Primary Data Deduplication – Large Scale Study and System Design

A. El-Shimi, R. Kalach, A. Kumar, J. Li, A. Oltean, S. Sengupta

Microsoft Corporation, Redmond (USA)

Primary Data Deduplication for File-based Storage

- Relatively recent interest vs. backup data dedup
- Driving forces
  - 50% year-over-year growth in file based data
  - #1 technology feature when choosing a storage solution
- Technology challenges
  - Continue to serve “primary” workload from same copy of data
  - Balance resource consumption (CPU/memory/disk I/O), dedup space savings, and dedup throughput

Key Requirements for Primary Data Deduplication

- Optimize for unique data
  - More than 50% of data could be unique (vs. 90+% duplication rates in backup data)
- Primary workload friendly
  - Maintain efficient access to data (both sequential and random I/O)
  - Deduplication cannot assume dedicated resources and must “yield” to primary workload
- Broadly used platform
  - Must run well on a low-end server
  - Huge variability in workloads and hardware platforms

Key Design Decisions

- Post-processing deduplication
  - Preserve latency/throughput of primary data access
  - Flexibility in scheduling dedup as background job on cold data
- Deduplication granularity and data chunking
  - Chunk-level: variable sized chunking, large chunk size (~80KB)
  - Modifications to Rabin fingerprint based chunking to achieve more uniform chunk size distribution
- Deduplication resource usage scaling slowly with data size
  - Reduced chunk metadata
  - RAM frugal chunk hash index
  - Data partitioning

Large Scale Study of Primary Datasets

- Used to drive key design decisions
- About 7TB of data spread across 15 globally distributed servers in a large enterprise
- Data crawled and chunked at different average chunk sizes
  - Using Rabin fingerprint based variable sized chunker
  - SHA-1 hash, size, compressed size, offset in file, file information logged for each chunk

Workload	Srvrs	Users	Total Data	Locations
Home Folders (HF)	8	1867	2.4TB	US, Dublin, Amster-dam, Japan
Group File Shares (GFS)	3	*	3TB	US, Japan
Sharepoint	1	500	288GB	US
Software Deploy-ment Shares (SDS)	1	†	399GB	US
Virtualization Libraries (VL)	2	†	791GB	US
Total	15		6.8TB	

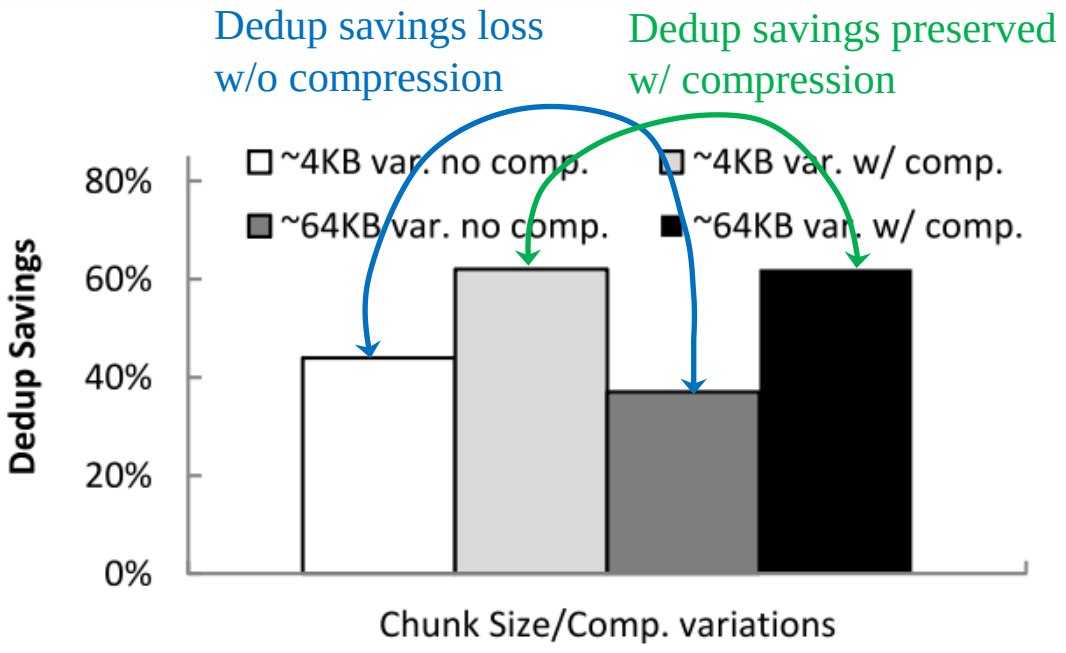
*Number of authors (users) assumed in 100s but not quantifiable due to delegated write access. †Number of (authors) users limited to < 10 server administrators.

Key Design Decisions

- Post-processing deduplication
  - Preserve latency/throughput of primary data access
  - Flexibility in scheduling dedup as background job on cold data
- Deduplication granularity and data chunking
  - Chunk-level: variable sized chunking, large chunk size (~80KB)
  - Modifications to Rabin fingerprint based chunking to achieve more uniform chunk size distribution
- Deduplication resource usage scaling slowly with data size
  - Reduced chunk metadata
  - RAM frugal chunk hash index
  - Data partitioning

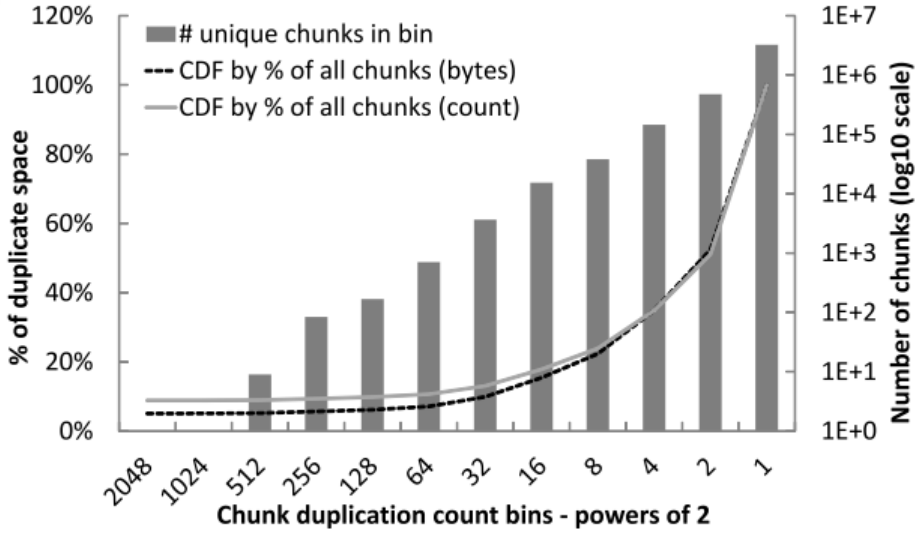
Average Chunk Size

- Compression compensates for savings decrease with higher chunk size
  - Compression is more efficient on larger chunks
- Use larger chunk size of ~64KB
  - Without sacrificing dedup savings
  - Reduce chunk metadata in the system



GFS-US dataset

Chunk Reference Count

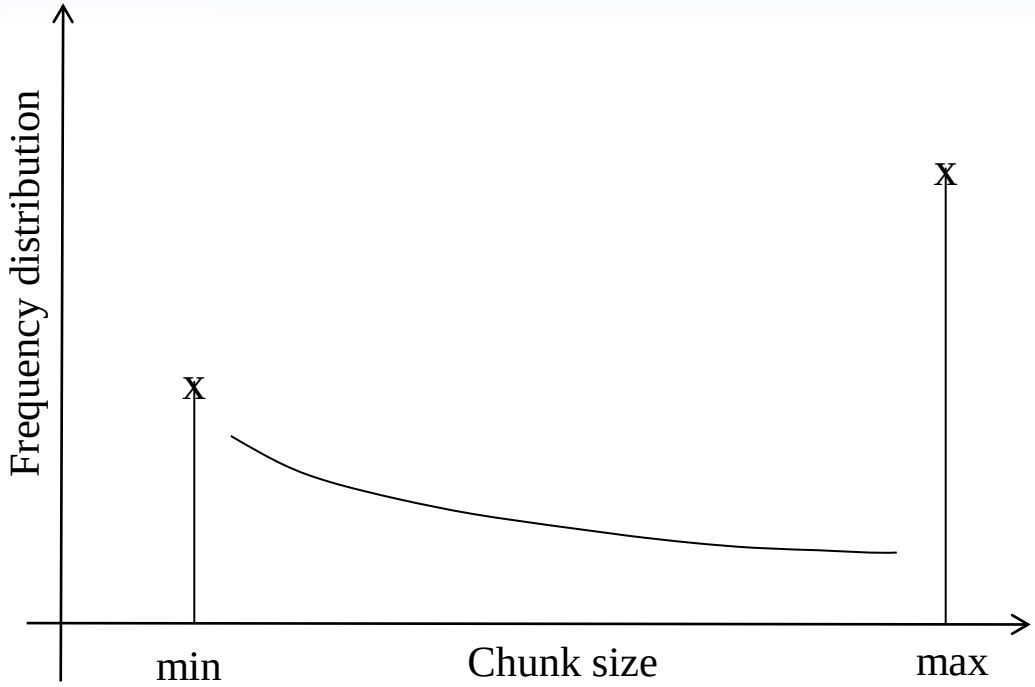


GFS-Japan-1 dataset

- Majority of duplicate bytes reside in middle portion of distribution
  - Not sufficient to dedup just high ref count chunks
- System needs to deduplicate all chunks that appear more than once
  - Implications on the chunk hash index design

Basic version of fingerprint based chunking

- Skewed chunk size distribution
  - Small chunk size => increase in chunk metadata in the system
  - Large chunks => reduced dedup savings, benefit of caching



- Forced chunk boundaries
  - Forced boundary at max chunk size is content independent, hence may reduce dedup savings

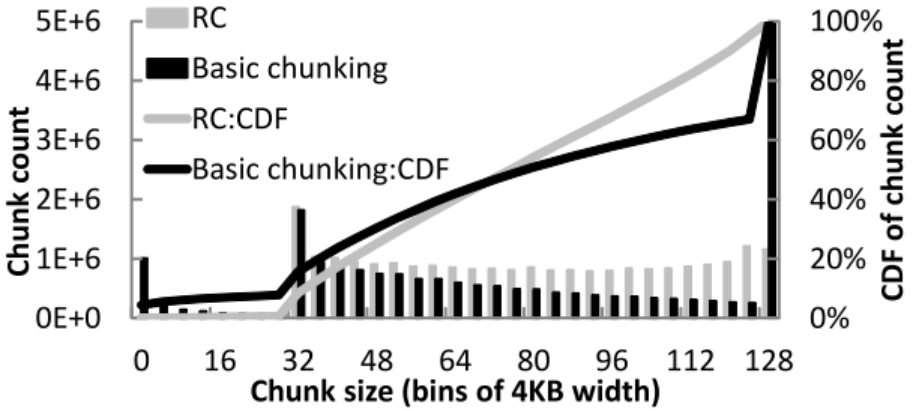
Regression Chunking Algorithm

- Goal 1: To obtain uniform chunk size distribution
- Goal 2: Reduce forced chunk boundaries at max size
- Basic idea
  - When max chunk size is reached, relax match condition to some suffix of bit pattern P
  - Match |P| - i bits of P, with decreasing priority for i=0,1, …, k
- Reduces probability of forced boundary at max size
  - $2x10^{-3}$ for k=1, 10^{-14} for k=4

Regression Chunking Algorithm contd.

- Maintains chunking throughput performance
  - Core matching loop checks against smallest prefix, break out only if match occurs
  - Single pass over data: remember match position for each relaxed suffix match

Regression Chunking Performance



GFS-US dataset

Uniform chunk size distribution

Dedup savings improvement

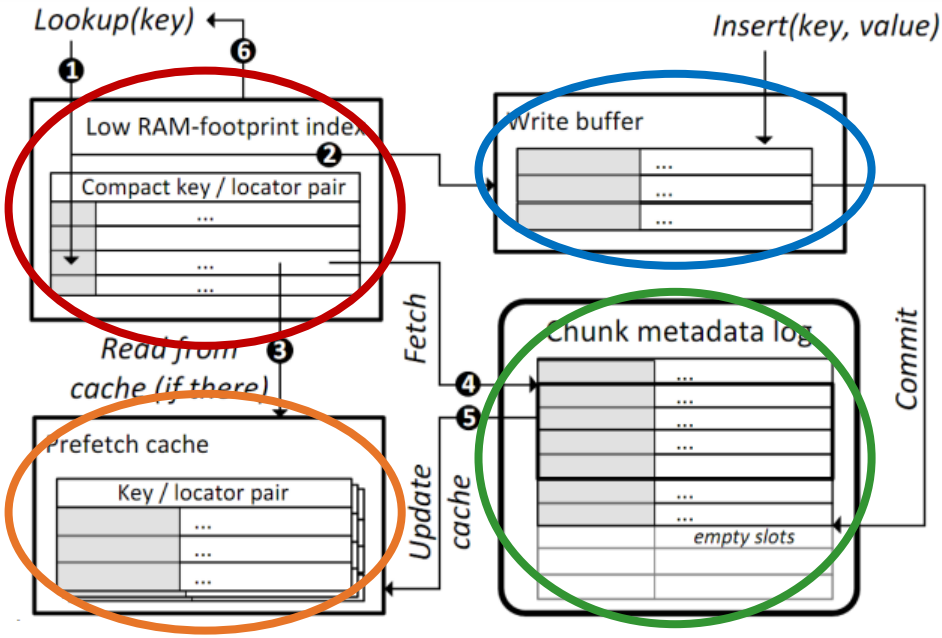
Dataset	Dedup Space Savings		
	Basic Chunking	Regression Chunking (RC)	RC Benefit
Audio-Video	2.98%	2.98%	0%
PDF	9.96%	12.70%	27.5%
Office-2007	35.82%	36.65%	2.3%
VHD	48.64%	51.39%	5.65%
GFS-US	36.14%	37.2%	2.9%

Key Design Decisions

- Post-processing deduplication
  - Preserve latency/throughput of primary data access
  - Flexibility in scheduling dedup as background job on cold data
- Deduplication granularity and data chunking
  - Chunk-level: variable sized chunking, large chunk size (~80KB)
  - Modifications to Rabin fingerprint based chunking to achieve more uniform chunk size distribution
- Deduplication resource usage scaling slowly with data size
  - Reduced chunk metadata
  - RAM frugal chunk hash index
  - Data partitioning

Chunk Indexing

- Log-structured organization
  - Chunk metadata organized in log-structured manner on disk
  - Insertions aggregated in write buffer in RAM and appended to log in single I/O
- Low RAM footprint index
  - Specialized hash table using variant of cuckoo hashing
  - 2-byte signature, 4-byte pointer per entry => 6-bytes of RAM per indexed chunk



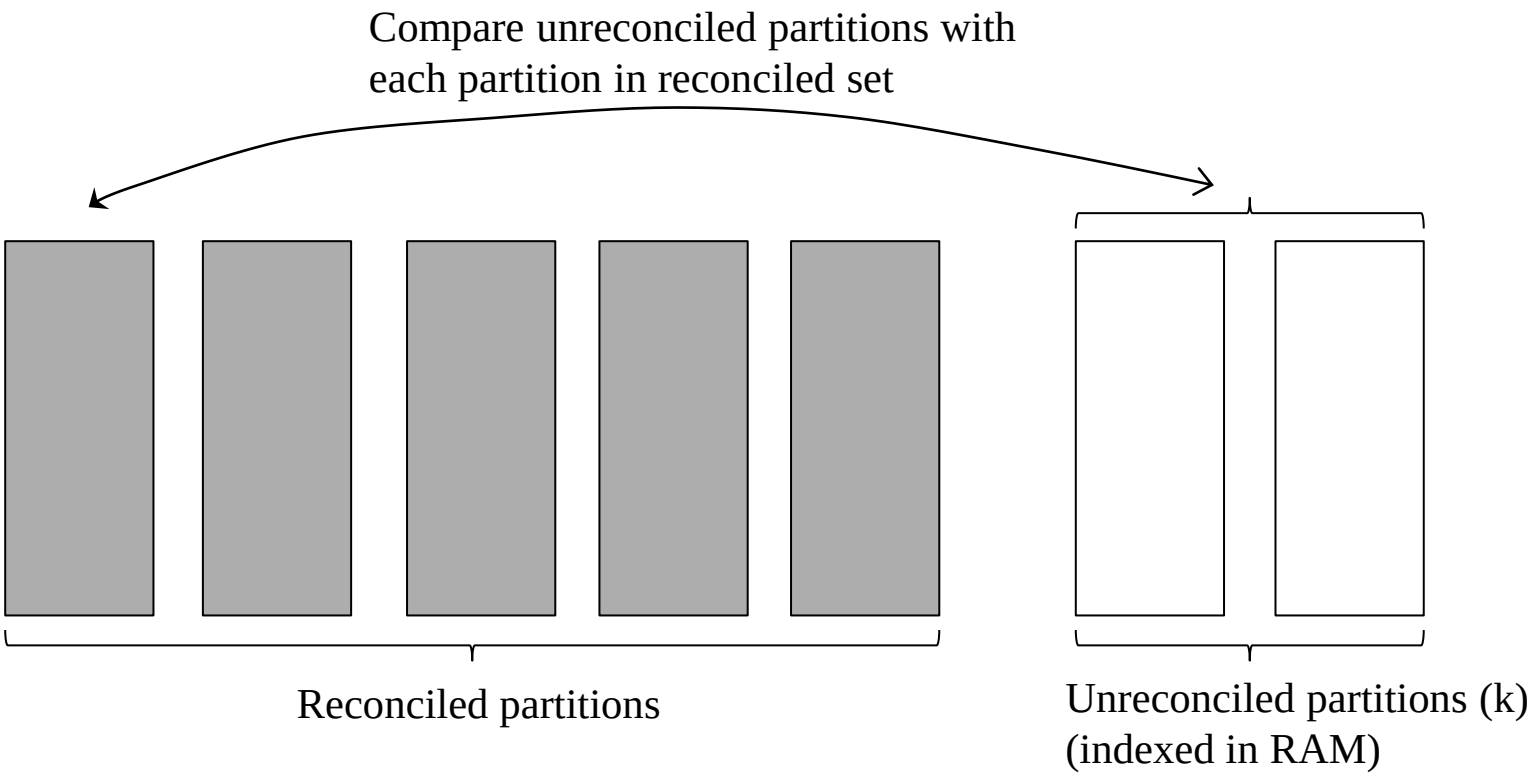
Chunk Indexing contd.

- Prefetch Cache
  - Prefetch chunk mappings for next 100-1000 chunks in same I/O
  - Exploit sequential predictability of chunk hash lookups
    - Locality expected to be less than in backup workloads
  - Prefetch cache sized at 100,000 entries (5MB of RAM)
  - About 1% of index lookups hitting disk (on all datasets evaluated)
    - Hash table acts as a bloom filter on new chunk lookups

Data Partitioning and Reconciliation

- Two-phase deduplication
  - Divide the data into disjoint partitions, and perform deduplication within each partition
  - Reconcile duplicates across partitions
- Reconciliation algorithm
  - Iterative procedure
  - Grow the set of reconciled partitions by considering some number of unreconciled partitions at a time
- Reconciliation Strategy
  - Selective reconciliation
  - Delayed reconciliation

Reconciliation of Data Partitions



k = #unreconciled partitions considered per iteration; provides trade-off between memory usage and reconciliation speed

Efficient partitioning strategies

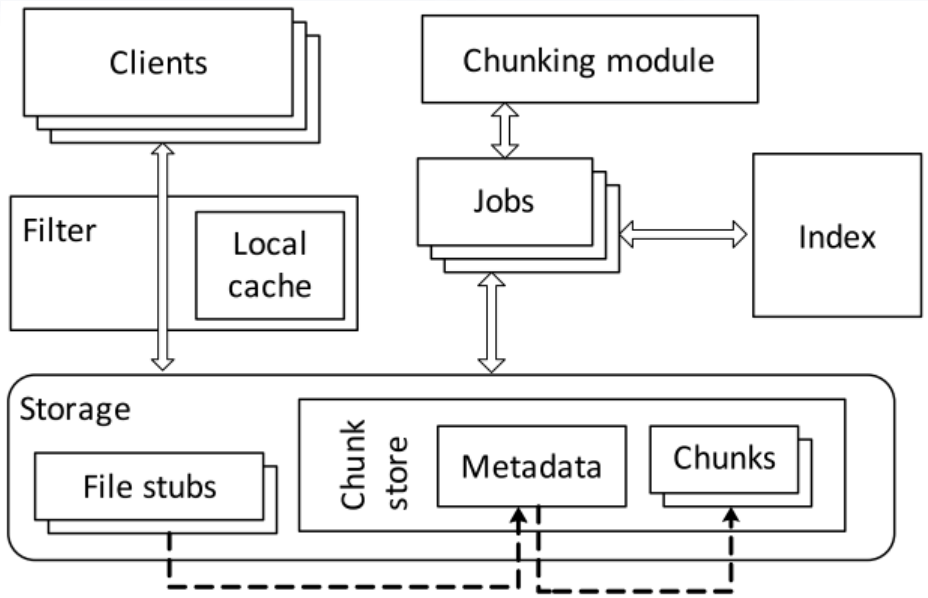
- Partition data and dedup within each partition
  - How close is dedup savings within partitions to that of global dedup?
- Partitioning by file type
  - Dedup savings almost as good as with global dedup
- Partitioning by file path
  - Partition by directory sub-trees (each partition ≤ 10% of total bytes)
  - Not as effective as partitioning by file type for preserving dedup savings
- Partitioning by system/volume

Dataset	Dedup Space Savings		
	Global	Clustered by	
		File type	File path
GFS-US	36.7%	35.6%	24.3%
GFS-Japan-1	41.1%	38.9%	32.3%
GFS-Japan-2	39.1%	36.7%	24.8%
HF-Amsterdam	15.2%	14.7%	13.6%
HF-Dublin	16.8%	16.2%	14.6%
HF-Japan	19.6%	19.0%	12.9%

Dedup amenable to partitioned processing

System Overview

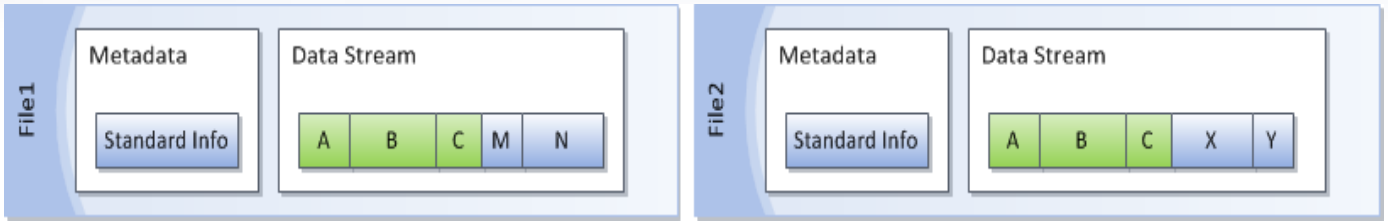
- Data path
  - Dedup filter
  - Chunk cache
  - File stub tx update
- Deduplication pipeline
  - Data chunking
  - Index lookups + insertions
  - Chunk Store insertions
- Background jobs
  - Garbage collection (in Chunk Store)
  - Data scrubbing



Deduplication and on-disk structures

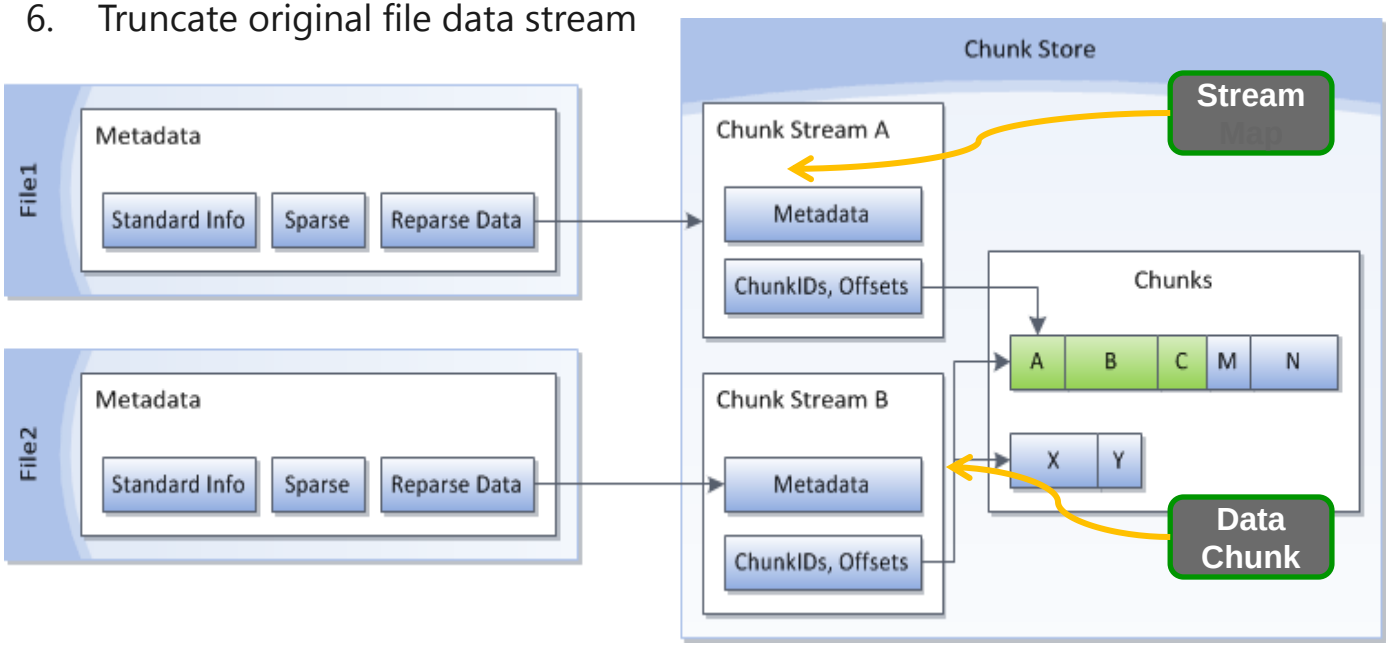
Phase I – Identify the duplicate data

1. Scan files according to policy
2. Chunk files intelligently to maximize recurring chunks
3. Identify common data chunks



Phase II – Optimize the target files

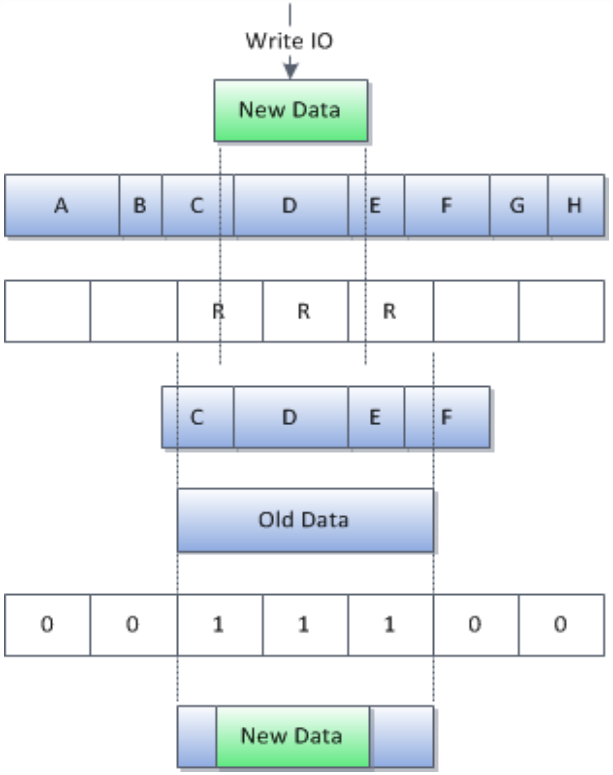
4. Single-instance data chunks in file stream order
5. Create stream metadata
6. Truncate original file data stream



Write path to Optimized File

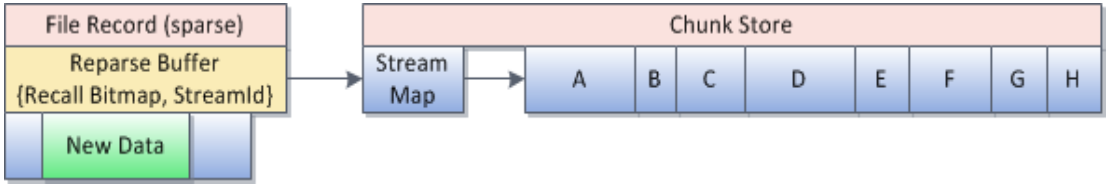
① Pre write File Layout

② Write flow



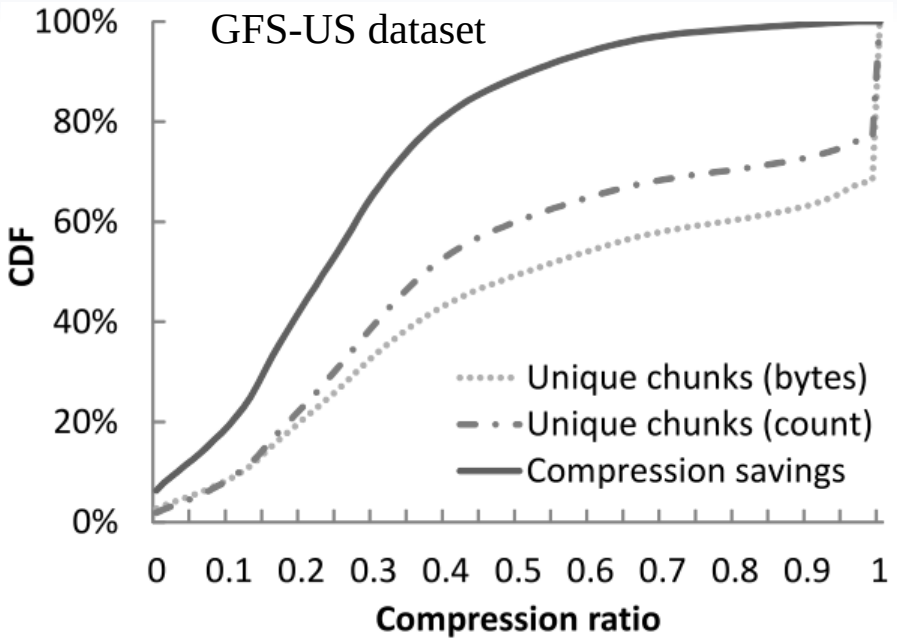
- Reduce latency for small writes to large files (e.g. OS image patching scenario)
- Recall granularity grows with file size
- Incremental Dedup will later optimize the new data
- GC cleans up unreferenced chunks (chunk “D” in example)

③ Post Write File Layout



Perf. Improvement: Chunk Compression

- Compression/decompression can have a significant perf impact
- Compression savings is skewed
  - 50% of unique chunks responsible for 86% of compression savings
  - 31% of chunks do not compress at all
- Solution: selective compression
  - Reduces cost of compression for large fraction of chunks
    - While preserving most of compression savings
  - Reduces decompression costs (reduce CPU pressure during heavy reads)
    - Also: use a cache for decompressed data (important for hotspots)
  - Heuristics for determining which chunks should not be compressed



Performance Evaluation – Throughput

- Quad-core Intel Xeon 2.27GHz machine, 12GB RAM
- Four scenarios, from combinations of
  - Index type (pure in-memory vs. memory/disk)
  - Data partitioning (off or on)

	Regular Index (Baseline)	Optimized index	Regular index w/ partitions	Optimized index w/ partitions
Throughput (MB/s)	30.6	28.2	27.6	26.5
Partitioning factor	1	1	3	3

GFS-US dataset

- Deduplication throughput
  - 25-30 MB/s (single thread performance)
  - Only about 10% decrease from baseline to least memory case
  - Three orders of magnitude higher than typical data ingestion rates of 0.03 MB/sec (Leung, et al.)

Performance Evaluation – Resource Usage

- RAM frugality
  - Index memory usage reduction of 24x vs. baseline
- Low CPU utilization
  - 30-40% per core
  - Enough room available for serving primary workload in multi-core modern file servers
- Low disk usage
  - Median disk queue depth is zero in all cases
  - At 75-th percentile, increase by 2-3; impact of index lookups going to disk and/or reconciliation

	Regular Index (Baseline)	Optimized index	Regular index w/ partitions	Optimized index w/ partitions
Partitioning factor	1	1	3	3
Index entry size (bytes)	48	6	48	6
Index memory usage	931MB	116MB	310MB	39MB
Single core utilization	31.2%	35.2%	36.8%	40.8%

Performance Evaluation – Parallelizability

- Parallel processing across datasets and CPU cores/disks
  - Disk diverse datasets
    - One session per volume in current implementation
  - One CPU core allocated per dedup session
    - One process and thread per deduplication session
  - No cross-dependencies in deduplication sessions (each session uses a separate index)
  - Aggregate dedup throughput scales as expected with number of cores (provided sufficient RAM is available)
- Workload scheduler
  - Assigns jobs (deduplication, GC, scrubbing) with CPU cores
  - Allocates memory per job
  - Keeps track of job activity (cancel jobs on memory or CPU pressure)

Summary

- Large scale study of primary data dedup
  - 7TB of data across 15 globally distributed servers in a large enterprise
- Primary data deduplication in Windows Server 2012
  - Design decisions driven by data analysis findings
- Primary workload friendly
  - Scale deduplication processing resource usage with data size
    - CPU/memory/disk IO
  - Data chunking and compression
  - Chunk indexing
  - Data partitioning and reconciliation
- Primary data serving, reliability, and resiliency aspects not covered in this paper