4-2012

Improving Backup and Restore Performance for Deduplication-based Cloud Backup Services

Stephen Mkandawire
University of Nebraska – Lincoln, smkandawire@gmail.com

IMPROVING BACKUP AND RESTORE PERFORMANCE FOR DEDUPLICATION-BASED CLOUD BACKUP SERVICES

by

Stephen Mkandawire

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Hong Jiang

Lincoln, Nebraska

April, 2012

IMPROVING BACKUP AND RESTORE PERFORMANCE FOR DEDUPLICATION-BASED CLOUD BACKUP SERVICES

Stephen Mkandawire, M.S.

University of Nebraska, 2012

Advisor: Hong Jiang

The benefits provided by cloud computing and the space savings offered by data deduplication make it attractive to host data storage services like backup in the cloud. Data deduplication relies on comparing fingerprints of data chunks, and store them in the chunk index, to identify and remove redundant data, with an ultimate goal of saving storage space and network bandwidth.

However, the chunk index presents a bottleneck to the throughput of the backup operation. While several solutions to address deduplication throughput have been proposed, the chunk index is still a centralized resource and limits the scalability of both storage capacity and backup throughput in public cloud environments. In addressing this challenge, we propose the Scalable Hybrid Hash Cluster (SHHC) that hosts a low-latency distributed hash table for storing fingerprints. SHHC is a cluster of nodes designed to scale and handle numerous concurrent backup requests while maintaining high fingerprint lookup throughput. Each node in the cluster features hybrid memory consisting of DRAM and Solid State Drives (SSDs) to present a large usable memory for storing the chunk index. Our evaluation with real-world workloads shows that SHHC is

consistently scalable as the number of nodes increases. The throughput increases almost linearly with the number of nodes.

The restore performance over the relatively low bandwidth wide area network (WAN) links is another drawback in the use of cloud backup services. High speed network connectivity is either too expensive for most organizations or reserved for special applications. Removing redundant data before transmitting over the WAN offers a viable option to improve network throughput during the restore operation. To that end, we propose Application-Aware Phased Restore (AAPR), a simple restore solution for deduplication-based cloud backup clients. AAPR improves restore time by removing redundant data before transmitting over the WAN. Furthermore, we exploit application awareness to restore critical data first and thus improve the recovery time. Our evaluations show that, for workloads with high redundancy, AAPR reduces restore time by over 85%.

ACKNOWLEDGEMENTS

I would like to express my thanks first and foremost to the Almighty God without whom all this is meaningless.

I would like to deeply thank my advisor, Dr. Hong Jiang, for his support and guidance. Thank you for being patient with me and guiding me not only academically but also on other natters of life. In you I found a true mentor and it's an honor and privilege to be advised by you. I would like to thank Dr. David Swanson and Dr. Lisong Xu for serving on my committee and for the knowledge you have imparted to me. Thank you for reading my draft thesis and all the suggestions.

To my loving wife, Nyambe, thank you for your support, your love, patience and understanding. Your sacrifice is priceless. To my son Wezi and daughter Tabo, thank you for your sacrifice and putting up with daddy's busy schedule.

I would like to thank Lei Xu and Jian Hu with whom I worked on the hybrid cluster. Thank you Lei for all the tips and help in debugging my code throughout this work. Special thanks to all members of the Abacus Distributed Storage Lab for your contributions and support. You are a great family to me.

To my friends and family at large, thank you so much for your prayers and support. Special thanks to all the academic and administrative staff the computer science engineering department at UNL for your support and the knowledge I have gained.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cloud Storage and Data Deduplication

Cloud computing is a computing paradigm in which hosted services are delivered to the user over a wide area network (WAN) using standard Internet protocols. The term “cloud” was coined from the use of the cloud symbol in network diagrams to represent a section in the Internet [6]. In such diagrams the cloud abstracts the details of that part of the network in order to, in most cases, present a view that is focused on the services provided by the cloud part of the network. Several distinctive features differentiate a cloud service from a traditional hosted service. Firstly, a cloud service is sold on demand, usually with pay-per-use or subscription model. In pay-per-use, the user only pays for how much of the services they use, for example, only the number of bytes transferred. Secondly, a cloud service is elastic – at any given time a user can get as little or as much as they need. The third differentiating feature is that a cloud service is fully managed by the service provider - the user only needs network connectivity to the service and enough local resources to use the service. The local resources vary depending on the type of service but can be as simple as a mobile smart phone. In the cloud business model service level agreements (SLA) make the provider accountable for the quality and reliability of

the service. This does not only protect the interests of the client but also clearly spells out exceptions between the parties to the agreement.

Examples of services hosted in the cloud include infrastructure services like servers and data storage. The others are platform and software services. Platform services include software and product development tools, whereas software services (software applications) encompass such services as web-based email and database processing. Figure 1.1 shows an overview of cloud computing (adapted from [33]).

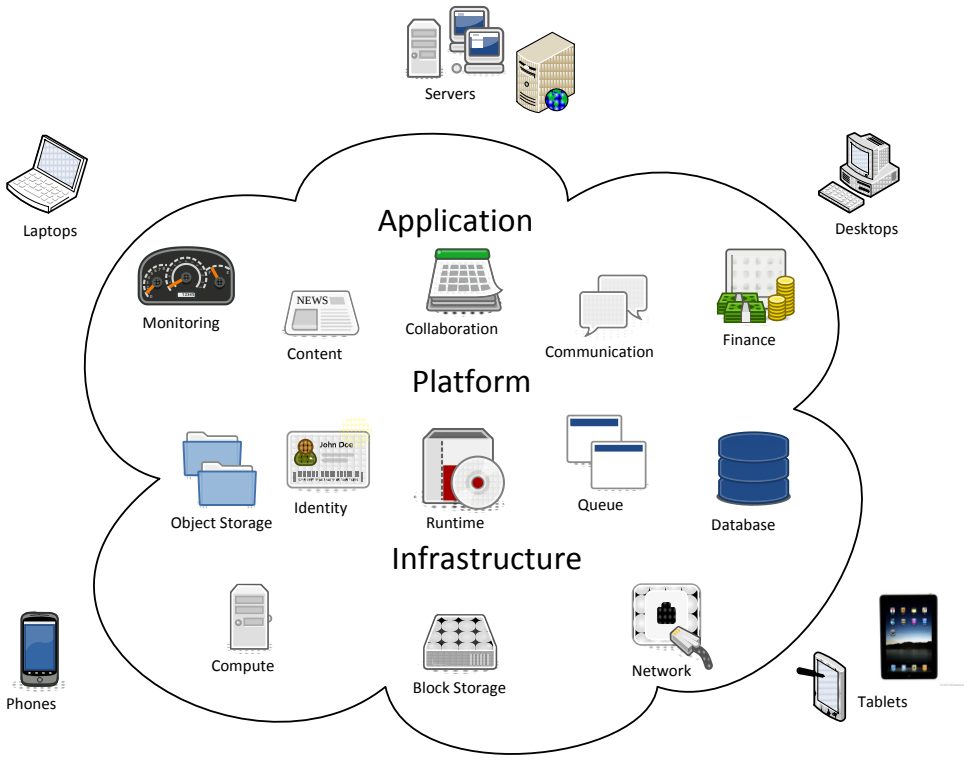


Figure 1.1: Overview of cloud computing.

Given the amount of data being generated annually and the need to leverage good storage infrastructure and technologies, data storage emerges as one of the top cloud computing services. In their 2010 digital universe study, IDC indicated that by 2020, the amount of digital information created and replicated will grow to over 35 trillion gigabytes. Further, the report indicates that a significant portion of digital information will be centrally hosted, managed, or stored in public or private repositories that today we call "cloud services" [3]. The 2011 digital universe study predicted that that in 2011 alone, the amount of digital information created and replicated would be more that 1.8 zettabyte (1.8 trillion gigabytes).

To process and protect such amounts of data efficiently, it is important to employ strategies such as data deduplication to improve both storage capacity and network bandwidth utilization. Further, for most organizations, especially small to medium businesses (SMBs), hosting such services in a public cloud proves to be more economical and efficient. In data deduplication, duplicate data is detected and only one copy of the data is stored, along with references to the unique copy of data, thus removing redundant data. Data deduplication can be performed at three levels, file level, block level (also called chunk level) and byte level, with chunk level being the most popular and widely deployed. For each of these deduplication types, files, data blocks or bytes are hashed and compared for redundancy detection. In general, there are four main steps in chunk level data deduplication, chunking, fingerprinting, index lookup and writing.

(i) *Chunking* - during the chunking stage, data is split into chunks of non-overlapping data blocks. The size of the data block can be either fixed or variable depending

on the chunking method used. The Fixed Size Chunking (FSC) method is used in the case of fixed data blocks, whereas the common method used to produce variable sized chunks is Content Defined Chunking (CDC).

(ii) *Fingerprinting* - using a cryptographic hash function (e.g., SHA-1), a fingerprint is calculated for each chunk produced from the chunking phase.

(iii) *Index lookup* – A lookup table (chunk index) is created containing the fingerprints for each unique data chunk. A lookup operation is performed for each fingerprint generated in step (ii) to determine whether or not the chunk is unique. If the fingerprint is not found in the lookup table, it implies that the data chunk is unique. The fingerprint is thus inserted in the table and the chunk is written to the data store in step (iv).

(iv) *Writing* – all unique data chunks are written to the data store.

Each chunk stored on the storage system using chunk-based deduplication has a unique fingerprint in the chunk index. To determine whether a given chunk is a duplicate or not, the fingerprint of the incoming data chunk is first looked up in the chunk index. Existence of a matching fingerprint (i.e., hash) in the index indicates that an identical copy of the incoming chunk already exists (i.e., has been stored earlier) and the system therefore only needs to store a reference to the existing data. If there is no match, the incoming chunk is unique and is stored on the system and its fingerprint inserted in the chunk index accordingly. Deduplication can be performed either ‘inline’ as the data is entering the storage system/device in real time, or as a ‘post-process’ after the data has

already been stored. Inline data duplication uses less storage space as the duplicate data is detected in real time before the data is stored.

Deduplication based cloud backup emerges as a suitable way of backing up huge amounts of data due to the advantages offered by both cloud computing and data deduplication. The customer can leverage the storage infrastructure offered by the cloud provider, whereas deduplication makes it possible for cloud provider to reduce storage capacity and network bandwidth requirements and optimize storage and networks. Backups benefit even more from deduplication because of the significant redundancy that exists between successive full backups of the same dataset.

While deduplication based cloud backup presents a good backup solution, it has its own shortcomings, with the major one being that of throughput. There are two important metrics that can be used in evaluating the performance of a backup system – the backup window (BW) and recovery time objective (RTO). The backup window is the period of time in which backups are allowed to run and complete on a system, whereas RTO is the amount of time between when a disaster happened and the time the business functions are restored. In this thesis, we address two main problems faced by deduplication based cloud backup systems. Firstly, the chunk index and associated fingerprint lookups present a bottleneck to the throughput and scalability of the system. And secondly, the constrained network bandwidth has a negative impact on the RTO.

(i) Throughput and scalability of chunk index (fingerprint store) and fingerprint lookup: during the backup operation, duplicate data is determined by first

consulting the chunk index. For a lager data set, it's not possible to store the whole index in RAM, forcing the index lookup to go to the disk and incurring disk I/O penalties. The system incurs longer latency as a result of the costly disk I/O. Furthermore, in a public cloud environment, the system has to handle hundreds of thousands of concurrent backup clients, thereby putting additional pressure on the throughput and scalability of the backup system.

(ii) The constrained bandwidth challenge: backup is not the primary goal of a backup system but the means to the goal, which is the ability to restore the backed up data in a timely manner when it's needed. A recent study [7] indicates that 58% of SMB cannot tolerate more than 4 hours of down time before they start experiencing negative effects on the business. A recent survey [8] indicates that 87% of enterprises rank the ability to recover data in a quick and effective way to be very important. However, low bandwidth WAN links present a challenge to cloud backup services that are expected to provide fast data restorations. It is therefore important to devise and employ methods of restoring data that increase the effective throughput of the data restore process.

We describe these problems in some details in our research motivation in the next chapter and present our proposed solutions in chapters 3 and 4 respectively.

1.2 Scope of the thesis

This thesis aims to improve both the backup and restore performances for duplication-based cloud backup services by addressing the issue of chunk index lookup bottleneck and by employing a method that does not transmit redundant data during the restore process.

1.2.1 Chunk index scalability and throughput

The server side maintains a chunk index that keeps chunk metadata. It is a key-value index mapping a fingerprint to, among other things, the location where the chunk is stored in the backend storage. In our solution to improve backup performance we focus only on the chunk index (hash store) and the fingerprint lookup process. The proposed solution can interface with any backend cloud storage using appropriate APIs.

1.2.2 Client side restore performance

Strategies to improve restore performance in deduplication-based cloud storage can be applied either on the server side or the client side. In this thesis we focus on improving restore performance from the client side. Duplicate data exists within backup data sets and therefore, we can improve the RTO by only requesting non-duplicate data chunks from the cloud. In order to do this, metadata for files that are backed up are maintained in file recipes used at the time of restore to generate unique hashes for a given restore data set. Further we apply application awareness to restore critical data first.

1.3 Main Contributions of the thesis

This thesis makes the following contributions

(i) Various schemes have been proposed to address the fingerprint lookup bottleneck problem and the associated disk I/O cost that results from having to access a large chunk index. However, none implements a distributed chunk index in a scalable cluster as a way of improving throughput and scalability. We propose a Scalable Hybrid Hash Cluster (SHHC) to maintain a low-latency distributed hash table for storing hashes. It is a distributed hash store and lookup service that can scale to handle hundreds of thousands of concurrent backup clients while maintaining high fingerprint lookup throughput. Results show that the hash cluster is consistently scalable as the number of nodes increases.

(ii) To improve restore performance, we propose Application-Aware Phased Restore (AAPR). AAPR is intended to be simple, effective and loosely coupled from the server. AAPR employs the concept of file recipes [5] to generates a set of unique hashes for a given restore data set. By transmitting only unique hashes over the WAN link, the restore time is improved. File recipes also help in the reconstruction of each file at the client side and to keep a loose coupling between the client and the server. We further apply application awareness to phase restore operations with critical data being recovered first.

1.4 System Assumptions

Our proposed hash cluster considers a public cloud scenario in which a cloud backup service serves hundreds of thousands of concurrent hash requests coming from different clients. This is a typical characteristic of a public cloud environment especially with the proliferation of mobile personal computing. In view of that, we further assume that the data sets stored in the backend storage will be very large (Petabyte scale). The system employs source deduplication in which all the chunking and hash generation is done at the client side. We further assume that the backup and restore operations are not taking place simultaneously. In real world systems, the cloud provider serves many different users and therefore, some users may be backing up while others may be in need of restore at the same time. Solutions to handle such situations would, among others, include scheduling and prioritizing restore traffic in the backend storage. Expected services can also be outlined in service level agreements. Our implementation uses chunk-based deduplication using the fixed size chunking algorithm. Furthermore, fault tolerance and availability of the hash cluster are out of the scope of this work.

The restore solution assumes the client data has already been backed up and exists at the server side. We generate file recipes at the time of backup and these are maintained by the user. A backup of the recipes can be kept in the cloud and downloaded if the local copy is not available. The local copy of recipes can be maintained on any offline media e.g. optical media or pen drive. This is possible because the size of file recipes is small and most organizations already have strategies to maintain backups on offline media.

1.5 Thesis outline

The remainder of this thesis is organized as follows. Chapter 2 presents our motivation and reviews related work. Chapter 3 presents the scalable hybrid hash cluster. In chapter 4 we describe the application awareness restore solution and we present our conclusions in chapter 5.

Chapter 2

Background and Motivation

In the previous chapter we introduced two challenges facing deduplication-based cloud backup services. In this chapter, we briefly present the background on existing research most relevant to the two challenges to motivate our research. In section 2.1, we describe the related work on and discuss our research motivation for scalable chunk index throughput for deduplication-based cloud backup services. Section 2.2 discusses existing work and motivates our study on improving restore performance for clients connected to deduplication-based cloud backup services via WAN.

2.1 Scalable chunk index throughput for deduplication based cloud backup services

In a deduplication system, fingerprints for all unique chunks stored in the storage system are maintained in a chunk index, usually a key/value hash table with the hash as the key and other chunk metadata as the value. When a new data stream arrives in the system, it is chunked using either the fixed-size chunking (FSC) or content-defined-chunking (CDC) algorithm to generate individual data chunks that are then fingerprinted by hashing (e.g., SHA-1). FSC, as employed in storage systems such as Venti [32], produces non-overlapping fixed-size data blocks, whereas CDC, used in systems like LBFS [15],

uses data content to produce variable-sized chunks. For each new data chunk, its fingerprint is checked against the chunk index in order to determine whether the new data chunk is a duplicate or not. Figure 2.1 shows an overview of the deduplication process. The chunk index and the associated lookup operations lie in the critical path of the deduplication process and present a bottleneck to the whole deduplication process and hence the backup service. In this thesis we refer to the server where the chunk index is stored as the '*hash node*'.

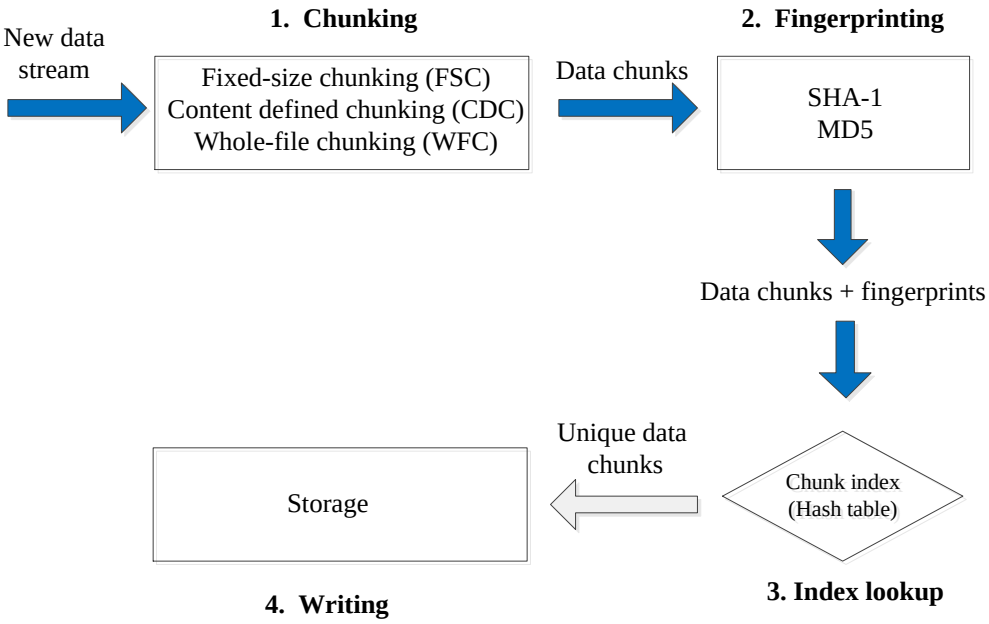


Figure 2.1: Overview of data deduplication

Keeping the whole index in RAM is ideal but only works for very small data sets. For most practical datasets, the chunk index is too big to fit in RAM and therefore the index lookup must go to disk. The system incurs longer latency as a result of the costly

disk I/O. However, a large chunk index is necessary to allow for Petabyte/Exabyte scale storage capacities.

The public cloud environment presents additional challenges that a deduplication backup system must address. In this environment, the backup system may have to serve hundreds of thousands of concurrent backup clients (ranging from enterprises to private individuals). Therefore, the main issues are that of scalability and throughput. The system has to scale to handle numerous concurrent users (i.e., very lager chunk index). In addition, the storage capacity of the system has to be very large (Petabyte/Exabyte scale) due to the number of users. Therefore, a scalable solution should allow for a large index while maintaining high fingerprint lookup throughput.

Various schemes [19, 20, 21, 22, 23] to address the disk I/O problem and improve the throughput of finger print lookups have been proposed. ChunkStash [19] is a flash-assisted inline deduplication scheme that makes use of SSD's good random-read properties to avoid disk I/O latency. It uses flash-aware data structures and stores the chunk index on flash. Each index lookup on flash is served only by one read operation. However, by keeping only a compact index in RAM, the raw storage capacity is limited. In addition ChunkStash does not deal with the problem of scaling to hundreds of thousands of concurrent backup requests.

Zhu et al [20] achieves high cache hit ratios and avoids 99% of disk accesses by using three techniques: a compact in-memory data structure for identifying new

segments, stream-informed segment layout, and locality preserved caching. However, the centralized chunk index is still a bottleneck when used in a public cloud environment.

Sparse Indexing [21] avoids the need for a full chunk index by using sampling and exploiting the inherent locality within backup streams to address the chunk-lookup disk bottleneck problem for large-scale backups. However, this does not use a distributed chunk index.

Most deduplication techniques [18, 19, 20, 34] require locality in backup datasets to provide reasonable throughput. Extreme Binning [21] takes a different approach by proposing a scalable deduplication technique for non-traditional backup workloads that exploits file similarity instead of locality to make only one disk access for chunk lookup per file. The work assumes no locality among consecutive files in a given window of time and therefore addresses datasets for which locality-based techniques perform poorly.

Dedupv1 [22] uses an SSD-based index system to optimize throughput. It takes advantage of the good random-read operations of SSDs to improve disk I/O performance. It avoids random writes inside the data path by delaying much of the I/O operations.

Maintaining the chunk index on a single server or as a centralized resource presents a performance bottleneck when scaling to hundreds of thousands of concurrent backup clients. While the above approaches reduce the latency of index lookups, none of them considers a distributed architecture of the chunk index, which, as we demonstrate, helps address the issues of scalability and throughput for deduplication-based backup systems in public cloud environments.

In order to establish this point, we developed a simulator and compared the throughput of a centralized fingerprint store (chunk index) approach to the clustered approach. For different number of nodes in the cluster, where one node represents the centralized approach, we issued hash queries at varying rates. The hash queries used were SHA-1 [13] fingerprints computed from 8KB data chunks. As shown in figure 2.2, the execution time decreases as we increase the number of nodes in the cluster for a given number of requests per unit time. This indicates the scalability of the distributed approach.

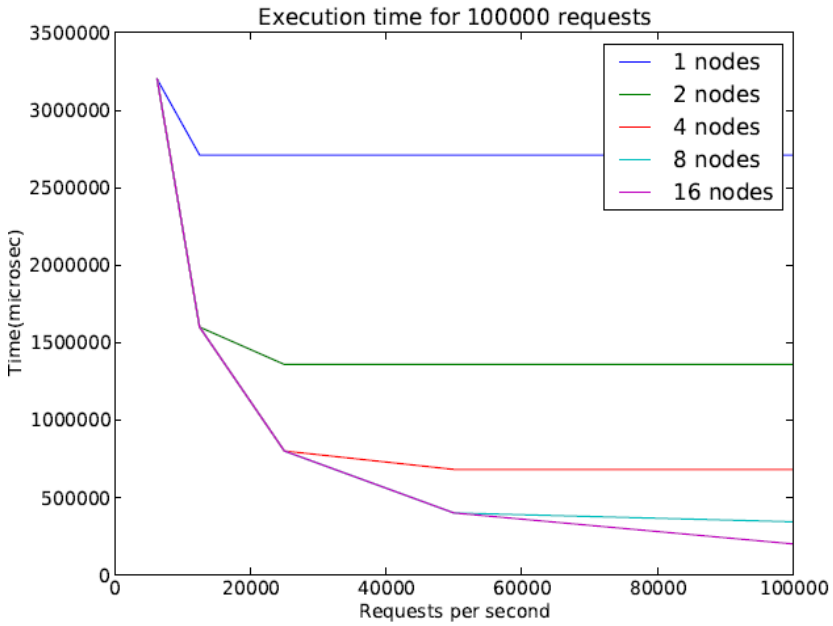


Figure 2.2: Throughput of fingerprint lookup operations

To address the chunk index throughput and scalability problem in cloud backup services, we propose a highly scalable and low latency hybrid hash cluster whose details we

present in chapter 3. It works in concert with other flash-based index lookup techniques that reduce disk I/O latency for index lookups.

2.2 Restore performance for WAN connected deduplication-based cloud backup clients

The Storage Networking Industry Association (SNIA) formed a special interest group in 2010 called Cloud Backup Recover & Restore (BURR) to focus on interoperability, solutions, best practices, and standards requirements for cloud backup, recover and restore. One of the cloud BURR user requirements [9] is that any cloud backup system should be able to provide fast recoveries locally. However, limited wide area network (WAN) bandwidth, poses a challenge on cloud backup services that have to transmit large amounts of data while satisfying the requirements of the ever shrinking backup windows and recovery time objectives. While the amount of digital data that needs to be backed up is growing rapidly and cloud computing popularity has been increasing steadily, the WAN link speeds have not experienced a corresponding growth. High speed network connectivity is either prohibitively expensive for most users or reserved for special applications. Results of a recent survey [8] in which 87% of enterprises rank the ability to recover data in a quick and effective way as very important underscores the importance of providing fast recoveries locally.

Despite a lot of active research in deduplication-based backup by the research community, relatively little attention has been paid to optimizing restore operations over

the WAN. One reason for this could be because most deduplication-based backup solutions are designed with offsite disaster recovery (DR) in mind. In offsite DR, data is backed up to a remoter repository (site) where the businesses can temporally relocate in a disaster situation. In this case, restoring data over the WAN is not a consideration as the user would recover from the remote site were the data is located.

However, not all disasters that cause complete data loss necessitate the relocation of business operations. Furthermore, in a typical cloud backup (especially in the public cloud) scenario, the user backs up to the cloud storage but the recovery site is located elsewhere at the user site or device. Therefore, it has become increasingly important, and thus a BURR user requirement, for a cloud backup service to provide fast recoveries locally. This entails restoring over the WAN.

Restore operations are not performed as frequent as backup operations. However, the demand for shorter restore time is more stringent than the demand for shorter backup window. This is because backup operations are usually pre-scheduled whereas most restore operations are never planned. If a disaster happens, it does so unexpectedly and disrupts business operations and necessitates restore. Therefore, the restore operation is usually a remedial or reactive operation, which is undertaken to restore the business operations. To minimize the negative effects of a disaster, the business or an individual wants to recover in the shortest possible time and therefore, making the requirement for shorter restore times more stringent. Backup is not the primary goal of a backup system but the means to the goal, which is the ability to restore the backed up data in a timely manner.

With the popularity of cloud computing and therefore cloud backup services, it is important to pay critical attention the restore performance over the WAN. Removing redundant data and/or compressing data before transmission over the WAN during the backup operation helps in improving bandwidth utilization and thus improves the backup performance. Likewise, removing redundant data from transmission over the WAN is one solution that can improve restore performance. Furthermore, by reducing the amount of data sent into and out of the cloud, the user can cut down on service monetary costs.

Various techniques to improve network throughput over low bandwidth links have been proposed. The low-bandwidth network file system (LBFS) [15], uses content defined chunking to exploit similarity between files or versions of the same file to save bandwidth for low bandwidth networks. TAPER [16] synchronizes a large collection of data across multiple geographically distributed replica locations using four pluggable redundancy elimination phases to balance the trade-off between bandwidth savings and computation overheads. It presents a multi-vendor interoperable universal data synchronization protocol that does not need any knowledge of the system's internal state to determine the version of the data at the replica. Shilane et al [18] builds upon the work in DDFS [19] to add stream informed delta compression to a deduplication system in order to not only eliminate duplicate regions of files but also compress similar regions of files. However, the emphasis of the study is on the replication of backup datasets in which backups are replicated from a backup server to a remote repository. It is silent on the application of the scheme to accelerate WAN based restores. Our work can benefit from delta replication to further compress data before transmission.

CABDedupe [1] captures the causal relationship among versions of dataset to remove the unmodified data from transmission not only during the backup operation but also during restore. To the best of our knowledge, this was the first deduplication scheme to specifically address restore performance for cloud backup services. However, the scheme only covers scenarios in which part of the data to be restored exists on the client after a disaster. The authors demonstrate that this is the case for about 23% to 34% of the data loss scenarios, mostly from virus attacks. However, there are cases of complete data loss even within the percentages shown above. Therefore, the scheme doesn't cover more than 77% of data lost scenarios. Given that background, we were motivated to build a solution focused on restore performance over the WAN that assumes complete data loss at the client and therefore can be applied to all data loss scenarios.

Our solution combines the elimination of redundant data and phasing the restore process to improve performance. The objective is to have a simple and efficient solution that is loosely coupled from the cloud storage backend. Loose coupling does not only simplify server and client implementation but also helps satisfy the BURR user requirement that a user should not be 'locked in' to one service provider. We achieve the loose coupling by using file recipes at the client.

In addition to the elimination of redundant data, we take advantage of the fact that not all data is needed at the same time and restore data in phases, with critical (i.e., more important and time-urgent) data first. This reduces the amount of data being restored at any particular instance. Removing redundant data improves effective bandwidth throughput and phasing the restore presents an appearance of quick restoration time as

the user is able to get back to business even though not all the data has been restored. With 70% of enterprises ranking the ability to easily and centrally select the appropriate data for backup as very important [8], such a strategy is workable and this is underscored by the fact that most organizations categorize up to 48% of their data and applications as mission critical [11].

To address the restore performance, we propose a simple, efficient and loosely coupled Application-Aware Phased Restore (AAPR) client, whose details we present in chapter 4.

Chapter 3

SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services

3.1 Introduction

The 2011 digital universe study [4] predicted that in 2011 alone, the amount of digital information created and replicated would be more than 1.8 zettabyte (1.8 trillion gigabytes). A 2010 study [3] indicates that by the year 2020, the amount of digital information created and replicated in the world will grow to almost 35 billion terabytes. It further indicates that nearly 75% of digital information is a copy, i.e., only 25% is unique. Therefore, data deduplication emerges as a natural solution to storing such amounts of data due to its space and bandwidth utilization efficiency. With the advantages' of the economics of scale provided by the cloud computing paradigm, it becomes attractive to host data storage services like backup in the cloud. Cloud-based backup services significantly benefit from deduplication because of the redundancy that exists between successive full backups of the same dataset.

However, for backup services that use inline, chunk-based deduplication schemes, the chunk index presents a throughput bottleneck to the whole operation. For most practical backup datasets, the index becomes too big to fit in RAM, forcing index queries

to go to the disk and thus incurring costly disk I/O penalties. Several solutions to the disk I/O problem and improving deduplication throughput have been proposed. However, scalability in both the storage capacity and the number of concurrent backup requests remains an issue in public cloud environments. In a typical public cloud environment, a backup service will have to handle hundreds of thousands of concurrent backup requests.

This chapter presents our approach to addressing the scalability and throughput problems of deduplication-based public cloud backup services. We propose a Scalable Hybrid Hash Cluster (SHHC) to host a low-latency distributed hash table for storing hashes. SHHC is a distributed hash store and lookup service that can scale to handle hundreds of thousands of concurrent backup requests while maintaining high fingerprint lookup throughput.

The rest of this chapter is organized as follows. Section 3.2 presents the design and implementation. We present our evaluation in section 3.3 and summarize the chapter in section 3.4.

3.2 Design and Implementation

3.2.1 Overall Architecture

Owing to huge amounts of data and the large number of users for public cloud backup services, SHHC is designed based on the following design considerations:

(i) The system should handle a large number of hash requests coming from different clients simultaneously. The objective, therefore, is to scale the chunk index to handle hundreds of thousands of concurrent backup requests while maintaining high fingerprint lookup throughput.

(ii) The system should accommodate very large datasets.

(iii) The system should use multiple backend storage nodes and the chunk index should be global - while the cluster is distributed, the hash table has a global view of the backend storage and, therefore, each chunk stored in the system is unique across all storage nodes. The global view eliminates the *storage node island effect* [28] in which duplicate data exists across multiple storage nodes.

The basic idea of our solution can be outlined as follows:

a) Instead of using a single hash node or a centralized chunk index, we use a hash cluster and distribute the hash store and lookup operations

b) Store the chunk index on solid state drives (SSD) and take advantage of the fast random-read property of SSDs to avoid the disk I/O issue. Several studies have demonstrated that the index can be accessed efficiently on SSD. Therefore, we can treat RAM/SDD as a large "hybrid" RAM without a heavy performance penalty. Due to SSD sizes, "hybrid" RAM is larger and cheaper per byte than traditional RAM. We can therefore support a much larger chunk index. This is a core feature of SHHC, hence the "hybrid" in the acronym.

c) Use a suitable in-ram data structure for indexing the hash store

Figure 3.1 shows the overall architecture of the proposed cloud-based back-up service.

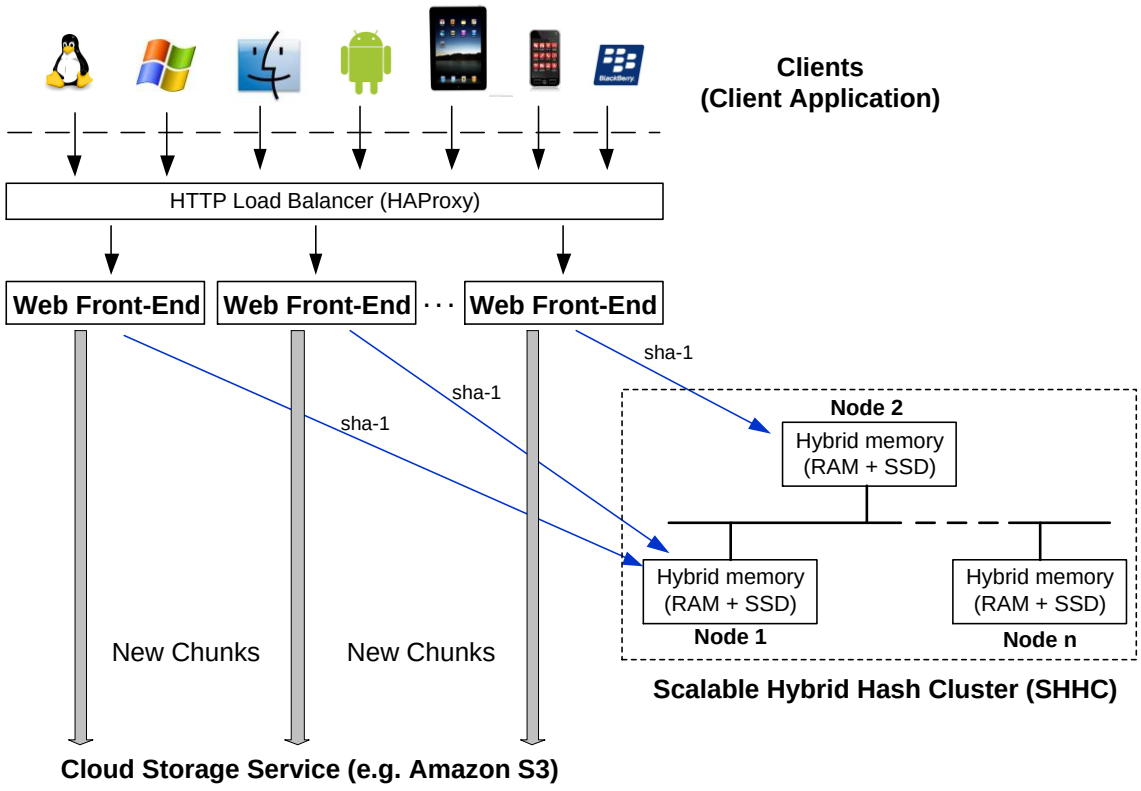


Figure 3.1: Overall architecture of the cloud-based back-up service

Our solution consists of four major components, the client application, web front-end cluster, hybrid hash cluster and the cloud backend storage.

3.2.1.1 Client application

This is an operating system specific application installed on a client device. It collects local changes to data, calculates fingerprints and performs hash queries. The chunk index is hosted in the hash cluster in the cloud. For each fingerprint computed, the client sends

a hash query to the hash cluster to determine if the hash already exists in the chunk index or not. The presence of a matching hash in the hash store means that the chunk data corresponding to the hash is already stored in the system. If the chunk doesn't already exist in the system, the client considers the chunk data represented by the hash as a new chunk that has not yet been backed up and sends it to the cloud for backup. SHHC assumes source deduplication in which chunking and fingerprinting are performed by the client.

3.2.1.2 Web front-end cluster

This is a highly scalable cluster of web servers that acts as an entry point into the cloud for the client. It responds to requests from the clients and generates an upload plan for each back-up request by querying hash nodes in the hash cluster for the existence of requested data blocks. If the data chunk is new, i.e., it doesn't exist in the system, the web cluster sends the new data blocks to the cloud backend for storage. One characteristic of backup datasets is that they exhibit a lot of locality among full backups of the same dataset [18, 19, 20]. To take advantage of this locality and data redundancy, the web cluster aggregates fingerprints from clients and sends them as a batch to the hash cluster.

3.2.1.3 Hybrid hash cluster

This is a novel scalable, distributed hash store and lookup service that can scale to handle hundreds of thousands of concurrent backup requests while maintaining high fingerprint lookup throughput. It is designed to be scalable, load balanced and of high fingerprint

store and lookup throughput. It can be considered as middleware between the cloud storage backend and the client.

As shown in Figure 3.1, we have designed the system using a multi-tier architectural model. The main idea is to separate the distributed hash cluster from the cloud storage backend. This separation offers several advantages, including:

- A highly optimized and scalable fingerprint storage and lookup mechanism.
- Functionality and resources can be added transparently to the cluster. For example, the hash cluster can transparently be scaled to thousands of nodes.
- Reduced network traffic to the cloud storage backend.
- As a specialized hash engine service, SHHC can connect to and work with any cloud storage service provider.

3.2.1.4 Cloud storage service

This is a cloud based multi-node storage backend for storing backup data.

3.2.2 The hybrid hash cluster

The hybrid hash cluster is a cluster of nodes hosting a low-latency distributed hash table for storing hashes. It's a hash engine designed for fingerprint store and lookup and keeps a global view of the backend storage. By using a low-latency distributed hash table (DHT) [31], SHHC can scale to handle numerous concurrent backup clients while maintaining high fingerprint lookup throughput.

Like the Chord [29], SHHC consists of a set of connected nodes, of which each holds a range of hashes. However, SHHC differs from the Chord in that, while the latter was designed for highly unstructured peer-to-peer environments, the former runs in a stable, structured and relatively static environment.

Apart from a distributed hash table, the other core feature of SHHC is its use of hybrid memory. Each node is made up of RAM and SSDs as shown in Figure 3.2. SHHC treats RAM/SDD as a large "hybrid" RAM without a heavy performance penalty. Due to SSD sizes, "hybrid" RAM is larger and cheaper per byte than traditional RAM. We can therefore, support a much larger chunk index.

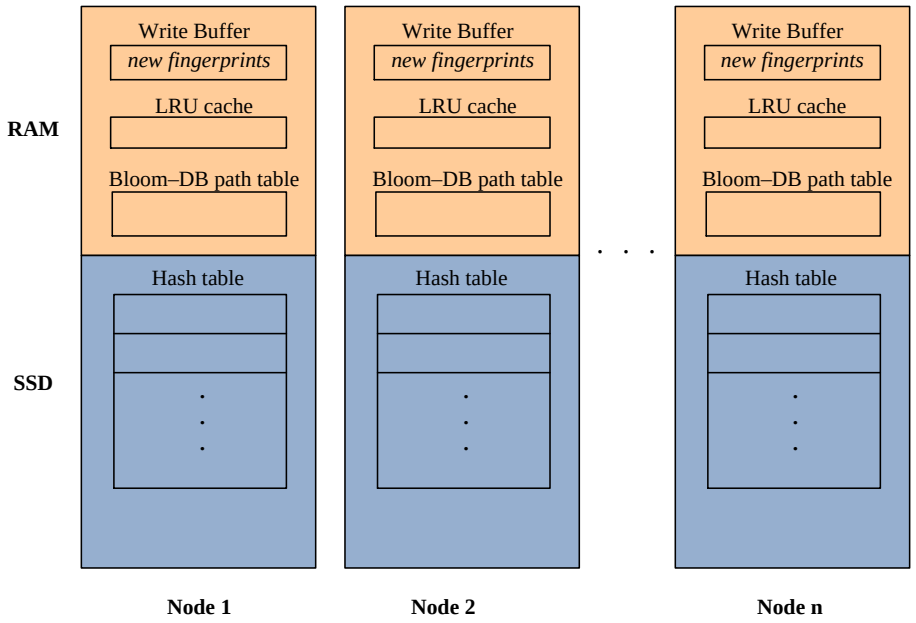


Figure 3.2: Hash node memory layout

In the current implementation, the hash table is stored on SSD as a Berkeley DB with a bloom filter used to represent membership of hashes in the DB. We keep a

<*bloomfilter, DB*> path in RAM. In order to accelerate hash lookup, we also maintain an in-RAM least recently used (LRU) cache for hashes. While SSDs have good random read properties, their random write performance is not equally good. To hide the random write weakness of SSDs, we use a write buffer in RAM. All the incoming hashes that are unique are added to the write buffer and when the write buffer is full, the hashes are written to SSD. Batching and delaying writes [22] to SSDs in this manner masks the random write performance weakness.

Figure 3.3 shows the workflow of an SSHC lookup operation. A typical hash lookup operation follows the following steps:

- Through the web front-end cluster, the client sends a fingerprint (SHA-1) to a corresponding hash node N.
- Node N attempts to locate this fingerprint in main memory. If it exists, node N informs the client that the data block indentified by this fingerprint has already been stored.
- Otherwise, a read miss is triggered. Node N then tries to locate this fingerprint in the hash table on SSD. If this fingerprint exists on SSD, node N loads it into the least recently used (LRU) cache in RAM and replies to the client. Otherwise node N writes a new entry in the hash table on SSD and notifies the client that the corresponding data does not exist in the cloud and asks the client to transmit the data.

- Node N maintains a LRU cache in RAM. If the LRU is full, it discards the least recently used fingerprints.

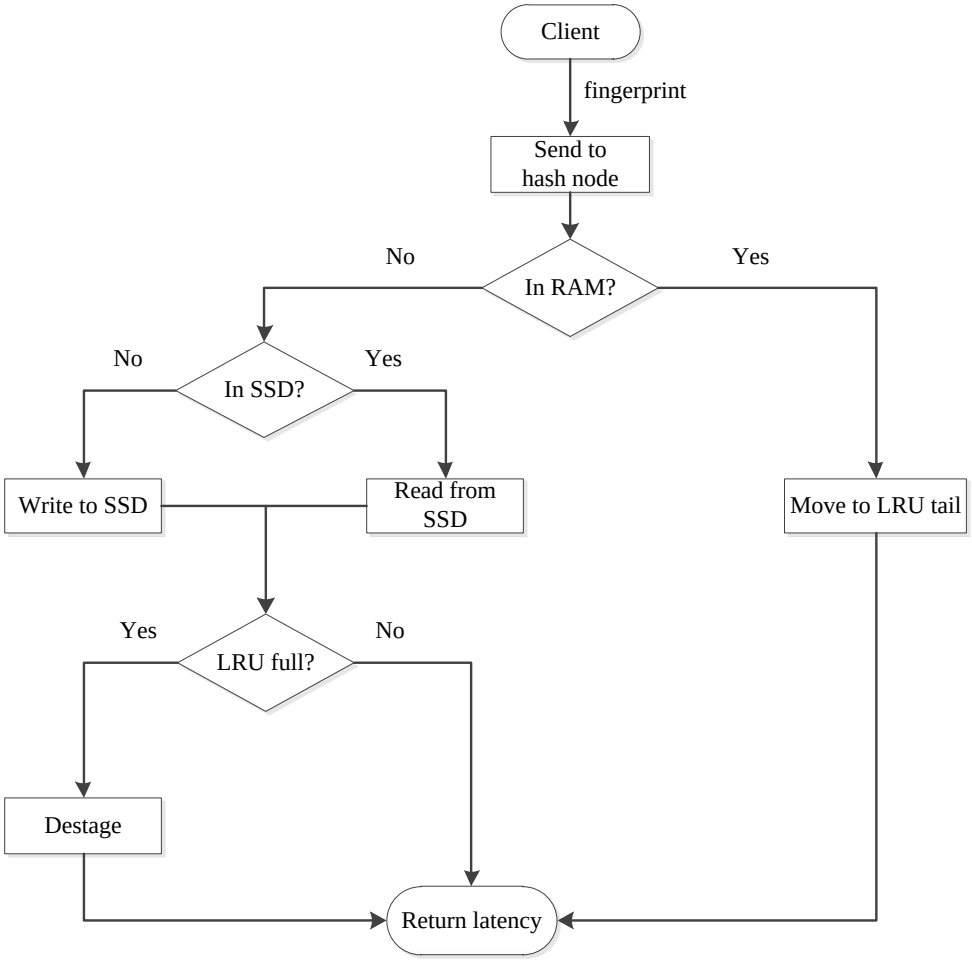


Figure 3.3: Flowchart of an SHHC lookup operation

3.3 Evaluation

In our evaluation, we would like to answer the flowing questions:

- Is the throughput of a distributed chunk index better than a single server (or centralized) index?
- Does the throughput scale with the number of nodes in the cluster?
- Is the load to each hash node in the cluster balanced?

To do so, we develop a hash cluster and evaluate the throughput of different cluster sizes by injecting fingerprints of real world workloads at different rates.

3.3.1 Experiment setup and datasets

We conduct our experiments with a cluster size of up to 5 nodes, each node with an Intel Xeon 2.53 GHz X3440 Quad-core Processor, 4-16GB RAM, SATA II 64GB SSD and 1GB NIC. All the nodes are running GNU/Linux Ubuntu Server 10.10 and are connected via a 1 GB/s Ethernet switch using CAT5e UTP cables. We use separate client machines to generate and send multiple workload traffic to the cluster. The characteristics of the workloads used in experiments are shown in Table 3.1.

Table 3.1: SHHC Workload characteristics

Workload	Fingerprints	% Redundancy	Distance
Web Server[30]	2,094,832	18	10,781
Home Dir[30]	2,501,186	37	26,326
Mail Server [30]	24,122,047	85	246,253
Time Machine	13,146,417	17	1,004,899

The time machine workload was collected from an OSX user data backed up over six months. The chunk size for this workload is 8KB while the chunk size of the other workloads is 4KB. In Table 3.1, % redundancy is the amount of redundancy in each workload expressed in percentage. The distance column shows the average distance between similar fingerprints in the list of hashes for each workload. We use this distance as a way of determining how much locality is in each workload. Workloads with shorter distances have higher spatial locality of data blocks.

3.3.2 Scalability and performance

We inject fingerprints of the workloads to different cluster sizes. For each evaluation, we use two client machines to generate and send hash queries to the cluster. Each client is implemented with a send buffer to aggregate hash queries and send them as a batch to the cluster. We evaluate the cluster performance with different client batch (send buffer) sizes. Batching queries before sending them to the cluster has a twofold advantage: i) it improves network bandwidth utilization and, ii) it preserves spatial locality of hash requests that are sent to the cluster. Spatial locality in backup datasets improves deduplication [18].

Figure 3.4 shows the overall throughput using batch sizes 1, 128, and 2048 per request. Batch size 1 represents the case when there is no batching used. Results show that batching improves throughput for all configurations of the cluster. Furthermore, throughput for lager batches is significantly better than the case without batching.

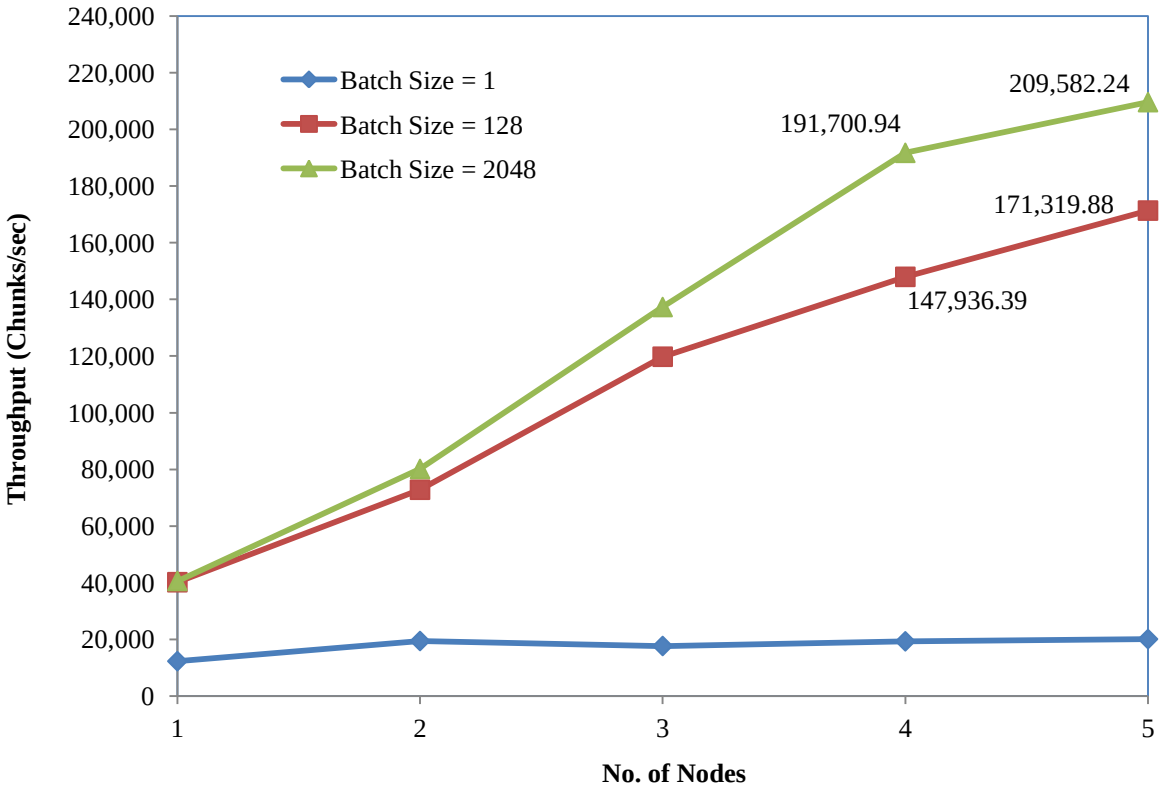


Figure 3.4: Scalable throughput

3.3.3 Load balancing

In evaluating load balancing, we analyze the hash table entries stored in each hash node for each workload. The percentage of the number of hashes stored in each node is as shown in Figure 3.5. The results indicate that our scheme is load balanced, with each node getting about 25% of hash requests for a four-node cluster.

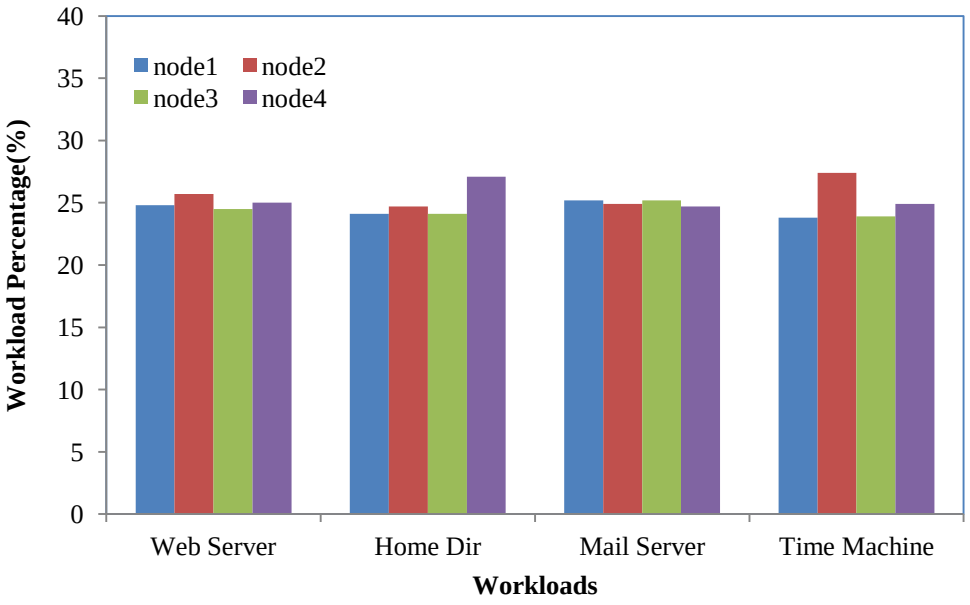


Figure 3.5: Hash distribution

3.4 Summary

Hosting storage services, such as backup, in the cloud has become very attractive due to the advantages offered by the cloud computing model. However, with the huge volume, high velocity and great variety of data to be stored, such services face various challenges. Data deduplication emerges as an efficient scheme to use in cloud backup storage systems as it reduces storage capacity requirements and also optimizes network bandwidth utilization. But for deduplication-based cloud backup services, the chunk index becomes a bottleneck to the throughput of the system. When the chunk index becomes too big to fit in RAM (a common scenario for most practical data sets), the index queries are forced to go to disk and thus incurring disk I/O penalties.

While solutions to address the disk I/O problem and improve deduplication throughput have been proposed, scalability of both the storage capacity and the volume of concurrent users remains an issue in public cloud environments.

In this thesis, we propose the Scalable Hybrid Hash Cluster (SHHC) which is able to scale to handle huge volumes of concurrent backup requests while maintaining high hash lookup throughput. Evaluation results show that the hash cluster is consistently scalable as the number of cluster nodes increases.

Chapter 4

AAPR: Application-Aware Phased Restore

4.1 Introduction

While there is a rich body of research on deduplication-based backups, much of it is focused on the backup operation and relatively little attention has been paid to the restore process, specifically restore over the wide area network (WAN) for cloud backups. With the use of cloud backup services becoming increasingly popular, the low WAN bandwidth presents a drawback in the use of such services as it negatively impacts restore times. One solution would be to increase bandwidth to boost network throughput in order to satisfy the need for fast restores over the WAN. However, this is prohibitively expensive. A more economical alternative that emerges is removing redundant data and/or compressing data before transmission, not only during the backup operation, but also during the restore operation. This does not only improve effective bandwidth throughput but also reduces cloud usage costs by reducing the amount of data sent into and out of the cloud. Another way to make use of low bandwidth effectively stems from the observation that most organizations don't need all the data to be restored at the same time in order to revive the disrupted business operations – instead they only need critical data first. Therefore, it is intuitive to restore data in phases, starting with critical data,

thereby shortening what would otherwise be longer restore times. Once the critical data is restored the business can resume its operations.

This chapter presents AAPR: Application-Aware Phased Restore, which is a simple, effective and loosely coupled restore solution for deduplication-based cloud backup clients. The solution combines the elimination of redundant data and phasing the restore process to improve performance. The objective is to have a simple and efficient solution which is loosely coupled from the cloud storage backend.

The rest of this chapter is organized as follows. Section 4.1.1 describes the concept of file recipe and the elimination of redundant data from datasets. It also discusses the categorization of application awareness. Section 4.2 presents the design and implementation. We present our evaluation in section 4.3 and the summary in section 4.4.

4.1.1 Eliminating redundant data in restore datasets

4.1.1.1 File Recipes

Central to our solution is the concept of file recipes [5]. A recipe is a synopsis of a file which, among other things, contains a sequence of SHA-1 fingerprints that identify all the chunks of data that belong to the file. Each hash uniquely maps to a particular chunk in the file. A file recipe forms the blueprint of a file. Our work is inspired by the observation that as a blueprint of a file, a file recipe contains a lot of information that can help in efficient WAN restorations. When used in a deduplication restore process, a

recipe presents several advantages including, i) allowing us to perform intra-file (intra-recipe) and inter-file (inter-recipe) redundancy elimination without handling the actual files or actual the dataset, ii) preserving the chronological order of bytes in the original file because the fingerprints in a recipe are arranged in a sequence. This helps with the reconstruction of the file. iii) client-side processing with minimal computation overhead since the total size of file recipes is very small compared to the actual dataset and, iv) recipes allow for loose coupling between the cloud backup client and the cloud backend storage. The client doesn't have to have a lot in common with the server. For example, to reconstruct a file, the only 'ingredients' needed for the recipe from the cloud are data chunks with the corresponding hashes they are mapped to. This simplifies both the client and server implementations.

We conducted experiments on some datasets to investigate the space overhead of file recipes. As shown in Table 4.1, the total size of the file recipes for a given dataset is very small and therefore, this makes it easy to store and manage them. We applied different chunking methods, content defined chunking (CDC) and fixed size chunking (FSC), to each dataset to see how the size of recipes is affected by the chunking method. Results show that, in general, for both types of chunking the total size of recipes is very small compared to the sizes of the actual dataset.

Table 4.1: Relationship of total recipe size to the dataset size

Dataset	Size(KB)	Recipe Size(KB)	Recipe size percentage
Virtual Machines (FSC)	226,073	552	0.2441
Virtual Machines (CDC)	226,073	294	0.1301
Video files(FSC)	7,321	18	0.2441
Video files(CDC)	7,321	9	0.1236

During the restore process hashes from the recipes for the files in the restore dataset are compared in order to eliminate redundant data. Figure 4.1 shows a sample of a file recipe. For clarity a hash is represented in hexadecimal format in this figure. In the actual implementation we store and manipulate the hashes as raw 20 byte fingerprints. Furthermore, file metadata information such those found in UNIX/LINUX *struct stat* can be added to the recipe as needed.

```
file name: /home/data/sample_file.pdf
backup time: 1331052002
No. of hashes: 15

af9626759adfcce6a003ae4df62e2a4953e1b44f
52d800af6db3f879fb8233ecfa88b975d0cd8ae5
ab5e5d9a339af7c3c102e7c543a453396acaca98
1fbb2ad365b42431fa41abf887ae74ef60b45923
a5852208f5b0212c29ff044ab6e1a23a4c1e9b5d
9f3f868920f2b8e516846dffc19648d9131e03f4
b503ca33070fda0448d845c98120b67581ca2f10
1b2ac70ecb94f8a320a72221b1c6d01927f30a8b
745af2e6587c612ddf8b0708c0c5b2a654b1a175
6715d18eff88007ff476e2e575c8b5147d112826
9bafe36fac10e46e6cf4f1e046bbef363f0db8bf
01e0943ecf9bd2a62237659e9d718dc1ec265c71
1ca9e1baec697c619a4aaed7d0019748d50e80cb
9fab497a8b26852177197e21c5da4b842160e346
ca39e05d8b07b6c92b99e9743ac501be8b09cd1f
```

Figure 4.1: Sample file recipe

4.1.1.2 File set redundancy

The other critical concept to AAPR is that of file set redundancy. We define a file set as a collection of files marked for backup/restore. It can be made up of a single file or multiple files. We broadly divide file redundancy into three categories: i) *intra-file set redundancy* - redundancy within the file set, ii) redundancy between versions (sessions) of the same file set – for example, redundancy between two or more backup/restore sessions of the same file set, and iii) *inter-file set redundancy* - redundancy between two or more file sets or streams of data. We make use of the observation that there exists redundancy within a particular file set - *intra-file set redundancy*. The importance of intra-file redundancy to this solution is that redundant data can be removed from the file set and the client can only request for unique chunk data from the cloud. To make the case for existence of intra-file set redundancy, we performed some experiments on some datasets. Results are shown in figure 4.2. and Table 4.2.

Table 4.2: Redundancy in single files

File Set	**Size**	**Total hashes**	**Unique Hashes**	**% Redundancy**
Ubuntu 11.10 server (iso)	683MB	87,334	87,075	0.2966
sample.mp4	117MB	14,243	14,237	0.0421
sample.pdf	262KB	33	33	0.0000
sample.mp3	30MB	3,762	3,762	0.0000
linux-2.5.75.tar	177MM	22,564	22,553	0.0488

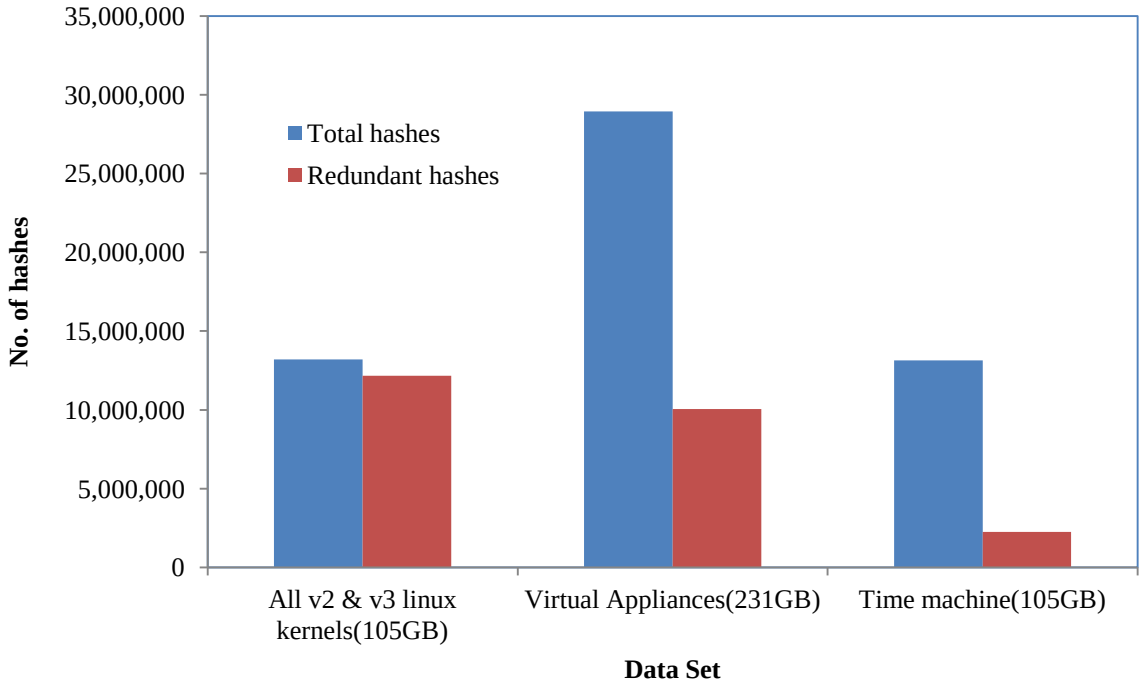


Figure 4.2: Intra-file set redundancy

In general, there isn't significant redundancy within a single file. This could be because current chunking methods are not able to extract much redundancy within a single file. However, recent studies [12] have revealed that modern files are actually not single files as they known to be; instead certain types of files are actually file systems. The findings in that research have strong implications on the design of next generation cloud-based storage systems. We therefore, foresee new schemes that will take advantage of such findings to extract more redundancy in single files. This will be beneficial especially for private individual cloud users whose datasets are not large. Since AAPR depends on intra-file set redundancy, it may not perform well on single file restores.

On the other hand, restoring single files is not the common case especially for the enterprise. An organization will usually backup or restore multiple files, which as demonstrated exhibit significant intra-file set redundancy.

4.1.1.3 Unique hash store

Following the results from figure 4.2 and since a file can be represented by a file recipe, we collect the hashes from the file recipes of all the files in a dataset and generate a unique hash store containing unique hashes only. This process is performed locally on the client side, after which chunk data from the cloud is request only for the hashes in the unique hash store. As described later, when the restore is phased, the size of the hash store is small enough to be maintained in RAM. In order to simplify the implementation while guaranteeing the correctness files restored, our implementation first builds a complete repository of all unique data chunks collected from the cloud before beginning the reconstruction. The local chunk repository is only needed during the restore process.

4.1.1.4 Application awareness

We further exploit application awareness to restore critical data first. This is archived by making the backup/restore client aware of what critical data is and then phasing out the restore process to restore the critical data first. Critical data is determined based on the application – hence application awareness.

Recovery point objective (RTO) is defined as the amount of time between the moment a disaster happened and the time the business functions are restored. Therefore,

it can be argued that RTO is satisfied by restoring what is essential to get the business back in operation.

4.1.1.5 Categorizing application awareness

In this thesis we define application awareness from two perspectives: system-level awareness and file-level awareness.

System level looks at the type of application, for example an email system. For an active web based email system, restoring inboxes for active and inactive users at the same time would unnecessarily lengthen the RTO for critical email boxes. Restoring emails for active users first and completing the rest of the restore in a ‘staggered fashion’ would satisfy the RTO in a more efficient and economical way. Even if not all inboxes are restored at the same time, the experience of the active users (and those are the ones that matter most at that time of restore) would be that of having experienced fast recovery. Furthermore, webmail systems such as gmail already have an option for the user to rank emails according to importance. Incorporating such awareness in restoring data for such a system may help improve the ‘apparent restore’ time for users.

File level application awareness deals with the type of files and file activity. For some users multimedia files might be more critical in restoring business operations than ordinary document files. On the other hand, files with certain file extensions might be more critical for some users than other file extensions. In some systems, file access time

might give an indication of how frequently certain files are used and hence their popularity. Typically it would be beneficial to restore more popular files first.

A five year study of file system metadata involving over 60000 Windows PC file systems found a significant temporal trend relating to, among other things, the popularly of certain file types[14]. The study indicates that eight filename extensions accounted for over 35% of files and nine filename extensions accounted for more than 35% of the bytes in the files and that these extensions remained popular for many years. The study further reveals that in more than 60% of the file systems considered, over 80% of files had never been changed since they were copied into the file system. A conclusion can be drawn that not all files in a given system are critical at the same time, and this indicates a need for application awareness in backup applications. Exploiting such application awareness can help restore the business back to operation quicker than otherwise.

4.2 Design and Implementation

In this section we present the architecture of AAPR: Application-Aware Phased Restore. AAPR employs source deduplication using the fixed chunking method. The implementation can broadly be divided into two parts, the server and the client. The client performs the fingerprinting and hash calculation while the server hosts the chunk index and the data store for storing all the unique chunks sent by the client. Figure 4.4 shows the overall system architecture. The details of the major system components are described below.

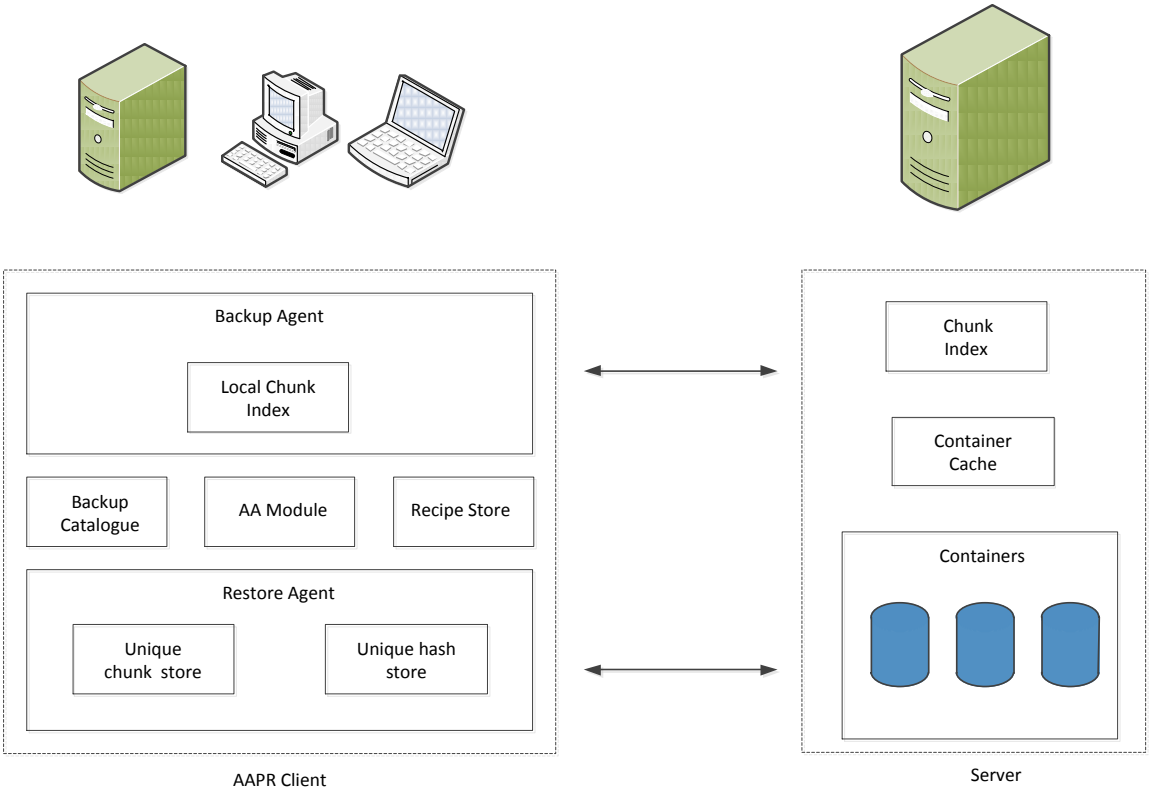


Figure 4.4: AAPR System Architecture

4.2.1 AAPR client

The AAPR client comprises the backup agent and restore agent as shown in figure 4.3.

4.2.1.1 Backup agent

The backup agent is responsible for chunking, fingerprinting and sending data to the storage server. It also generates file recipes for all backed up files. To backup data the backup agent performs several operations as follows.

- It takes a file from the backup dataset and splits it into 8KB chunks using the fixed size chunking algorithm and calculates a SHA-1 fingerprint for each chunk.
- For each chunked file, the backup agent generates a file recipe, keeping a sequence of all hashes that belong to the file in the order in which the chunks appear in the file. When the chunking and fingerprinting is complete, the recipe is saved in the recipe store.
- To determine whether a given chunk has already been sent to the server or not, the agent checks the local chunk index and buffers the <*hash-chunk*> pair if the chunk is not a duplicate. It's important to maintain a chunk index at the client side to avoid sending unnecessary queries to the server. Otherwise the client would have to send a query to the server to establish whether or not a chunk exists even for chunks that have previously been sent. This wastes bandwidth especially for datasets that exhibit a lot of redundancy.
- Each unique chunk is paired with its corresponding hash and the <*hash-chunk*> pairs are batched and sent to the server.
- When the backup operation is complete the backup agent generates a backup catalogue containing all the files that have been successfully backed up. The application awareness module (AA module) then generates a list of all the files in the catalogue that are critical and will need to be restored first.

At the server-side the server checks for existence of each of the received chunk. Note that in a real world scenario the server would be handling requests from other users. Therefore, an incoming chunk from one client may already be present in the backend

storage. In our implementation, if the chunk already exists, the server simply drops the hash-chunk pair. The server doesn't have to return any chunk location or reference information to the client. Since the reconstruction information is kept in the recipe at the client side, the server will only need to respond with the requested fingerprint and its corresponding data chunk at restore time.

4.2.1.2 Recipe store

The recipe store is responsible for storing all the recipes for the files. We have implemented it as a Berkley DB [25] key-value hash table indexed by a file name.

4.2.1.3 Application awareness module (AA Module)

The application-awareness module determines which files are critical. As earlier described, application-awareness can vary depending on the system and the user. In this thesis, we have used access time to determine critical files. Files from the backup catalogue whose access time is newer than a certain time are considered critical and put on the critical list. In our experiments, we randomly modify access time for 20% of files in the dataset. This is based on the fact that in more than 60% of the file systems considered by Agrawal et al [14], over 80% of files had never been changed since they were copied into the file system. Therefore, we can safely estimate that based on access time, only about 20% of the files in our scenario are critical. While it's possible to manually select which files are critical, it is clear that doing so is not only inefficient but also not scalable especially for a large size dataset with a larger critical data size.

4.2.1.4 Restore agent

The restore agent is responsible for getting data chunks from the server and reconstructing files. It works in conjunction with the backup catalogue, the AA module and the recipe store to complete its operations. By using file recipes, AAPR avoids the use of chunking and fingerprint in the restore operation. In order to restore data the restore agent performs several operations as follows.

- The restore agent gets a list of files to restore from the backup catalogue or the critical list(when using application awareness)
- For each file, the agent retrieves a file recipe from the recipe store. It gathers the fingerprints from the file recipes and queries the unique hash store to determine whether each of those hashes is unique or not. Unique hashes are inserted in the unique hash store and also batched for sending to the server.
- Upon receiving the chunks from the server, the client stores each chunk in the local chunk store. The chunk store only keeps unique chunks.
- When all the chunks have been received, the agent starts to reconstruct the files. Deferring the file reconstruction until all the chunks have been received has several advantages including ease of implementation of the reconstruction logic. The reconstruction can also easily be parallelized with guaranteed correctness since all the operations on the chunk store are read operations.

Note that with application awareness, the critical files are restored first and the above steps are repeated for the remainder of the files.

4.2.1.5 Unique hash store

The unique hash store is responsible for storing all the unique hashes form the file recipes. For the size of datasets used in our experiments, we maintain this as an in-RAM data structure. Other efficient chunk index schemes could be used for very large data sets. Application awareness helps to keep the local hash store to manageably smaller sizes.

4.2.1.6 Local chunk store

The local chunk store is a temporary repository of all the chunks retrieved from the server. With current storage drive capacities it's easy to allocate space at the client for this purpose. Besides, the chunk store is only temporary and is needed only during the restore time. In practice systems already use additional space for recovery, for example, snapshots. Another example is that of Oracle 10g R2 which uses a "flash recovery area" [26].

4.2.2 Server

The server hosts the chunk index for all clients of the cloud backup service. The chunk index supports global, exact deduplication i.e. it's not a sampled index and the storage backend (all containers in Figure 4.4) have the same view of the index. Only one client is used in our experiments. Each unique chunk is stored in a container on SSD with its corresponding hash as a <*hash, chunk*> pair. The container size was set to 2048 <*hash, chunk*> pair entries, giving a total size of about 16.8MB for each container. To accelerate

the read performance, a container cache is maintained in RAM. Upon a miss in the container cache, all the chunks in the container where the chunk is located are fetched into the cache. Storing chunks as <*hash, chunk*> pairs in a container simplifies the implementation of the prefetching logic. The container cache uses a least recently used (LRU) replacement policy. If the cache is full and a container needs to be fetched, 2048 least recently used <*hash, chunk*> pairs are evicted from cache. The average cache hit rate observed was about 99%.

4.3 Evaluation

In this section we present the performance evaluation of AAPR. We built a client and server to evaluate the performance of our solution. In order to make performance evaluations, the implementation of the restore client is such that we can run the restore agent with or without application awareness. To use application awareness the agent uses the list of files generated by the AA module to restore the needed files. The restore of the remainder of the files is run thereafter.

4.3.1 Experiment setup and datasets

The experiments were conducted on a server and client with the following hardware specifications. The server was configured with Intel Xeon 2.53 GHz X3440 Quad-core Processor, 16GB RAM, SATA II 64GB SSD and 1GB NIC. The client had similar configuration and both were running GNU/Linux Ubuntu Server 11.10. The client and

server were connected via a 1 GB/s Ethernet switch using CAT5e UTP. The Ethernet was rate limited to the required bandwidth in order to impose a WAN bandwidth scenario.

We used three datasets with different redundancy characteristics as shown in Table 4.3. The datasets consists of version 2 Linux kernels [26] and virtual machine appliances [27]. The third dataset is a collection of user videos.

Table 4.3: AAPR Workload Characteristics

Dataset	Total hashes	Unique hashes	Redundancy (%)
Linux kernels	383,738	44,504	88.4
Virtual appliances	703,969	585,824	16.78
Videos	937,143	937, 143	0

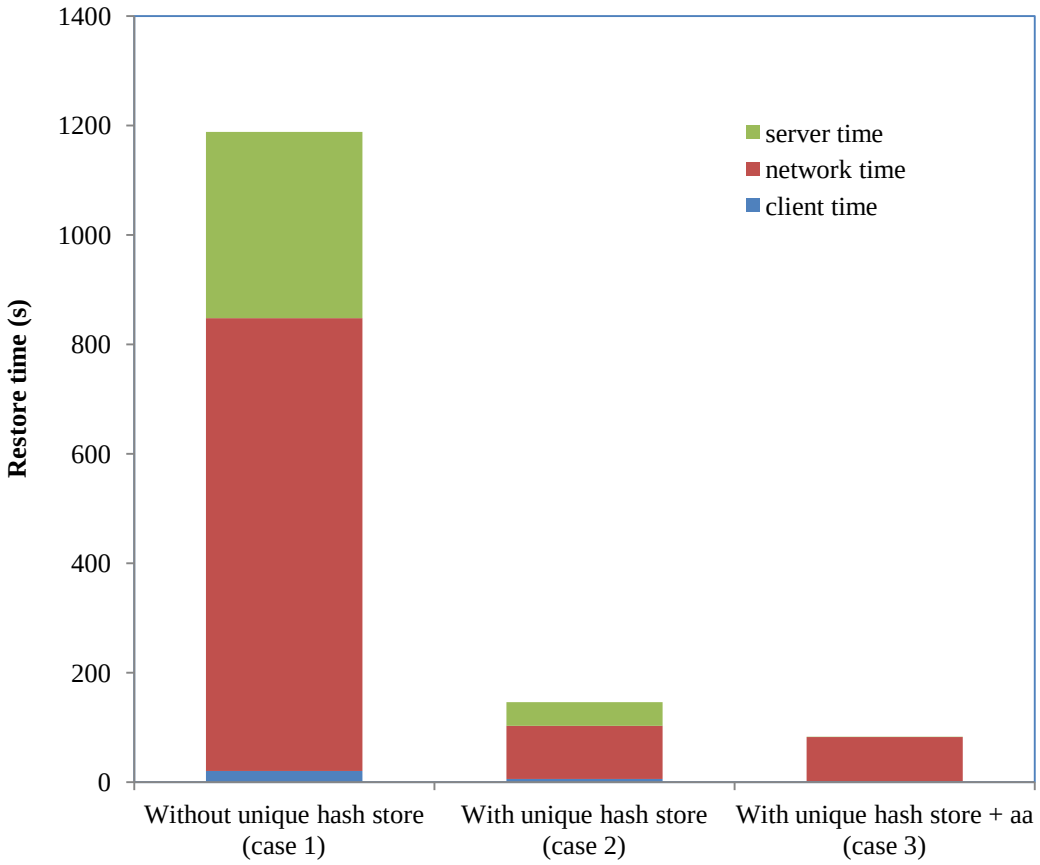
4.3.2 Restore time

To assess restore performance we restored each data set under different setups shown below. We performed a full backup of each data set and restored the whole data set.

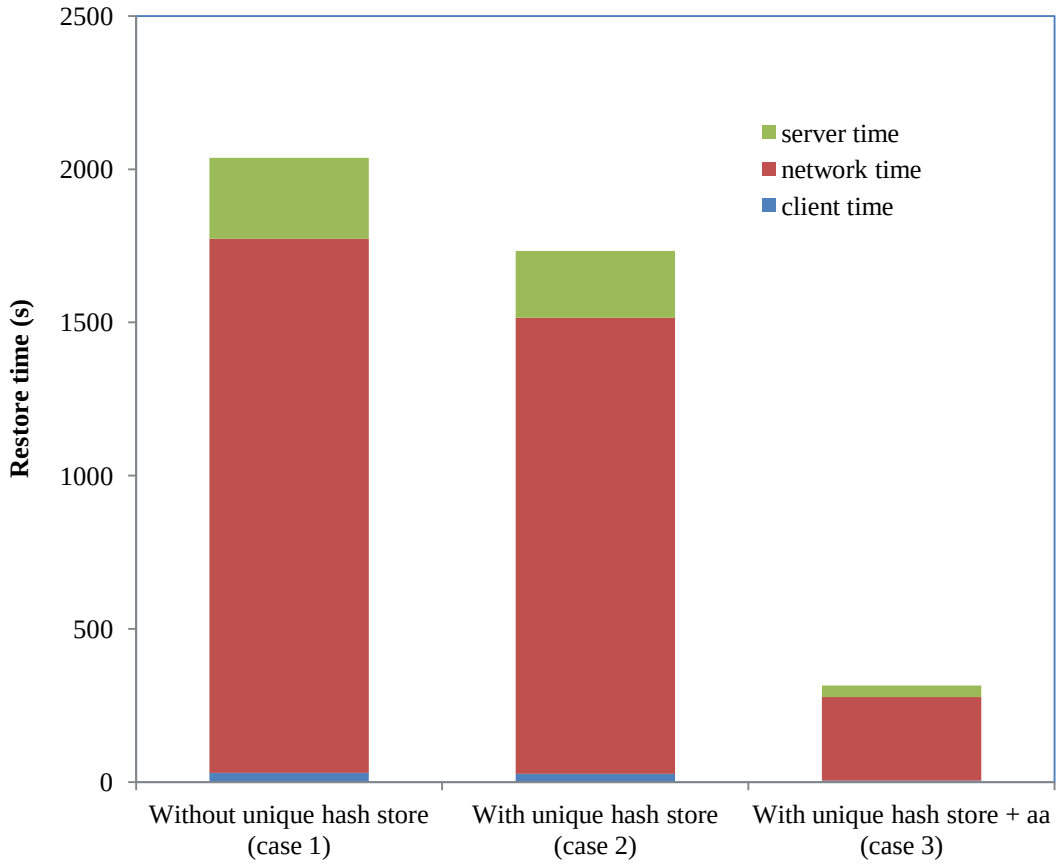
- *Case 1:* This setup runs the restore agent without using the unique hash store and without application-awareness
- *Case 2:* In this setup, the restore agent uses the unique hash store but without application-awareness

- *Case 3:* This setup runs the restore agent using the unique hash store and application-awareness

The restore times for Linux and virtual appliances workloads are shown in figure 4.5 (a) and Figure 4.5(b) In general, restore time can be divided into three categories, i) *client computer time (client time)* – this the processing time in the client, ii) *network time* – total transmission time and, iii) *server time* – processing time in the server. The network connection between the client and the server was rate-limited to 4MB/s. In order to evaluate the effect of application awareness, we define two important parameters: *apparent system restore time* and *actual system restore time.* The apparent restore time is the time it takes to restore only the critical data. It is 'apparent' because not all the backed up data is restored (only the critical data) even though the user can get back to business and thus stratifying the RTO. The actual restore time is the time it takes to restore the entire dataset.



(a) Linux kernels



(b) Virtual Appliances

Figure 4.5: Restore performance for different workloads

Our datasets exhibit different data redundancies. The Linux kernels have very high redundancy; the virtual appliances have minimal redundancy while the user videos in this case exhibit no redundancy. Figure 4.5 shows that the performance gap between a restore client using the unique hash store (case 2) and one without a unique hash store (case 1) is related to the redundancy in the dataset. The higher the redundancy, the bigger the gap. In incorporating application awareness, we randomly changed the access time for

20% of files in each dataset to signify critical files that were restored with application awareness. Clearly, network time is the dormant potion of the restore time.

Figure 4.6 shows the data transmitted over the WAN for each workload. To restore data, the restore agent sends hashes to the server and for each hash received, the server sends back a <*hash-chunk data*> back to the client. Using 8KB data chunks the total data transferred is determined as follows.

$$Total\ data = NH_T * H_S + NH_{T*}(H_S + C_S)$$

Where NH_T is the number of hashes sent to the server, H_S is the hash size in bytes (20 for SHA-1) and C_S is the chunk size in bytes (8KB in this thesis).

As expected, the reduction in data transmitted increases with the redundancy found in a dataset. This indicates that datasets with higher redundancy benefit more from our scheme. The Linux workload shows more that 85% reduction in data transferred over the WAN.

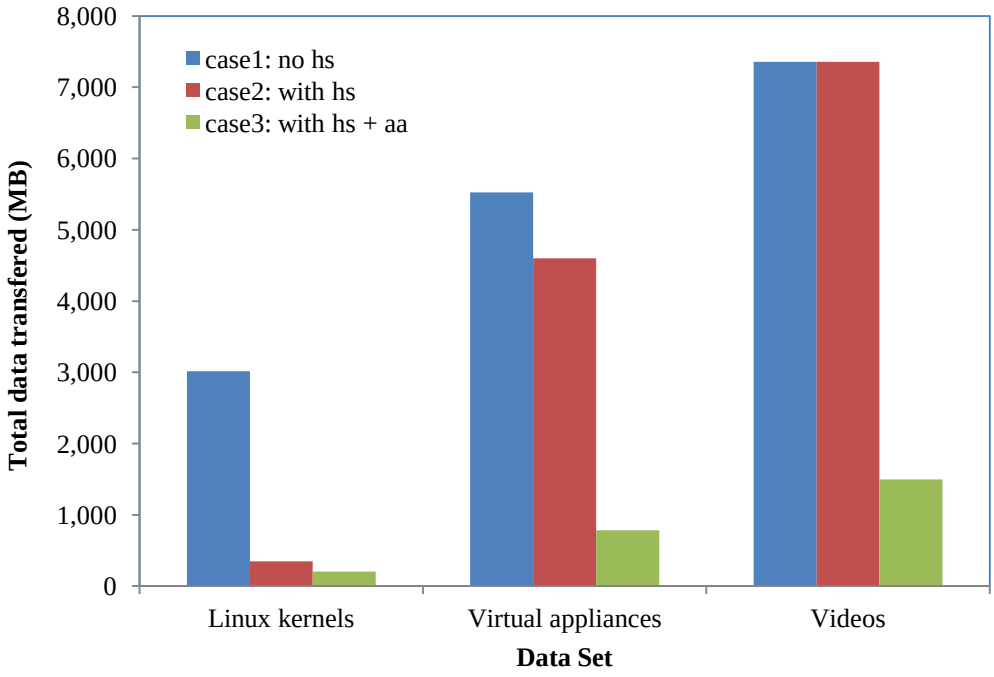


Figure 4.6: Data transferred over the WAN

In the second part of our evaluation, we compared the performance of our scheme to that of CABdedupe [1], which is the state-of-the-art. CABdedupe works by capturing the causal relationship among versions of dataset to remove the unmodified data from transmission. It assumes that after a disaster, part of the data will be available at the client. Given a list of files whose content has changed since the last backup, CABdedupe chunks the files and calculates fingerprints in order to determine which of the chunks from the modified file remained intact at the client. If all the data is lost, the scheme has little effect and all the data chunks for the given hashes must be retrieved from the server. Therefore, in this scenario, we can represent CABdedupe using case 1 of our scheme in which the client sends requests for all the hashes for a given dataset.

To evaluate the performance for the scenario when part of the data can be found at the client, we implemented a simple version of the CABdedupe restore workflow. 60% and 30% of the files in the Linux and virtual appliance dataset were randomly modified, after being backed up, by overwriting the lower half of each file with randomly generated data. Therefore, part of the data for each files remained intact. For both cases the network connection was rate limited to 4MB/s.

The performance results for the Linux and virtual machine workloads are show in Figures 4.7 and 4.8 respectively. Figure 4.8 shows that AAPR is not very effective when there is very little redundancy in the dataset e.g. 16.78% for the virtual appliance workload.

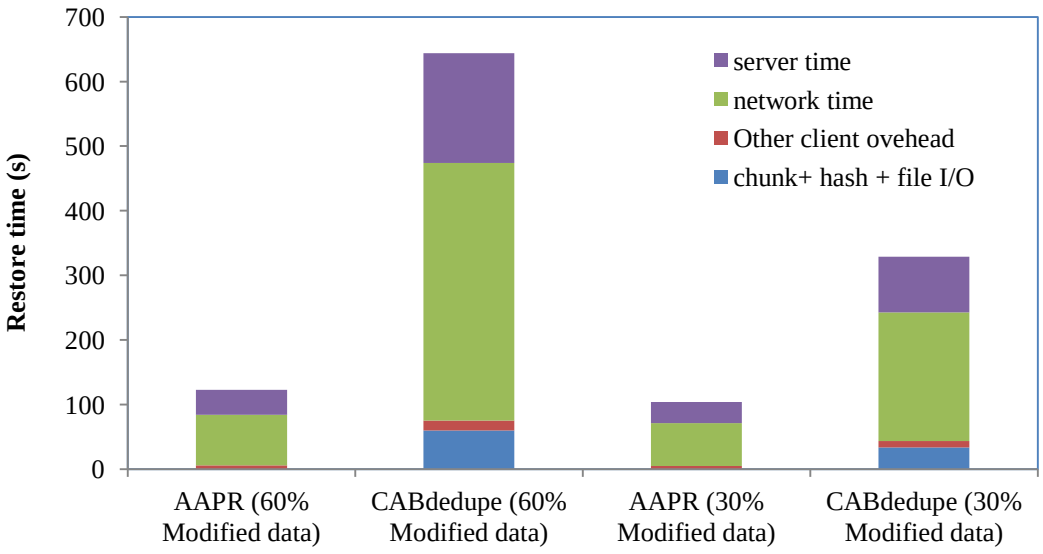


Figure 4.7: Comparison of AAPR and CABdedupe schemes – Linux workload

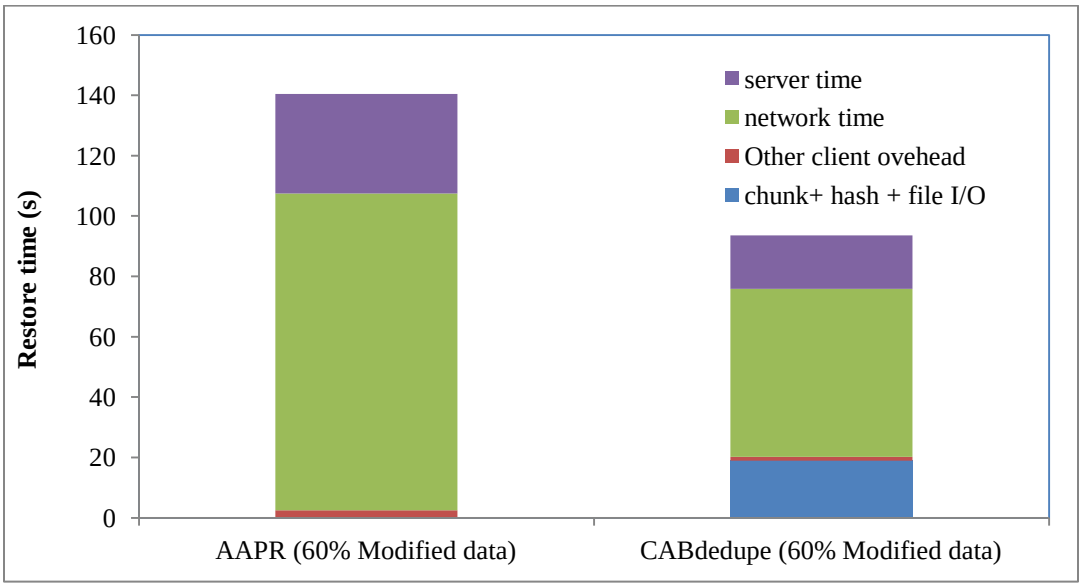


Figure 4.8: Comparison of AAPR and CABdedupe schemes – Virtual machines

4.4 Summary

Deduplication-based cloud backup services have increased in popularity partly due to the economics of scale provided by cloud computing. However, restore performance over low bandwidth internet links still remains a drawback in the use of these services. While the amount of digital data that needs to be backed up is growing rapidly and cloud computing is becoming ubiquitous, the internet (WAN) link speeds have not experienced corresponding growth.

Increasing bandwidth to improve network throughput in order to satisfy the need for fast restores over the WAN is very expensive. The alternative that emerges is

removing redundant data and/or compressing data before transmission over the WAN not only during the backup operation but also during the restore operation. Given that background, we proposed AAPR, a simple solution focused on restore performance over the WAN. Furthermore, we exploit application awareness to restore critical data first and thus improve the recovery point objective.

Our evaluations with real world workloads show that AAPR performs well in reducing data transferred over the network with over 85% reduction in data transferred for workloads with high redundancy.

Chapter 5

Conclusions

In this thesis, we investigated problems relating to backup and restore performance for deduplication-based cloud backup services. We investigated scalability and throughput for backup operations. We have also looked at restore performance for deduplication-based cloud backup clients. This thesis has made the following contributions:

(i) *Scalable Hybrid Hash Cluster (SHHC):* During the backup operation, duplicate data is determined by first consulting the chunk index. For a lager data set, it's not possible to store the whole index in RAM forcing the index lookup to go to the disk and incurring disk I/O penalties. Furthermore, in a public cloud environment the system has to serve numerous concurrent backup requests. This puts additional pressure on the throughput and scalability of the backup system.

Various schemes have been proposed to address the fingerprint lookup bottleneck problem and the associated disk I/O problem. However, none have explored the use of a distributed chunk index as a way of improving throughput and scalability. We propose a novel Scalable Hybrid Hash Cluster (SHHC) which hosts a low-latency distributed hash table for storing hashes. It is a distributed hash store and lookup service that can scale to handle hundreds of thousands concurrent backup requests while maintaining high fingerprint lookup throughput.

Results show that the hash cluster is consistently scalable and the throughput increases almost linearly with the number of nodes.

(ii) *Application-Aware Phased Restore (AAPR):* Deduplication based cloud backup services have increased in popularity but low bandwidth WAN links present a challenge to backup services that are expected to provide fast data restorations. High speed network connectivity is either too expensive for most users or reserved for special applications. The alternative that emerges is removing redundant data before transmission over the WAN during the restore operation.

To this end, we propose AAPR, a simple solution focused on restore performance over the WAN. AAPR is also loosely coupled from the cloud storage backend. We achieve the lose coupling by using file recipes at the client. Furthermore, we exploit application awareness to restore critical data first and thus improve the recovery point objective. Our evaluations with real world workloads show that AAPR is effective in reducing restore time

Bibliography

[1] Y. Tan, H. Jiang, D. Feng, L. Tian, and Z. Yan. CABdedupe: A Causality-Based Deduplication Performance Booster for Cloud Backup Services. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 1266-1277, 2011.

[2] L. Xu, J. Hu, S. Mkandawire, and H. Jiang. SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services in Data Centers. In *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 61-65, June 2011.

[3] The 2010 Digital Universe Study: A Digital Universe Decade – Are You Ready? http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf

[4] The 2011 Digital Universe Study: Extracting Value from Chaos. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-r.pdf

[5] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, and A. Perrig. Opportunistic use of content addressable storage for distributed file systems. *In Proceedings of the 2003 USENIX Annual Technical Conference*, 2003.

[6] http://searchcloudcomputing.techtarget.com/definition/cloud-computing

[7] http://cibecs.com/resource-center/white-papers/2010-data-loss-survey/

[8] CIBECS 2011 Report: Business Data Loss Survey. http://cibecs.com/wp-content/uploads/2011/09/Survey-2011_Aug-E.pdf

[9] Cloud Backup and Recovery Requirements. http://www.snia.org/sites/default/education/tutorials/2011/spring/cloud/BaigAshar_Cloud_BURR_Requirements_2.pdf

[10] The Role of WAN Optimization in Cloud Infrastructures. http://www.snia.org/sites/default/education/tutorials/2011/spring/cloud/TsengJosh_Role_of_WAN_Opt_Cloud_3-18-2011.pdf

[11] 2010 Data Protection Trends ESG Research Report. http://www.enterprisestrategygroup.com/2010/04/2010-data-protection-trends/

[12] T. Harter, C. Dragga, M. Vaughn, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. A file is not a file: understanding the I/O behavior of Apple desktop applications. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*, 2011.

[13] NIST. Secure Hash Standard (SHS). In FIPS Publication 180-1 (1995). http://www.itl.nist.gov/fipspubs/fip180-1.htm

[14] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," In *ACM Transactions on Storage*, Volume 3 Issue 3, October 2007.

[15] A. Muthitacharoen, B. Chen, and D. Mazi. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01)*, 2001.

[16] N. Jain, M. Dahlin, and R. Tewari. TAPER: tiered approach for eliminating redundancy in replica synchronization. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05),* 2005.

[17] P. Shilane, M. Huang, G. Wallace, and W. Hsu. WAN Optimized Replication of Backup Datasets Using Stream-Informed Delta Compression. In *Proceedings of the 10th USENIX Conference on File and storage Technologies (FAST'12),* 2012.

[18] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies(FAST'08),* pages 269–282, 2008.

[19] B. Debnath, S. Sengupta, and J. Li. ChunkStash: speeding up inline storage deduplication using flash memory. In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIXATC'10),* page 16, 2010.

[20] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th Conference on File and Storage Technologies* (FAST '09), 2009.

[21] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS),* 2009.

[22] D. Meister and A. Brinkmann. dedupv1: Improving deduplication throughput using solid state drives (SSD). In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1-6, 2010.

[23] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*, 2011.

[24] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proceedings of the USENIX Annual Technical Conference (ATEC '99)*, 1999.

[25] Oracle Database Documentation Library. http://www.oracle.com/pls/db102/ homepage.

[26] The Linux Kernel Archives. http://www.kernel.org/pub/

[27] http://www.thoughtpolice.co.uk/vmware/

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

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. In *IEEE/ACM Transactions on Networking*, pages 17- 32, 2003.

[30] R. Rangaswami. http://users.cis.fiu.edu/~raju/WWW/

[31] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. In *Communications of the ACM* , Volume 46 Issue 2, February 2003.

[32] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Data Storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02)*, 2002.

[33] http://en.wikipedia.org/wiki/File:Cloud_computing.svg

[34] 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. In *Proceedings of the 2011 USENIX Annual Technical Conference (USENIXATC'11),* 2011.

.