# Collocated Deduplication with Fault Isolation for Virtual Machine Snapshot Backup

Wei Zhang, Michael Agun, Tao Yang
Department of Computer Science
University of California, Santa Barbara, CA 93106

## ABSTRACT

A cloud environment that hosts a large number of virtual machines (VMs) has a high storage demand for frequent backup of system image snapshots. Deduplication of data blocks can lead a big reduction of redundant blocks when their signatures are identical. However it is expensive and less fault-resilient to perform a global deduplication using signatures and let a data block share by many virtual machines. This paper studies a VM-centric scheme which collocates a lightweight backup service with other cloud services in a cluster and it integrates multiple duplicate detection strategies that localize deduplication as much as possible within each virtual machine. It also organizes the write of small data chunks into large file system blocks so that each underlying file block is associated with one VM for most of cases. Our analysis shows that this VM centric scheme can provide better fault tolerance while using a small amount of computing and storage resource. This paper provides a comparative evaluation of this scheme in accomplishing a high deduplication efficiency while sustaining a good backup throughput.

## 1. INTRODUCTION

In a cluster-based cloud environment, each physical machine runs a number of virtual machines as instances of a guest operating system and their virtual hard disks are represented as virtual disk image files in the host operating system. Frequent snapshot backup of virtual disk images can increase the service reliability. For example, the Aliyun cloud, which is the largest cloud service provider by Alibaba in China, automatically conducts the backup of virtual disk images to all active users every day. The cost of supporting a large number of concurrent backup streams is high because of the huge storage demand. Using a separate backup service with full deduplication support [7, 11] can effectively identify and remove content duplicates among snapshots, but such a solution can be expensive. There is also a large amount of network traffic to transfer data from the host machines to the backup facility before duplicates are removed.

This paper seeks for a low-cost architecture option that collocates a backup service with other cloud services and uses a minimum amount of resources. We also consider the fact that after deduplication most data chunks are shared by several to many virtual machines. Failure of shared data chunks can have a catastrophic effect and many snapshots of virtual machines would be affected. The previous work in deduplication focuses on the efficiency and approximation of finger print comparison, and has not addressed fault tolerance together with deduplication. Thus we also seek deduplication options that yield better fault isolation.

The paper studies and evaluates an integrated approach which uses multiple duplicate detection strategies based on version detection, inner VM duplicate search, and controlled cross-VM comparison. This approach is VM centric by localizing duplicate detection within each VM and by packaging data chunks from the same VM into a file system block as much as possible. By narrowing duplicate sharing within a small amount of data chunks, this scheme can afford to allocate extra replicas of these shared chunks for better fault resilience. Localization also brings the benefits of parallelism exploitation so backup operations can run simultaneously without a central bottleneck. This VM-centric solution uses a small amount of memory while delivering a decent deduplication efficiency. We have developed a prototype system that runs a cluster of Linux machines with Xen. The backup storage uses a standard distributed file system with data replication and block packaging.

The rest of this paper is organized as follows. Section **??** reviews background and related work. Section **??** discusses the design options for snapshot backup with a VM-centric approach. Section **??** analyzes the benefit of our approach for fault isolation. Section 5 describes our system architecture and implementation details. Section **??** is our experimental evaluation that compare with the other approaches. Section **??** concludes this paper.

## 2. BACKGROUND AND RELATED WORK

At a cloud cluster node, each instance of a guest operating system runs on a virtual machine, accessing virtual hard disks represented as virtual disk image files in the host operating system. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations take place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed

data.

The previous work for storage backup has extensively studied data deduplication techniques that can eliminate redundancy globally among different files from different users. Backup systems have been developed to use content fingerprints to identify duplicate content [7, **?**]. Offline deduplication is used in [**?**, **?**] to remove previously written duplicates during idle time. Several techniques have been proposed to speedup searching of duplicate fingerprints. For example, the data domain method [11] uses an in-memory Bloom filter and a prefetching cache for data chunks which may be accessed. An improvement to this work with parallelization is in [10, **?**]. The approximation techniques are studied in [2, 4, **?**] to reduce memory requirements with the tradeoff of a reduced deduplication ratio.

Additional inline deduplication techniques are studied in [5, 4, **?**]. All of the above approaches have focused on optimization of deduplication efficiency, and none of them have considered the impact of deduplication on fault tolerance in the cluster-based environment that we have considered in this paper. We will describe the motivation of using the cluster-based approach for running the backup service and then present our solution with fault isolation.

## 3. DESIGN CONSIDERATION AND OPTIONS

As discussed earlier, collocating the backup service on the existing cloud cluster avoids the extra cost to acquire a dedicated backup facility and reduces the network bandwidth consumption in transferring the un-deduplicated raw data for backup. Figure 1 illustrates the cluster architeture where each physical runs a backup service and a distribued file system (DFS) [**?**, **?**] serves a backup store for the snapshopts. The previous study shows that deduplication can compress the backup copies effectively in a 10:1 or even 15:1 rage. Therefore the portion of space in a cluster allocated for snapshots of data should not dominate the cluster storage usage.

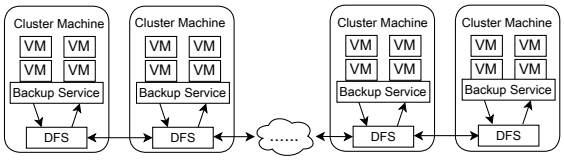



**Figure 1: Collocated VM Backup System.**

We discuss the design considerations as follows.

- *Deduplication localization, sharing minmization, and fault tolerance.*

  Because a data chunk is compared with signatures collected from other VMs during the deduplication process, only one copy of duplicates is stored in the backup storage and this artificially creates data dependency among different VM users. Content sharing via deduplication affects fault isolation since machine failures happen at daily basis 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 incrase fault isolation and resilience. Thus from the fault tolerance point of view, duplicate sharing among mutiple VMs is discourage. On the other hand, we need to seek for a tradeoff since there are a significant number of duplicates cross VMs.

- **Packaging data chunks as file system blocks.** Since the file block size in the Hadoop and GFS is uniform and large with 64MB as a default setting, the content chunk in a deduplication system is of nonuniform size with 4KB or 8KB on average. We need to build an intermediate layer that supports large snapshot writing with append operations and infrequent snapshot access. Packaging that maps data chunks to file system blocks can create data dependence among VMs since a file block can be shared even more VMs. Thus we need to consider a minimum assoocation of a fie system block to VMs in the packaging process.

Because of collocation of this snapshot service with other existing cloud services, cloud providers wish that the backup service only consumes small resources with a minimal impact to the existing cloud services. The key usage of resource for backup is memory for storing and comparing the fingerprints. We will consider the approximation techniques with less memory consumption studied in [2, 4] along with the fault isolation consideration discussed below.

A deduplication scheme compares the fingerprints of the current snapshot with its parent snapshot and also other snapshots in the entire cloud without consideration of . We call this as the VM-oblivious (VO) approach. In designing and selecting a duplication algorithm, we have considered the following options.

- **Version-based change detection**. VM snapshots can be backed up incrementally by identifying file blocks that have changed from the previous version of the snapshot [3, **?**, **?**]. Active snapshots will contain all the information needed to restore the virtual disk and when deleting a snapshot, data referenced by other snapshots are removed. While full signature comparison can deliver additional reduction [4, **?**, **?**], content change detection falls into a VM-centric approach since deduplication is localizable. We are seeking for additional optimzations to improve duplication efficency and design the assoication of data chunks with the underlying file blocks so that file block sharing among VMs is minimized.

- **sampled Index**. One alternative approach to reducing the use of memory space is to use a sampled index with prefetching, proposed by Guo and Efstathopoulos[4]. An evaluation with our test data shows that using a sampled index can achieve a high deduplication

effiency with a single machine setup. One key problem is that the algorithm is VM oblivious and not easy to be adopted for VM-centric. Another problem is to design a distributed version in deduplicating large bodies of data. To use a distributed memory version of the sampled index, every deduplicate request may access a remote machine for index lookup, which incurs a significant overhead.

Another possibiliy is to use this approach partially in our VM centric solution by indexing the most popular data chunks. For a small set of popular data chunks, the prefetching strategy used in the sampled index will not work well because the spatial locality is limited among popular data chunks. In our test data, on average the number of consecutive data chunks is 7, which is too small to be effective for index sampling.

- **Stateless Data Routing**. Another approach for duplicate comparison is to use a content-based hash partitioning algorithm called stateless data routing [**?**] that divides the deduplication work with an approximation, which is similar to Exreme Binning[2]. Again such an approach is VM-oblivious while each request does incur a network routing latency, and then additional overhead for a bloomer filter based disk lookup [**?**] or a disk index access [2].

With these considerations in mind, we study a VM-centric appraoch (called VC) for a backup service co-hosted in the existing set of machines and resource usage is friendly to the existing applications. We will first discuss and analyze the integration of the VM-centric deduplication strategies with fault isolation, and then present an architecture and implementation design with deletion support.

# 4. VM-CENTRIC SNAPSHOT DEDUPLICATION

Our VC design has the following objectives:

- 1) Localize the deduplication and data blocking within each VM as much as possible so that any failure of a file system block mainly affects the associated VM. Localizing the snapshot data deduplication within a VM improves the system by increasing data independence between different VM backups, simplifying snapshot management and statistics collection, and facilitating parallel execution of snapshot operations.

- 2) Maintain a competitive deduplication effiency while being resource-friendly to other existing cloud services. Cross-VM duplication can be desirable since there are many cloud images use widely-used software and libraries and their data blocks are duplicate. As the result, different VMs tend to backup large amount of highly similar data.

- 3) Mininimize the number of data chunks shared among VMs and add extra replicas to incrase fault resilience for such data chunks.

## 4.1 Key VC Strategies

- **Local duplicate search.** We start with the standard dirty bit approach in a coarse grain segment level and use the Xen virtual device driver to implement a feature called "changed block tracking" for the storage device and the dirty bit setting is maintained in a coarse grain level we call it a segment. In our implementation, the segment size is 2MB. Since every write for a segment will touch a dirty bit, the device driver maintains dirty bits in memory and cannot afford a small segment size.

  It should be noted that dirtybit tracking is supported or can be easily implemented in many major virtualization solution vendors. The VMWare hypervisor has an API to let external backup application know the changed areas since last backup. Xen doesn't directly support it, However, their open-source architecture allows anyone to extend the device driver, thus enabling changed block tracking. We implement dirty bit tracking this way in Alibaba's platform. The Microsoft SDK provides an API that allows external applications to monitor the VM's I/O traffic, therefore changed block tracking can be implemented externally.

  Since the best deduplication uses a nonuniform chunk size in the average of 4K or 8K [**?**], we conduct additional local inner-VM deduplication by comparing chunk signatures within a dirty segment from its parent snapshot. We load the chunk fingerprints in the corresponding segment from the parent and perform fingerprint matching for further inner-VM deduplication. The amount of memory for maintaining those fingerprints is small, as we only load one segment at a time. For example, with a 2MB segment, there are about 500 fingerprints to compare.

- **Cross-VM deduplication with popular chunks and replication support** This step accomplishes the standard global fingerprint comparison as conducted in the previous work [**?**]. One key observation is that the inner deduplication has removed many of the duplicates. There are fewer deduplication opportunities across VMs while the memory and network consumption for global comparison is more expensive. Thus our approximation is that the global fingerprint comparison only searches for the top $k$ most popular items. This dataset is called CDS (common data set). The popularity of a chunk is the number of data chunks from different VMs that are duplicates of this chunk after the inner VM deduplication. This number can be computed periodically on a weekly basis. Once the popularity of all data chunks is collected, the system only maintains the top $k$ most

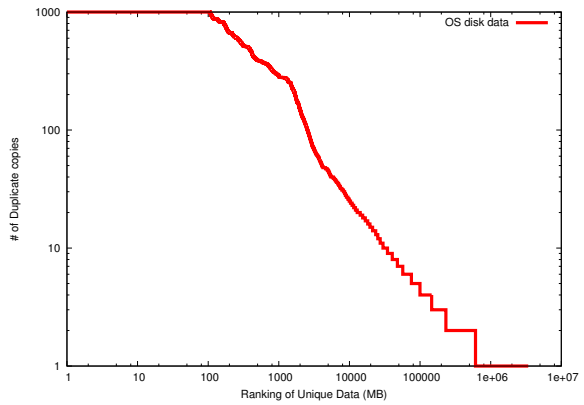



**Figure 2: Duplicate frequency versus chunk ranking in a log scale.**

popular chunk signatures in a distribued shared memory.

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

- **VM-centric file system block management.** When a chunk is not detected as a duplicate to any existing chunk, this chunk will be written to the file system. Since the backend file system typically uses a large block size such as 64MB, each VM will accumulate small local chunks. We manage this accumulation process using an append-store scheme and discuss this in details in Section 5. The system allows all machines conduct the backup in parallel in different machines, and each machine conducts the backup of one VM at a time, and thus only requires a write buffer for one VM.

  CDS chunks are stored in a separate append-store instance. In this way, each file block for non-CDS chunks is associated with one VM and does not contain any CDS chunks.

## 4.2 Impact on deduplication efficiency

We analyze how the choice of value $k$ impacts the deduplication efficiency. The analysis is based on the charactristics of the VM snapshot traces studied from Alibaba's production user data. Our previous study shows that the popularity of data chunks after inner VM deduplication follows a Zipf-like distribution[**?**] and its exponent $\alpha$ is ranged between 0.65 and 0.7. [**?**]. Table 1 lists parameters used in this analysis.

*[Need to find a place to put these numbers in: Total number of chunks in 350 snapshots: 1,546,635,485. Total number of chunks after localized dedup: 283,121,924. Total number of unique chunks: 87,692,682.]*

As summarized in Table 1, let $c$ be the total number of data chunks. $c_u$ be the total number of fingerprints in the

| | |
|---|---|
| $c$ | the total amount of data chunks |
| $c_u$ | the total amount of unique fingerprints after inner VM deduplication |
| $f_i$ | the frequency for the $i$th most popular fingerprint |
| $\delta$ | the percentage of duplicates detected in inner VM deduplication |
| $\sigma$ | the number of unique non-CDS chunks over the number of the CDS chunks. |
| $p$ | the number of machines in the cluster |
| $D$ | the amount of unique data on each machine |
| $C$ | the average data chunk size. Our setting is 4K. |
| $s$ | the average size of file system blocks in the distributed file system. The default is 64MB. |
| $m$ | memory size on each node used by VC |
| $E$ | the size of an popular data index entry |
| $N_1$ | the average number of non-CDS file system blocks in a VM |
| $N_2$ | the average number of CDS file system blocks in a VM |
| $N_o$ | the average number of file system blocks in a VM for VO |

**Table 1: Modeling parameters and symbols.**

global index after complete deduplication, and $f_i$ be the frequency for the $i$th most popular fingerprint. By Zipf-like distribution, $f_i = \frac{f_1}{i^\alpha}$. Since $\sum_{i=1}^{c_u} f_i = c$,

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

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

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

The $k$ most popular fingerprints can cover the following number of chunks after inner VM deduplication:

$$f_1 \sum_{i=1}^{k} \frac{1}{i^\alpha} \approx f_1 \int_1^k \frac{1}{x^\alpha} dx \approx f_1 \frac{k^{1-\alpha}}{1-\alpha}.$$

Deduplication efficiency of VC using top $k$ popular chunks is the percentage of duplicates that can be detected:

$$\frac{c(1-\delta) + f_1 \frac{k^{1-\alpha}}{1-\alpha}}{c(1-\delta) + \delta c - c_u} = \frac{(1-\delta) + \delta(\frac{k}{c_u})^{1-\alpha}}{1 - \frac{c_u}{c}}. \tag{2}$$

Let $p$ be the number of physical machines in the cluster, $m$ be the memory on each node used by the popular index, $E$ be the size of an index entry, $D$ be the amount of unique data on each physical machine, and $C$ be the average chunks size. We store the popular index using a distributed shared memory hashtable such as MemCacheD. Then $k$ and $c_u$ can be expressed as: $k = p * m/E$, and $c_u = p * D/E$.

The overall deduplication efficiency of VC is

$$\frac{(1-\delta)+\delta(\frac{m*C}{D*E})^{1-\alpha}}{1-\frac{c_u}{c}}.$$

where $(\frac{m*C}{D*E})^{1-\alpha}$ represents the percentage of the remaining chunks detected as duplicates after inner VM deduplication. Figure 3 shows measured CDS coverage in our 35 VM dataset. You can see from the graph that the coverage actually increases as the datasize increases, which indicates that $\alpha$ increases with the datasize (a fixed-$\alpha$ model would predict constant coverage). This makes the CDS even more effective, as the coverage of the CDS increases as more data is added.

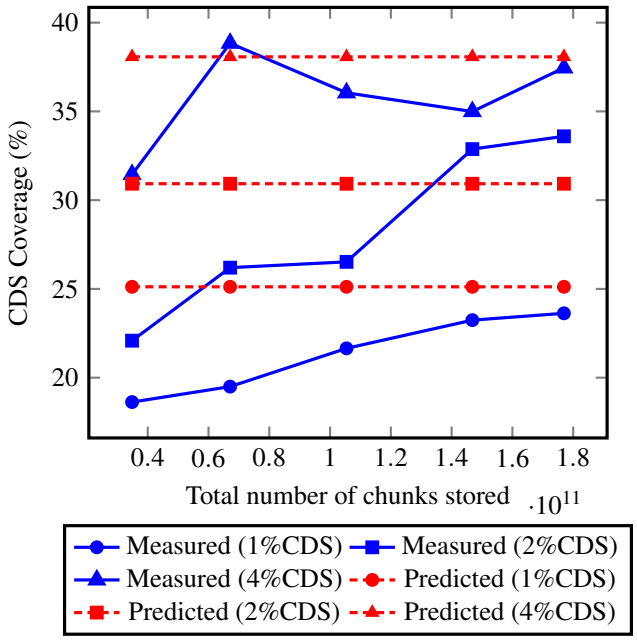


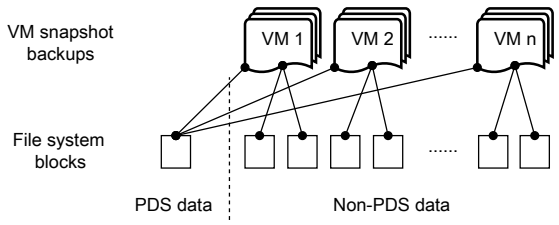

**Figure 3: fixed-alpha predicted vs. actual CDS coverage as data size increases.**

As illustrated in Figure 5*I am not sure where this reference should point, but I don't think it is pointing to the right figure*, when the number of machines at each cluster increases, the number of total VMs increases. Then $k$ increases since more memory is available to host the popular chunks index. But for each physical machine, the number of VMs remains the same, and thus $D$ is a constant. Then the overall deduplication efficiency of VC remains a constant.

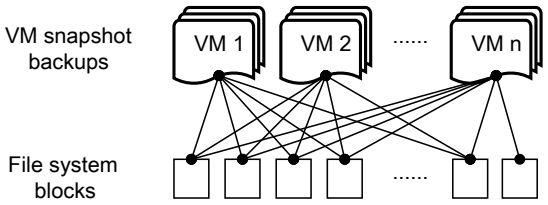
## 4.3 Storage Space and Impact on Fault Tolerance

The replication degree of the backup storage is $r$ for regular file blocks and $r = 3$ is a typical setting in the distributed file system [**?**, **?**]. In the VC approach, a special replica degree $r_c$ used for CDS blocks where $r_c > r$. Notice that the ratio of non CSD data size vs CDS data size for each VM is

$$\sigma = \frac{c*\delta(1-(\frac{k}{c_u})^{1-\alpha})}{k}.$$




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




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

**Figure 4: Difference of sharing data under VO and VC approaches**

Thus storage cost for VO with full deduplication is $c_u * r$ and for VC, it is

$$k * r_c + k * \sigma * r.$$

In our experiment with Alibaba data, the ratio $\sigma$ is 162. Thus allocation of extra replicas for CDS only introduces a small amount of extra space cost. Figure **??** shows the storage cost ratio of VC and VO when $r$=3, and $r_c$ varies from 3 to 10. The result shows that the storage cost for adding extra replication for CDS is insignificant.

Next we compare the impact of losing $d$ machines to the the VC and VO approaches.

In characterizing the reliability of VM backups in our model, we consider the likely hood that a file system block fails, given some number of storage machine failures. Every time a filesystem block fails, we say that we have lost data for that virtual machine, so it is no longer available. In reality it may be only one snapshot that is affected, but it is the user who must decide which snapshots are important, so we consider the worst case. We use filesystem blocks rather than a deduplication data chunk as our unit of failure because the DFS keeps filesystem blocks as its base unit of storage.

To compute the probablity of losing snapshots of a virtual machine, we estimate the number of file system blocks per VM in each apporach. We can build a bipartite graph representing the association from unique file system blocks to their corresponding VMs. An association edge is drawn from a file block to a VM if this file is used by this VM. For VC, each VM has an averge number of $N_1$ file system blocks for non-CDS data. It also refers an average of $N_2$ file system blocks for CDS data For VO, each VM has an average of $N_o$ file system blocks and let $V_o$ be the averge number of VMs shared by each file system block. Figure **??** illustates the

bipartite association.

In VC, each non-CDS file system block is associated with one VM while CDS file system blocks are shared among VMs (at most $V$ VMs). Thus,

$$V * N_1 * s = pD\frac{\delta}{\delta+1} \text{ and } V * N_2 * s \leq pD\frac{1}{\delta+1} * V.$$

For the VO approach,

$$V * N_o * s = pDV_o.$$

Then

$$N_1 = \frac{pD\delta}{Vs(\delta+1)},\ N_2 \leq \frac{pD}{s(\delta+1)},\ \text{and } N_o = \frac{pDV_o}{sV}.$$

Since each file block (with default size $s = 64MB$) contains many chunks (on average 4KB), each file block contains the hot low-level chunks shared by many VMs, and it also contains rare chunks which are not shared. Figure **??** shows the number of VMs shared by each file block. In our experiment, we find that $V_o \approx 0.2V$ when backing up VMs one by one. We can observe that $N_1 + N_2 << N_o$. If the backup for multiple VMs is conducted concurrently, there would be more VMs shared by each file block on average.

Figure 5 shows the average number of VMs sharing a block in the global index as more VMs are added in VO. The first 15 VMs are all windows machines, so that explains the initial higher values, but for the most part the average number of links is between 1.5 and 2. Figure 6 shows the average number of VMs sharing a filesystem block (FSB) in the global index, but only for the 15 windows machines (the last machine has a much larger hard-drive, and therefore more unique data, which explains the drop-off in the last datapoint). This is a reasonable assumption, that a cluster will have one OS dominating, and for our small dataset the number of links to an FSB increases as more VMs of the same OS are added. You can see in the plot however that even though a FSB contains 16K chunks, the average number of links to an FSB is only several times the number of average links to a chunk. This is because even in the VO global deduplication model, as long as writes are batched a single FSB will tend to contain chunks from only a small number of VMs. Below we show the significant impact the number of links has on VM backup reliability.

The snapshot availability of is a VM is the likelyhood that there is no data loss for all its file blocks. With replication degree $r$, the likelyhood of a file block block is the probability that all of its replicas appear in $d$ failed machines. Namely, $\binom{d}{r}/\binom{p}{r}$.

When there are $r \leq d < r_c$ machines failed and then there is no CDS data loss, the snapshot avaiability of a VM in the VC approach is and is

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

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

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

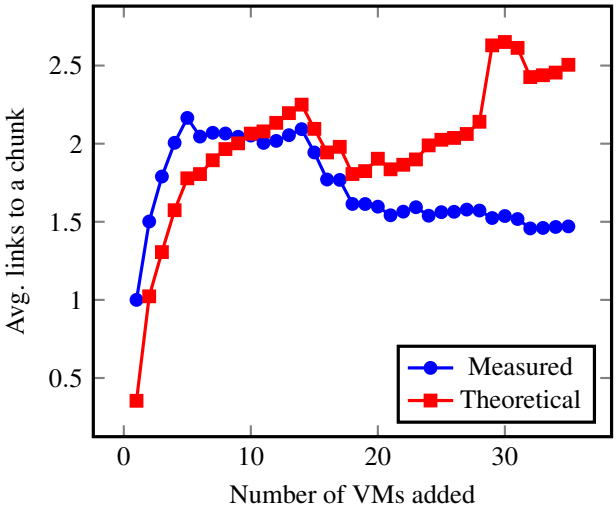



**Figure 5: Average number of VMs sharing a data chunk in the global index**

The snapshot availability of a VM in the VO approach is

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

Figure 8 shows the reliability of VM backups as storage nodes fail for different numbers of links to average blocks in the index. We chose 5 and 11 for the VO and VC block links counts because those are what we measured in our dataset, and chose higher numbers to account for much larger datasets. As you can see from the graph, even small changes in the number of links to VO blocks has a significant impact on reliability, while even a 100 times increase in the VC block link counts only slightly affects VC reliability. The key factor placed is that $N_1 + N_2 << N_o$, caused by the fact that the VM-centric approach localizes deduplication and packs data blocks for one VM as much as possible. The extra replicaton for CDS blocks also signficantly increaes the snapshot availability even when a CDS file block is shared by every VM.

Figure 9 shows the advantages of increasing the replication factor for CDS blocks. the numbers in the graph are only meant to be relative, as we assume higher CDS block links than we have measured in our dataset (to account for a much larger body of data). It is easy to see though that increasing the replication of just the CDS blocks (which are the most popular blocks) can have a positive impact on the overall reliablity of the VM backups. These figures together show the advantages of the VM-centric model, and the advantages that separating the replication factor for popular blocks can have on reliability.

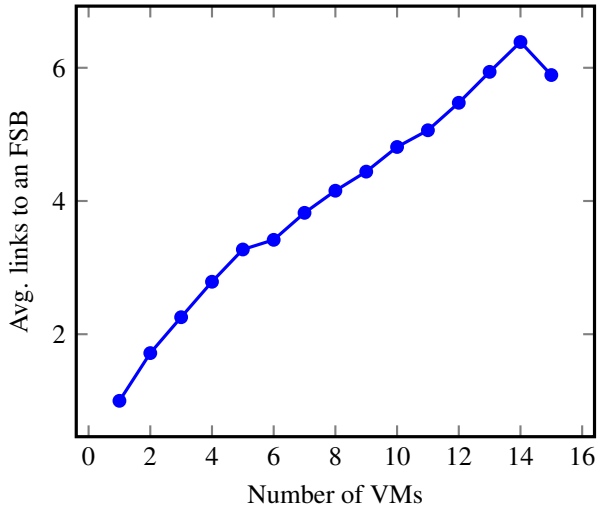



**Figure 6: Measured Average number of VMs sharing a 64MB Filesystem Block in the global index (for VO)**

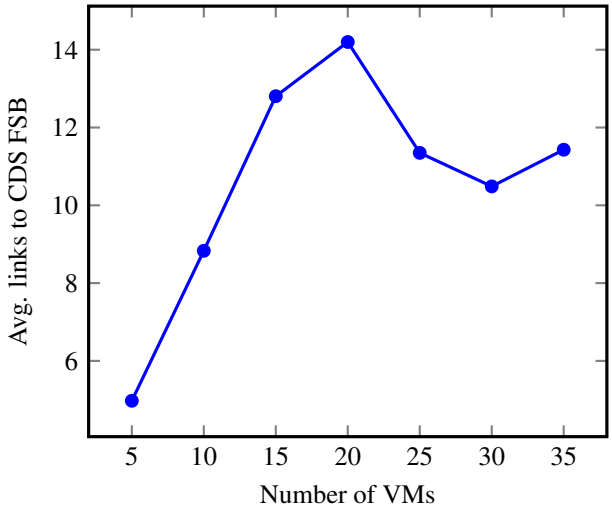



**Figure 7: Measured Average number of VMs sharing a 64MB CDS Filesystem Block for (VC with 5% CDS)**

# 5. ARCHITECTURE AND IMPLEMENTATION DETAILS

Our system runs on a cluster of Linux machines with Xen-based VMs. A distributed file system (DFS) manages the physical disk storage and we use QFS [**?**]. All data needed for VM services, such as virtual disk images used by runtime VMs, and snapshot data for backup purposes, reside in this distributed file system. One physical node hosts tens of VMs, each of which access its virtual machine disk image through the virtual block device driver (called TapDisk[9] in Xen).

## 5.1 Components of a cluster node

As depicted in Figure **??**, there are four key service components running on each cluster node for supporting backup and deduplication: 1) a virtual block device driver, 2) a snapshot deduplication component, 3) an append store client to store and access snapshot data, and 4) a CDS client to support CDS index access. We will further discuss our deduplication scheme in Section **??**.

We use the virtual device driver in Xen that employs a bitmap to track the changes that have been made to virtual disk. When the VM issue a disk write, the bits corresponding to the segments that covers the modified disk region are set, thus letting snapshot deduplication component knows these segments must be checked during snapshot backup. After the snapshot backup is finished, snapshot deduplication component acknowledges the driver to resume the dirty-bits map to a clean state. Every bit in the bitmap represents a fix-sized (2MB) region called *segment*, indicates whether the segment is modified since last backup. Hence we could treat segment as the basic unit in snapshot backup similar to file in normal backup: a snapshot could share a segment with previous snapshot it is not changed. As a standard practice, segments are further divided into variable-sized chunks (average 4KB) using content-based chunking algorithm, which brings the opportunity of fine-grained deduplication by allowing data sharing between segments.

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 a snapshot recipes or a segment recipe, the data structures includes reference pointers to the actual data location.

## 5.2 A VM-centric snapshot store for backup data

We build a snapshot storage on the top of a distribued file system. Following the VM-centric idea for the purpose of fault isolation, each VM has its own snapshot store, contaning new data chunks which are considered to be non-duplicates. There is also a special store containing all PDS chunks shared amony different VMs. The replication degree of file blocks of snapshot stores in the underlying file system Extra replication of this store is added as discussed in Section **??**. As shown in Fig.12, we explain the data structure of snapshot stores as follows.

- The PDS snapshot contains a set of commonly used data chunks and is accessed by its offset and size in the corresponding DSF file. The PDS index uses the offset and size as a reference in its index structure.
- Each non-PDS snapshot store is divided into a set of containers and each of them is approximately 1G bytes. The reason we dividie the snapshot into containers is to simplify the compact process conducted perioidcally. There are chunks deleted without any reference from other snapshots and to reclaim the space of these useless chunks, a compaction routine can work on one

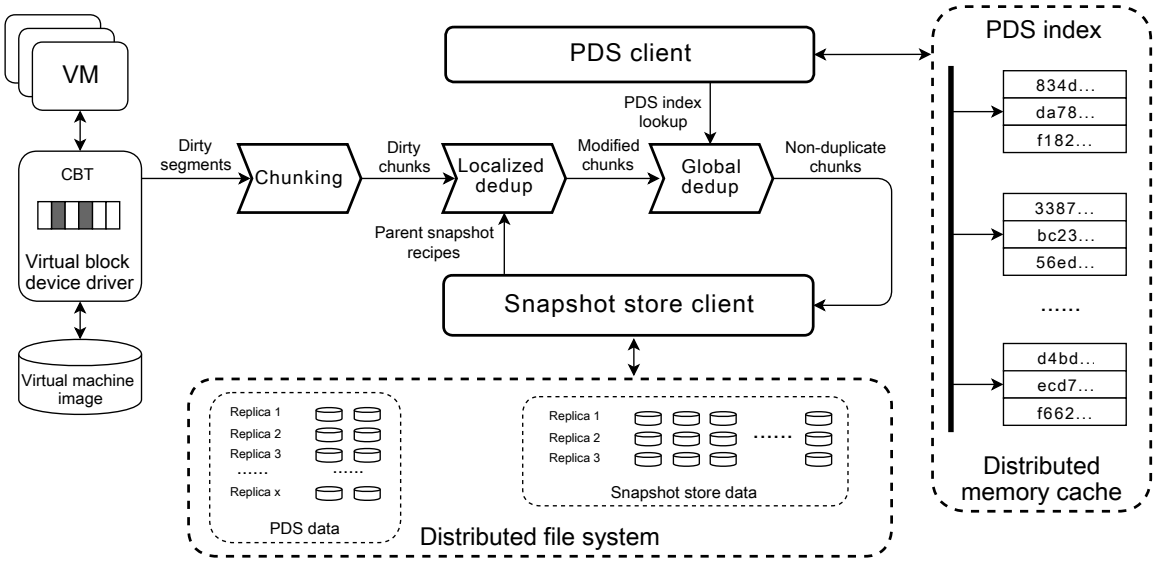



**Figure 10: System Architecture**

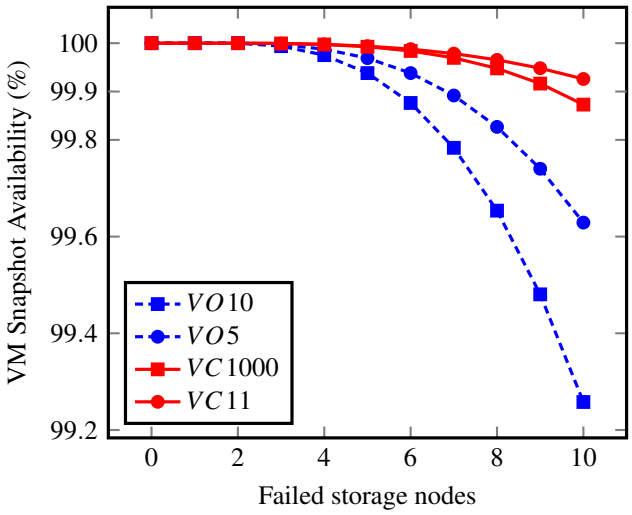



**Figure 8: Availability of VM backups as nodes fail for VO and VC models for varying average block link counts (i.e. average number of VMs that use a block)**

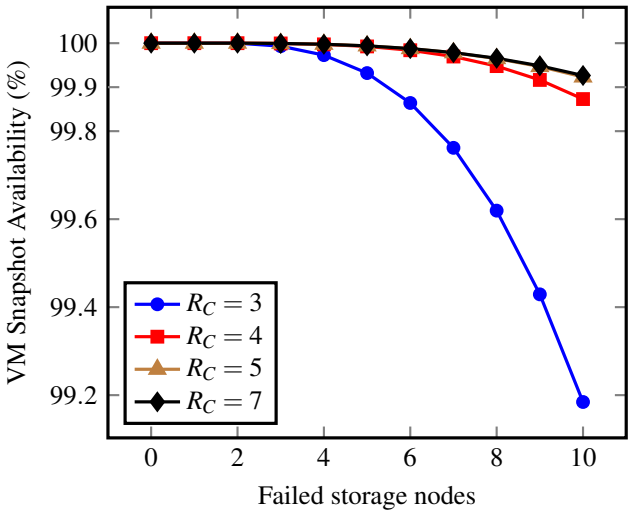



**Figure 9: Availability of VM backups as nodes fail in the VC model for different CDS Replication factors (Non-CDS replication fixed at 3, and average CDS block links set to 1000)**

container at a time and copy used data chunks to another container.

– Each container is divided into a set of chunk data groups. Each group is composed of a set of data chunks and is the basic unit for the snapshot in data access and retrieval. In writing a chunk group, data chunks in this group are compressed together and then stored. When accessing a particular chunk, its chunk group is retrieved from the disk storage and uncompressed. Given the high spatial locality in snapshot data accessing [**?**, **?**], retrieval of data chunks by group naturally fits in the prefetching scheme to speedup snapshot access. A typical chunk group contains 100 to 1000 chunks, with an average size of 200-600KB. Chunk groupping also reduces the container index size as we discuss below. Given the average chunk size 4KB, the index size for a 1GB container reduces from 10MB to 100KB when the chunk group size is 100.

– Each data container is represented by three data files in the DFS: 1) the container data file holds the actual content of data chunks, 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 saviing all the deletion requests within the container. A VM snapshot store typically has a small number of containers because each container is fairly large with an average size of 1GB bytes, it maintains a limited number of snapshots (e.g. 10 in the Alaibaba case), and new snapshot data chunks can be effectively compressed in chunk groups in additon to deduplciation.

– We maintain a chunk counter and assign the current number as a chunk ID (called CID) within this container as a reference of a new chunk added to a container. Since data chunks are always appended to the snapshot store, a CID is monotonically increasing. A data chunk reference stored in the index of snapshot reeipts is composed of two parts: a container ID (2 bytes) and CID (6 bytes). Once a snapshop is to be accessed access, the receipt in this snapshot will point to either a data chunk in the PDS or a reference number with a container ID and CID. With a container ID, the corresponding container index file is accessed and the chunk group is identified using this CID. Once this chunk group is loaded to memory, its header contains the exact offset of the corresponding chunk and the content is then accessed from the memory buffer.

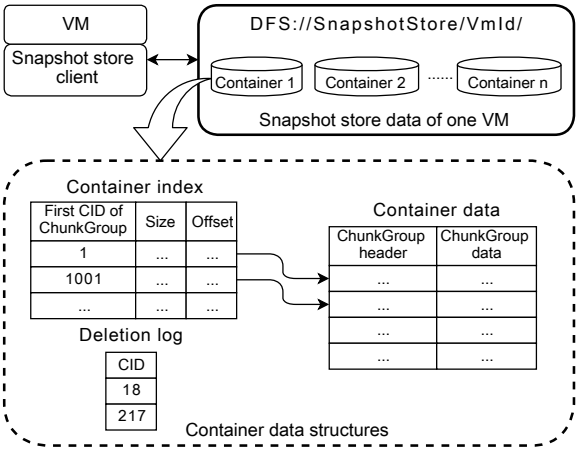



**Figure 11: Data structure of VM snapshot stores.**

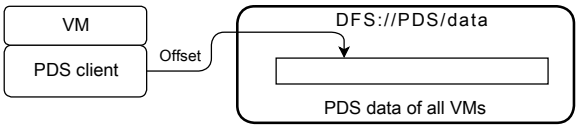



**Figure 12: System architecture for PDS data store.**

The snapshot store supports three API calls.

- *Put(data)* places data chunk into the snapshot store and returns a reference to be stored in the recipe metadata of a snapshot.

  The write requests to append a data chunk to a VM store are accumulated in the client side. When the number reaches the group size $g$, the snapshot store client compresses the accmnulated chunk group, adds a chunk group index in the begining, and then append the header and data to the corresponding VM file, Then a new container index entry is also created and is written the corresponding container index file.

  The writing of PDS data chunks is conducted periodically when there is a new PDS caclulation. Since the PDS dataset is small, a new PDS file is created during the periodical update.

- *Get(reference)*. The fetch operation for the PDS data chunk is straightforward since each refererence contains the offset and size within the PDS underlying file. We also maintain a small data cache for the PDS data service to speedup the process.

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

- *Delete(reference)*. A data chunk can be deleted when a snapshot expires or gets deleted explictly by a user. We will discuss the snapshot deletion shortly in the following subsection. When deletion requests issued for a specific container, those requests are logged into the container's deletion log file initially and thus a lazy delection strategy is exercised. Once CIDs appear in the deletion log, they will not be referenced by any future snapshot and can be safely deleted when needed. Periodically, the sanpshot store picks those containers with an excessive number of deletion requests to compact and reclaim the corrwsponding 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 scan the old container, copying all the chunks that are not found in the deletion log to the new container, creating new chunk groups and indices. However, every chunk's CID is plainly copied rather than re-generated. This does not affect the sorted order of CIDs in new container, but just leaving holes in CID values. As the result, all data references stored in upper level recipes are unaffected, and the data reading process is as efficient as before.

### 5.3 Fast VM-centric Snapshot Deletion with a Two-Phase Apprxoimation

In a busy VM cluster, snapshot deletions are as frequent as snapshot creations. The VM-centric snapshot storage design simplifies the deletion process since we need to look for the useless chunks within each VM when deleting a snapshot. The PDS data chunks are commonly shared all VMs and we donot consider their referene counting during snapshot deletion. The selection of PDS data chunks is updated periodically independent of snapshot deletion process.

While we can use the standand mark-sweep technique [**?**], it takes time to conduct this process everytime there is a snapshot deletion request. In the case of Alibaba, snapshot backup is conducted automatically and there are about 10 snapshot stored for every user. Then when there is a new snapshort created every day, there will be a snapshort expired everyday to maintain a balanced storage use. Given a large number of snapshot deletion requests, we are seeking a fast solution with a very low resource usage to delete snapshorts with an approximation.

With this in mind, we develop a two-phase *approximated* deletion strategy to trade deletion accuracy for speed and resource usage. Our method sacrifaces a tiny percentage of storage leakage to effectively identify unused blocks in $O(n)$ speed, with $n$ being the logical amount of chunks to be deleted.

**Snapshot Summaries.** with this in mind, we compute a bloom-filter fingerprint summary for all non-PDS chunks used by a snapshot during its creation time.

**Approximate Deletion Phase-1** The goal of approximate deletion phase-1 to fast identify unused blocks which are no longer referenced by other snapshots after a snapshot deletion. Instead of scanning the entire append store indices, we merge the type-1 summaries of all valid snapshots. Since each VM has uniform bloom filter parameters to create snapshot summaries, such merged summeries give us a compact representation of all block fingerprints that are still in use. Thus by the property of bloom filter, if a fingerprint is not found in merged summaries, we are certain that block is no longer used by any valid snapshot, it would be then added to append store's deletion log. However, there is a small false-positive probility which would indetify unused data as in use, resulting in temporary storage leakage.

**Approximate Deletion Phase-2** We designed the second phase of approxmite deletion to solve the temporary storage leakage problem mentioned above. In this phase we scan the entire append store indices, using the merged type-2 snapshot summaries to check if any of them are not referenced by existing snapshots. We cannot simply repeat the phase 1 multiple times to reduce temporary storage leakage, because:

1. After several runs of phase-1, it is proven that the merged type-1 summaries cannot sieve remaining unused blocks, due to the false-positive property of bloom filter.
2. The recipes of deleted snapshots have been removed from the system, thus we are not able to obtain the deleted block fingerprints from any metadata, the only way to discover them is to scan the append store indices.

**Discussion** For one VM's snapshots, let $L_{temp}$ be the amount of temporary storage leakage, $D_{del}$ be the average amount of data that should be physically deleted in one snapshot deletion, $P_1$ be the false-positive rate of merged bloom filter type-1, $N_1$ be the number of runs of deletion phase-1 during the gap of two deletion phase-2 operations, then we have:

$$L_{temp} = N_1 * P_1 * D_{del} \tag{3}$$

If we let approxmite deletion phase-2 be triggered when $L_{temp}/D_{del}$ is accmulated to exceed certain threshold $t$, then:

$$\frac{L_{temp}}{D_{del}} = N_1 * P_1 > t \Rightarrow N_1 > \frac{t}{P_1} \tag{4}$$

Let $P_2$ be the false-positive rate of merged snapshot type-2 summaries, $N_2$ be the number of runs of deletion phase-2, $L_{perm}$ be the permanent storage leakage resulting from the approxmite deletion phase-2, it can be calculated as follows:

$$L_{perm} = N_2 * P_1 * P_2 * D_{del} \tag{5}$$

For example, letting $P_1 = 0.01$ and $t = 1.0$, we would expect deletion phase-2 be triggered once for every $T/P_{bl} = 100$ runs of deletion phase-1. On a node that hosts 20 VMs and each VM deletes one snapshot per day, there would be only 1 deletion phase-2 scheduled for every 5 days, which is sufficiently small to prevent the heavy I/O workload of deletion phase-2 from disturbing normal VM opeations.

## 6. IMPLEMENTATION

### 6.1 Snapshot Backup

### 6.2 Snapshot Read

### 6.3 Snapshot Deletion

The following steps would take place during an approximate deletion:

1. **Creating bloom filter** Scan all the living snapshot recipess and their segment recipes, for every reference pointing to append store, add it to the bloom filter.
2. **Check existance** For every data reference in the deleted snapshot recipe and its segment recipes, check the existance of that data reference in bloom filter. If not found, it is safe to delete that piece of data from append store because no living snapshots has referenced it.

The overall time of running a approximate deletion for one snapshot deletion would be scanning all the living snapshots and deleted snapshots, since operations on the in-memory

bloom filter can be done in parallel and is much faster than loading recipes from DFS:

$$T = (N_{SS} + 1) * T_{scan\_recipes} \quad (6)$$

Using the example and analysis in previous section, this approximate deletion can be done in 5 minutes. Memory usage of the bloom filter depends on its false-positive probility $P_{bl}$, when set $P_{bl}$ to 0.01, the memory footprint of approximate deletion is about 15 MB.

# 7. EVALUATION

We have implemented and evaluated a prototype of our VO scheme on a Linux cluster of 8-core AMD FX-8120 at 3.1 GHz with 16 GB RAM. Our implementation is based on Alibaba's Xen cloud platform [1, **?**]. Each machine is equiped with a a distributed file system (QFS) running in the cluster manages six 3TB disks with default replication degree set to 3. The CDS replication is set to 5. Objectives of our evaluation are: 1) Study the deduplication efficiency of the VC approach and compare with an alternative design. 2) Evaluate the backup throughput performance VC for a large number of VMs. 3) Examine the impacts of VC for fault isolation.

## 7.1 Settings

We have performed a trace-driven study using a 1323 VM dataset collected from 100 Alibaba Aliyun's cloud nodes [**?**]. The production environment tested has about 1000 machines with 25 VMs on each machine. For each VM, the system keeps 10 automatically-backed snapshots in the storage while a user may instruct extra snapshots to be saved. The backup of VM snapshots is completed within a few hours every night. Based on our study of its production data, each VM has about 40GB of storage data usage on average including OS and user data disk. All data are divided into 2 MB fix-sized segments and each segment is divided into variable-sized content chunks [6, 8] with an average size of 4KB. The signature for variable-sized blocks is computed using their SHA-1 hash. Popularity of data blocks are collected through global counting and the top 1% will fall into CDS, as discussed in Section **??**.

Since it's impossible to perform large scale analysis without affecting the VM performance, we sampled two data sets from real user VMs to measure the effectiveness of our deduplication scheme. Dataset1 is used study the detail impact of 3-level deduplication process, it compose of 35 VMs from 7 popular OSes: Debian, Ubuntu, Redhat, CentOS, Win2003 32bit, win2003 64 bit and win2008 64 bit. For each OS, 5 VMs are chosen, and every VM come with 10 full snapshots of it OS and data disk. The overall data size for this 700 full snapshots is 17.6 TB.

Dataset2 contains the first snapshots of 1323 VMs' data disks from a small cluster with 100 nodes. Since inner-VM deduplication is not involved in the first snapshot, this data set helps us to study the CDS deduplication against user-related data. The overall size of dataset2 is 23.5 TB.

## 7.2 Deduplication Efficiency

Figure **??** shows the overall impact of 3-level deduplication on dataset1. The X axis shows the overall impact in (a), impact on OS disks in (b), and impact on data disks in (c). Each bar in the Y axis shows the data size after deduplication divided by the original data size. Level-1 elimination can reduce the data size to about 23% of original data, namely it delivers close 77% reduction. Level-2 elimination is applied to data that could pass level-1, it reduces the size further to about 18.5% of original size, namely it delivers additional 4.5% reduction. Level-3 elimination together with level 1 and 2 reduces the size further to 8% of original size, namely it delivers additional 10.5% reduction. Level 2 elimination is more visible in OS disk than data disk, because data change frequency is really small when we sample last 10 snapshots of each user in 10 days. Nevertheless, the overall impact of level 2 is still significant. A 4.5% of reduction from the original data represents about 450TB space saving for a 1000-node cluster.

Figure **??** shows the impact of different levels of deduplication for different OS releases. In this experiment, we tag each block in 350 OS disk snapshots from dataset1 as "new" if this block cannot be deduplicated by our scheme and thus has to be written to the snapshot store; "CDS" if this block can be found in CDS; "Parent segment" if this block is marked unchanged in parent's segment recipe. "Parent block" if this block is marked unchanged in parent's block recipe. With this tagging, we compute the percentage of deduplication accomplished by each level. As we can see from Figure **??**, level-1 deduplication accomplishes a large percentage of elimination, this is because the time interval between two snapshots in our dataset is quite short and the Aliyun cloud service makes a snapshot everyday for each VM. On the other hand, CDS still finds lots of duplicates that inner VM deduplication can't find, contributing about 10% of reduction on average.

It is noticeable that level-1 deduplication doesn't work well for CentOS, a significant percentage of data is not eliminated until they reach level-3. It shows that even user upgrade his VM system heavily and frequently such that data locality is totally lost, those OS-related data can still be identified at level-3.

In general we see a stable data reduction ratio for all OS varieties, ranging from 92% to 97%, that means the storage cost of 10 full snapshots combined is still smaller than the original disk size. And compare to today's widely used copy-on-write snapshot technique, which is similar to our level-1 deduplication, our solution cut the snapshot storage cost by 64%.

# 8. EXPERIMENTS

Our main test-bed is an cluster of 6 machines, each of which is equiped with a 8-core AMD FX-8120 at 3.1 GHz

| PDS replication degree | Total space (GB) |
|---|---|
| 3 | 3065 |
| 4 | 3072 |
| 5 | 3079 |
| 6 | 3086 |
| 7 | 3093 |

**Table 2: Storage space cost under different PDS replication degree**

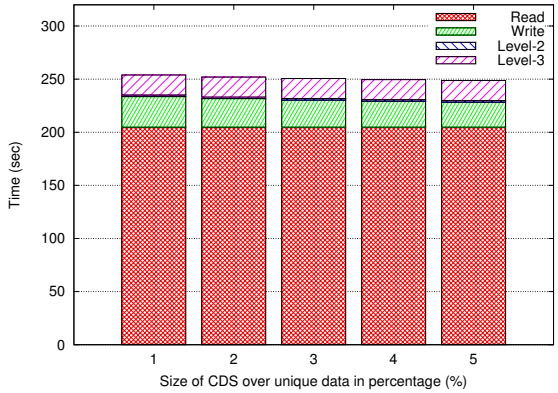



**Figure 13: Backup times for varying CDS sizes**

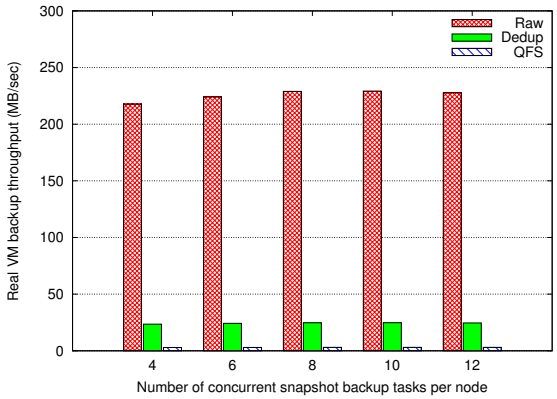
with 16 GB RAM, running Linux. A distributed file system (QFS) runs in the cluster manages six 3TB disks with default replication degree set to 3. Our data set consists of 350 virtual machine image snapshots taken from Alibaba.com's public VM cloud in China. We select 35 VMs from the most popular 7 OSes: Debian, Ubuntu, Redhat, CentOS, Win2003 32bit, win2003 64 bit and win2008 64 bit. For each OS, 5 VMs are chosen, and every VM come with 10 full snapshots.

## 8.1 Replication cost

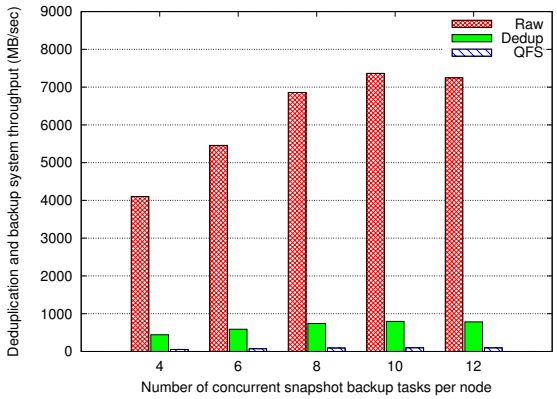
## 8.2 System Performance

**Single VM Backup** We start the evaluation of our system by examine the normal warmed-up backup scenario - taking a snapshot of a VM which already has old backups exist. To simulate this scenario, we pick one 40GB VM from the data set, make an initial snapshot backup, then evaulate the system performance on the second snapshot backup. We repeat such procedure under different CDS memory usage settings, the results of backup time break down are shown in Figure 13.

It's easy to see that the time of I/O dominates the backup process, our deduplication procedure only takes a tiny fraction of the total backup time. Larger memory permits us to use larger CDS index to cover more of global index, as a result the amount of data to be writen is reduced and so do the writing time. Inside the deduplication procedure, the time costs of level-1 is plainly zero, and the costs of level-2 and level-3 are almost identical under different settings because




(a) Real VM backup performance




(b) Deduplication and storage system performance

**Figure 14: Throughput per-node with concurrent snapshot backup tasks**

the number of fingerprints to process are just the same.

**Parallel VM Backup** We then evaluate the performance of writing snapshots in parallel. To begin, on each node we let 4 VMs writing snapshot concurrently, and gradually increase number of VMs to 12 to saturate our system capability. We observed the per-node throughput peaked at 2700 MB/s when writing 10 VM snapshots in parallel, which is far beyond our QFS file system capability. The reason behind it is our efficient deduplication architecture and compression greatly reduce the amount of data that need to be writen to the file system. The main bottleneck here is our QFS only manages one disk per node, making it inefficient to utilize the benefits of parallel disk access. We expect our achitecture can perform even better in production cluster which normally has more than ten disks on each node.

## 8.3 Comparison with Other Approaches

### *8.3.1 Stateless Routing with Binning*

We compare our system with Stateless Routing with Binning (SRB) in terms of both backup time and deduplication efficiency. SRB is the same as the stateless routing algo-

rithm given in the Extreme Binning[**?**] paper. the deduplication efficiency is compared in Figure 15, showing the deduplication efficiency (percent of duplicate chunks which are deduped) of SRB and VC with several different CDS sizes. From the graph you can see that VC provides similar or better deduplication efficiency in all cases. In Figure 16 you can see simulated backup times using actual throughput numbers from our test backup cluster, and VC outperforms SRB mainly due to significantly fewer index lookups (since SRB uses hash-partitioning, all lookups are essentially random and caching will be largely innefective). *Should we use one of the two bar charts, or the graph? I am thinking the graph, but I'll leave the two bar charts in the paper until we decide? If we use the bar charts some text will have to be changed. We could also add each of the CDS sizes to the bar chart and have 4 bars instead of 2.*

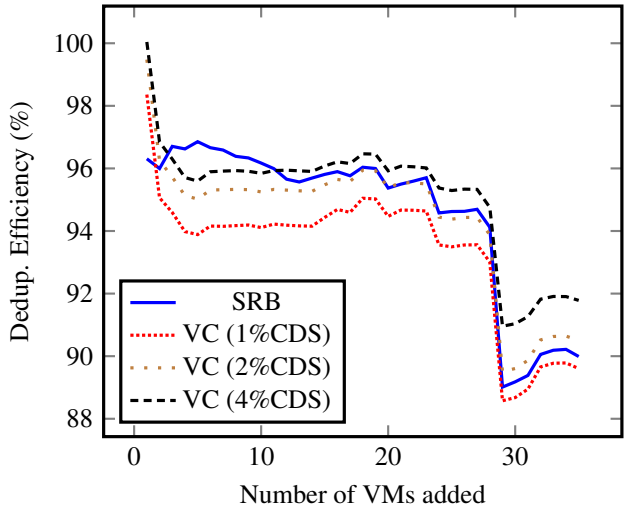



**Figure 15: Deduplication Efficiency** ($efficiency = \frac{d-du}{d-du_{complete}}$) **comparison between Stateless Routing with Binning and VC.**

## 9. CONCLUSION

In this paper we propose a VM-centric deduplication scheme for snapshot backup in VM cloud for maximizing fault isolation and tolerance. Inner-VM deduplication localizes backup data dependency and exposes more parallelism while cross-VM deduplication with a small common data set effectively covers a large amount of duplicated data. The scheme organizes the write of small data chunks into large file system blocks so that each underlying file block is associated with one VM for most of cases. Our solution accomplishes the majority of potential global deduplication saving while still meets stringent cloud resources requirement. Our analysis shows that this VM centric scheme can provide better fault tolerance while using a small amount of computing and storage resource.

[Talk about more what we learn from Evaluation] Evaluation using real user's VM data shows our solution can accomplish 75% of what complete global deduplication can do. Compare to today's widely-used snapshot technique, our scheme reduces almost two-third of snapshot storage cost. Finally, our scheme uses a very small amount of memory on each node, and leaves room for additional optimization we are further studying.

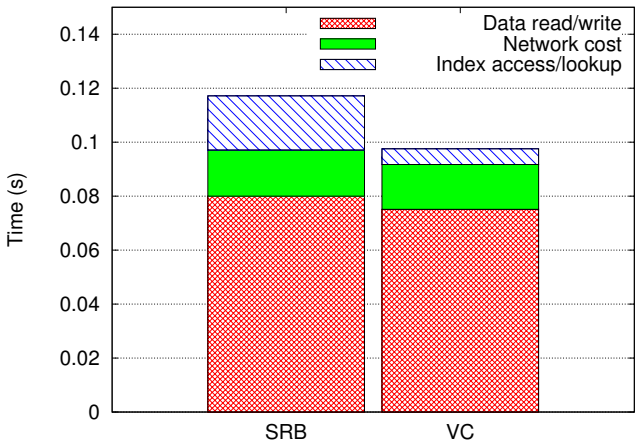



**Figure 16: Time to backup a dirty segment under SRB and VC approach**

## 10. REFERENCES


[1] Aliyun Inc. 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 *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pages 1–9, 2009.

[3] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in SAN cluster file systems. page 8, June 2009.

[4] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. page 25, June 2011.

[5] 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*, pages 111–123, 2009.

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

[7] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[8] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research

in Computing Technology, Harvard University, 1981.

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

[10] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: A scalable high-throughput exact deduplication approach for network backup services. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–14, May 2010.

[11] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.