UNIVERSITY OF CALIFORNIA
Santa Barbara

Collocated Data Deduplication for Virtual Machine Backup in the Cloud

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Wei Zhang

Committee in Charge:

Professor Tao Yang, Chair

Professor Jianwen Su

Professor Rich Wolski

September 2014

The Dissertation of
Wei Zhang is approved:

Professor Jianwen Su

Professor Rich Wolski

Professor Tao Yang, Committee Chairperson

January 2014

Collocated Data Deduplication for Virtual Machine Backup in the Cloud

by

Wei Zhang

To my family.

Acknowledgements

First of all, I want to take this opportunity to thank my advisor Professor Tao Yang for his invaluable advice and instructions to me on selecting interesting and challenging research topics, identifying specific problems for each research phase, as well as preparing for my future career. I would also like to thank my committee members, Professor Rich Wolski and Professor Jianwen Su, for their pertinent and helpful instructions on my research work.

I also want to thank members and ex-members of my research group, Michael Agun, Gautham Narayanasamy, Prakash Chandrasekaran, Xiaofei Du, and Hong Tang, for their incessant support in the forms of technical discussion, system co-development, cluster administration, and other research activities.

I owe my deepest gratitude to my parents, for their love and support, which give me the confidence and energy to overcome all past, current, and future difficulties.

Finally and most importantly, I would like to express my gratitude beyond words to my wife, Jiayin, who has been encouraging and inspiring me since we met in love. We went through all the difficult times together, and depended on each other in this foreign place one Pacific Ocean away from our homeland. Every step forward I have made is unimaginable without Jiayin's support.

This dissertation study was supported in part by NSF IIS-1118106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Alibaba or the National Science Foundation.

Curriculum Vitæ

Wei Zhang

Education

2002 – 2005	Master of Science in Computer Science, Tsinghua University, Beijing, China.
1997 – 2001	Bachelor of Science in Electrical Engineering, Wuhan University, Wuhan, China.

Publications

Michael Daniel Agun, Tao Yang, Wei Zhang. "Asynchronous Source-side Deduplication for Cloud Data Backup". *To be submitted for publication.*

Wei Zhang, Michael Daniel Agun, Tao Yang, Hong Tang. "VM-Centric Snapshot Deduplication for Converged Cloud Architectures". *Submitted for publication.*

Wei Zhang, Tao Yang, Gautham Narayanasamy, Hong Tang. "Low-Cost Data Deduplication for Virtual Machine Backup in Cloud Storage". *In Proceedings of the 5th USENIX Workshop on Hot Topics in File and Storage Technologies (HotStorage'13)*, July 2013.

Wei Zhang, Hong Tang, Hao Jiang, Tao Yang, Xiaogang Li, Yue Zeng. "Multi-level Selective Deduplication for VM Snapshots in Cloud Storage". *In Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, June 2012.

Wen Ye, Wei Zhang, Rich Wolski. "Simulation-Based Augmented Reality for Sensor Network Development". *In Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

Wei Zhang, Haoxiang Lin. "A Versatile Cryptographic File System for Linux". *In Journal of Micro Computer Systems, China*, 2006.

Wei Zhang, Fuchun Sun. "An Improved FCM Algorithm Based On Search Space Smoothing". *In Proceedings of the 1st IASTED International Conference on Computational Intelligence (CI'05)*, July 2005.

Abstract

Collocated Data Deduplication for Virtual Machine Backup in the Cloud

Wei Zhang

Cloud platforms that host a large number of virtual machines (VMs) have high storage demand for frequent backups of VM snapshots. Content signature based deduplication is necessary to eliminate excessive redundant blocks. While dedicated backup storage systems can be used to reduce data redundancy, such an architecture is expensive and introduces huge network traffic in a large cluster. This thesis research is focused on a low-cost backup and deduplication service collocated with other cloud services to reduce infrastructure and network cost.

The previous research for cluster-based data deduplication has concentrated on various inline solutions. The first part of the thesis work is a highly parallel batched solution with synchronized backup scalable for a large number of virtual machines. The key idea is to separate duplicate detection from the actual storage backup, and to partition global index and detection requests among machines using fingerprint values. Then each machine conducts duplicate detection partition by partition independently with minimal memory consumption. Another optimization is to allocate and control buffer space for exchanging detection requests and duplicate summaries among machines. The resource requirement in terms of memory and disk usage for the proposed solution is very small while the backup efficiency in terms of overall throughput and time is not compromised. Our evaluation validates this and shows a satisfactory backup throughput in a large cloud setting.

The second part of the thesis work is a VM-centric collocated backup service with inline deduplication. The key difference compared to the previous work is its novelty in fault resilience and low resource usage. We propose a multi-level selective deduplication scheme which integrates similarity-guided and popularity-guided duplicate elimination under a stringent resource requirement. This scheme uses popular common data to facilitate fingerprint comparison, localizes deduplication as much as possible within each VM, and associates underlying file blocks with one VM for most of cases. The main advantage of this scheme is that it strikes a balance between inner and inter VM deduplication, increasing parallelism and improving reliability. Our analysis shows that this VM-centric scheme can provide better fault tolerance while using a small amount of computing and storage resource. We have conducted a comparative evaluation of this scheme on its competitiveness in terms of deduplication efficiency and backup throughput.

Professor Tao Yang
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The ubiquity of the cloud computing has resulted in the widespread availability of cluster-based services and applications accessible through the Internet. Examples include online storage services, big data analytics, and e-commerce websites. In such a cluster-based cloud environment, each physical machine runs a number of virtual machines as instances of a guest operating system to contain different kind of user applications, and their data is stored in virtual hard disks which are represented as virtual disk image files in the host operating system. Frequent snapshot backup of virtual disk images is critical to increase the service reliability and protect data safety. 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. If such VM snapshot data is plainly backed up without any duplicate reduction, storage waste would be extreme high. For example, a fresh Windows server 2008 installation costs 25 GB on the virtual disk, and they are almost certainly duplicated with other Windows VM instances[51, 33]. Using a separate backup service with full deduplication support [42, 62] 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.

Unlike the previous work dealing with general file-level backup and deduplication inside a centralized storage appliance, our research is focused on virtual disk image backup in a cluster of machines. Although we treat each virtual disk as a file logically, its size is very large. On the other hand, we need to support parallel backup of a large number of virtual disks in a cloud every day. One key requirement we face at a busy cloud cluster is that VM snapshot backup should only use a minimal amount of system resources so that most of resources is kept for regular cloud system services or applications. Thus our objective is to exploit the characteristics of VM snapshot data and pursue a cost-effective deduplication solution. Another goal is to decentralize VM snapshot backup and localize deduplication as much as possible, which brings the benefits for increased parallelism and fault isolation.

Despite its importance, storage deduplication remains a challenging task for cloud storage providers, especially for resource-constrained large-scale public VM clouds. In recognizing this importance and the associated challenges, my thesis is that

It is possible to build an efficient, scalable, and fault-tolerant backup service supporting highly accurate deduplication with sustainable high throughput.

In other words, the main goal of this study is to answer the following question. Given a large scale VM cluster running tens of thousand of VMs, how can such a snapshot backup service identify whether a piece of data in VM disk really needs a backup? And how to manage backup data under complex sharing relationship? This dissertation investigates techniques in building a snapshot storage architecture that provides low-cost deduplication support for large VM clouds. In particular, it contains the following contributions to establish my thesis:

- The design of a low-cost multi-stage parallel deduplication solution for automatically batched deduplication.
- An inline multi-level deduplication scheme with similarity-guided local detection and popularity-guided global detection.
- The development and analysis of a VM-centric storage approach which considers fault isolation and integrates multiple duplicate detection strategies.

- Fast and efficient garbage collection with approximate snapshot deletion and leak repair.

In general, this dissertation study is built upon a large body of previous research in storage deduplication and distributed storage systems. The goal of this work is to provide low-cost deduplication support and storage management for large-scale resource-constrained VM clouds. This solution should be collocated with existing cloud infrastructure services and require no dedicated hardware support. It also needs to accomplish good deduplication efficiency and does not comprise the data reliability due to the inherent volatility of commodity hardware. Under the above consideration, our system is designed to support multiple levels of deduplication depending on backup requirements. The objective is to provide satisfactory deduplication efficiency and high throughput with the emphasis on low resource usage and fault tolerance.

1.2 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 gives the background of thesis problems. Chapter 3 presents the overall VM snapshot storage architecture. Chapter 4 describes a synchronous and parallel deduplication scheme for batched complete deduplication. Chapter 5 studies a multi-level selective deduplication solution for inline deduplication. Chapter 6 describes a VM-centric storage management with approximate deletion. Chapter 7 concludes this dissertation and briefly discusses some potential future work.

Chapter 2

Background

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. Virtual machine backup is typically conducted through a snapshot, which is a copy of the virtual machine's disk files at a given time. Performing VM snapshot backup for the entire cluster is difficult for several reasons:

- **Huge storage demand.** The size of each VM snapshot varies from tens of gigabytes to multiple terabytes while the total snapshot size of an entire cloud cluster is often in a petabyte level. Such data needs to be backed up daily or weekly.

- **Big network bandwidth cost.** Sending such a large amount of snapshot data requires a large network bandwidth and may impede normal application activities.

As a result, data deduplication is necessary in order to reduce the storage and bandwidth costs of backup in a cloud cluster. But adding deduplication at a cloud scale brings new challenges against previous developed techniques:

- **Limited resources.** Without dedicated hardware support in backup appliances, VM backup and data deduplication in cloud must not race for CPU and memory resources with existing VM activities. Compare to dedicated backup appliances which are typically equipped with tens of GB of memory and multiple processors, backup service in the cloud are only allowed to use no more than a few hundred MB of memory, and its CPU usage must always stay low.

- **Garbage collection for shared data.** Data management for deduplicated storage is complicated due to the need of tracking use of every shared data block. In order to reclaim disk space, the reference counts of shared data blocks will need to be updated whenever a snapshot is added or deleted. Managing reference counting for data blocks at KB level in a PB level storage is extreme difficult.

- **Little semantic information.** For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations take place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data.

In summary, deduplication systems designed for legacy network storage that are attached to physical servers are too expensive and complex for virtual machines. Both cloud users and providers are seeking low-cost solutions to effectively backup their data, otherwise they will be forced to backup less amount of data or less frequently, which in return will compromise the data safety and recovery time. For example, Electronic Arts, which deploys its data analytics platform in Amazon's AWS using several hundreds of VMs, avoids using Amazon's built-in snapshot storage and chooses to backup hand-picked data manually for cost reasons.

To that end, this dissertation proposes low-cost deduplication storage architecture that is collocated with other cloud services. We have learned a lot from various deduplication systems built in the past for different backup scenarios. In this chapter we will review the literature and discuss several system design options at high-level. Section 2.1 introduces the origin and applications of data deduplication. Section 2.2 talks the optimization and trade-off in designing a deduplication storage system. Section 2.3 discusses design options of backup storage architecture.

2.1 Deduplication

Backup storage for VM snapshots in the cloud contains petabytes of data from many VMs, the data redundancy in such a data set is very high[30, 5]. If we treat each snapshot as a single data object, the similarities between snapshots can not be utilized. Therefore we need better method to break files into data blocks. In the recent decade, data deduplication techniques are emerged to solve this data redundancy problem in backup systems. Most of data deduplication methods can be categorized by their granularity:

- **By file:** Old storage systems detect the duplicated files by compare their hash value or byte-by-byte. If two files are the same, then only one copy of data is saved, duplicated copies will be saved as metadata plus a pointer to the data. Thus this method can not detect any redundant data between different files.

- **By fixed size block:** Many popular file systems or tools can catch data redundancy at block level. It breaks files into many small fixed size blocks, and such block is the basic unit for comparison. Typical applications based on this method are Windows shadow copy, ZFS[3] snapshot, and Rsync[52]. While this method does better than simple file comparison, it can only be used in pair-wise comparison between different versions of the same logical data object, and any insertion/deletion will completely destroy the block similarity after the position where modifications are taken place.

- **By variable size block:** This method breaks data into variable size blocks, and the block boundary is only decided by local data content, so it is also called content-based chunking. Because of the chunk boundaries are only decided by local data content, this method is modification-resistant. It resolves the limitations in the previous two methods, and provides best deduplication efficiency.

Content-based chunking is first proposed by Manber[35]. He uses Rabin's fingerprints[43, 18] to sample data in order to find similar files. This technique computes fingerprints of all possible sub-strings of a certain length in a file and chooses a subset of these fingerprints based on their values, e.g., if $(fingerprint(s)\ mod\ a) = b$, where s is a sub-string, a and b are pre-defined values, then the position of sub-string s is considered as a breakpoint. These breakpoints provide modification-resistant boundaries that separate the file into many small chunks. A sequence of hash values of such chunks is a compact representation of a file, that is then used to compare against other fingerprinted files.

There are pathological cases in finding breakpoints, for example, a long extent of zeros may never contain a break point. Thus, LBFS[37] introduces min and max thresholds to the chunk size: when scanning the file sequentially for breakpoints, if the distance between current sub-string and previous breakpoint is less than the minimum threshold, then it is skipped; if the distance reaches the maximum threshold, then a breakpoint is enforced. This method also reduces the computation because less sub-strings are checked. In order to increase the chance of finding a breakpoint between two thresholds, Kave[32] proposes the two divisors idea, in addition to have a standard breaking condition, a weaker breaking condition is used to find backup breakpoints.

Policroniades[41] gives out experimental results and present a comparative analysis to three deduplication methods: First, the content based chunking is always the best strategy to discover redundancy in any data sets. Second, the fixed size block method can sometimes provides close results compare to content based chunking, and it brings significant less overhead to computation and data management. Finally, for I/O intensive primary storage system, data compression is still the most efficient approach to save disk space, inline data reduction through deduplication has too much random I/O overhead, making it difficult to justify the limited space savings in a primary storage system.

Data deduplication is extensively used in saving bandwidth for data synchronization. The popular rsync[52] protocol exploits the similarity of two directory trees using fixed size blocks deduplication. LBFS uses content-based chunking to improve the NFS protocol, it avoids transmitting redundant chunks which already exist at the other side. But data deduplication has its cost. Content-based chunking algorithm needs to scan the whole file looking for breakpoints, which is very slow (about 20MB/s using a single 2GHz core). StoreGPU[6] exploits the possibility of using GPU to chunk the file, their results are exciting. Another option is to use parallel chunking algorithm to improve the chunking speed, so that future's many-core chips will be helpful. Meanwhile, stripping files is not a quite big issue in cloud storage scenario, because it is expected to be done at the client side before files are sent to the cloud.

Since content-based chunking strips files into data blocks and identify them by content hash, it is very natural to build CAS system base on it. Jumbo Store and others[25, 11, 59] encodes directory tree snapshots into a graph whose nodes are data blocks and edges are hash value pointers. Directory metadata, file metadata and data chunk are all represented in the form of variable-size data blocks that identified by content hash. A object which encapsulate metadata also has a list of hash identifiers that point to other blocks it contains. Such a deduplication storage system has some great advantages in terms of space efficiency, file transmission efficiency (through a LBFS-like protocol), and data integrity checking (by verifying the hash).

Cumulus[53] is a client side backup tool to store file system snapshots in exist cloud storage such as S3. To create a snapshot, Cumulus uses content-based chunking to break files into data blocks, then compare (by hash) them to the list of stored blocks (in local logs). New blocks will be packed and compressed into a segment file, and sent to server. The advantage of Cumulus is portability, because it makes the deduplication backup totally opaque to the server. However, without the deduplication support from protocol and server side, it's hard to have efficient storage management, and from server's perspective, snapshots from many users still contain much redundant data.

2.2 Optimization and Trade-off

Many previous studies on data deduplication focus on the disk bottleneck problem, which refers to the key performance challenge in finding duplicate blocks. Let's consider a single machine in cluster which hosts many VMs and has 10 TB of virtual disks in total, if it needs to be backed up daily, then the performance target would be 115 MB/sec on each machine. Given a block size of 4 KB, a deduplication system must process approximately 28,900 blocks per second. An in-memory index of all block fingerprints could easily achieve this performance, but the size of the index would limit system size and increase system cost. If we use 20 bytes as block fingerprint size, then supporting 10 TB worth of unique blocks, would require 50 GB just to store the fingerprints. An alternative approach is to maintain an on-disk index of block fingerprints and use a cache to accelerate block index accesses. Unfortunately, a traditional cache would not be effective for this workload. Since fingerprint values are random, there is no spatial locality in the block index accesses. With low cache hit ratios, most index lookups require disk operations. If each index lookup requires a disk access which may take 10 ms and 8 disks are used for index lookups in parallel, the write throughput will be about 6.4MB/sec, which is far from the throughput goal of deduplicating at 115 MB/sec for our cloud backup scenario, and don't forget that this goal is conservative since VM users generally want the backup task to finish in tens of minutes without affecting their normal application activities.

Massive efforts have been put into the optimization of searching duplicate fingerprints in large index. They can be summarized into three categories:

- **Reduce disk index access.** The data domain method [62] uses an in-memory Bloom filter and a locality-based prefetching cache to intercept fingerprint lookup which may access the on-disk index. Bloom filter[13] helps to tell if a fingerprint is new, and prefetching cache helps to quickly find consequential duplicates. Improvements to this work with parallelization can be found in [56, 58, 38].

- **Index sampling.** A sampled index can be put into memory[34] or NAND flash[29] to accelerate fingerprint lookup. This sampled index is 100 times or more smaller than the full index so that they can be accessed quickly and easily. Once there's a hit in the sampled index, system loads the fingerprints that are next to the hit one on disk, so that it can efficiently catches duplicates based on spatial locality.

- **Similarity based sharding.** Several similarity based techniques[19] such like Extremely Binning[12, 22, 48, 49] group big files by similarity metrics such that similar files are likely to fall into the same group. Then block level deduplication is performed within the group. While all the group indices are still on the disk, it can deduplicate an entire file by only load one group index into memory. This approach can easily scale-out, but it misses small amount of duplicates because the similarity algorithm does not guarantee always mapping a file to the most similar group.

Most of approaches discussed above focus on optimization of inline deduplication performance by sacrificing a small percent of deduplication efficiency. In our cloud backup scenario, we must always satisfy the low resource usage constraints while providing sustainable good throughput. As a result, we consider to sacrifice other performance metrics to achieve our design goals, which we describe as below:

Given we are designing scalable low-cost collocated deduplication solutions, the effectiveness of a deduplication system is determined by the extent to which it can achieve three mutually competing goals: deduplication efficiency, throughput, and job turnaround time. Deduplication efficiency refers to how well the system can detect and share duplicate data units which is its primary compression goal. Throughput refers to the rate at which data can be transferred in and out of the system, and constitutes the main performance metric. Job turnaround time is the total time for a backup job from submit to finish. All three metrics are important. Good deduplication efficiency reduces the storage cost. High throughput is particularly important because it can enable fast backups, minimizing the length of a backup window. Job turnaround time reflects how fast primary storage can be released from backup related I/O activity. Among the three goals, it is easy to optimize any two of them, but not all. To get good deduplication efficiency, it is necessary to perform data indexing for duplicate detection. The indexing metadata size grows linearly with the capacity of the system. Keeping this metadata in memory, would yield good throughput. But the amount of available RAM would set a hard limit to the scalability of the system. Partition indexing metadata man remove the scalability limit, but significantly hurt job turnaround time. Finally, we can optimize for both throughput and turnaround time, , but then we lose deduplication as there's no cheap solution to search all metadata quickly. Achieving all three goals is a non-trivial task.

Another less obvious but equally important problem is duplicate reference management: duplicate data sharing introduces the need to determine who is using a particular data unit, and when it can be reclaimed. The computational and space complexity of these reference management mechanisms grows with the amount of supported capacity. Real world experience has shown that the cost of reference management for garbage collection (upon addition and deletion of data) has become one of the biggest bottlenecks.

This dissertation studies two low-cost deduplication design options with different trade-off. Chapter 4 gives a synchronous parallel deduplication design that sacrifices the job turnaround time to achieve high throughput and deduplication efficiency. Chapter 5 describes an multi-level selective deduplication solution that comprises deduplication efficiency to improve throughput and job turnaround time.

2.3 Architecture Options

The way that backup storage architecture evolves has been following the pace of primary storage, so we will give a glimpse on primary storage's evolution first. Organizations start with building their virtualization infrastructure using the traditional servers-connected-to-storage-over-a-network architecture, which cant adapt to the ever-changing demands of virtualization. In addition to slow performance, network storage has become the single biggest source of cost and complexity in virtualized environments. The network storage-based architecture worked well for physical servers that served relatively static workloads. Virtualization, and now Cloud Computing, has made data centers extremely dynamic[8]; virtual machines are created on the fly, move from server to server and depend heavily on shared resources. These characteristics make the management of virtual machines and their underlying physical infrastructure extremely complex: Data volumes are growing at a rapid pace in the data center, thanks to the ease of creating new VMs. In the enterprise, new initiatives like desktop virtualization contribute to this trend. Service providers deal with an even larger number of VMs as they build data centers to serve customers who cant afford the cost and management overhead that virtualization requires. This growing pool of VMs is exerting tremendous cost, performance and manageability pressure on the traditional architecture that connects compute to storage over a multi-hop network.

Google[27] and other leading cloud-generation companies such as Amazon, Yahoo and Microsoft(Azure)[20] realized that a network-storage based approach would not work for their data centers. They built software technology (such as Google File System) that could glue a large number of commodity servers with local storage into a single cluster. This approach allowed Google to build a converged compute and storage infrastructure that used commodity servers with local storage as its building block[28]. Google File System runs across a cluster of servers and creates a single pool of local storage that can be seamlessly accessed by applications running on any server in the cluster. It provides high availability to applications by masking failures of hard disks and even complete servers. Google File System allowed Google to build data centers with massively scalable compute and storage, without incurring the costs and performance limitations associated with network storage.

In cooperate with the changes of primary storage in cloud, backup storage architecture has to evolve as well. A dedicated backup storage appliance sits aside the cloud does provide some good properties, such like it deduplicates all the backup data together so it provides good deduplication efficiency, and because it uses dedicated hardware to fingerprint lookups, it leaves very small resource footprint on the client side. However, its weakness on high-cost and bandwidth demand make it unsuitable for large scale cloud backup scenario. This is especially true for public clouds whose users are very sensitive to the storage pricing.

To make VM snapshot backup efficient and economy for large scale cloud, we must seek for a low-cost architecture that can be collocated with existing storage and other cloud services. To avoid excessive bandwidth usage on transmitting backup data, it must process fingerprint lookups at the client side and provides good deduplication efficiency. In addition, the resource footprint of deduplication must stay very low in order to minimize the influence to normal VM activities. Finally, it must provides efficient mechanism to manage the complicated data sharing relationship on deduplicated data.

Chapter 3

System Overview

At a cloud cluster node, each instance of a guest operating system runs on a virtual machine, accessing virtual hard disks represented as virtual disk image files in the host operating system. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations take place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data. Backup systems have been developed to use content fingerprints to identify duplicate content [42, 44]. As discussed earlier, collocating a backup service on the existing cloud cluster avoids the extra cost to acquire a dedicated backup facility and reduces the network bandwidth consumption in transferring the raw data for backup. Figure 3.1 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 [27, 46]. 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.

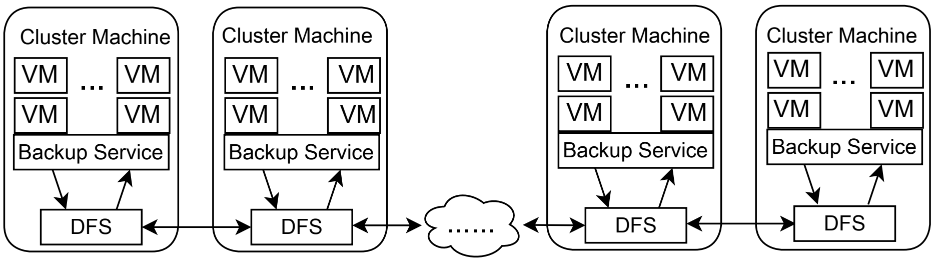


Figure 3.1: VM snapshot backup running on a converged cloud cluster.

3.1 Snapshot Service Overview

We will briefly introduce the abstraction of our storage architecture here and discuss the implementation details in chapter 6. Our architecture is built on the Aliyun platform which provides the largest public VM cloud in China based on Xen [10]. A typical VM cluster in our cloud environment consists of from hundreds to thousands of physical machines, each of which can host tens of VMs.

A GFS-like distributed file system holds the responsibility of managing physical disk storage in the cloud. All data needed for VM services, which include runtime VM disk images and snapshot backup data, reside in this distributed file system. During the VM creation, a user chooses her flavor of OS distribution and the cloud system copies the corresponding pre-configured base VM image to her VM as the OS disk, and an empty data disk is created and mounted onto her VM as well. All these virtual disks are represented as virtual machine image files in our underline runtime VM storage system. The runtime I/O between virtual machine and its virtual disks is tunneled by the virtual device driver (called TapDisk[55] at Xen). To avoid network latency and congestion, our distributed file system place the primary replica of VM's image files at physical machine of VM instance. During snapshot backup, concurrent disk write is logged to ensure a consistent snapshot version is captured.

Figure 3.2 shows the architecture view of our snapshot service at each node. The snapshot broker provides the functional interface for snapshot backup, access, and deletion. The inner-VM deduplication is conducted by the broker to access meta data in the snapshot data store and we discuss this in details in Section 5.3.1. The cross-VM deduplication is conducted by the broker to access a popular data set (PDS) (will discuss in Section 5.3.2, whose block hash index is stored in a distributed memory cache.

The snapshot store supports data access operations such as *Get*, *Append* and *Delete*. Other operations include data block traverse and resource usage report. The snapshot data does not need to be co-located with VM instances, and in fact they can even live in a different cluster to improve the data reliability: when one cluster is not available, we are still able to restore its VMs from another cluster which holds its snapshot data.

Under the hood of snapshot store, it organizes and operates snapshot data in the distributed file system. We let each virtual disk has its own snapshot store, and no data

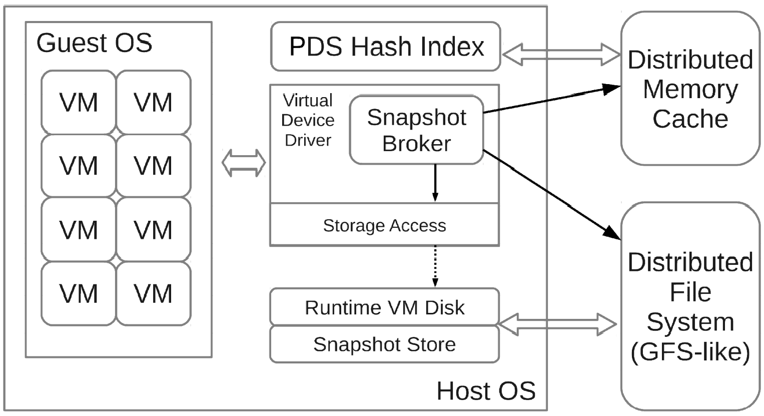


Figure 3.2: Snapshot backup architecture abstraction

is shared between any two snapshot stores, thus achieve great fault isolation. For those selected popular data that shared by many VM snapshot stores, we could easily increase its availability by having more replications.

3.2 VM Snapshot Metadata Hierarchy

The representation of each snapshot in the backup storage has a two-level index structure in the form of a hierarchical directed acyclic graph as shown in Figure 3.3. A VM image is divided into a set of segments and each segment contains content blocks of variable-size, partitioned using the standard chunking technique with 4KB as the average block size. The snapshot metadata contains a list of segments and other meta

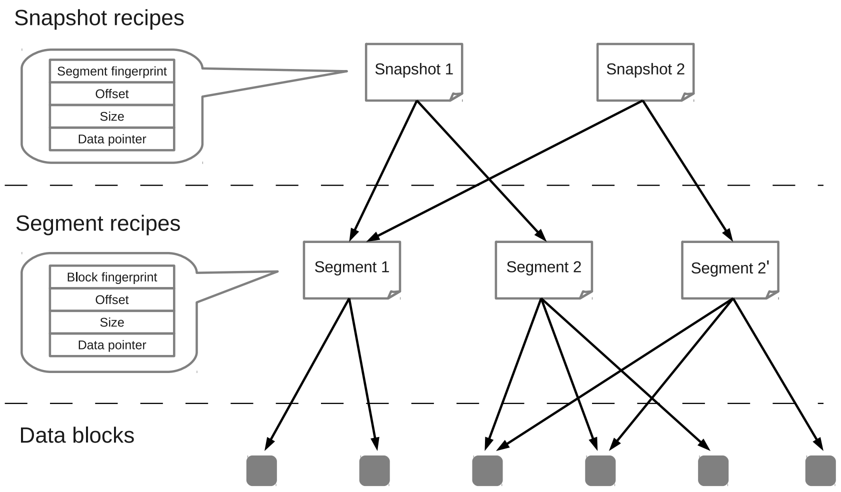


Figure 3.3: An example of snapshot representation.

data information. Segment metadata contains its content block fingerprints and reference pointers. If a segment is not changed from one snapshot to another, indicated by a dirty bit embedded in the virtual disk driver, its segment metadata contains a reference pointer to an earlier segment. For a dirty segment, if one of its blocks is duplicate to another block in the system, the block metadata contains a reference pointer to the earlier block.

Chapter 4

Synchronous and Parallel Approach for Batched Deduplication

4.1 Introduction

This chapter introduces a low-cost architecture option and considers that a backup service uses the existing cloud computing resource. Performing deduplication adds significant memory cost for comparison of content fingerprints. Since each physical machine in a cluster hosts many VMs, memory contention happens frequently. Cloud providers often wish that the backup service only consumes small or modest resources with a minimal impact to the existing cloud services. Another challenge is that deletion of old snapshots compete for computing resource as well, because data dependence created by duplicate relationship among snapshots adds processing complexity.

The traditional approach to deduplication is an inline approach which follows a sequence of block reading, duplicate detection, and non-duplicate block write to the backup storage. In this chapter we introduce an synchronous and parallel approach for batched deduplication. This solution is suitable for cloud which typically conduct automatic backup of all the VMs during system's spare time. Our key idea is to first perform parallel duplicate detection for VM content blocks among all machines before performing actual data backup. Each machine accumulates detection requests and then performs detection partition by partition with minimal resource usage. Fingerprint based partitioning allows highly parallel duplicate detection and also simplifies reference counting management.

While our synchronous batched solution accomplishes perfect deduplication efficiency while maintaining low resource usage, the trade-off of this approach is that the job turnaround time for each individual VM backup task becomes longer. In addition, every machine has to read dirty segments twice and that some deduplication requests are delayed for staged parallel processing. However, with our careful parallelism and buffer management, this multi-stage detection scheme can provide a sufficient throughput for VM backup.

The rest of this chapter is organized as follows. Section 4.2 describes our synchronous processing steps. Section 4.3 is our experimental evaluation that compares with other approaches. Section 4.4 reviews the related works. Section 4.5 concludes this chapter.

4.2 Multi-stage Synchronous Deduplication Design

We consider deduplication in two levels. The first level uses coarse-grain segment dirty bits for version-based detection [21, 54]. Our experiment with Alibaba's production dataset shows that over 70 percentage of duplicates can be detected using segment dirty bits when the segment size is 2M bytes. This setting requires OS to maintain segment dirty bits and the amount of space for this purpose is negligible. In the second level of deduplication, content blocks of dirty segments are compared with the fingerprints of unique blocks from the previous snapshots. Our key strategies are explained as follows.

- **Separation of duplicate detection and data backup.** The second level detection requires a global comparison of fingerprints. Our approach is to perform duplicate detection first before actual data backup. That requires a prescanning of dirty VM segments, which does incur an extra round of VM reading. During VM prescanning, detection requests are accumulated. Aggregated deduplicate requests can be processed partition by partition. Since each partition corresponds to a small portion of global index, memory cost to process detection requests within a partition is small.

- **Buffered data redistribution in parallel duplicate detection**. Let *global index* be the meta data containing the fingerprint values of unique snapshot blocks in all VMs and the reference pointers to the location of raw data. A logical way to distribute detection requests among machines is based on fingerprint values of content blocks. Initial data blocks follows the VM distribution among machines and the detected duplicate summary should be collected following the same distribution. Therefore, there are two all-to-all data redistribution operations involved. One is to map detection requests from VM-based distribution to fingerprint based distribution. Another one is to map duplicate summary from fingerprint-based distribution to VM based distribution. The redistributed data needs to be accumulated on the disk to reduce the use of memory. To minimize the disk seek cost, outgoing or incoming data exchange messages are buffered to bundle small messages. Given there are $p \times q$ partitions where p is the number of machines and q is the number of fingerprint-based partitions at each machine, space per each buffer is small under the memory constraint for large p or q values. This counteracts the effort of seek cost reduction. We have designed an efficient data exchange and disk data buffering scheme to address this.

We assume a flat architecture in which all p machines that host VMs in a cluster can be used in parallel for deduplication. A small amount of local disk space and memory on each machine can be used to store global index and temporary data. The real backup storage can be either a distributed file system built on this cluster or use another external storage system.

We use the two level metadata hierarchy as discussed in section 3.2. The snapshot metadata contains a list of segments and other meta data information. Segment metadata contains its content block fingerprints and reference pointers. If a segment is not changed from one snapshot to another, indicated by a dirty bit embedded in the virtual disk driver, its segment metadata contains a reference pointer to an earlier segment. For a dirty segment, if one of its blocks is duplicate to another block in the system, the block metadata contains a reference pointer to the earlier block.

The data flow of our multi-stage duplicate detection is depicted in Figure 4.1. In Stage 1, each machine independently reads VM images that need a backup and forms duplicate detection requests. The system divides each dirty segment into a sequence of chunk blocks, computes the meta information such as chunk fingerprints, sends a request to a proper machine, and accumulates received requests into a partition on the local temporary disk storage. The partition mapping uses a hash function applied to the content fingerprint. Assuming all machines have a homogeneous resource configuration, each machine is evenly assigned with q partitions of global index and it accumulates corresponding requests on the disk. There are two options to allocate buffers at each machine. 1) Each machine has $p \times q$ send buffers corresponding to $p \times q$ partitions in the cluster since a content block in a VM image of this machine can be sent to any of these partitions. 2) Each machine allocates p send buffers to deliver requests to p machines; it allocates p receive buffers to collect requests from other machines. Then

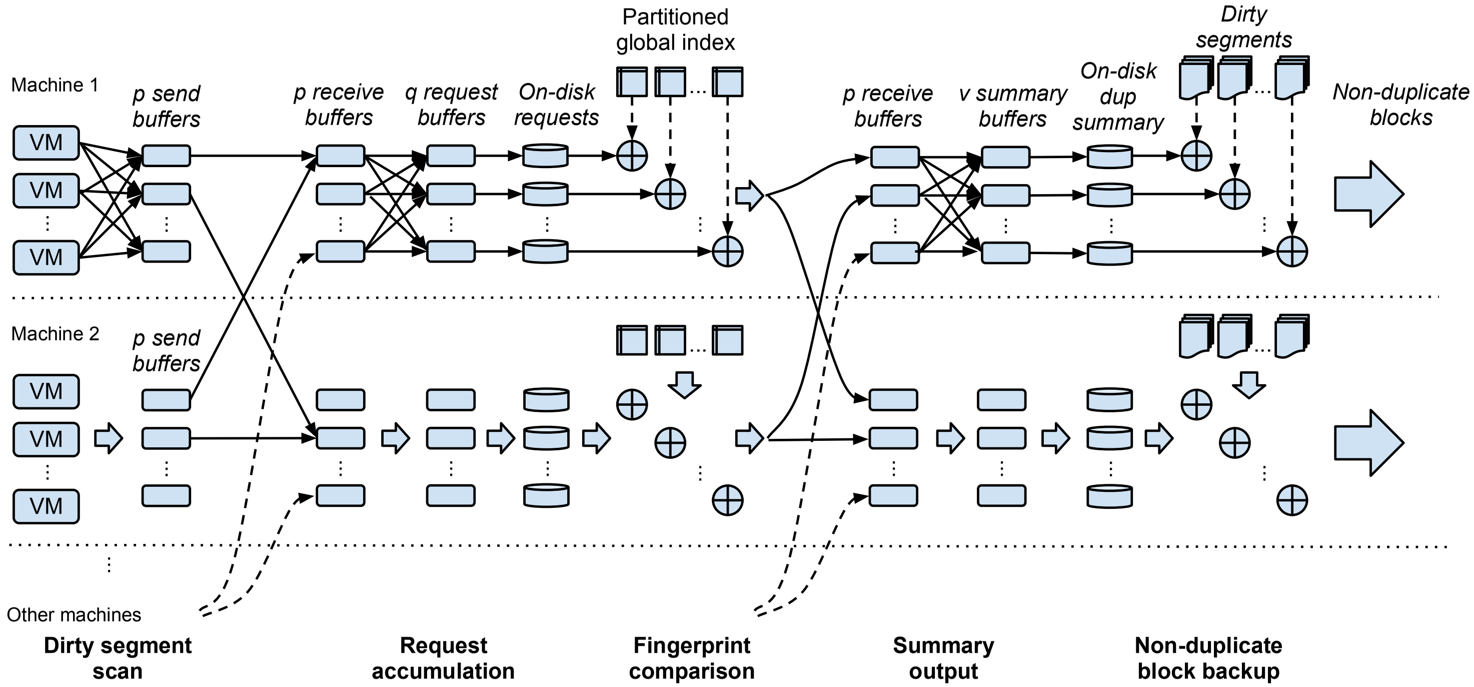


Figure 4.1: Processing flow of Stage 1 (dirty segment scan and request accumulation), Stage 2 (fingerprint comparison and summary output), and Stage 3 (non-duplicate block backup).

the system copies requests from each of p receive buffers to q local request buffers, and outputs each request buffer to one of the request partitions on the disk when this request buffer becomes full. Option 2, which is depicted in Figure 4.1, is much more efficient than Option 1 because $2p+q$ is much smaller than $p \times q$, except for the very small values. As a result, each buffer in Option 2 has a bigger size to accumulate requests and that means less disk seek overhead.

Stage 2 is to load disk data and perform fingerprint comparison at each machine one request partition at a time. At each iteration, once in-memory comparison between an index partition and request partition is completed, duplicate summary information for segments of each VM is routed from the fingerprint-based distribution to the VM-based distribution. The summary contains the block ID and the reference pointer for each detected duplicate block. Each machine uses memory space of the request partition as a send buffer with no extra memory requirement. But it needs to allocate p receive buffers to collect duplicate summary from other machines. It also allocates v request buffers to copy duplicate summary from p receive buffers and output to the local disk when request buffers are full.

Stage 3 is to perform real backup. The system loads the duplicate summary of a VM, reads dirty segments of a VM, and outputs non-duplicate blocks to the final backup storage. Additionally, the global index on each machine is updated with the meta data of new chunk blocks. When a segment is not dirty, the system only needs to output the segment meta data such as a reference pointer. There is an option to directly read dirty blocks instead of fetching a dirty segment which can include duplicate blocks. Our experiment shows that it is faster to read dirty segments in the tested workload. Another issue is that during global index update after new block creation, the system may find some blocks with the same fingerprints have been created redundantly. For example, two different VM blocks that have the same fingerprint are not detected because the global index has not contained such a fingerprint yet. The redundancy is discovered and logged during the index update and can be repaired periodically when necessary. Our experience is that there is a redundancy during the initial snapshot backup and once that is repaired, the percentage of redundant blocks due to concurrent processing is insignificant.

The above steps can be executed by each machine using one thread to minimize the use of computing resource. The disk storage usage on each machine is fairly small for storing part of global index and accumulating duplicate detection requests that contain fingerprint information. We impose a memory limit M allocated for each stage of processing at each machine. The usage of M is controlled as follows and space allocation among buffers is optimized based on the relative ratio between the cross-machine network startup cost and disk access startup cost such as seek time. Using a bigger buffer can mitigate the impact of slower startup cost.

- For Stage 1, M is divided for 1) an I/O buffer to read dirty segments; 2) $2p$ send/receive buffers and q request buffers.

- For Stage 2, M is divided for 1) space for hosting a global index partition and the corresponding request partition; 2) p receive buffers and v summary buffers.

- For Stage 3, M is divided for 1) an I/O buffer to read dirty segments of a VM and write non-duplicate blocks to the backup storage; 2) summary of duplicate blocks within dirty segments.

Snapshot deletion. Each VM will keep a limited number of automatically-saved snapshots and expired snapshots are normally deleted. We adopt the idea of mark-and-sweep [29]. A block or a segment can be deleted if its reference count is zero. To delete useless blocks or segments periodically, we read the meta data of all snapshots and compute the reference count of all blocks and segments in parallel. Similar to the multi-stage duplicate detection process, reference counting is conducted in multi-stages. Stage 1 is to read the segment and block metadata to accumulate reference count requests in different machines in the fingerprint based distribution. Stage 2 is to count references within each partition and detect those records with zero reference. The backup data repository logs deletion instructions, and will periodically perform a compaction operation when its deletion log is too big.

4.3 System Implementation and Experimental Evaluations

We have implemented and evaluated a prototype of our multi-stage deduplication scheme on a cluster of dual quad-core Intel Nehalem 2.4GHz E5530 machines with 24GB memory. Our implementation is based on Alibaba's Xen cloud platform [1, 60]. Objectives of our evaluation are: 1) Analyze the deduplication throughput and effectiveness for a large number of VMs. 2) Examine the impacts of buffering during metadata exchange.

We have performed a trace-driven study using a 1323 VM dataset collected from a cloud cluster at Alibaba's Aliyun. 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. Each VM image is divided into 2 MB fix-sized segments and each segment is divided into variable-sized content blocks with an average size of 4KB. The signature for variable-sized blocks is computed using their SHA-1 hash.

The seek cost of each random IO request in our test machines is about 10 milliseconds. The average I/O usage of local storage is controlled about 50MB/second for backup in the presence of other I/O jobs. Noted that a typical 1U server can host 6 to 8 hard drives and deliver over 300MB/second. Our setting uses 16.7% or less of local storage bandwidth. The final snapshots are stored in a distributed file system built on the same cluster.

The total local disk usage on each machine is about 8GB for the duplicate detection purpose, mainly for global index. Level 1 segment dirty bits identify 78% of duplicate blocks. For the remaining dirty segments, block-wise full deduplication removes about additional 74.5% of duplicates. The final content copied to the backup storage is reduced by 94.4% in total.

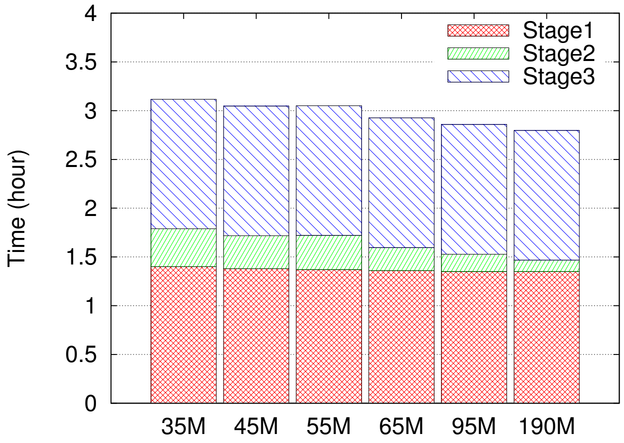


Figure 4.2: Parallel processing time when memory limit varies.

Figure 4.2 shows the total parallel time in hours to backup 2500 VMs on a 100-node cluster a when limit *M* imposed on each node varies. This figure also depicts the time breakdown for Stages 1, 2, and 3. The time in Stages 1 and 3 is dominated by the two scans of dirty segments, and final data copying to the backup storage is overlapped with VM scanning. During dirty segment reading, the average number of consecutive dirty segments is 2.92. The overall processing time does not have a significant reduction as M increases to 190MB. The aggregated deduplication throughput is about 8.76GB per second, which is the size of 2500 VM images divided by the parallel time. The system runs with a single thread and its CPU resource usage is 10-13% of one core. The result shows the backup with multi-stage deduplication for all VM images can be completed in about 3.1 hours with 35MB memory, 8GB disk overhead and a small CPU usage. As we vary the cluster size p, the parallel time does not change much, and the aggregated throughput scales up linearly since the number of VMs is $25p$.

Table 4.1 shows performance change when limit M=35MB is imposed and the number of partitions per machine (q) varies. Row 2 is memory space required to load a partition of global index and detection requests. When $q = 100$, the required memory is 83.6 MB and this exceeds the limit M =35MB. Row 3 is the parallel time and Row 4 is the aggregated throughput of 100 nodes. Row 5 is the parallel time for using Option 1 with $p \times q$ send buffers described in Section 4.2. When q increases, the available space per buffer reduces and there is a big increase of seek cost. The main network usage before performing the final data write is for request accumulation and summary output. It lasts about 20 minutes and each machine exchanges about 8MB of metadata per second with others during that period, which is 6.25% of the network bandwidth.

Table 4.1: Performance when M=35MB and q varies.

#Partitions (q)	100	250	500	750	1000
Index+request (MB)	83.6	33.5	16.8	11.2	8.45
Total Time (Hours)	N/A	3.12	3.15	3.22	3.29
Throughput (GB/s)	N/A	8.76	8.67	8.48	8.30
Total time (Option 1)	N/A	7.8	11.7	14.8	26

4.4 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.

Backup systems have been developed to use content fingerprints to identify duplicate content [42, 44]. Offline deduplication is used in [24, 7] to remove previously written duplicate blocks during system's idle time. Several techniques have been proposed to speedup searching of duplicate fingerprints. For example, the data domain method [62] uses an in-memory Bloom filter and a prefetching cache for data blocks which may be accessed. An improvement to this work with parallelization is in [56, 58]. As discussed in Section 4.1, there is no dedicated resource for deduplication in our targeted setting and low memory usage is required so that the resource impact to other cloud services is minimized. The approximation techniques are studied in [12, 29] to reduce memory requirement with a trade-off of the reduced deduplication ratio. In comparison, this paper focuses on full deduplication without approximation.

Additional inline deduplication techniques are studied in [34, 29, 48]. All of the above approaches have focused on such inline duplicate detection in which deduplication of an individual block is on the critical write path. In our work, this constraint is relaxed and there is a waiting time for many duplicate detection requests. This relaxation is acceptable because in our context, finishing the backup of required VM images within a reasonable time window is more important than optimizing individual VM block backup requests.

4.5 Concluding Remarks

The contribution of this work is a low-cost multi-stage parallel deduplication solution. Because of separation of duplicate detection and actual backup, we are able to evenly distribute fingerprint comparison among clustered machine nodes, and only load one partition at time at each machine for in-memory comparison.

The proposed scheme is resource-friendly to the existing cloud services. The evaluation shows that the overall deduplication time and throughput of 100 machines are satisfactory with about 8.76GB per second for 2500 VMs. During processing, each machine uses 35MB memory, 8GB disk space, and 10-13% of one CPU core with a single thread execution. Our future work is to conduct more experiments with production workloads.

The major limitation of our synchronous approach is that all VM snapshots must be processed together in order to achieve maximum aggregated throughput, as a result, small VMs need to wait big VMs during the duplicate detection phase until all VMs finishes stage 2. Since the small VM disks need to be put under copy-on-write protection longer, their disk I/O performance will be slow down and additional disk space are needed to store the modified segments during backup. Another limitation is that VM user cannot choose the time of taking snapshot freely. Therefore, our synchronous approach is ideal for private cloud in which the IT admin needs to backup all the VMs together, but it may not be suitable for public cloud which demands real-time snapshot backup operation. We will discuss our solution for on demand inline deduplication in the next chapter.

Chapter 5

Multi-level Selective Approach for Inline Deduplication

5.1 Introduction

In a public virtualized cloud environment such as ones provided by Amazon EC2[2] and Alibaba Aliyun[1], VM snapshots are created on user's requests. Because the snapshot captures the virtual disk state at a specific point of time, virtual disk under snapshot backup operation must be protected by copy-on-write until the operation finishes. However, copy-on-write introduces additional disk I/O and space overheads, users are typically advised to reduce their disk I/O activities to the minimum before making a snapshot backup operation, such activities typically include their web and database services. As a result, it is critical to perform the backup operation in the shortest amount of time to make the VM back to work.

Our synchronous batched deduplication introduced in chapter 4 does not apply to such situation which requires fast inline deduplication. Based on the requirement that we are now pursuing low-cost solution with short job turnaround time, we choose to design a deduplication solution which will sacrifice the deduplication efficiency to satisfy the cost and time constraints. By observations on the VM snapshot data from production cloud, we found snapshot data duplication can be easily classified into two categories: *inner-VM* and *cross-VM*. Inner-VM duplication exists between VM's snapshots, because the majority of data are unchanged during each backup period. On the other hand, Cross-VM duplication is mainly due to widely-used software and libraries such as Linux and MySQL. As the result, different VMs tend to backup large amount of highly similar data.

With these in mind, we have developed a distributed multi-level solution to conduct segment-level and block-level inner-VM deduplication to localize the deduplication effort when possible. It then makes cross-VM deduplication by excluding a small number of popular popular data blocks from being backed up. Our study shows that popular data blocks occupy significant amount of storage space while they only take a small amount of resources to deduplicate. Separating deduplication into multi levels effectively accomplish the major space saving goal compare the global complete deduplication scheme, at the same time it makes the backup of different VMs to be independent for better fault tolerance, which we will discuss more in chapter 6.

The rest of the paper is arranged as follows. Section 5.2 discusses the requirements and design options, Section 5.3 presents our multi-level selective deduplication scheme, Section 5.4 presents our evaluation results on the effectiveness of multi-level deduplication for snapshot backup. Section 5.5 discusses on some background and related work. Section 5.6 concludes this chapter.

5.2 Requirements and Design Options

We discuss the characteristics and main requirements for VM snapshot backup in a cloud environment. which are different from a traditional data backup.

1. *Cost consciousness.* There are tens of thousands of VMs running on a large-scale cluster. The amount of data is so huge such that backup cost must be controlled carefully. On the other hand, the computing resources allocated for snapshot service is very limited because VM performance has higher priority. At Aliyun, it is required that while CPU and disk usage should be small or modest during backup time, the memory footprint of snapshot service should not exceed 500 MB at each node.

2. *Fast backup speed.* Often a cloud has a few hours of light workload each day (e.g. midnight), which creates an small window for automatic backup. Thus it is desirable that backup for all nodes can be conducted in parallel and any cen-

tralized or cross-machine communication for deduplication should not become a bottleneck.

3. *Fault tolerance.* The addition of data deduplication should not decrease the degree of fault tolerance. It's not desirable that small scale of data failure affects the backup of many VMs.

There are multiple choices in designing a backup architecture for VM snapshots. We discuss the following design options with a consideration on their strengths and weakness.

1. *An external and dedicated backup storage system.* In this architecture setting, a separate backup storage system using the standard backup and deduplication techniques can be deployed [62, 12, 34]. This system is attached to the cloud network and every machine can periodically transfer snapshot data to the attached backup system. A key weakness of this approach is communication bottleneck between a large number of machines in a cloud to this centralized service. Another weakness is that the cost of allocating separate resource for dedicated backup can be expensive. Since most of backup data is not used eventually, CPU and memory resource in such a backup cluster may not be fully utilized.

2. *A decentralized and co-hosted backup system with full deduplication.* In this option, the backup system runs on an existing set of cluster machines. The dis-

advantage is that even such a backup service may only use a fraction of the existing disk storage, fingerprint-based search does require a significant amount of memory for fingerprint lookup of searching duplicates. This competes memory resource with the existing VMs.

Even approximation [12, 34] can be used to reduce memory requirement, one key weakness the hasn't been addressed by previous solutions is that global content sharing affects fault isolation. Because a content chunk is compared with a content signature collected from other users, this artificially creates data dependency among different VM users. In large scale cloud, node failures happen at daily basis, the loss of a shared block can affect many users whose snapshots share this data block. Without any control of such data sharing, we can only increase replication for global dataset to enhance the availability, but this incurs significantly more cost.

With these considerations in mind, we propose a decentralized backup architecture with multi-level and selective deduplication. This service is hosted in the existing set of machines and resource usage is controlled with a minimal impact to the existing applications. The deduplication process is first conducted among snapshots within each VM and then is conducted across VMs. Given the concern that searching duplicates across VMs is a global feature which can affect parallel performance and complicate failure management, we only eliminate the duplication of a small but popular data set while still maintaining a cost-effective deduplication ratio. For this purpose, we exploit the data characteristics of snapshots and collect most popular data. Data sharing across VMs is limited within this small data set such that adding replicas for it could enhance fault tolerance.

5.3 Multi-level Selective Deduplication Scheme

5.3.1 Similarity Guided Inner-VM Deduplication

The first-level deduplication is logically localized within each VM. Such localization increases data independency between different VM backups, simplifies snapshot management and statistics collection during VM migration and termination, and facilitates parallel execution of snapshot operations.

The inner VM deduplication contains two levels of duplicate detection efforts and the representation of each snapshot is correspondingly designed as a two-level level index data structure in the form of a hierarchical directed acyclic graph as shown in Figure 3.3. An image file is divided into a set of segments and each segment contains hundreds of content blocks from the bottom level. These blocks are of variable-size, partitioned using the standard chunking technique [35] with 4KB as the average block size. Segment metadata (called segment recipe) records its content block hashes and data pointers. The snapshot recipe contains a list of segments and other meta data information.

- *Level 1 Changed segment tracking.* We start with the changed block tracking approach in a coarse grain segment level. In our implementation with Xen on an Alibaba platform, the segment size is 2 MB and the device driver is extended to support tracking changed segments using a dirty bitmap. Since every write for a segment will touch a dirty bit, the device driver maintains dirty bits in memory and cannot afford a small segment size. It should be noted that dirty bit tracking is supported or can be easily implemented in major virtualization solution vendors. For example, the VMWare hypervisor has an API to let external backup applications know the changed areas since last backup. The Microsoft SDK provides an API that allows external applications to monitor the VM's I/O traffic and implement such changed block tracking feature.

- *Level 2 Similarity guided segment comparison.* Since the best deduplication uses a non-uniform chunk size in the average of 4KB or 8KB [31], we conduct additional local similarity guided deduplication on a snapshot by comparing chunk fingerprints of a dirty segment with those in *similar* segments from its parent snapshot. We define two segments are similar if their content signature is the same. This segment content signature value is defined as the minimum value of all its chunk fingerprints computed during backup and is recorded in the snapshot metadata (called recipe). Note that this definition of content similarity is an approximation [17]. When processing a dirty segment, its similar segments can be found easily from the parent snapshot recipe. Then recipes of the similar segments are loaded to memory, which contain chunk fingerprints to be compared. To control the time cost of search, we set a limit on the number of similar segments recipes to be fetched. For a 2MB segment, its segment recipe is roughly 19KB which contains about 500 chunk fingerprints and other chunk metadata, by limiting at most 10 similar segments to search, the amount of memory for maintaining those similar segment recipes is small. As part of our local duplicate search we also compare the current segment against the parent segment at the same offset.

We choose this two-level structure because in practice we observe that during each backup period only a small amount of VM data are added or modified. As the result, even the metadata of two snapshots can be highly similar, thus aggregating a large number of content blocks as one segment can significantly reduce the space cost of snapshot meta data.

How can we choose the length of a segment? Instead of using variables-sized segments, we take a simple approach to let every segment being aligned to the page boundary of each virtual image file. For Aliyun, each VM image file is represented as a virtual disk file format (called *vhd* at Xen) and we use a dirty bit to capture if a page (or segment) of a virtual disk file has been modified or not to ease the segment-level deduplication. A dirty bit array is used to indicate which segments have been modified or not. Each page (segment) size in our implementation uses 2 MB, which contains a large number of content blocks.

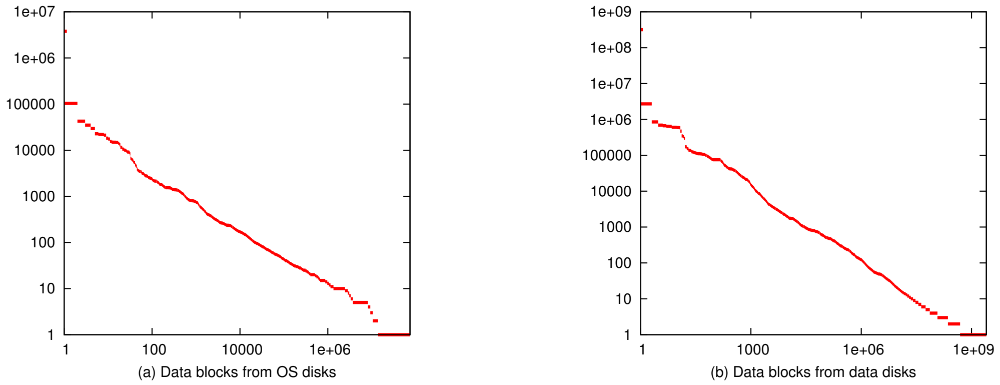
Once level-2 deduplication is activated for those segments that have been modified, it requires memory to load block fingerprints from the corresponding parent snapshot's segment. This scheme processes one segment at time and each segment of 2 MB contains about 500 content blocks on average given 4KB is the average block size. That only takes a tiny amount of space to hold their fingerprints.

5.3.2 Popularity guided Cross-VM Deduplication with PDS

The level-3 deduplication is to identify duplicated data blocks among multiple VMs through the index cache of popular data set (PDS). PDS is the most popular content blocks among snapshots across all VMs. Each index entry contains the block fingerprint and a reference pointer to the location of its real content in the snapshot content block store.

At the runtime, the PDS index resides in a distributed memory lookup table implemented using Memcached [26] to leverage the aggregated memory in a cluster. The usage of memory at each machine is small and thus this scheme does not lead to a memory resource contention with the existing cloud services. PDS raw data stored in the distributed file system has multiple copies in different machines for the purpose of fault tolerance and while providing high read throughput.

To control the size of searchable popular data in this global setting, we focus on those items that are most popular based on the historical snapshot data and the popular analysis is conducted periodically to ensure meta data freshness to match the latest data access trend. There are two advantages to exploit this. First, a smaller PDS reduces overall resource requirement while covering most of data duplication. Second, knowing this small set of data is shared heavily makes the fault tolerance management easier because we can replicate more copies to mitigate the failure.



(a) Data blocks from OS disks

(b) Data blocks from data disks

Figure 5.1: Number of duplicates vesus ranking

In Aliyun's VM cloud, each VM explicitly maintains one OS disk, plus one or more data disks. During VM's creation, its OS disk is directly copied from user's chosen base image. Given the separation of OS disks and data disks, we study their characteristics separately. We expect that data related to OS and popular software installations are not frequently modified or deleted, to facilitate analysis we call them *OS-related* data, and the rest of data, either from OS or data disks, are called *user-related* data.

We have studied the popularity of popular blocks in the OS and data disks from a dataset containing over 1,000 VMs, taking their first snapshots to watch the cross-VM duplication pattern (scale of OS disk sampling is smaller due to performance impact to user VMs). Figure 5.1 shows the duplicate count for unique data blocks sorted by their ranking in terms of the duplication count. Y axis is the popularity of a data block in a log scale measured its duplicate count among snapshots. X axis is the identification of data blocks in a log scale sorted by their duplicate rank. The rank number 1 is the block with the highest number of duplicates. These two curves exhibit that the popularity of popular blocks partially follows a Zipf-like distribution.

Base on the Zipf-like distribution pattern, many previous analysis and results on web caching[4, 15] may apply. In general this indicates a small set of popular data dominates data duplication. Although we already know OS-related data are heavily duplicated among OS disks, it still surprises us that user-related data follow a similar pattern. Therefore, we collect the PDS by performing global reference counting through map-reduce for all blocks in snapshot store, and select the most popular ones base on the memory limitation of PDS hash index.

The PDS collected from user-related data is generally proportional to the data size. As discussed in Section 5.4, selecting about 1% of data (after level 1 and level 2 deduplication) can cover about 38% of data blocks. Consider we allow maximum 25 VMs per machine, each VM has about 30 GB of user-related data, having 10 snapshots in its snapshot store and the data change ratio during each backup is 10%, this translates to 15 GB PDS data per machine. Consider each PDS hash index entry cost about 40 bytes and the average block size is 4KB, this leads to a 1:100 ratio so the memory cost by PDS hash index is about 150 MB per machine. On the other hand, the PDS from OS-related data is only relevant to the number of OS releases. Our experiment on 7 major OS releases shows that about 100 GB of data is never modified, and we expect it won't grow over 200 GB in the near future. So it would cost the entire cluster 2 GB of memory to store its hash index, or 20 MB per machine on a 100 node cluster. On average each OS disk has about 10 GB of data that can be completely eliminated in this way. Thus in total the PDS index size per node takes less than 170 MB in a large cluster bigger than 100 nodes, covering over 47% of blocks in OS and data disks after inner-VM deduplication. This memory usage is well below the limit required by our VM services.

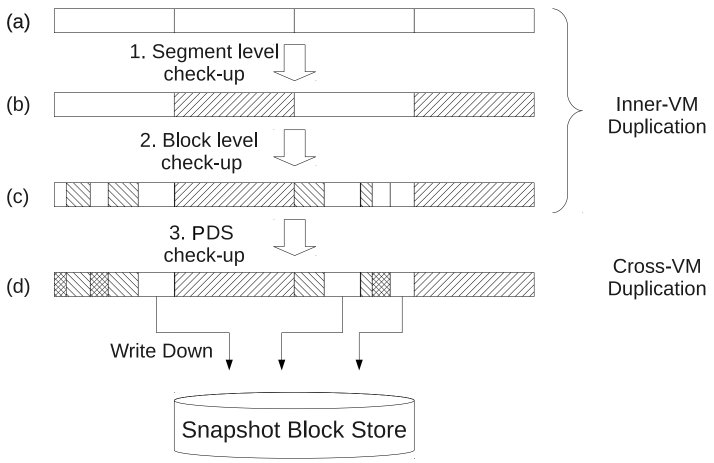


Figure 5.2: Illustration of snapshot deduplication dataflow.

5.3.3 Illustration of multi-level deduplication process

We illustrate the steps of 3-level deduplication in Figure 5.2, which can be summarized as 5 steps:

1. *Segment level checkup.* As shown in Figure 5.2 (a), when a snapshot of a plain virtual disk file is being created, we first check the dirty bitmap to see which segments are modified. If a segment is not modified since last snapshot, it's data pointer in the recipe of the parent snapshot can be directly copied into current snapshot recipe (shown as the shadow area in Figure 5.2 (b)).

2. *Block level checkup.* As shown in Figure 5.2 (b), for each dirty segment, we divide it into variable-sized blocks, and compare their signatures with the similar

segment recipes in the previous snapshot (called parent snapshot). For any duplicated block, we copy the data pointer directly from the parent segment recipe.

3. *PDS checkup.* For the remaining content blocks whose duplicate status is unknown, Figure 5.2 (d) shows a further check to compare them with the cached signatures in the PDS by querying the PDS hash index. If there is a match, the corresponding data pointer from the PDS index is copied into the segment recipe.

4. *Write new snapshot blocks :* If a data block cannot be found in the PDS index, this block is considered to be a new block and such a block is to be saved in the snapshot store, the returned data pointer is saved in the segment recipe.

5. *Save recipes.* Finally the segment recipes are saved in the snapshot block store also. After all segment recipes are saved, the snapshot recipe is complete and can be saved.

If there is no parent snapshot available, which happens when a VM creates its first snapshot, only PDS-based checkup will be conducted. Most of the cross-VM duplication, such as OS-related data, is eliminated at this stage.

5.4 System Implementation and Experimental Evaluations

We have implemented the snapshot deduplication scheme on the Aliyun's cloud platform. Objectives of our experimental evaluation are: 1) Analyze the popularity of content data blocks and the popularity of hot items. 2) Assess the effectiveness of 3-level deduplication for reducing the storage cost of snapshot backup. 3) Examine the impacts of PDS size on deduplication ratio.

5.4.1 Experimental setup

At Aliyun our target is to backup cluster up to 1000 nodes with 25 VMs on each. Based on the data studied, each VM has about 40 GB of storage data usage on average, OS disk and data disk each takes about 50% of storage space. The backup of VM snapshots is completed within two hours every day, and that translates to a backup throughput of 139 GB per second, or 500TB per hour. For each VM, the system keeps 10 automatically-backed snapshots in the storage while a user may instruct extra snapshots to be saved.

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 PDS deduplication against user-related data. The overall size of dataset2 is 23.5 TB.

All data are divided into 2 MB fix-sized segments and each segment is divided into variable-sized content blocks [35, 43] 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 PDS, as discussed in Section 5.3.2.

5.4.2 Effectiveness of Multi-level Deduplication

Figure 5.3 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

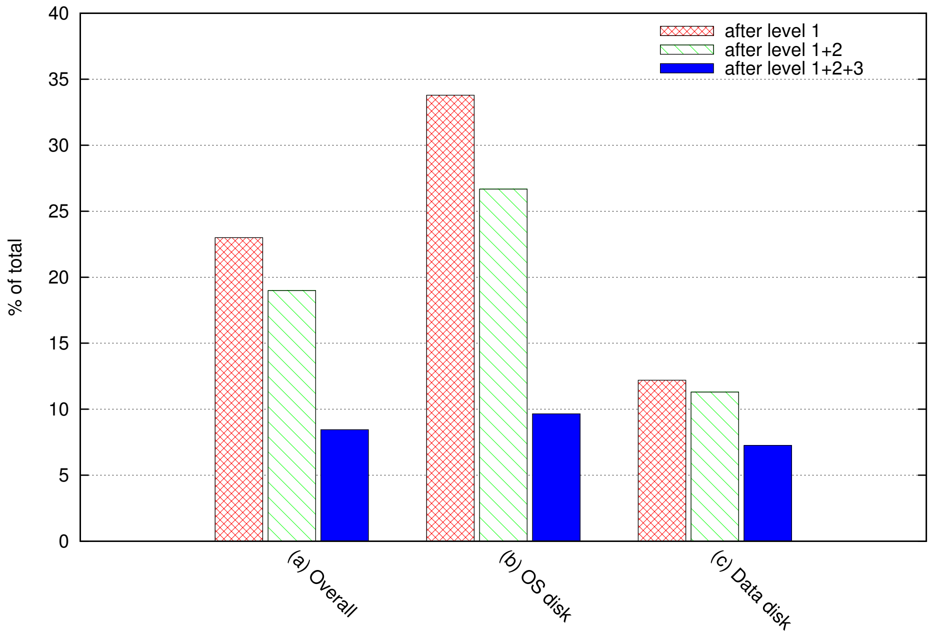


Figure 5.3: Impacts of 3-level deduplication. The height of each bar is the data size after deduplication divided by the original data size and the unit is percentage.

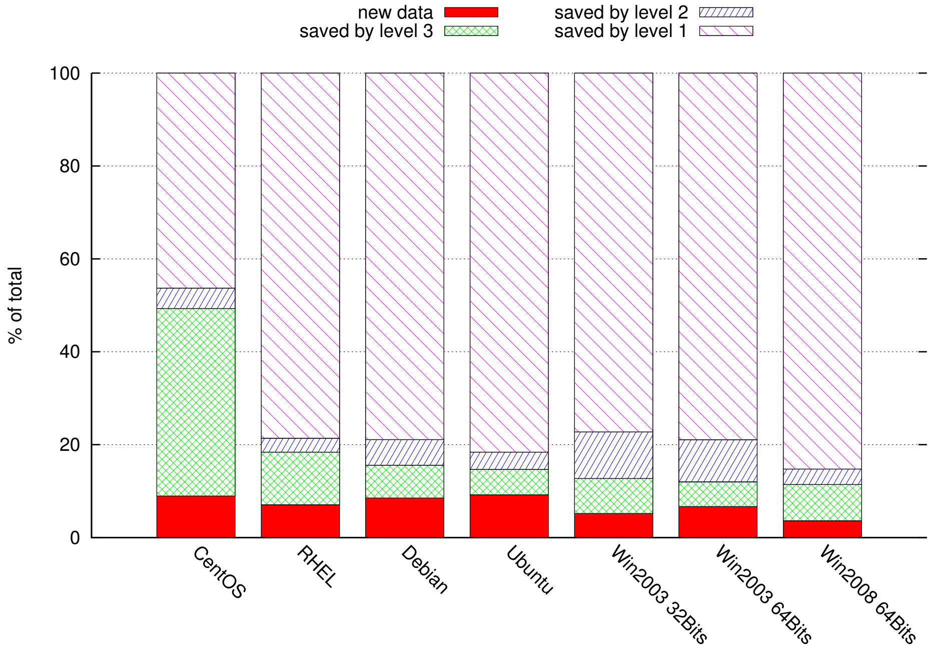


Figure 5.4: Impact of 3-level deduplication for OS releases.

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 5.4 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; “PDS” if this block can be found in PDS; “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 5.4, 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, PDS 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%.

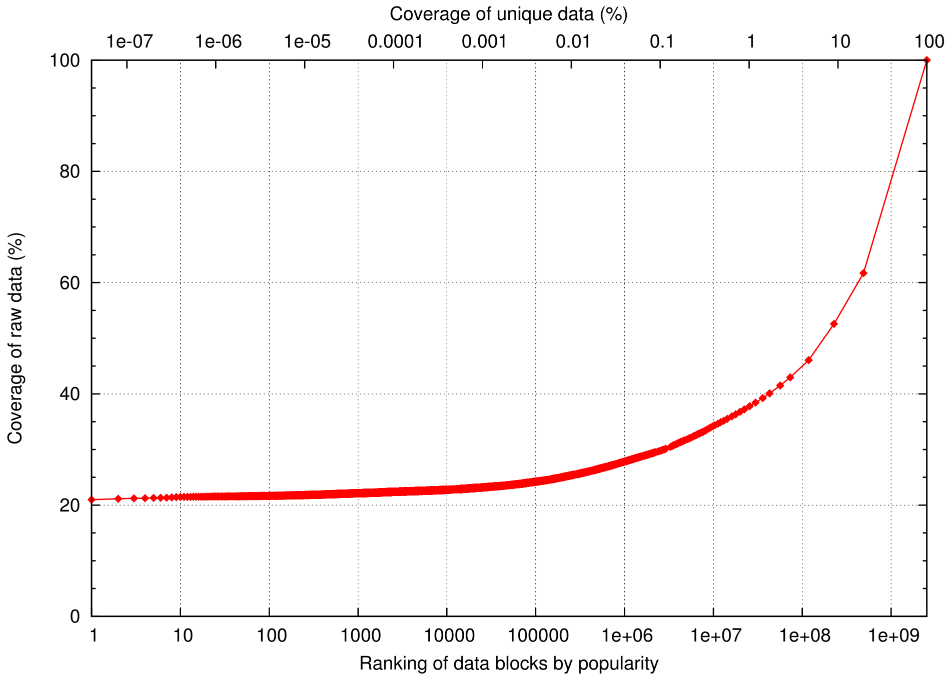


Figure 5.5: Cumulative coverage of popular popular user data blocks.

5.4.3 Coverage of popular data blocks

One of our biggest advantage is small memory footprint for deduplication, because we only eliminate a small amount of highly popular data. we use dataset2 to demonstrate the coverage of popular data blocks on user-related data, as shown in Figure 5.5. The X axis is the rank of popular user data blocks, and Y shows how much raw data can be covered given the size of PDS. Let S_i and F_I be the size and duplicate count of the i-th block ranked by its duplicate rank. Then Y axis is the coverage of the popular dataset covering data items from rank 1 to rank i. Namely

$$\frac{\sum_{i=1}^{i} S_i * F_i}{\text{Total data size}}.$$

Thus with about 1% of blocks on data disks, PDS can cover about 38% of total data blocks appeared in all data snapshots. The corresponding PDS index uses no more than 150 MB memory per machine, which can be easily co-located with other cloud services.

Because there are a limited number of OS releases, we study popular blocks among OS disks loaded with the same OS release. In Figure 5.6, we list a conservative popularity study in 7 major OS versions supported in Aliyun. For every block in the base image, we classify this block as "unchanged" if this block has appeared in all snapshots of the same OS release even they are used by different VMs. Figure 5.6 shows that for each OS, at least 70% of OS blocks are completely unchanged among all snapshots of the same OS. Some latest release of OS tends to have a higher percentage of content change while old release tends to have more variations of content versions. That can be interpreted as that users with very old version of operating systems have a lower tendency to update their OS versions and this causes a larger discrepancy among OS snapshots of these users.

Based on the above analysis, we have selected a small set of most popular OS blocks, which is about 100 GB OS data and its corresponding PDS index takes about 1 GB memory space in total. They can cover sufficiently over 70% of OS-related data.

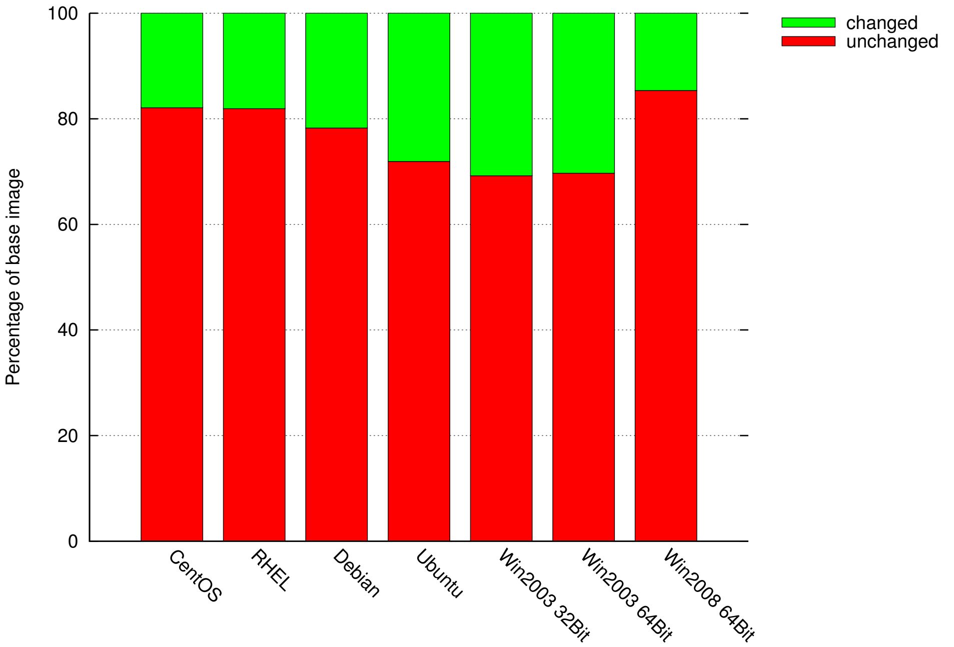


Figure 5.6: Percentage of completely popular blocks among different VMs for the same OS release.

5.4.4 A comparison of perfect and PDS-based deduplication

After level-1 and level 2 elimination, we find that the complete deduplication would reduce the storage cost further by 50%. If we put all these unique data into PDS, we could achieve complete deduplication, but fingerprint lookup in such huge PDS hash index would become a bottleneck as discussed in many pervious works. So we use dataset2 to evaluate how much space saving of deduplication can be achieved when varying the PDS size.

Figure 5.7 shows the relationship between PDS cache size and relative space saving ratio compared to the full deduplication. The unit of PDS size is gigabytes. We define *space saving ratio* as the space saving of PDS method divided by full deduplication space reduction. With a 100 GB PDS data (namely 1 GB PDS index) can still accomplish about 75% of what perfect deduplication can do.

With dataset2, Figure 5.8 shows how PDS space saving ratio compared to the full deduplication is affected by the dataset size. In this experiment we first set out a goal of space saving ratio completed, then watch how much data needs to be placed in PDS cache to achieve this goal. From the graph we can see a 75% saving ratio lead to a stable ratio between PDS size and data size, which requires 1% of data to be placed in PDS.

When we deal with a large cluster of 1,000 nodes, we expect that using 1% of data disks can cover more than what we have seen from this 1323 VM dataset, assuming

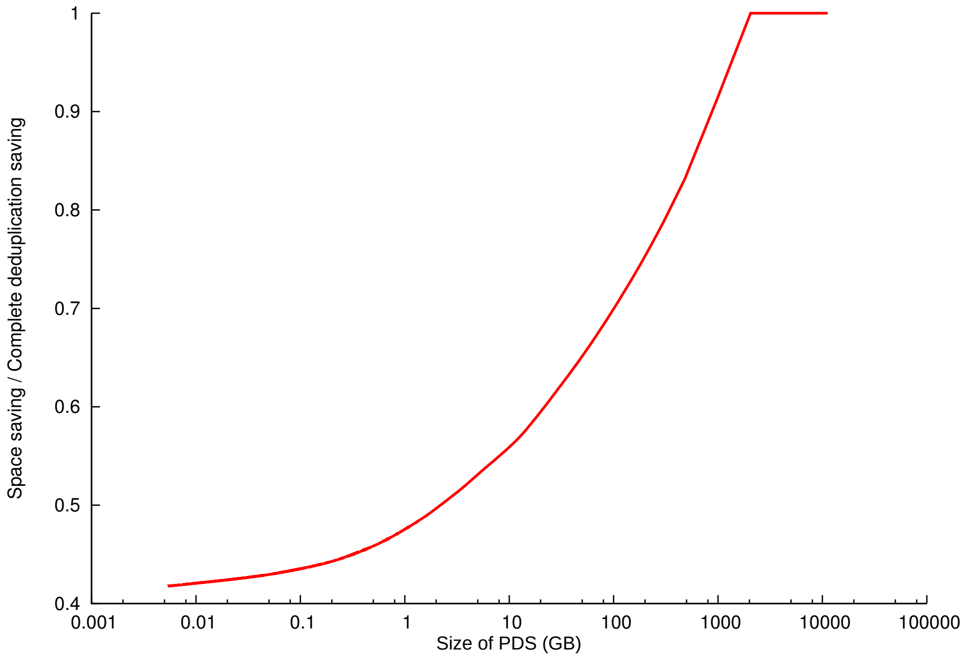


Figure 5.7: Relative ratio of space size compared to full deduplication when PDS size changes.

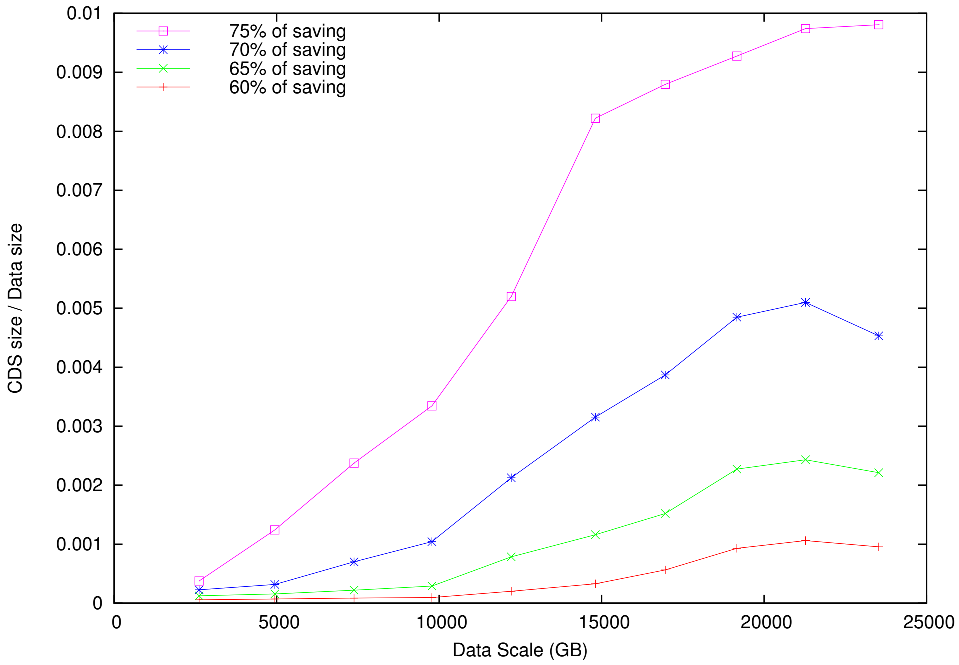


Figure 5.8: The relative size ratio of PDS over the data size.

that the average behavior of every subcluster with 100 machines exhibits a similar popularity. In addition to this, PDS for OS disks will become even more effective when there are more VMs sharing the same collection of OS releases.

5.5 Related Work

In a VM cloud, several operations are provided for creating and managing snapshots and snapshot trees, such as creating snapshots, reverting to any snapshot, and removing snapshots. For VM snapshot backup, file-level semantics are normally not provided. Snapshot operations are taken place at the virtual device driver level, which means no fine-grained file system metadata can be used to determine the changed data. Only raw access information at disk block level are provided.

VM snapshots can be backed up incrementally by identifying file blocks that have changed from the previous version of the snapshot [21, 54, 50]. The main weakness is that it does not reveal content redundancy among data blocks from different snapshots or different VMs.

Data deduplication techniques can eliminate redundancy globally among different files from different users. Backup systems have been developed to use content hash (finger prints) to identify duplicate content [42, 44]. Today's commercial data backup systems (e.g. from EMC and NetApp) use a variable-size chunking algorithm to detect duplicates in file data [35, 23]. As data grows to be big, fingerprint lookup in such schemes becomes too slow to be scalable. Several techniques have been proposed to speedup searching of duplicate content. For example, Zhu et al. [62] tackle it by using an in-memory Bloom filter and prefetch groups of chunk IDs that are likely to be accessed together with high probability. It takes significant memory resource for filtering and caching. NG et al. [39] use a related filtering technique for integrating deduplication in Linux file system and the memory consumed is up to 2 GB for a single machine. That is still too big in our context discussed below.

Duplicate search approximation [12, 34, 57] has been proposed to package similar content in one location, and duplicate lookup only searches for chunks within files which have a similar file-level or segment-level content fingerprints. That leads to a smaller amount of memory usage for storing meta data in signature lookup with a trade-off of the reduced recall ratio.

5.6 Concluding Remarks

In this chapter we propose a multi-level selective deduplication scheme for snapshot service in VM cloud. Similarity based inner-VM deduplication localizes backup data dependency and exposes more parallelism while popularity based cross-VM deduplication with a small popular data set effectively covers a large amount of duplicated data. Our solution accomplishes the majority of potential global deduplication saving while still meets stringent cloud resources requirement. 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. In future we may conduct more field study on large VM clusters to better study the VM data duplication patterns.

Chapter 6

VM-centric Storage Management with Approximate Deletion

6.1 Introduction

In this chapter, we discuss a low-cost storage and management architecture 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 a few shared data chunks can have a broad effect and many snapshots of virtual machines could be affected. The previous work in deduplication focuses on the efficiency and approximation of fingerprint comparison, and has not addressed fault tolerance issues together with deduplication. Thus we also seek deduplication options that yield better fault isolation. Another issue considered is that that garbage collection after deletion of old snapshots also competes for computing resources. Sharing of data chunks among by multiple VMs needs to be detected during garbage collection and such dependencies complicate deletion operations.

The key contribution of this work is the development and analysis of a VM-centric approach which considers fault isolation and integrates multiple duplicate detection strategies supported by similarity guided local deduplication and popularity guided global deduplication. This approach localizes duplicate detection within each VM and packages only data chunks from the same VM into a file system block as much as possible. By narrowing duplicate sharing within a small percent of common data chunks and exploiting their popularity, this scheme can afford to allocate extra replicas of these shared chunks for better fault resilience while sustaining competitive deduplication efficiency. In addition, our VM-centric design allows garbage collection to be performed in a localized scope and we propose an approximate deletion scheme to reduce this cost further. Localization also brings the benefits of greater ability to exploit parallelism so backup operations can run simultaneously without a central bottleneck. This VM-centric solution uses a small amount of memory while delivering reasonable deduplication efficiency.

The rest of this chapter is organized as follows. Section 6.2 reviews the background and discusses the design options for snapshot backup with a VM-centric approach. Section 6.3 describes our system architecture and implementation details. Section 6.4 analyzes the trade-off and benefits of our approach. Section 6.5 describes out approximate deletion algorithm. Section 6.6 is our experimental evaluation that compares with other approaches. Section 6.7 reviews the related works. Section 6.8 concludes this chapter.

6.2 Design Considerations

Our key design consideration is VM dependence minimization during deduplication and file system block management.

- *Deduplication localization.* Because a data chunk is compared with fingerprints collected from all VMs during the deduplication process, only one copy of duplicates is stored in the storage, this artificially creates data dependencies among different VM users. Content sharing via deduplication affects fault isolation since machine failures happen periodically in a large-scale cloud and loss of a small number of shared data chunks can cause the unavailability of snapshots for a large number of virtual machines. Localizing the impact of deduplication can increase fault isolation and resilience. Thus from the fault tolerance point of view, duplicate sharing among multiple VMs is discouraged. Another disadvantage of sharing is that it complicates snapshot deletion, which occurs frequently when snapshots expire regularly. The mark-and-sweep approach [29, 14] is effective for deletion, but still carries a significant cost to count if a data chunk is still shared by other snapshots. Localizing deduplication can minimize data sharing and simplify deletion while sacrificing deduplication efficiency, and can facilitate parallel execution of snapshot operations.

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

Another consideration is the computing cost of deduplication. Because of collocation of this snapshot service with other existing cloud services, cloud providers will want the backup service to only consume small resources with a minimal impact to the existing cloud services. The key resource for signature comparison is memory for storing the fingerprints. We will consider the approximation techniques with reduced memory consumption along with the fault isolation considerations discussed below.

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

We will first present an VM-centric architecture and implementation design with deletion support that can be integrated with our multi-level selective deduplication scheme, then discuss and analyze the integration of the VM-centric deduplication strategies with fault isolation.

6.3 Snapshot Storage Architecture

6.3.1 Components of a Cluster Node

Our VM cloud runs on a cluster of Linux machines with Xen-based VMs and an open-source package for the distributed file system called QFS [40]. 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[55] in Xen).

As depicted in Figure 6.1, there are four key service components running on each cluster node for supporting backup and deduplication: 1) a virtual block device driver, 2) a snapshot deduplication agent, 3) a snapshot store client to store and access snapshot data, and 4) a PDS client to support PDS metadata access.

We use the virtual device driver in Xen that employs a bitmap to track the changes that have been made to the virtual disk (CBT). Every bit in the bitmap represents a fixed-sized (2MB) segment, indicating whether the segment has been modified since last backup. Segments are further divided into variable-sized chunks (average 4KB) using a content-based chunking algorithm [32], 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.

6.3.2 A VM-centric Snapshot Store for Backup Data

We build the snapshot storage on the top of a distributed file system. Following the VM-centric idea for the purpose of fault isolation, each VM has its own snapshot store,

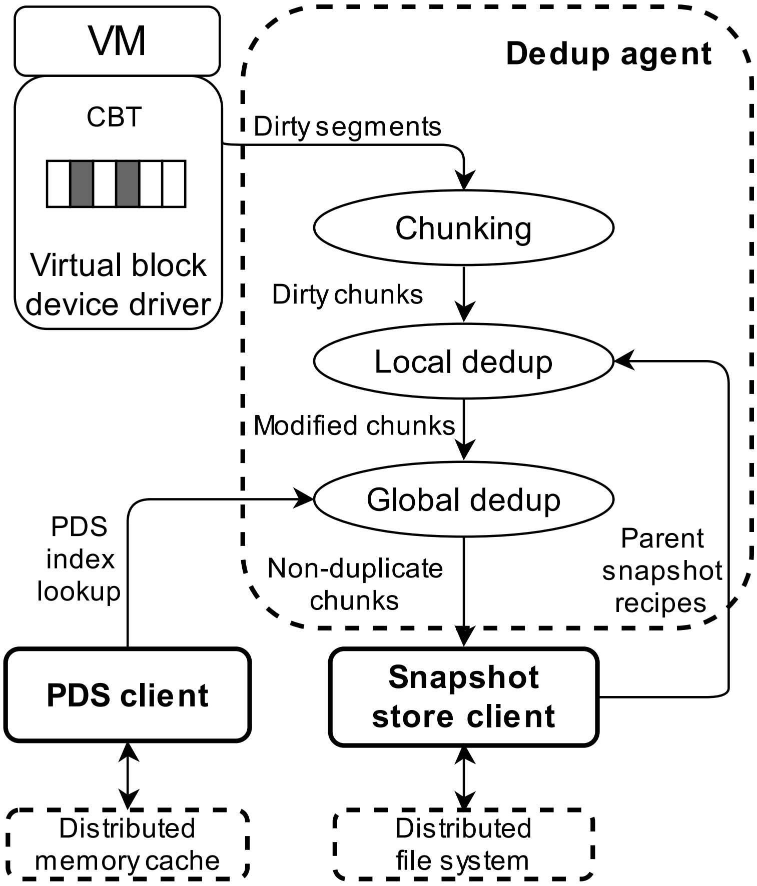


Figure 6.1: System architecture and data flow during snapshot backup

containing new data chunks which are considered to be non-duplicates. As shown in Figure 6.2, we explain the data structure of the snapshot stores as follows.

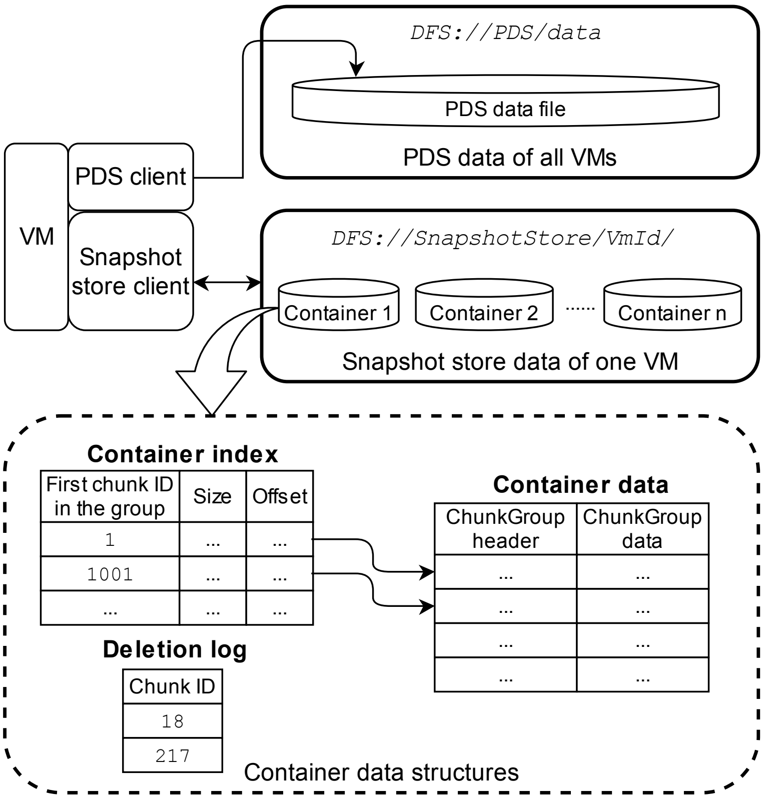


Figure 6.2: Data structure of a VM snapshot store.

There is an independent store containing all PDS chunks shared among different VMs as a single file. Each reference to a PDS data chunk in the PDS index is the offset within the PDS file. Additional compression is not applied because for the data sets we have tested, we only observed limited spatial locality among popular data chunks. On average the number of consecutive PDS index hits is lower than 7. Thus it is not very effective to group a large number of chunks as a compression and data fetch unit.

For the same reason, we decide not to take the sampled index approach [29] 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 [29, 45], retrieval of a data chunk group naturally works well with prefetching. A typical chunk group contains 1000 chunks in our experiment.

Each non-PDS data container is represented by three files in the DFS: 1) the container data file holds the actual content, 2) the container index file is responsible for translating a data reference into its location within a container, and 3) a chunk deletion log file records all the deletion requests within the container.

A non-PDS data chunk reference stored in the index of snapshot recipes is composed of two parts: a container ID with 2 bytes and a local chunk ID with 6 bytes. Each container maintains a local chunk counter and assigns the current number as a chunk ID when a new chunk is added to this container. Since data chunks are always appended to a snapshot store during backup, local chunk IDs are monotonically increasing. When a chunk is to be accessed, the segment recipe contains a reference pointing to a data chunk in the PDS store or in a non-PDS VM snapshot store. Using a container ID, the corresponding container index file of this VM is accessed and the chunk group is identified using a simple chunk ID range search. Once the chunk group is loaded to memory, its header contains the exact offset of the corresponding chunk ID and the content is then accessed from the memory buffer.

Our snapshot store supports three API calls for block data operations:

Append(). For PDS data, the chunk is appended to the end of the PDS file and the offset is returned as the reference. Note that PDS append may only be used during PDS recalculation. For non-PDS data, this call places a chunk into the snapshot store and returns a reference to be stored in the recipe metadata of a snapshot. The write requests to append data chunks to a VM store are accumulated at the client side. When the number of write requests reaches a fixed group size, the snapshot store client compresses the accumulated chunk group, adds a chunk group index to the beginning of the group, and then appends the header and data to the corresponding VM file. A new container index entry is also created for each chunk group and is written to the corresponding container index file.

Get(). The fetch operation for the PDS data chunk is straightforward since each reference contains the file offset, and the size of a PDS chunk is available from a segment recipe. We also maintain a small data cache for the PDS data service to speedup common data fetching. To read a non-PDS chunk using its reference with container ID and local chunk ID, the snapshot store client first loads the corresponding VM's container index file specified by the container ID, then searches the chunk groups using their chunk ID coverage. After that, it reads the identified chunk group from DFS, decompresses it, and seeks to the exact chunk data specified by the chunk ID. Finally, the client updates its internal chunk data cache with the newly loaded content to anticipate future sequential reads.

Delete(). Chunk deletion occurs when a snapshot expires or gets deleted explicitly by a user and we discuss this in more details in next subsection. When deletion requests are issued for a specific container, those requests are simply recorded into the container's deletion log initially and thus a lazy deletion strategy is exercised. Once local chunk IDs appear in the deletion log, they will not be referenced by any future snapshot and can be safely deleted when needed. This is ensured because we only dedup against the direct parent of a snapshot, so the deleted snapshot's blocks will only be used if they also exist in other snapshots. Periodically, the snapshot store identifies those containers with an excessive number of deletion requests to compact and reclaim the corresponding disk space. During compaction, the snapshot store creates a new container (with the same container ID) to replace the existing one. This is done by sequentially scanning the old container, copying all the chunks that are not found in the deletion log to the new container, and creating new chunk groups and indices. Every local chunk ID however is directly copied rather than re-generated. This process leaves holes in the chunk ID values, but preserves the order and IDs of chunks. As a result, all data references stored in recipes are permanent and stable, and the data reading process is as efficient as before. Maintaining the stability of chunk IDs also ensures that recipes do not depend directly on physical storage locations, which simplifies data migration.

Additional APIs such as **Scan()**, **Compact()**, **Create()** and **Remove()** are provided for higher level operations. Scan() is used by the map-reduce procedure to collect the most popular blocks among all snapshot stores. Compact() is called when the system determines a container has too much unclaimed space and needs to be reclaimed. Create() and Remove() initialize and delete a container respectively.

6.4 Analysis of VM-centric Approach

In this section we give out analysis of the impacts of our VM-centric deduplication scheme. As introduced in chapter 5.3, our VM-centric approach put the cluster wide popular data into PDS which prevent each VM from keeping their own copies. We will show that by adding extra replication to PDS data, this strategy brings advantages to both fault tolerance and deduplication efficiency. The parameters we will use in our analysis below are defined in Table 6.1.

6.4.1 Impact on Deduplication Efficiency

Choosing the value *k* for the most popular chunks affects the deduplication efficiency. We analyze this impact based on the characteristics of the VM snapshot traces

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

Table 6.1: Modeling parameters

studied from application datasets. A previous study shows that the popularity of data chunks after local deduplication follows a Zipf-like distribution[16, 9] and its exponent α is ranged between 0.65 and 0.7 [60]. Figure 5.1 illustrates the Zipf-like distribution of chunk popularity.

By Zipf-like distribution, $f_i = f_1/i^{\alpha}$. The total number of chunks in our backup storage which has local duplicates excluded is $c(1-\delta)$, this can be represented as the sum of each unique fingerprint times its frequency:

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

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

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

Thus putting the k most popular fingerprints into PDS index can remove the following number of chunks during global deduplication:

$$f_1 \sum_{i=1}^{k} \frac{1}{i^{\alpha}} \approx f_1 \int_1^k \frac{1}{x^{\alpha}} dx \approx f_1 \frac{k^{1-\alpha}}{1-\alpha} = c(1-\delta)\sigma^{1-\alpha}.$$

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

$$E_c = \frac{c\delta + c(1-\delta)\sigma^{1-\alpha}}{c - c_u}. \tag{6.1}$$

We store the PDS index using a distributed shared memory hash table such as Memcached and allocate a fixed percentage of memory space per physical machine for top

k popular items. As the number of physical machines (p) increases, the entire cloud cluster can host more VMs; however, ratio σ which is k/c_u remains a constant because each physical machine on average still hosts a fixed constant number of VMs. Then the overall deduplication efficiency of VC defined in Formula 6.1 remains constant. Thus the deduplication efficiency is stable as p increases as long as σ is a constant.

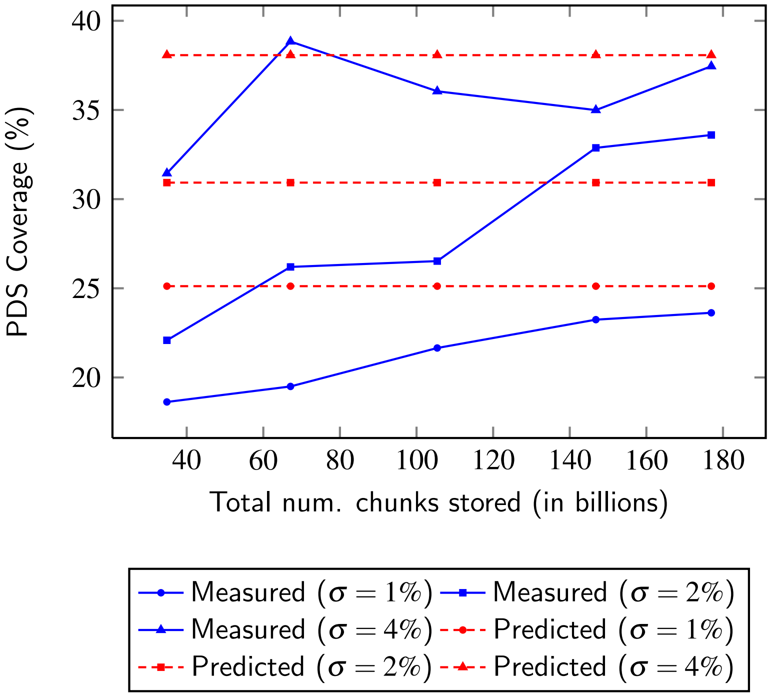
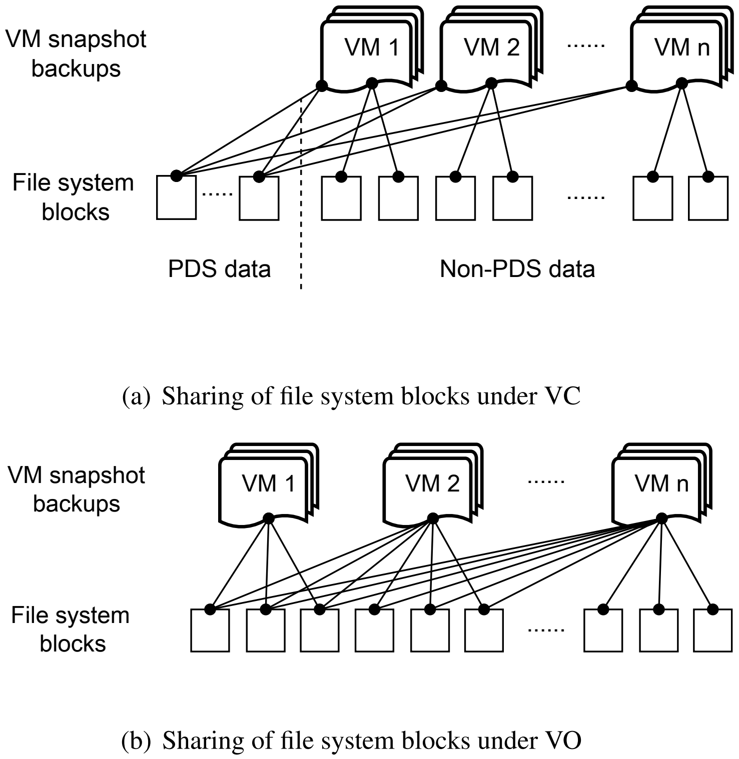


Figure 6.3: Predicted vs. actual PDS coverage as data size increases.

Ratio $\sigma^{1-\alpha}$ represents the percentage of the remaining chunks detected as duplicates in global deduplication due to PDS. We call this PDS coverage. Figure 6.3 shows predicted PDS coverage using $\sigma^{1-\alpha}$ when α is fixed at 0.65 and measured PDS coverage in our test dataset. $\sigma = 2\%$ represents memory usage of approximately 100MB



(a) Sharing of file system blocks under VC

(b) Sharing of file system blocks under VO

Figure 6.4: Bipartite association of VMs and file system blocks under (a) VC and (b) VO.

memory per machine for the PDS. While the predicted value remains flat, measured PDS coverage increases as more VMs are involved. This is because the actual α value increases with the data size.

6.4.2 Impact on Fault Isolation

The replication degree of the backup storage is r for regular file system blocks and $r = 3$ is a typical setting in distributed file systems [27, 46]. Since σ is small (e.g. 2%

in our experiments), the impact of replication on storage increase is very small even when choosing r_c/r ratio as 2 or 3.

Now we assess the impact of losing d machines to the VC and VO approaches. A large r_c/r ratio can have a positive impact on full availability of VM snapshot blocks. We use an FSB rather than a deduplication data chunk as our unit of failure because the DFS keeps file system blocks as its base unit of storage. To compute the full availability of all snapshots of a VM, we derive the probability of losing a snapshot FSB of a VM by estimating the number of file system blocks per VM in each approach. As illustrated in Figure 6.4, we build a bipartite graph representing the association from unique file system blocks to their corresponding VMs in each approach. An association edge is drawn from an FSB to a VM if this block is used by the VM.

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

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

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

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

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

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

When E_c is close to E_o, N_1 is much smaller than N_o. Figure 6.5 shows the average number of file system blocks for each VM in VC and in VO and N_1 is indeed much smaller than N_o in our tested dataset.

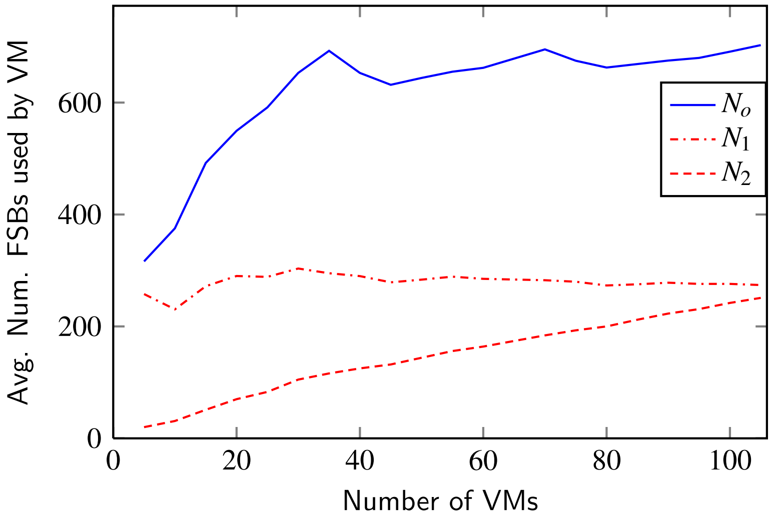


Figure 6.5: Measured average number of 64MB FSBs used by a single VM. For VC both the number of PDS and Non-PDS FSBs used are shown.

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

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

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

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

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

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

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

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

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

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

We have considered a worst case scenario that every PDS FSB is shared by all VMs in the VC approach, which leads to a large N_2 value. Even with that, the availability of VC

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

snapshots is still much higher than VO and there are two reasons for this: 1) N_1 is much smaller than N_o as discussed previously. 2) $A(r) < A(r_c)$ because $r < r_c$. Table 6.2 lists the $A(r)$ values with different replication degrees, to demonstrate the gap between $A(r)$ and $A(r_c)$.

6.5 Approximate Snapshot Deletion with Leak Repair

In a busy VM cluster, snapshot deletions can occur frequently. Deduplication complicates the deletion process because space saving relies on the sharing of data and it requires the global references to deleted chunks to be identified before they can be safely removed. The complexity of our distributed environment obviates reference counting as an option, and while the mark-and-sweep techniques can be used and optimization can be considered [14], it still takes significant resources to conduct reference counting every time there is a snapshot deletion. We seek a fast solution with low resource usage

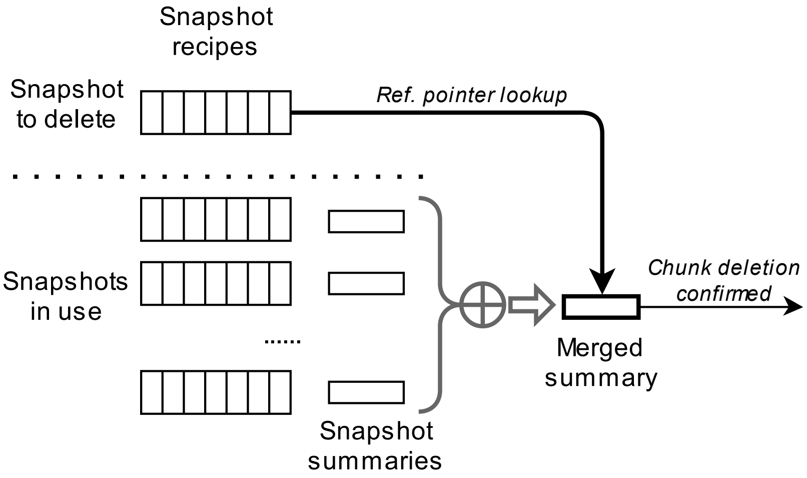


Figure 6.6: Approximate deletion merges existing snapshot summaries to check block reference validity contained by a deleted snapshot

to delete snapshots and our VM-centric design simplifies the deletion process. Since PDS is small and separated, we can focus on unreferenced non-PDS chunks within each VM. Another resource-saving strategy we propose is an *approximate* deletion strategy to trade deletion accuracy for speed and resource usage. Our method sacrifices a small percent of storage leakage to efficiently identify unused chunks.

We depict our approximate deletion process in Fig.6.6, this procedure contains three aspects.

- **Computation for snapshot reference summary.** Every time there is a new snapshot created, we compute a Bloom-filter with z bits as the reference summary vector for all non-PDS chunks used in this snapshot. The items we put into the summary vector are all the references appearing in the metadata of the snapshot.

For each VM we preset the vector size according to estimation of VM image size, given h snapshots stored for a VM, there are h summary vectors maintained. We adjust the summary vector size and recompute the vectors if the VM size changes substantially over time. This can be done during the periodic leakage repair stage described below.

- **Approximate deletion with fast summary comparison.** When there is a snapshot deletion, we need to identify if chunks to be deleted from that snapshot are still referenced by other snapshots. This is done approximately and quickly by comparing the reference of deleted chunks with the merged reference summary vectors of other live snapshots. The merging of live snapshot Bloom-filter vectors uses the bitwise OR operator and the merged vector still takes z bits. Since the number of live snapshots h is limited for each VM, the time and memory cost of this comparison is small, linear to the number of chunks to be deleted.

  If a chunk's reference is not found in the merged summary vector, we are sure that this chunk is not used by any live snapshots, thus it can be deleted safely. However, among all the chunks to be deleted, there are a small percentage of unused chunks which are misjudged as being in use, resulting in storage leakage.

- **Periodic repair of leakage**. Leakage repair is conducted periodically to fix the above approximation error. This procedure compares the live chunks for each

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

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

$$U = u + (h-1)\Delta u.$$

Each Bloom filter vector has z bits for each snapshot and let j be the number of hash functions used by the Bloom filter. Notice that a chunk may appear multiple times in these summary vectors; however, this should not increase the probability of being a 0 bit in all h summary vectors. Thus the probability that a particular bit is 0 in all h

summary vectors is $(1-\frac{1}{z})^{jU}$. Then the misjudgment rate of being in use is:

$$\varepsilon = (1-(1-\frac{1}{z})^{jU})^{j}. \tag{6.2}$$

For each snapshot deletion, the number of chunks to be deleted is nearly identical to the number of newly added chunks Δu. Let R be the total number of runs of approximate deletion between two consecutive repairs. We estimate the total leakage L after R runs as:

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

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

$$\frac{L}{U} = \frac{R\Delta u\varepsilon}{u+(h-1)\Delta u} > \tau \Longrightarrow R > \frac{\tau}{\varepsilon} \times \frac{u+(h-1)\Delta u}{\Delta u}. \tag{6.3}$$

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 6.2, each summary vector requires $z = 10U = 122.5$ million bits or 15MB. From Equation 6.3, 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.

6.6 System Implementation and Experimental Evaluations

We have implemented and evaluated a prototype of our VC scheme on a Linux cluster of machines with 8-core 3.1Ghz AMD FX-8120 and 16 GB RAM. Our implementation is based on Alibaba cloud platform [1, 60] 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 [22, 12]. SRB executes a distributed deduplication by routing a data chunk to one of cluster machines [22] using a min-hash function discussed in [12]. Once a data chunk is routed to a machine, the chunk is compared with the fingerprint index within this machine locally.

6.6.1 Settings

We have performed a trace-driven study using a production dataset [60] from Alibaba Aliyun's cloud platform with about 100 machines. Each machine hosts up to 25 VMs and each VM keeps 10 automatically-generated snapshots in the storage system while a user may instruct extra snapshots to be saved. The VMs of the sampled data set use popular operating systems such as Debian, Ubuntu, Redhat, CentOS, win2008 and win2003. Based on our study of production data, each VM has about 40GB of storage data on average including OS and user data disk. The fingerprint for variable-sized chunks is computed using their SHA-1 hash [35, 43].

6.6.2 Fault Isolation and Snapshot Availability

Table 6.3 shows the availability of VM snapshots when there are up to 20 machine nodes failed in a 100-node cluster and a 1000-node cluster. We have assumed a worst-case scenario that a PDS block is shared by every VM. Our results show that even the worst case, VC still has a significantly higher availability than VO as the number of failed machines increases. For example, with 5/100 machines failed and 25 VMs per machine, VO with 93.256% availability would lose data in 169 VMs while VC with 97.763% loses data for 56 VMs. The key reason is that for most data in VC, only a single VM can be affected by the loss of a single FSB. Since most FSBs contain chunks for a single VM, VMs can depend on a smaller number of FSBs.

Although the loss of a PDS block affects many VMs, by increasing replication for those blocks we minimize the effect on VM snapshot availability. Figure 6.7 shows the impact of increasing PDS data replication. While the impact on storage cost is small, a replication degree of 6 has a significant improvement over 4, but the availability is about the same for $r_c = 6$ and $r_c = 9$ (beyond $r_c = 6$ improvements are minimal).

Failures (d)	VM Snapshot Availability(%)			
	$p = 100$		$p = 1000$	
	VO	VC	VO	VC
3	99.304248	99.773987	99.999321	99.99978
5	93.256135	97.762659	99.993206	99.997798
10	43.251892	76.093998	99.918504	99.97358
20	0.03397	5.613855	99.228458	99.749049

Table 6.3: Availability of VM snapshots for VO and VC.

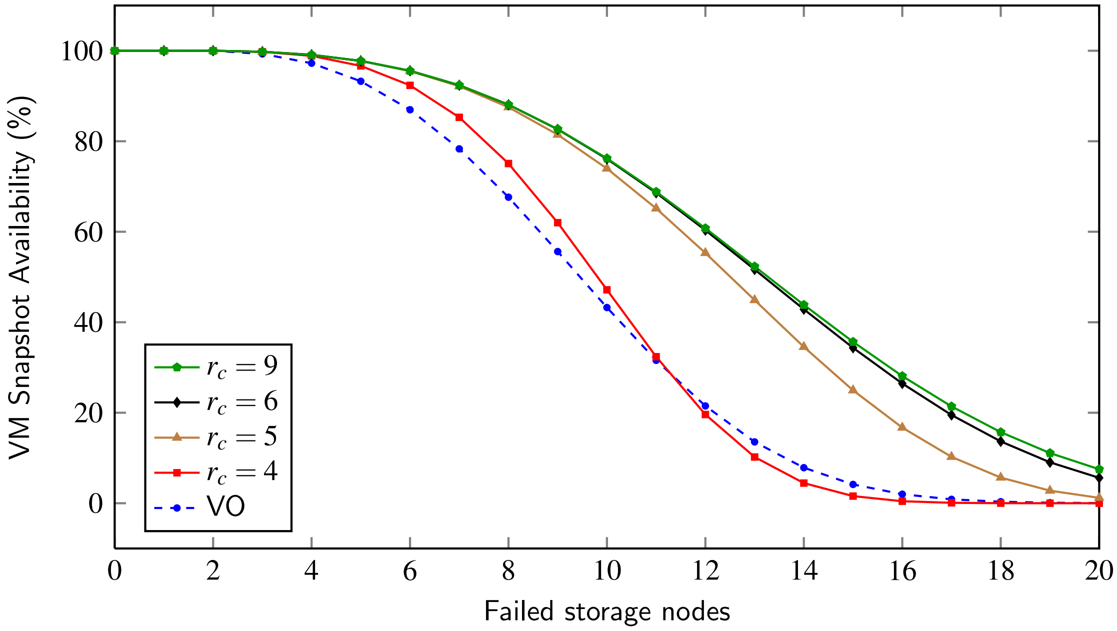


Figure 6.7: Availability of VM snapshots in VC with different PDS replication degrees

6.6.3 Deduplication Efficiency

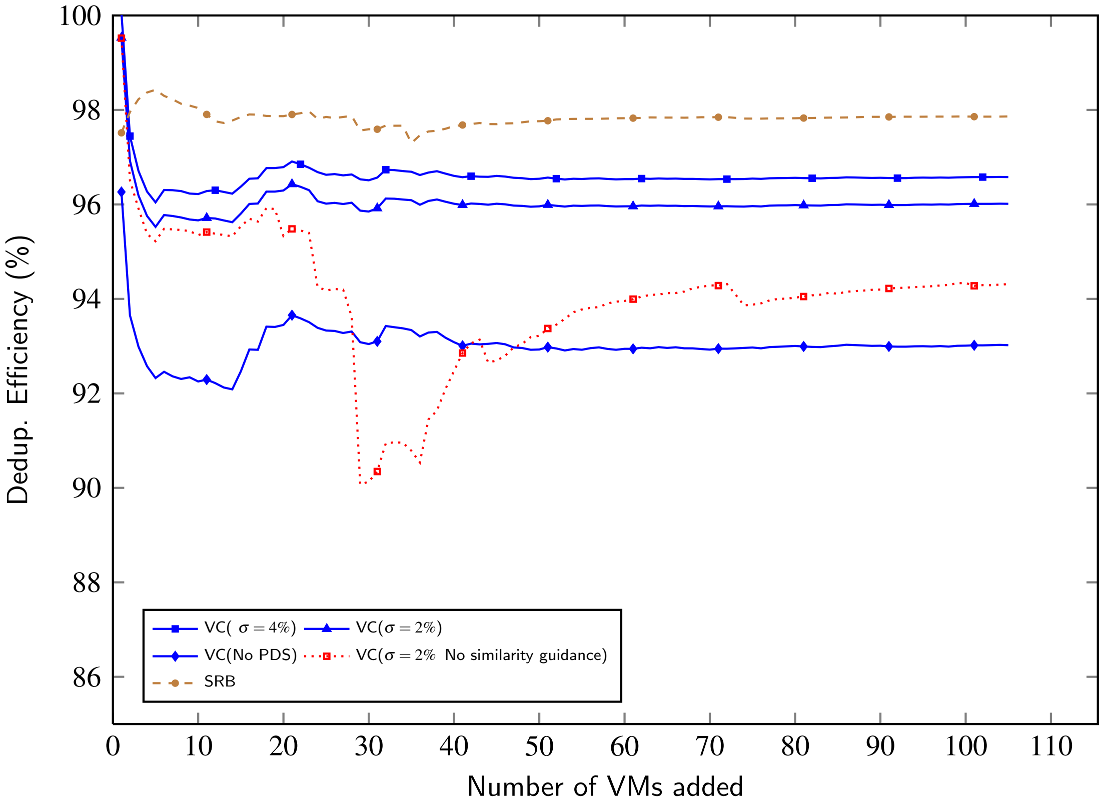


Figure 6.8: Deduplication efficiency of VC and SRB.

Figure 6.8 shows the deduplication efficiency for SRB and VC, namely the percent of duplicate chunks which are detected and removed. With $\sigma = 2\%$, memory usage for PDS index lookup per machine is about 100MB per machine and the deduplication efficiency can reach over 96.33%. When $\sigma = 4\%$, the deduplication efficiency can reach 96.9% while space consumption increases to 200MB per machine. The loss of efficiency in VC is caused by the restriction of the physical memory available in the cluster for fast in-memory PDS index lookup. SRB can deliver up to 97.79% deduplication efficiency, which is slightly better than VC. Thus this represents a trade-off that VC provides better fault tolerance and fast approximate deletion with competitive deduplication efficiency.

Figure 6.8 also shows the curve of VC without local similarity search. There is a big efficiency drop in this curve when the number of VMs is about 30. The reason is that there are VMs in which data segments are moved to another location on disk, for example when a file is rewritten rather than modified in place, a dirty-bit or offset based detection would not be able to detect such a movement and similarity search becomes especially important. We have found that in approximately 1/3 of the VMs in our dataset this movement happens frequently. In general, adding local similarity-guided search increases deduplication efficiency from 93% to over 96%. That is one significant improvement compared to the work in [60] which uses the parent segment at the same offset to detect duplicates instead of similarity-guided search.

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

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

Table 6.4: Resource usage of concurrent backup tasks at each machine

6.6.4 Resource Usage and Processing Time

Storage cost of replication. When the replication degree of both PDS and non-PDS data is 3, the total storage for all VM snapshots in each physical machine takes about 3.065TB on average before compression and 0.75TB after compression. Allocating one extra copy for PDS data only adds 7GB in total per machine. Thus PDS replication degree 6 only increases the total space by 0.685% while PDS replication degree 9 adds 1.37% space overhead, which is still small.

Memory and disk bandwidth usage with multi-VM processing. We have further studied the memory and disk bandwidth usage when running concurrent VM snapshot backup on each machine with $\sigma = 2\%$. Table 6.4 gives the resource usage when running 1 or multiple VM backup tasks at the same time on each physical machine. "CPU" column is the percentage of a single core used. "Mem" column includes 100MB memory usage for PDS index and other space cost for executing deduplication tasks such as receipt metadata and cache. “Read” column is controlled as 50MB/s bandwidth usage with I/O throttling so that other cloud services are not impacted too much. The peak raw storage read performance is about 300MB/s and we only use 16.7% with this collocation consideration. “Write” column is the I/O write usage of QFS and notice that each QFS write triggers disk writes in multiple machines due to data replication. 50MB/s dirty segment read speed triggers about 16.4MB/s disk write for non duplicates with one backup task.

Table 6.4 shows that a single backup task per node can complete the backup of the entire VM cluster in about 1.31 hours. Since there are about 25 VMs per machine, we could execute more tasks in parallel at each machine. But adding more backup concurrency does not shorten the overall time significantly because of the controlled disk read time.

Processing Time breakdown. Figure 6.9 shows the average processing time of a VM segment under VC and SRB. VC uses $\sigma = 2\%$ and 4%. It has a breakdown of processing time. “Snapshot read/write” includes snapshot reading and writing from disk, and updating of the metadata. “ Network transfer” includes the cost of transferring raw and meta data from one machine to another during snapshot read and write. “Index access/comparison” is the disk, network and CPU time during fingerprint comparison. This includes PDS data lookup for VC and index lookup from disk in VO after Bloom filter lookup. For VC, the change of σ does not significantly affect the overall backup speed as PDS lookup takes only a small amount of time. The network transfer time for VC and SRB is about the same, because the amount of raw data they transfer is comparable. SRB spends slightly more time for snapshot read/write because during each snapshot backup, SRB involves many small bins, while VC only involves few containers with a bigger size. Thus, there are more opportunities for I/O aggregation in VC to reduce seek time. SRB has a higher cost for index access and fingerprint comparison because most of chunk fingerprints are routed to remote machines for comparison while VC handles most of chunk fingerprints locally.

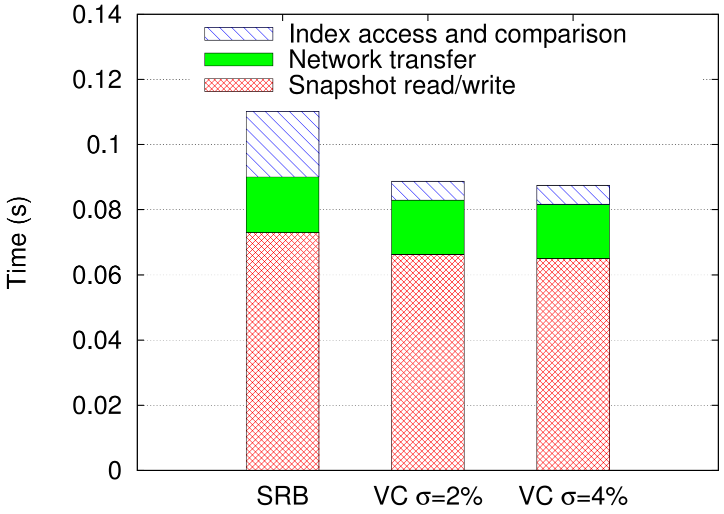


Figure 6.9: Average time to backup a dirty VM segment under SRB and VC

Throughput of software layers. Table 6.5 shows the average throughput of software layer when when I/O throttling is not applied to control the usage. “Backup”

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 6.5: Throughput of software layers per machine under different concurrency

column is the throughput of the backup service per machine. “Snapshot store” is the write throughput of the snapshot store layer. The gap between this column and “Backup” column is caused by significant data reduction by dirty bit and duplicate detection. Only non-duplicate chunks trigger a snapshot store write. “QFS” column is the write request traffic to the underlying file system after compression. For example, with 148MB/second write traffic to the snapshot store, QFS write traffic is about 35.3MB/second after compression. The underlying disk storage traffic will be three times greater with replication. The result shows that the backup service can deliver up to 5.46GB/second without I/O restriction per machine with 6 concurrent backup tasks. With a higher disk storage bandwidth available, the above backup throughput would be higher.

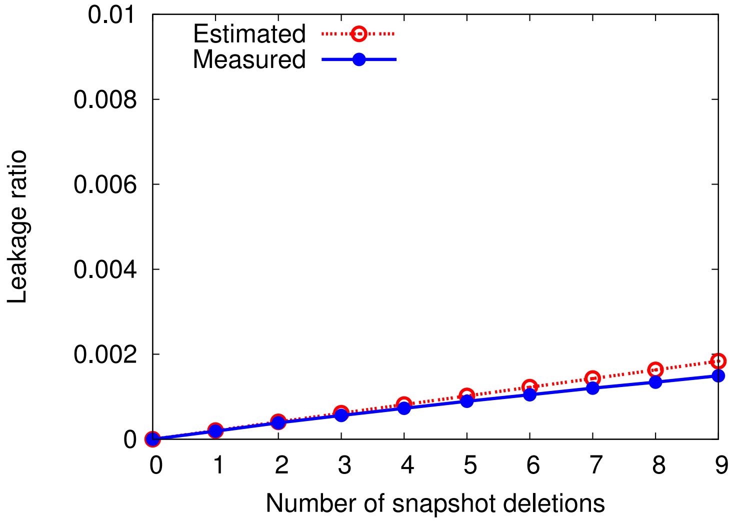


Figure 6.10: Accumulated storage leakage by approximate snapshot deletions ($\Delta u/u = 0.025$)

6.6.5 Effectiveness of Approximate Deletion

Figure 6.10 shows the average accumulated storage leakage in terms of percentage of storage space per VM caused by approximate deletions. The top dashed line is the predicted leakage using Formula 6.3 from Section 6.5 given $\Delta u/u = 0.025$, while the solid line represents the actual leakage measured during the experiment. The Bloom filter setting is based on $\Delta u/u = 0.025$. After 9 snapshot deletions, the actual leakage ratio reaches 0.0015 and this means that there is only 1.5MB space leaked for every 1GB of stored data. The actual leakage can reach 4.1% after 245 deletions.

6.7 Related Work

Since it is expensive to compare a large number of chunk signatures for deduplication, several techniques have been proposed to speedup searching of duplicate fingerprints. For example, the data domain method [62] 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 [56, 58, 36]. The approximation techniques are studied in [12, 29, 60] to reduce memory requirements with the trade-off of a reduced deduplication ratio. Additional inline deduplication techniques are studied in [34, 29, 48] and a parallel batch solution for cluster-based deduplication is studied in [61]. 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 work.

In designing a VC duplication solution, we have considered and adopted some of the following previously-developed techniques. 1) *Changed block Tracking*. VM snapshots can be backed up incrementally by identifying data segments that have changed from the previous version of the snapshot [21, 54, 50, 47]. Such a scheme is VM-centric since deduplication is localized. We are seeking for a trade-off since global signature comparison can deliver additional compression [29, 22, 12]. 2) *Stateless Data Routing*. One approach for scalable duplicate comparison is to use a content-based hash partitioning algorithm called stateless data routing by Dong et al. [22] Stateless data routing divides the deduplication work with a similarity approximation. This work is similar to Extreme Binning by Bhagwat et al. [12] and each request is routed to a machine which holds a Bloom filter or can fetch on-disk index for additional comparison. While this approach is VM-oblivious, it motivates us to use a combined signature of a dataset to narrow VM-specific local search. 3) *Sampled Index.* One effective approach that reduces memory usage is to use a sampled index with prefetching, proposed by Guo and Efstathopoulos[29]. 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.

6.8 Concluding Remarks

In this chapter we propose a collocated backup service built on the top of a cloud cluster to reduce network traffic and infrastructure requirements. The key contribution is a VM-centric data management architecture that integrate with multi-level deduplication to maximize fault isolation while delivering competitive deduplication efficiency. Similarity guided local search reduces cross-VM data dependency and exposes more parallelism while global deduplication with a small common data set eliminates popular duplicates. VM-specific file block packing also enhances fault tolerance by reducing data dependencies. The design places a special consideration for low-resource usages as a collocated cloud service. Evaluation using VM backup data shows that VC strikes a trade-off and can accomplish 96.33% or 96.9% of what complete global deduplication can do. The availability of snapshots increases substantially with a small replication overhead for popular inter-VM chunks.

We consider our approach has the potential to become a general purpose storage backend build into the cloud to serve dropbox-like general file system backups.

Chapter 7

Conclusion and Future Works

Deduplication in storage backup with an ever-increasing demand faces scalability and throughput challenges. This dissertation investigates scalable techniques in building a VM snapshot storage system that provides low-cost deduplication support for large VM clouds. In particular, our work focuses on three specific aspects of VM snapshot deduplication: a synchronous solution for batched duplicate detection, a multi-level selective approach for inline deduplication, and data management on the top of cloud file system with an emphasis on fault tolerance and garbage collection. Our system has been implemented on Xen-based Linux VM clusters.

This dissertation begins with studying a low-cost multi-stage parallel deduplication solution for synchronous backup. We split the original inline deduplication process into different stages to facilitate parallel batch processing. We first performing parallel duplicate detection for VM content blocks among all machines before performing actual data backup. Each machine accumulates detection requests and then performs detection partition by partition with a minimal resource usage. Fingerprint based partitioning allows highly parallel duplicate detection and also simplifies reference counting management. Because the design separation of duplicate detection and actual backup, we are able to conduct distributed fingerprint comparison in parallel among clustered machine nodes, and only load one partition at time at each machine for in-memory comparison. This makes the proposed scheme very resource-friendly to the existing cloud services.

We have also presented a multi-level selective deduplication scheme for on-demand snapshot backup service. Our approach utilizes similarity guided inner-VM deduplication to localize backup data dependence and exposes more parallelism. Meanwhile cross-VM deduplication uses a small popular data set to effectively cover a large amount of content blocks. Our solution achieves the majority of what perfect global deduplication can accomplish while meeting stringent resource requirements. Compared to the existing snapshot techniques, our scheme reduces about two-third of snapshot storage cost. Finally, our scheme uses a very small amount of memory on each node, and leaves room for employing other optimization techniques.

The key contribution of our VM-centric data management is VM-specific file block packing that enhances fault tolerance by maximizing data localization and reducing cross VM sharing. In addition, such a design allows garbage collection to be performed in a localized scope and we propose an approximate deletion scheme to further reduce the cost. As supported by theoretical analysis and experimental studies, the availability of VM snapshots increases substantially with a small replication overhead for popular inter-VM chunks.

The design of our VM-centric deduplication approach has a potential of being applied to general purpose cloud backup. The idea of collecting popular data becomes more interesting based on in data commonality among users. However, the access to popular data could be a bottleneck in the I/O path if not designed carefully, these issues remain to be solved.

Bibliography

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

[2] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[3] ZFS. http://www.sun.com/software/solaris/zfs.jsp.

[4] L. A. Adamic and B. A. Huberman. Zipfs law and the Internet. *Glottometrics*, 3(1):143–150, 2002.

[5] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *Trans. Storage*, 3(3):9, 2007.

[6] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu. StoreGPU: exploiting graphics processing units to accelerate distributed storage systems. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 165–174, New York, NY, USA, 2008. ACM.

[7] C. Alvarez. NetApp Deduplication for FAS and V-Series Deployment and Implementation Guide. NetApp. Technical Report TR-3505 , 2011.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb. 2009.

[9] S. K. Baek, S. Bernhardsson, and P. Minnhagen. Zipf's law unzipped. *New Journal of Physics*, 13(4):043004, Apr. 2011.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177, Oct. 2003.

[11] J. a. Barreto and P. Ferreira. Efficient file storage using content-based indexing. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–9, New York, NY, USA, 2005. ACM.

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

[13] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese. Beyond bloom filters. *ACM SIGCOMM Computer Communication Review*, 36(4):315, Aug. 2006.

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

[15] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *IN INFOCOM*, pages 126 – 134, 1999.

[16] L. Breslau, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, pages 126–134 vol.1. IEEE, 1999.

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

[18] A. Z. Broder. Identifying and Filtering Near-Duplicate Documents. In *COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag.

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

[20] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows azure storage: A highly available cloud storage

service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

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

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

[23] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. HYDRAstor: a Scalable Secondary Storage. In *FAST '09: Proccedings of the 7th conference on File and storage technologies*, pages 197–210, Berkeley, CA, USA, 2009. USENIX Association.

[24] EMC. Achieving storage efficiency through EMC Celerra data deduplication. White Paper, 2010.

[25] K. Eshghi, M. Lillibridge, L. Wilcock, G. Belrose, and R. Hawkes. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 123–138, Berkeley, CA, USA, 2007. USENIX Association.

[26] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004:5–, Aug. 2004.

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

[28] H. S. Gunawi, N. Agrawal, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and J. Schindler. Deconstructing Commodity Storage Clusters. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 60–71, Washington, DC, USA, 2005. IEEE Computer Society.

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

[30] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei. An empirical analysis of similarity in virtual machine images. In *Proceedings of the Middleware*

2011 Industry Track Workshop on - Middleware '11, pages 1–6, New York, New York, USA, Dec. 2011. ACM Press.

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

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

[33] E. K. Lee and C. A. Thekkath. Petal: distributed virtual disks. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 84–92, New York, NY, USA, 1996. ACM.

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

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

[36] C. Marshall. Efficient and Safe Data Backup with Arrow. Master's thesis, University of California, Santa Cruz, June 2008.

[37] A. Muthitacharoen, B. Chen, and D. Mazières. A Low-bandwidth Network File System. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 174–187, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[38] P. Nath, B. Urgaonkar, and A. Sivasubramaniam. Evaluating the usefulness of content addressable storage for high-performance data intensive applications. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 35–44, New York, NY, USA, 2008. ACM.

[39] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. C. Lee, and J. C. S. Lui. Live deduplication storage of virtual machine images in an open-source cloud. In *Middleware*, pages 81–100, 2011.

[40] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, J. Kelly, C. Zimmermanand, D. Adkins, T. Subramaniam, and J. Fishman. The quantcast file system.

In *Proceedings of the 39th international conference on Very Large Data Bases*, VLDB'13, pages 2–2, Berkeley, CA, USA, 2013. VLDB.

[41] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems data. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, page 6, Berkeley, CA, USA, 2004. USENIX Association.

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

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

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

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

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

[47] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger. Metadata Efficiency in Versioning File Systems. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 43–58, Berkeley, CA, USA, 2003. USENIX Association.

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

[49] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM*, pages 149–160, 2001.

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

[51] J. C. Tang, C. Drews, M. Smith, F. Wu, A. Sue, and T. Lau. Exploring patterns of social commonality among file directories at work. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 951–960, New York, NY, USA, 2007. ACM.

[52] A. Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, The Australian National University, 1999.

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

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

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

[56] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: A scalable high-throughput exact deduplication approach for network backup services. In *IEEE MSST'10*, pages 1–14, May 2010.

[57] W. Xia, H. Jiang, D. Feng, and Y. Hua. SiLo: a similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput. pages 26–28, June 2011.

[58] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan. Debar: A scalable high-performance de-duplication storage system for backup and archiving. In *IEEE IPDPS*, pages 1–12, 2010.

[59] L. You. *Efficient Archival Data Storage*. PhD thesis, University of California, Santa Cruz, June 2006.

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

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

[62] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08*, pages 1–14, 2008.