UNIVERSITY OF TORONTO

Avoiding the Disk Bottleneck in the Data Domain Deduplication File System

Qi Zhao
Yilin Han

Conetent

- → Introduction
- → Architecture
- → Acceleration Methods
- → Experimental Results
- → Discussion

Introduction

- Disk-based deduplication storage has emerged as the new-generation storage system for enterprise data protection to replace tape libraries.
- Deduplication removes redundant data segments to compress data into a highly compact form and makes it economical to store backups on disk instead of tape.
- This paper describes techniques employed in the production Data Domain deduplication file system to relieve the disk bottleneck.

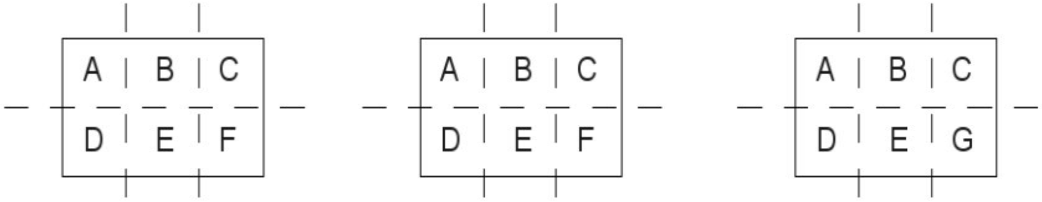
Traditional Backup solution

1. Low Cost
2. High Performance
3. Traditional Solution has been to use tape libraries as secondary storage for disaster recovery.

Tape VS Disk

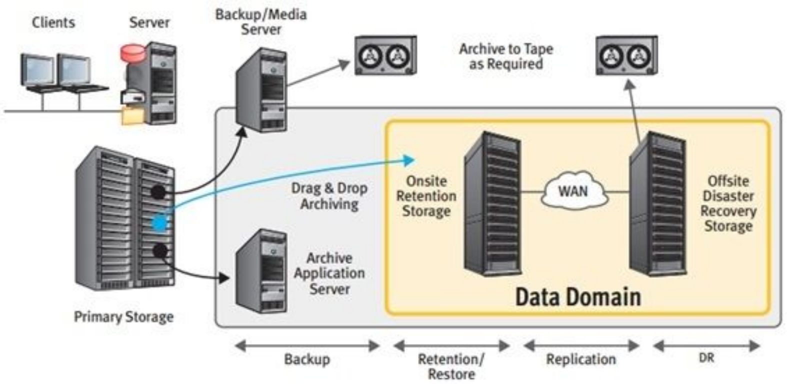
Tape	Hard disk
Sequential access	Direct access
Cheap	Expensive than tape
Random access slow	Access in milliseconds

Motivation



document1.docx	document2.docx	document_new.docx	
6 MB	6 MB	6 MB	
Without De-dup			
6 MB	6 MB	6 MB	= 18 MB
With De-dup			
6 MB	0 MB	1 MB	= 7 MB

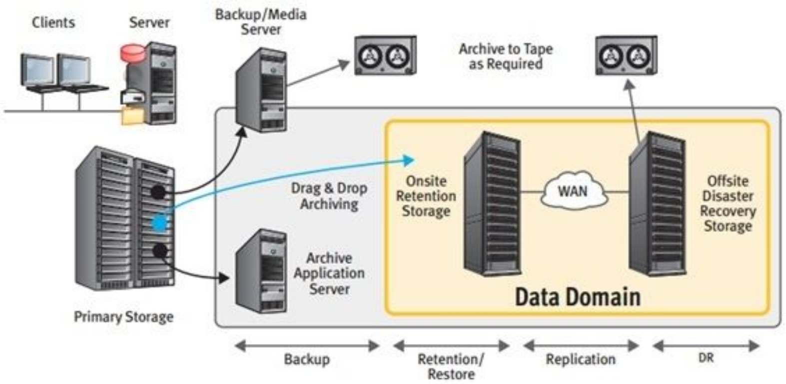
Challenges



1. Performance Challenges: Finding duplicate segments.

2. Given a segment of size of 8KB and a performance target of 100 MB/sec=> a system must process 12,000 segments per second

Challenges

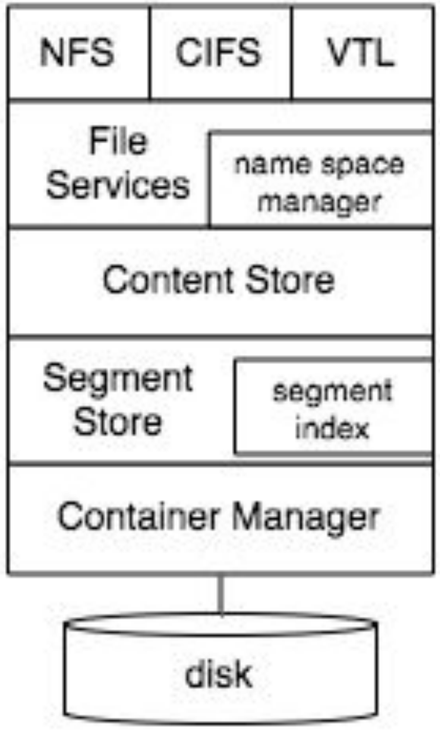


1. Byte Comparison
2. In-memory index of all segment fingerprints
3. On-disk index of all segment fingerprints with a cache to accelerate access

Architecture

Data Domain File System (DDFS)

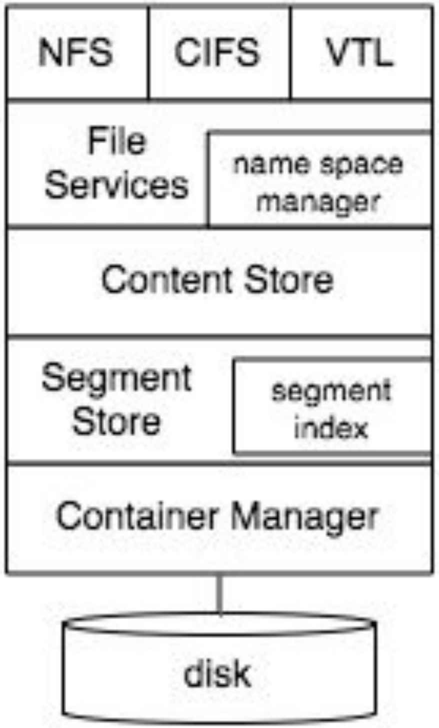
01 | Content Store
02 | Segment Store
03 | Container Manager



Architecture

Content Store

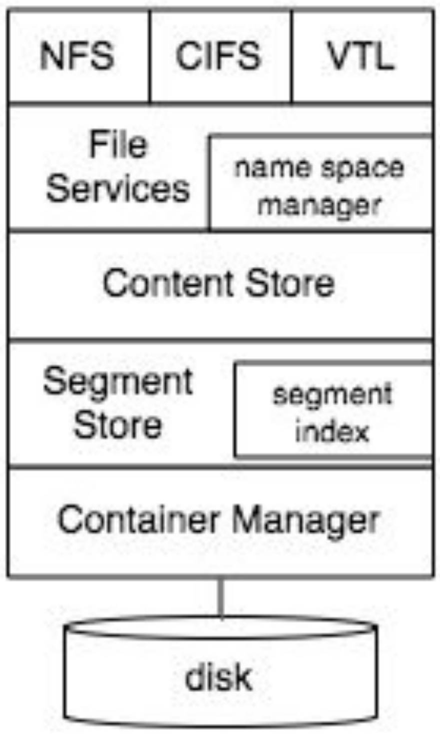
01 | Anchoring
02 | Segment Fingerprinting
03 | Segment Mapping



Architecture

Segment Store

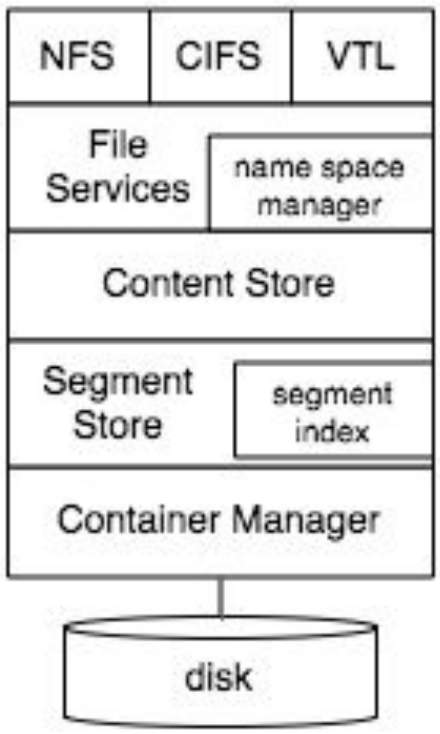
01 | Segment Filtering
02 | Container Packing
03 | Segment Indexing



Architecture

Segment Store : Read

01 | Segment Lookup
02 | Container Retrieval
03 | Container Unpacking



Architecture

Container Manager

01 | Fixed container size
02 | Large granularity
03 | Immutable

Figure 2: Containers are self-describing, immutable, units of storage several megabytes in size. All segments are stored in containers.

Acceleration Methods

Summary Vector

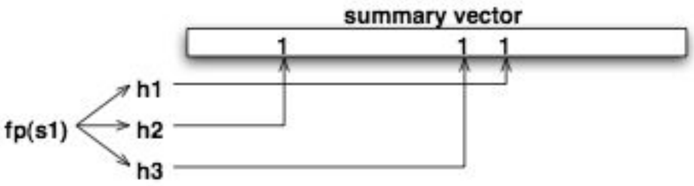
01 | Purpose: reduce disk I/O time of none exists

02 | Implementation: Bloom Filter

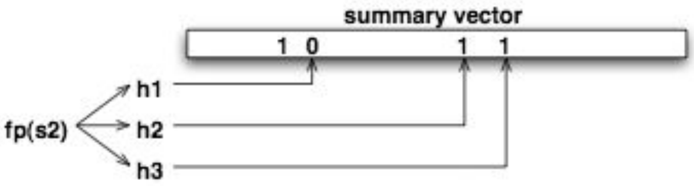
03 | Three operations → Init(), Insert(fingerprint), Lookup(fingerprint)

04 | Tradeoff: false positive

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}.$$



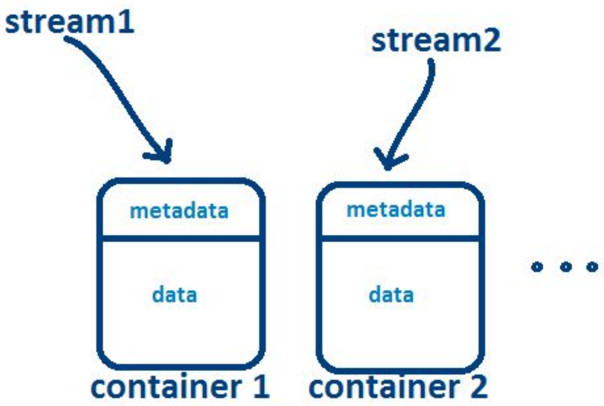
(a) Inserting segment s1 to the Summary Vector



(b) Looking up segment s2 in the Summary Vector

Acceleration Methods (Cont.)

Stream-Informed Segment Layout (SISL)



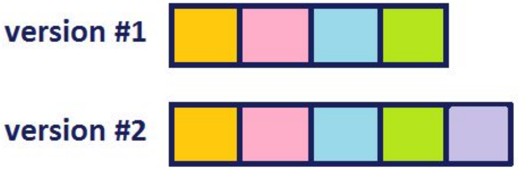
Segments appear in similar sequence

01 | Assumption and Observation → Segment duplicate locality

02 | Implemented in Content Store, and Segment store

03 | Process to create SISL

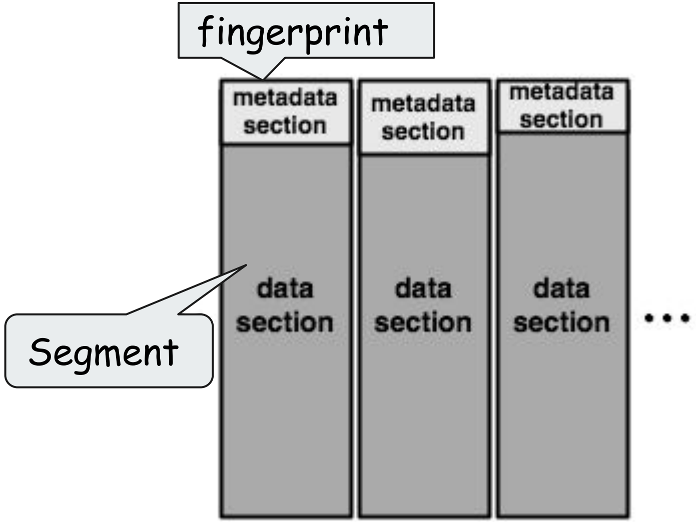
04 | Parallelization

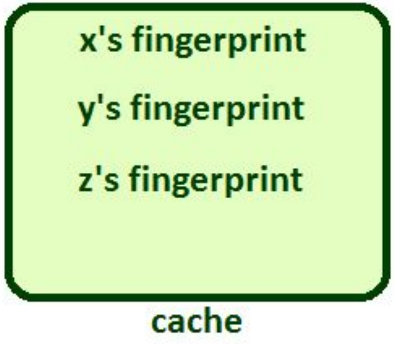


Acceleration Methods (Cont.)

Locality Preserved Caching (LPC)

01 | Motivation: high miss ratio
02 | Segment Cache
03 | LPC Process
04 | Container based LRU caching strategy





Combination of Acceleration Methods

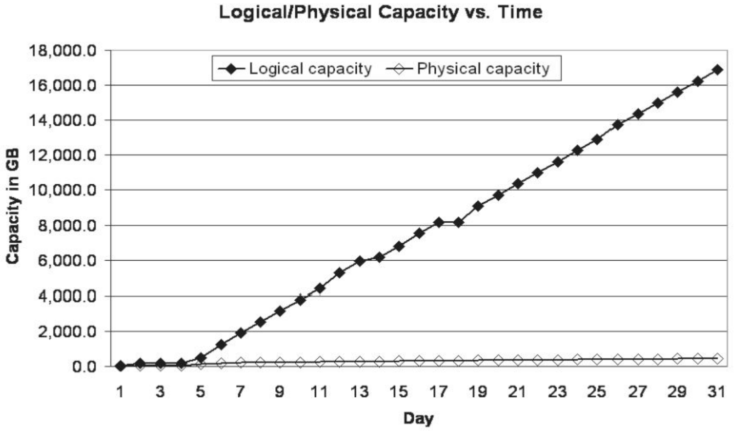
For an incoming segment for write, the algorithm does the following:

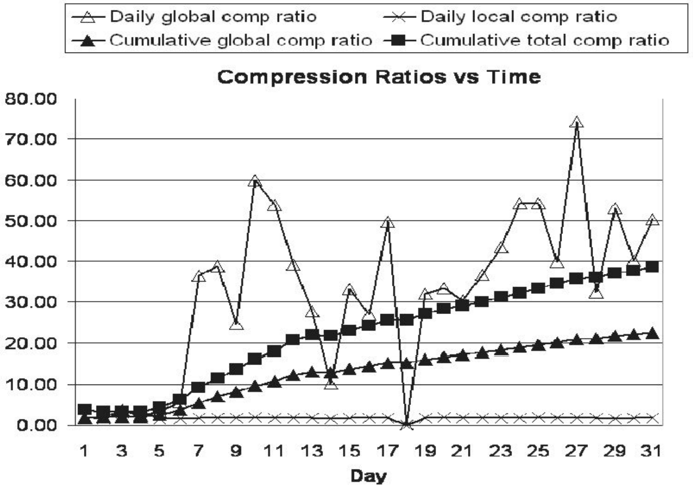
- ➔ **Check segment cache**
  - ◆ **If it is in, the segment is a duplicate**
- ➔ **Check the summary vector**
  - ◆ **If it is not in, write the segment into container (it is a new segment)**
- ➔ **Lookup segment index for container ID**
  - ◆ **If it is in, the segment is a duplicate (but not in cache)**
    - **Fetch all the container metadata into segment cache**
    - **Leverage LRU to remove old one then fetch (full cache)**
- ➔ **Write the segment into container**

Experimental Results

	Min	Max	Average	Standard deviation
Daily global compression	10.05	74.31	40.63	13.73
Daily local compression	1.58	1.97	1.78	0.09

Table 1: Statistics on Daily Global and Daily Local Compression Ratios at Data Center A





Experimental Results (Cont.)

	Exchange data		Engineering data	
	# disk I/Os	% of total	# disk I/Os	% of total
no Summary Vector and no Locality Preserved Caching	328,613,503	100.00%	318,236,712	100.00%
Summary Vector only	274,364,788	83.49%	259,135,171	81.43%
Locality Preserved Caching only	57,725,844	17.57%	60,358,875	18.97%
Summary Vector and Locality Preserved Caching	3,477,129	1.06%	1,257,316	0.40%

Table 4: Index and locality reads. This table shows the number disk reads to perform index lookups or fetches from the container metadata for the four combinations: with and without the Summary Vector and with and without Locality Preserved Caching. Without either the Summary Vector or Locality Preserved Caching, there is an index read for every segment. The Summary Vector avoids these reads for most new segments. Locality Preserved Caching avoids index lookups for duplicate segments at the cost an extra read to fetch a group of segment fingerprints from the container metadata for every cache miss for which the segment is found in the index.

Experimental Results (Cont.)

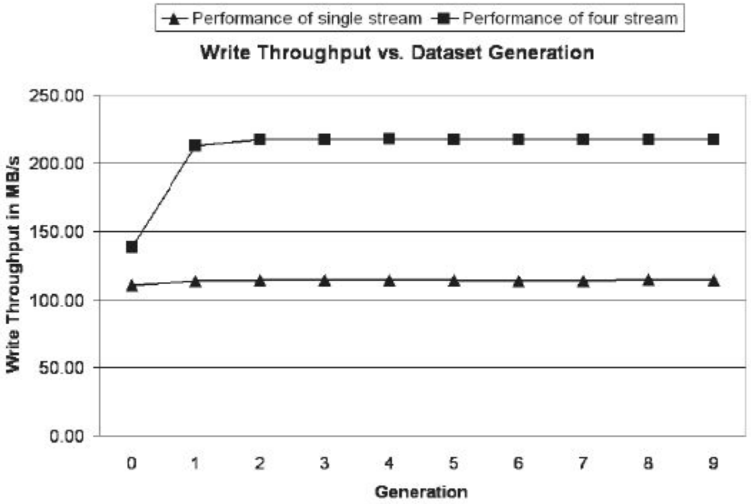


Figure 8: Write Throughput of Single Backup Client and 4 Backup Clients.

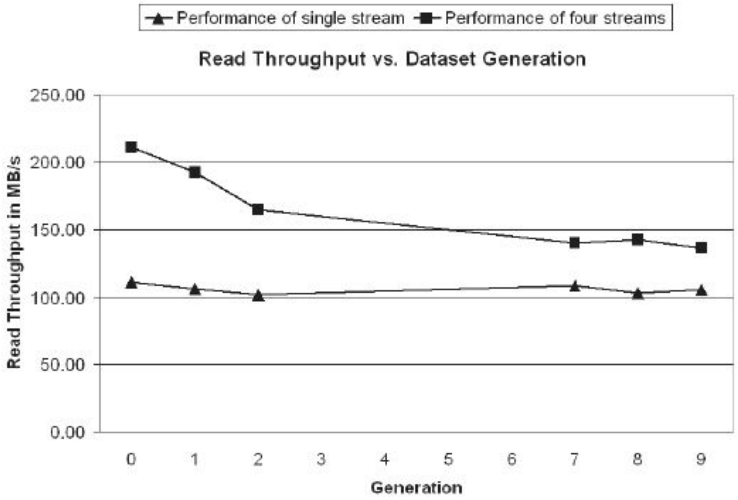


Figure 9: Read Throughput of Single Backup Client and 4 Backup Clients

Discussion

Advantages:

- high compression rate/ high performance
- low cost

Disadvantages:

- extra CPU overhead
- only inline deduplication (cannot be used online)

Discussion

- Summary Vector (Bloom Filter) increment problems
- Fragmentation
- Duplication detection when multiple streams writing in parallel
- Recovery
- SHA-1 Collision (1/2^160)

Thanks for listening