VM-Centric Snapshot Deduplication with Fault Isolation for Converged Cloud Architectures

Wei Zhang[⋆]*, Michael Agun*[⋆]*, Tao Yang*[⋆]*, Rich Wolski*[⋆]*, and Hong Tang*[†]
[⋆]*University of California at Santa Barbara* [†]*Alibaba Inc.*

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 cloud 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 running the backup service on a converged cloud architecture that cohosts many other services. This paper describes an evaluation of this scheme to assess its deduplication efficiency and fault resilience.

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 [7, 18]. In such an environment, each physical machine runs a number of virtual machines (VMs) and their virtual disks are stored as disk image files, typically using a local (possibly shared) file system. Frequent snapshot backup of virtual disk images can increase the service reliability by allowing VMs to restart from their latest snapshot in the event of a server failure. However, because the disk blocks that make up the VMs are often the same from snapshot to snapshot and are shared among VMs, deduplication of redundant content blocks [14, 25] is necessary to substantially reduce the storage demand.

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 undeduplicated backup data wastes huge amount of network bandwidth that would otherwise be available to user VMs. Even when the storage system implements deduplication internally such dedicated backup storage is either difficult to scale out (which can comprise the overall scalability of the converged architecture) or it must forego global deduplication in order to gain scalability (which increases backup cost).

This paper considers a backup service for converged cloud architectures that is co-located with user VMs, sharing the cloud computer, network, and storage resource with them. We describe methodologies to address the following two challenges: 1) simple deduplication and snapshot deletion with competitive efficiency and a small resource usage without affecting other collocated cloud services; 2) improved fault isolation for the sharing of deduplicated data, given that node failures in converged architecture are the norm rather than exception.

Version-based detection has been used to identify file blocks that have not changed across snapshots so their re-

dundant storage can be avoided [5, 21, 20]. Alternatively, techniques that compare block "fingerprints" to identify duplicates that exist among all files regardless of their origin [8, 6, 2] have also gained in popularity. One consequence of aggressive deduplication is the possible loss of failure resilience. Separate files share the same physical copies of blocks that are logically duplicated among them. If a shared block is lost, all files that share that block are affected. Thus, there exists a tension between the additional redundancy that snapshot creation is intended to provide, and the storage efficiency that deduplication makes possible. The previous work on deduplication has not considered the impact of duplicate sharing on fault tolerance.

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. We detail the storage, performance, and fault isolation properties of our method and verify these properties empirically using a prototype system that we have developed to run as a co-located back-up service in converged IaaS cloud environments. Our work focuses on sharing only "popular" data blocks (through deduplication and limited replication) across virtual machines while using localized deduplication within each VM to achieve both storage savings and fault isolation. Since the underlying storage file system block size is typically bigger than the average data chunk size used for deduplication, packaging content from different VMs into the same file block creates a "false" sharing that affects fault isolation. Thus we choose to avoid such VM-oblivious packaging. The cost associated with localization, however, is in the lost storage efficiency that cross-VM deduplication makes possible. To recover some of this efficiency we allow a small number of the most popular data blocks to be deduplicated (and then replicated) across VMs.

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 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.

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 [7, 18]. 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.

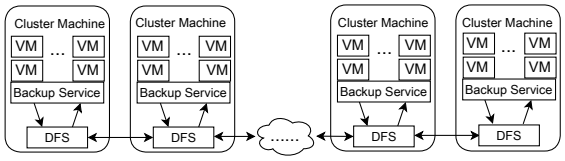


Figure 1: VM snapshot backup running on a converged cloud cluster.

File backup systems have been developed to use fingerprints generated for data "chunks" to identify duplicate content [14, 16]. In a simple implementation, it is expensive to compare a large number of fingerprints so several techniques have been proposed to improve duplicate identification. For example, the data domain method [25] uses an in-memory Bloom filter and a prefetching cache for data chunks which may be accessed. Approximation techniques are studied in [2, 8, 23] to reduce memory requirements at the cost of some loss of deduplication efficiency. 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 or speed, and none of them have considered the impact of deduplication on fault resilience in a cloud environment. To do so, we examine the following properties of VM snapshot deduplication for converged IaaS architectures.

Local versus Global Deduplication – If deduplication is implemented strictly in the storage system, a data chunk is compared with fingerprints collected from all VMs during the deduplication process, and only a small number of copies of duplicates are stored in the storage. This sharing artificially creates hidden data dependencies among different VMs owned by different who contract for isolation by the IaaS cloud platform. That is, the unit of failure visible to the cloud user is the VM, but the unit of sharing in a deduplicated data store is the data chunk.

Indeed, the greater the deduplication efficiency, the less the data redundancy is maintained, and the greater the "hidden" data dependence between VMs that must be hosted as if they were physically isolated. Thus, restricting deduplication to be local to each VM (but across snapshots) aligns the units of storage with the isolation properties guaranteed by the cloud at the possible expense of storage efficiency.

Additionally, storage efficient deduplication can be computationally expensive to implement in a distributed setting. In particular, duplicate detection is logically a "global" operation over all data chunks under consideration. By localizing duplicate search, the computational requirements for deduplication can be reduced, because fewer chunks must be compared. Further, deduplication of VMs can easily be implemented in parallel.

Another disadvantage of cross-VM 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, especially in a distributed setting.

Thus, localizing deduplication to the VM can align fault isolation properties and guarantees, improve the computational efficiency of snapshot maintenance, and simplify snapshot deletion over non-localized approaches at the possible expense of storage efficiency.

Units of Storage versus Units of Sharing – The file system block size in a distributed file system such as the Hadoop file system and the Google file system [7] is large (e.g. 64MB) so as to optimize file system performance for big data. Typically, deduplication implementations use smaller block sizes (4K bytes to 8K bytes on average [9]). In addition to the failure correlation that cross-VM data deduplication can introduce, disk block sharing of data chunks is another potential source of hidden data dependence. Thus a VM-centric solution should consider how data chunks from VMs are distributed among large file system blocks when a large-scale distributed file system is used for snapshot storage.

Moderating Resource Usage – In a converged setting, 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 must share the resources available to other cloud services as illustrated in Figure 1, and has to minimize its impact on user-deliverable performance.

For the remainder of this paper, we term the traditional deduplication approach *VM-oblivious* (VO) because it manages duplicate data chunks without consideration of VMs. With the above considerations in mind, 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.

In designing a VC deduplication 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 to the snapshot history for each VM. 1) *Stateless Data Routing*. One approach to scalable duplicate comparison is to use content-based hash partitioning 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 for Memory Parsimony*. 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

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.

VM-local duplicate search optimization – We use the standard version-based detection [5, 21] to identify changed content with dirty bits in a coarse grain segment level. The reason to choose a coarse grain 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.

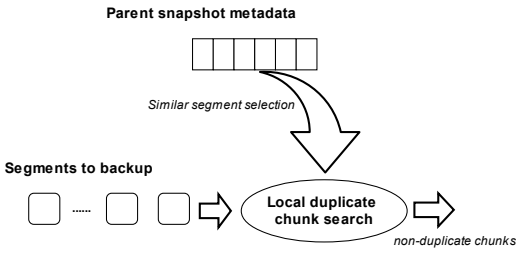


Figure 2: Similarity-guided local duplicate detection

Since the best deduplication uses a non-uniform chunk size with an average of 4KB or 8KB [9], we conduct ad-

ditional 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 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 small. As part of our local duplicate search we also compare the current segment against the parent segment at the same offset.

Global deduplication using popular chunks – 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.

Figure 3 shows the distribution of chunk popularity for a data trace from Alibaba with 2500 VMs discussed in Section 5. The distribution is Zipf-like and popular chunks dominate the distribution. We denote σ as the percentage of unique data chunks belonging to PDS and from the evaluation in Section 5, we find that σ with a range of 2 to 4% can deliver a fairly competitive deduplication efficiency.

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

VM-centric file system block management – When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written to the file system. Since

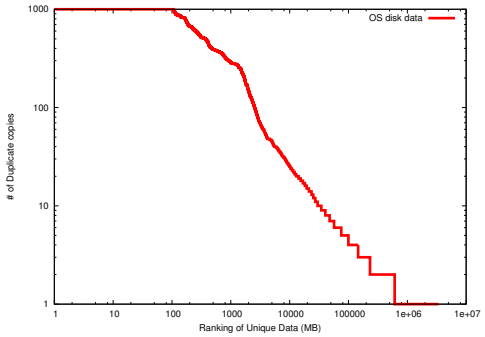


Figure 3: Duplicate frequency versus chunk ranking in a log scale after local deduplication.

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

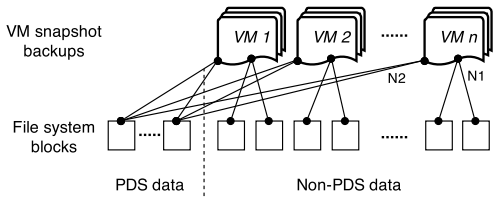
Table 1: Modeling parameters

the backend file system typically uses a large block size such as 64MB, each VM will accumulate small local chunks. 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.

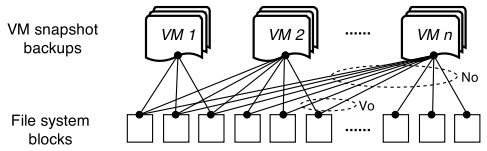
If we do not separate the popular chunks from the less-popular, the popular chunks are dispersed across all of the filesystem blocks in the storage system. Then we cannot leverage the file system features to provide extra replication to popular and shared chunks because we have to add extra replication to each file block as long as it contains a popular chunk.

3.1 Fault Isolation

Now 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



(a) Sharing of file system blocks under VC



(b) Sharing of file system blocks under VO

Figure 4: Bipartite association of VMs and file blocks.

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 1.

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. 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. Then by counting outgoing edges from VMs in Figure 4(a), we get:

$$n * N_1 = \text{ Number of non-PDS file system blocks in VC.}$$

For VO, by counting outgoing edges from VMs in Figure 4(b) with parameters defined in Table 1:

$$n * N_o = V_o * \text{ Number of file system blocks in VO.}$$

Since we choose 2-4% of unique chunks for PDS and Section 5.2 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 5 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.

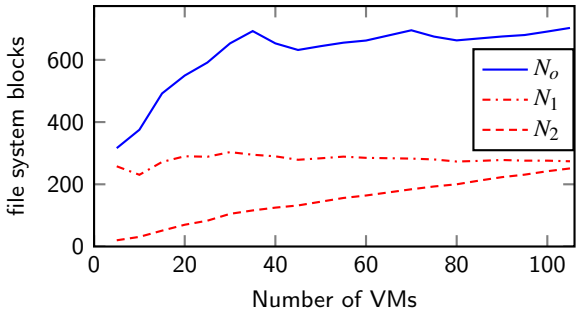


Figure 5: Measured average number of 64MB file system blocks used by a single VM in VC and VO.

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 the total of p machines. 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: $r \le d < r_c$ and $r_c \le d$.
$r \le d < r_c$: 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.
$r_c \le d$: 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

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

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 2 lists the $A(r)$ values with different replication degrees, to demonstrate the gap between $A(r)$ and $A(r_c)$.

3.2 Snapshot Deletion with Leak Repair

Snapshot deletions can occur frequently since old snapshots become less useful. General deduplication complicates the deletion process because sharing of dupli-

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

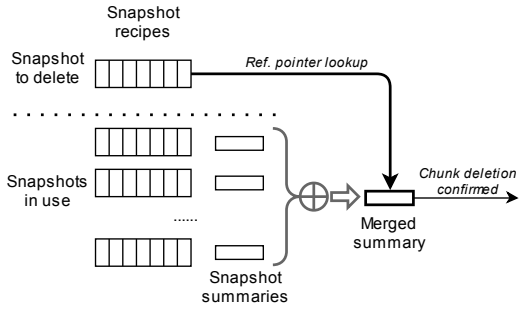


Figure 6: Fast approximate deletion

cates requires a global reference counting [8, 3] 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 independently per VM, 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 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 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.

  One advantage of the above fast method is that it can finish and free storage usage immediately, while other offline methods (e.g. [8, 3]) can't. That is important for storage accounting as users pay what are used 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. For example, the space requirement is about 85MB for a VM of size 40GB in our tested dataset. This is much less expensive than the VM-oblivious mark-and-sweep which scans snapshot chunks from all VMs, even with optimization [8].

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 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:

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

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.

4 Prototype Implementation

Our prototype system runs on a cluster of Linux machines with Xen-based VMs and the QFS [13] 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[22] in Xen).

4.1 Per Node Software Components

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. 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],

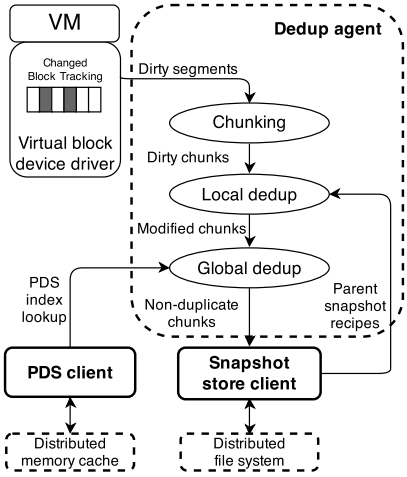


Figure 7: Data flow during snapshot backup

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 for Backup Data

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 8, 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

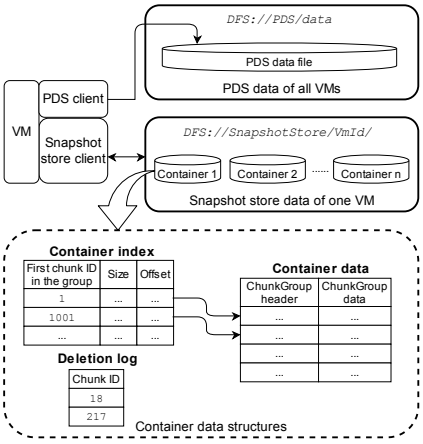


Figure 8: VM snapshot store data structures

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, 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. 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 (the overall deletion strategy was discussed in detail in Section 3.2). 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, 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 distributed deduplication by routing data chunks to 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 based on [2].

Settings. We have performed a trace-driven study based on a production dataset [23] from Alibaba Aliyun's cloud platform with about 2500 VMs, running on 100 physical 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. Each VM has about 40GB of storage data on average including OS and user data disk. Each physical machine deals with about 10TB of snapshot data and each 1TB data represents one snapshot version of 25 VMs. 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 [12, 15].

5.1 Snapshot Availability

Table 3 shows the availability of VM snapshots when there are up to 20 machines failed in a cluster with $p = 100$ physical machines. 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. The availability for a 1000-node cluster is also listed, assuming file block sharing patterns among VMs remain the same from $p = 100$ setting to $p = 1000$. With different p values, how the availability varies when the number of failures in the cluster changes.

Our results show that VC has a significantly higher availability than VO as the number of failed machines increases from 3 to 20 in this table. With $p = 100$ and 3 machine failures, there are 2500 VMs in total. VO with 69.5% availability could lose data in 763 VMs while VC with 99.7% loses data for 8 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. Since most blocks contain chunks for a single VM, VMs can depend on a smaller number of blocks, 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 there is a good availability for PDS data, VC still loses most of non-PDS blocks given the replication degree for non PDS is 3. On the other hand, when $p = 1000$, the percentage of machine failure is then 2% and VC outperforms VO significantly in availability in this case by delivering a meaningful availability with 99.62%.

Figure 9 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 not so visible 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 3: Availability of VM snapshots for VO ($r = 3$) and VC ($r_c = 6$, $r = 3$).

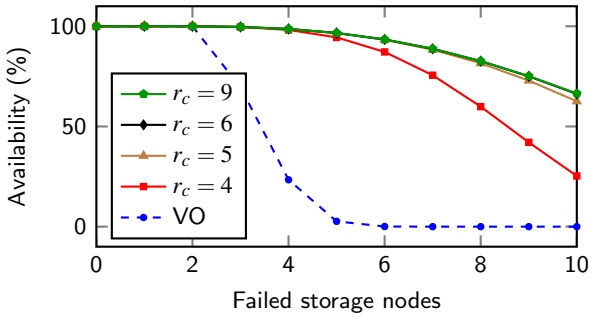


Figure 9: Availability of VM snapshots in VC with different PDS replication degrees on a $p = 100$-node cluster.

5.2 Deduplication Efficiency

Table 4 shows the deduplication efficiency for SRB and VC with different settings. Deduplication efficiency is defined as the percent of duplicate chunks which are removed compared 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%. When $\sigma = 4\%$, Column 4 shows that the deduplication efficiency can reach 96.6% while more index space is allocated per machine for PDS fingerprint lookup. 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 in Column 5 can deliver up to 97.86% deduplication efficiency, which is slightly better than VC. Thus this represents a tradeoff that VC provides better fault tolerance and fast approximate deletion with a slight reduction in deduplication efficiency.

Column 2 of Table 4 shows deduplication efficiency of VC without using PDS and it achieves 93.02% and

VC				VO
2%PDS no-simi	No PDS	2%PDS	4%PDS	SRB
94.31%	93.02%	96.01%	96.58%	97.86%

Table 4: Deduplication efficiency for different VC settings and Stateless Routing with Binning (SRB).

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

missed duplicates take up-to 3.56% of the original space, which is about 356GB per physical machine. Column 1 of Table 4 shows deduplication efficiency of VC without local similarity search. The reason similarity search provides such an improvement 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, 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 [23] which uses the parent segment at the same offset to detect duplicates instead of similarity-guided search.

In general, our experiments show that version 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. 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 to the dirty segments. 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.

5.3 Resource Usage and Processing Time

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 5 gives the resource usage when run-

ning 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 recipe metadata and cache. The "Read" column is controlled to 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" 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 5 shows that when each machine conducts backup one VM at a time, the backup of the entire VM cluster completes 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 in this case because of the controlled disk read bandwidth usage.

It should be noted that global fingerprint space requirement of VC is about 2% of all unique fingerprints with σ=2%. SRB uses an approximated method to look for a match within a bin and its memory space usage is about 190MB per machine. That is compared with 118MB used in VC with $\sigma = 2\%$. Using a bloom filter with an index cache would require more memory space [6].

Processing Time breakdown without I/O throttling. Table 6 shows the average time breakdown for processing a dirty VM segment in milliseconds under VC and Stateless Routing with Binning (SRB). VC uses $\sigma = 2\%$ and 4%. 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
VC $\sigma = 4\%$	65.072	16.626	5.784

Table 6: 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 7: Throughput of software layers per machine under different concurrency and without I/O throttling.

Throughput of software layers without I/O throttling. Table 7 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 148MB/second write 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 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.

5.4 Effectiveness of Approximate Deletion

Table 8 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

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

Table 8: Accumulated storage leakage by approximate snapshot deletions ($\Delta u/u = 0.025$)

comparing the size of remaining data in use after deletion to the correct number. Row 1 in Table 8 is the deletion step. 3 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 and such a repair is needed every 12 days to the leakage under 5%. This experiment shows that the leakage of our approximate snapshot deletion is very small, below the estimated number.

The space cost for each snapshot deletion is insignificant. Leakage repair for each VM needs less than 85MB of memory considering each reference consumes 8 bytes plus 1 mark bit and each VM snapshot has 40GB backup data with about 10 million chunks. This VM-specific leakage repair takes less than half an hour for each VM, when disk bandwidth is controlled to 50MB/s.

6 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 up to 96.58% of what complete global deduplication can do, and the simple and approximate snapshot deletion effectively manages deleted chunks with a much lower 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. 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.

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.

[3] 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.

[4] A. Broder. On the Resemblance and Containment of Documents. In *SEQUENCES '97: Proc. of Compression and Complexity of Sequences*, page 21. IEEE, 1997.

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

[6] 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.

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

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

[9] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *SYSTOR'09*, page 1. ACM.

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

[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. USENIX.

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

[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 *VLDB'13*, pages 2–2. VLDB.

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

[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 *ATC'08*, pages 143–156. USENIX.

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

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

[19] 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.

[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.

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

[22] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. *ATC'05*, page 22.

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

[24] W. Zhang, T. Yang, G. Narayanasamy, and H. Tang. Low-cost data deduplication for virtual machine backup in cloud storage. In *HotStorage'13*. USENIX.

[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. USENIX.