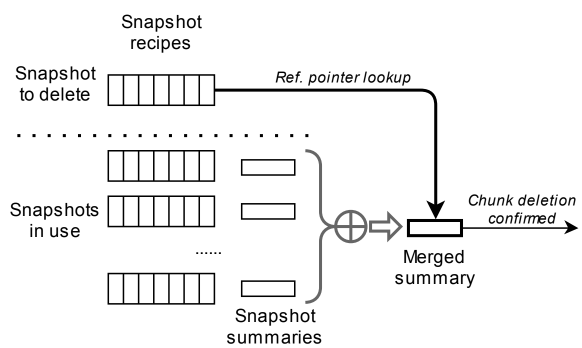


Figure 6: Approximate deletion

Snapshot deletions can occur frequently since old snapshots become less useful. General deduplication complicates the deletion process because sharing of duplicates requires a global refence counting to identify if a chunk can be safely removed without any reference. Since our VM-centric design restricts sharing of data chunks only to a small dataset, we can greatly simplify the deletion process by focusing on unreferenced non-PDS chunks within each VM. This process can be conducted by independent among VMs, and thus results in a simpler flow control and lower resource usage.

We further propose 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 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 estimation of 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 the periodic leakage repair stage described below.

- **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 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 bitwise OR operator and the merged vector still takes z bits. Since the number of live snapshots h is limited for each VM, the time and memory cost of this comparison is small, linear to the number of chunks to be deleted.

  If a chunk's reference is not found in the merged summary vector, we are sure that this chunk is not used by any live snapshots, thus 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.

- **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 the entire snapshot store. Since it is a VM-specific procedure, the space and time cost is relatively small compared to the tranditional mark-and-sweep which scans snapshot chunks from all VMs. For example, consider each reference consumes 8 bytes plus 1 mark bit. A VM that has 40GB backup data with about 10 million chunks will need less than 85MB of memory to complete a VM-specific mark-and-sweep process in less than half an hour, assuming 50MB/s disk bandwidth is allocated.

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

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

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,

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

For example in our tested dataset, $h = 10$ and each snapshot adds about 0.1-5% of new data. Thus we take $\Delta u/u \approx 0.025$. 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.

5 Prototyping and Implementation Details

Our system runs on a cluster of Linux machines with Xen-based VMs and an open-source package for the distributed file system called QFS [13]. All data needed for 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[22] in Xen).

5.1 Components of a Cluster Node

As depicted in Figure 7, there are four key service components running on each cluster node for supporting backup and deduplication: 1) a virtual block device driver, 2) a snapshot deduplication 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 (CBT). 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 [10], which brings the opportunity of fine-grained deduplication. When the VM issues a disk write, the dirty bit for the corresponding segment is set and this indicates such a segments needs to be checked during snapshot backup. After the snapshot backup is finished, the driver resets the dirty bit map to a clean state. For data modification during backup, 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.

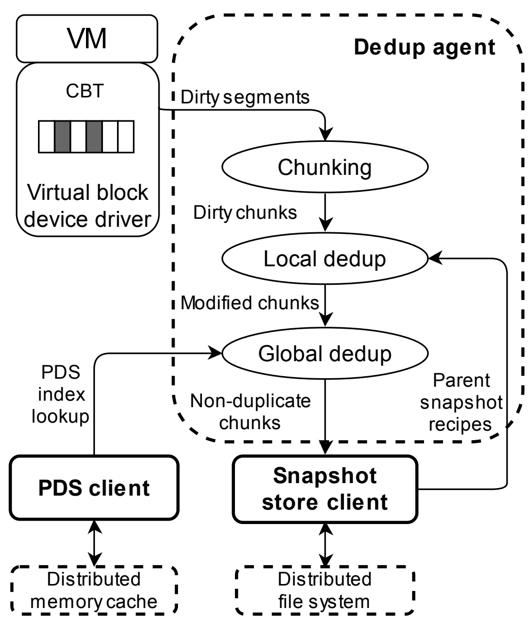


Figure 7: System architecture and data flow during snapshot backup

5.2 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. As shown in Figure 8, we explain the data structure of the snapshot stores as follows.

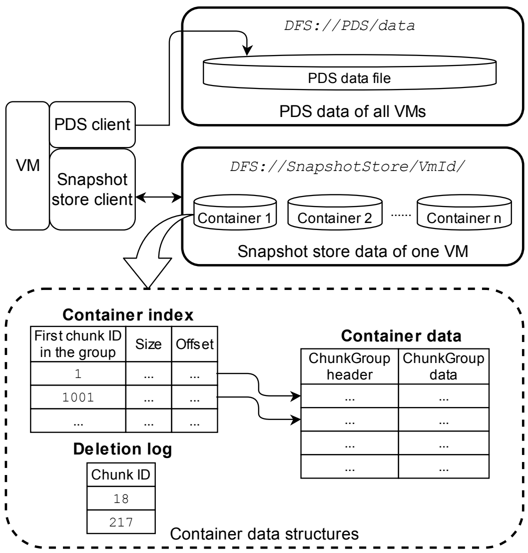


Figure 8: Data structure of a VM snapshot store.

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

Each non-PDS data 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 [8, 17], 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 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 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 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.

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. Note that PDS append 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 metadata of a snapshot. The write requests to append data chunks to a VM store are accumulated at the client side. When the number of write requests reaches a fixed group size, the snapshot store client compresses the accumulated chunk group, adds a chunk group index to the beginning of the group, and then appends the header and data to the corresponding VM file. A new container index entry is also created for each chunk group and is written to the corresponding container index file.

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

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

Delete(). Chunk deletion occurs when a snapshot expires or gets deleted explicitly by a user and we discuss this in more details in next subsection. When deletion requests are issued for a specific container, those requests are simply recorded into the container's deletion log initially and thus a lazy deletion strategy is exercised. Once local chunk IDs appear in the deletion log, they will not be referenced by any future snapshot and can be safely deleted when needed. This is ensured because we only dedup against the direct parent of a snapshot, so the deleted snapshot's blocks will only be used if they also exist in other snapshots. Periodically, the snapshot store 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.

6 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 6. Our evaluation objective is to study the benefit in fault tolerance and deduplication efficiency of VC, and assess its backup throughput and resource usage.

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

6.1 Settings

We have performed a trace-driven study using a production dataset [23] from Alibaba Aliyun's cloud platform with about 100 machines. Each machine hosts up to 25 VMs and each VM keeps 10 automatically-generated snapshots in the storage system while a user may instruct extra snapshots to be saved. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, win2008 and win2003. Based on our study of production data, each VM has about 40GB of storage data on average including OS and user data disk. The fingerprint for variable-sized chunks is computed using their SHA-1 hash [12, 15].

6.2 Fault Isolation and Snapshot Availability

Table 3 shows the availability of VM snapshots when there are up to 20 machine nodes failed in a 100-node cluster and a 1000-node cluster. We have assumed a worst-case scenario that a PDS block is shared by every VM. Our results show that even the worst case, VC still has a significantly higher availability than VO as the number of failed machines increases. For example, with 5/100 machines failed and 25 VMs per machine, VO with 93.256% availability would lose data in 169 VMs while VC with 97.763% loses data for 56 VMs. The key reason is that for most data in VC, only a single VM can be affected by the loss of a single FSB. Since most FSBs contain chunks for a single VM, VMs can depend on a smaller number of FSBs.

Although the loss of a PDS block affects many VMs, by increasing replication for those blocks we minimize the effect on VM snapshot availability. Figure 9 shows the impact of increasing PDS data replication. While the impact on storage cost is small, a replication degree of 6 has a significant improvement over 4, but the availability

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

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

is about the same for $r_c = 6$ and $r_c = 9$ (beyond $r_c = 6$ improvements are minimal).

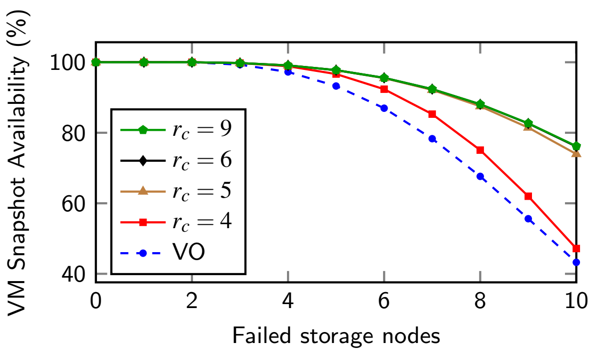


Figure 9: Availability of VM snapshots in VC with different PDS replication degrees

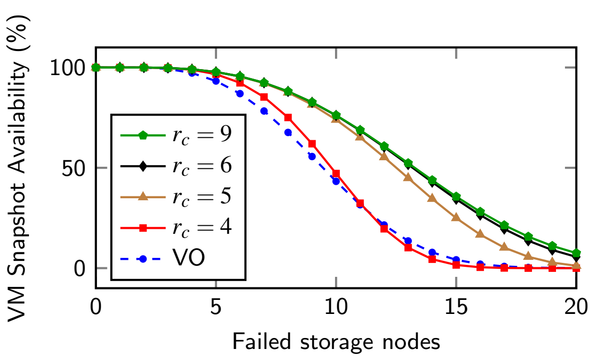


Figure 10: Availability of VM snapshots in VC with different PDS replication degrees *decide which one*

6.3 Deduplication Efficiency

Figure 11 shows the deduplication efficiency for SRB and VC, namely the percent of duplicate chunks which are detected and removed. With $\sigma = 2\%$, memory usage for PDS index lookup per machine is about 100MB per machine and the deduplication efficiency can reach over 96.33%. When $\sigma = 4\%$, the deduplication efficiency can reach 96.9% while space consumption increases to 200MB per machine. The loss of efficiency in VC is caused by the restriction of the physical memory available in the cluster for fast in-memory PDS index lookup. SRB can deliver up to 97.79% deduplication efficiency, which is slightly better than VC. Thus this represents a tradeoff that VC provides better fault tolerance and fast approximate deletion with competitive deduplication efficiency.

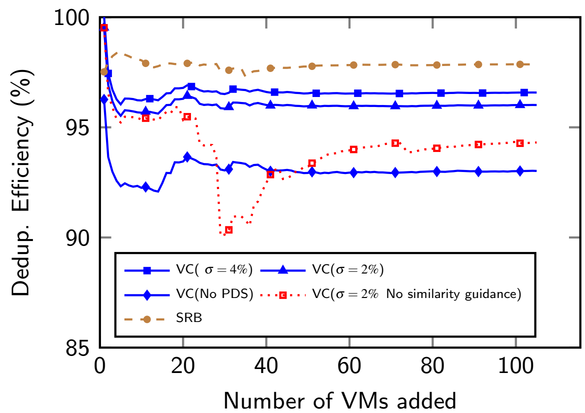


Figure 11: Deduplication efficiency of VC and SRB.

Figure 11 also shows the curve of VC without local similarity search. There is a big efficiency drop in this curve when the number of VMs is about 30. The reason is that 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, a dirty-bit or offset based detection would not be able to detect such a movement and similarity search becomes especially important. 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 93% to over 96%. That is one significant improvement compared to the work in [23] which uses the parent segment at the same offset to detect duplicates instead of similarity-guided search.

In general, our experiments show that dirty-bit detection at the segment level can reduce the data size to about 24.14% of original data, which leads to about a 75.86% reduction. Similarity-guided local search can further reduce the data size to about 12.05% of original, namely it delivers a 50.08% reduction to the dirty segments. The popularity-guided global deduplication with $\sigma = 2\%$ can reduce the data further to 8.6% of its original size, so it provides additional 28.63% reduction to the remaining data.

6.4 Resource Usage and Processing Time

Storage cost of replication. When the replication degree of both PDS and non-PDS data is 3, the total

Tasks	CPU	Mem (MB)	Read (MB/s)	Write (MB/s)	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 4: Resource usage of concurrent backup tasks at each machine

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 and disk bandwidth usage with multi-VM processing. We have further studied the memory and disk bandwidth usage when running concurrent VM snapshot backup on each machine with $\sigma = 2\%$. Table 4 gives the resource usage when running 1 or multiple VM backup tasks at the same time on each physical machine. "CPU" column is the percentage of a single core used. "Mem" column includes 100MB memory usage for PDS index and other space cost for executing deduplication tasks such as receipt metadata and cache. "Read" column is controlled as 50MB/s bandwidth usage with I/O throttling so that other cloud services are not impacted too much. The peak raw storage read performance is about 300MB/s and we only use 16.7% with this collocation consideration. "Write" column is the I/O write usage of QFS and notice 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 4 shows that a single backup task per node can complete the backup of the entire VM cluster in about 1.31 hours. Since there are about 25 VMs per machine, we could execute more tasks in parallel at each machine. But adding more backup concurrency does not shorten the overall time significantly because of the controlled disk read time.

Processing Time breakdown. Figure 12 shows the average processing time of a VM segment under VC and SRB. VC uses $\sigma = 2\%$ and 4%. It has a breakdown of processing time. "Snapshot read/write" includes snapshot reading and writing from disk, and updating of the metadata. " Network transfer" includes the cost of transferring raw and meta data from one machine to another during snapshot read and write. "Index access/comparison" is the disk, network and CPU time during fingerprint comparison. This includes PDS data lookup for VC and index lookup from disk in VO after Bloom filter lookup. For VC, the change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time. 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 has a higher cost for index access and fingerprint comparison because most of chunk fingerprints are routed to remote machines for comparison while VC handles most of chunk fingerprints locally.

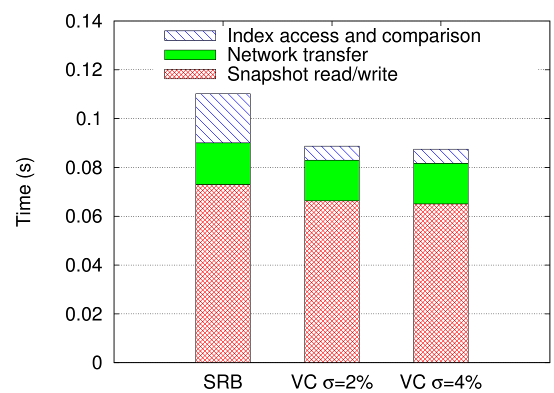


Figure 12: Average time to backup a dirty VM segment under SRB and VC

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 5: Throughput of software layers per machine under different concurrency

Throughput of software layers. Table 5 shows the average throughput of software layer when when I/O throttling is not applied to control the usage. "Backup" column is the throughput of the backup service per machine. "Snapshot store" is the write throughput of the snapshot store layer. The gap between this column and "Backup" column is caused by significant data reduction by dirty bit and duplicate detection. Only non-duplicate chunks trigger a snapshot store write. "QFS" column is the write request traffic to the underlying file system after compression. For example, with 148MB/second write

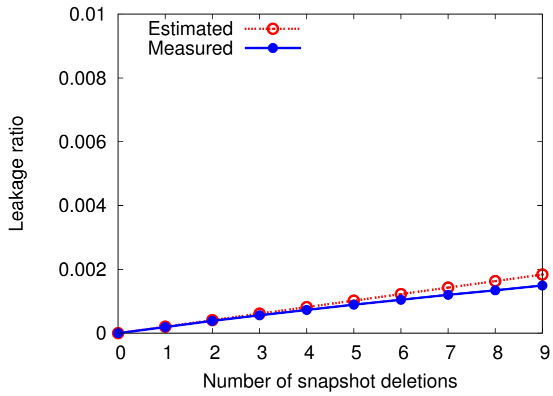


Figure 13: Accumulated storage leakage by approximate snapshot deletions ($\Delta u/u = 0.025$)

traffic to the snapshot store, QFS write traffic is about 35.3MB/second after compression. The underlying disk storage traffic will be three times greater with replication. The result shows that the backup service can deliver up to 5.46GB/second without I/O restriction per machine with 6 concurrent backup tasks. With a higher disk storage bandwidth available, the above backup throughput would be higher.

6.5 Effectiveness of Approximate Deletion

Figure 13 shows the average accumulated storage leakage in terms of percentage of storage space per VM caused by approximate deletions. The top dashed line is the predicted leakage using Formula 2 from Section 4 given $\Delta u/u = 0.025$, while the solid line represents the actual leakage measured during the experiment. 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.

7 Conclusion

The main contribution of this paper is a low-profile and VM-centric deduplication scheme to maximize fault isolation while delivering competitive deduplication efficiency using a small amount of system resource. Evaluation using this scheme strikes a tradeoff and restricted cross-VM duplicate detection can accomplish 96.33% or 96.9% of what complete global deduplication can do, and the approximate snapshot deletion effectively manages deleted chunks with a low-resource usage. The availability of snapshots increases substantially when adding more replication for popular cross-VM chunks and packaging chunks from the same VM in one file system block.

References

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

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

[3] F. C. Botelho, P. Shilane, N. Garg, and W. Hsu. Memory efficient sanitization of a deduplicated storage system. In *Proceedings of the 11th USENIX conference on File and storage technologies*, FAST'13, pages 81–94, Berkeley, CA, USA, 2013. USENIX Association.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[25] 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.