Primary Data Deduplication – Large Scale Study and System Design

Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Oltean, Jin Li,
Sudipta Sengupta
Microsoft Corporation, Redmond, WA, USA

ATC 2012

Background

- **Primary data**
  - No regular backup circle
  - Smaller deduplication ratio than backup dataset’s
  - Focus on performance of read/write
- **Backup dataset**
  - Regular backup circle
  - About 90+% deduplication ratio
  - Focus on data reliability and storage saving

Challenges

- Low deduplication ratio needs new strategies for unique chunks rather than traditional strategies in backup dataset.
- Maintain efficient access to primary data and avoid degradation because of deduplication process
- Balance resource consumption (CPU/memory/disk I/O), space saving, and throughput without dedicated hardware

Large Scale Study of Primary Dataset

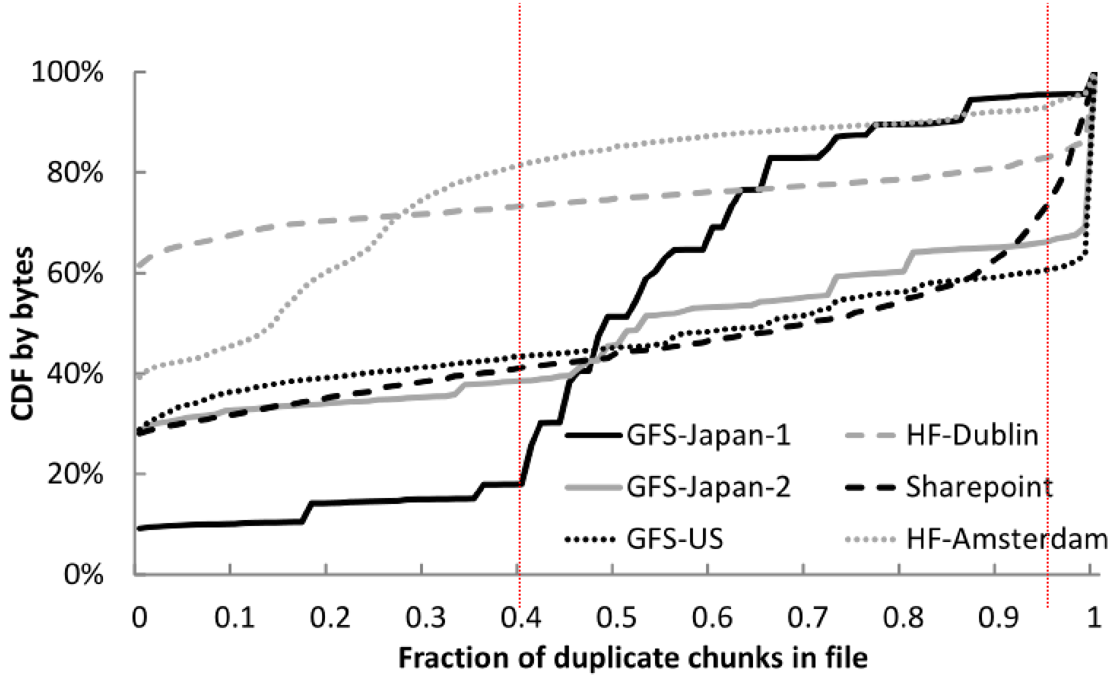
- **Used to get findings and motivations**
- HF(Documents, Photos, Music, etc.): created/modified/accessed by a single user.
- GFS: shared files in workgroups, created/modified by a single user, accessed by many users.
- Sharepoint: office documents in workgroups, created/modified/accessed by many users.
- SDS: created once by administrators, accessed by many users.
- VL: file shares containing virtualization image files, created/updated by administrators, accessed by many users.

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

Table 1: Datasets used for deduplication analysis. *Number of authors (users) assumed in 100s but not quantifiable due to delegated write access. †Number of (authors) users limited to < 10 server administrators.

Whole-file vs. Sub-file Dedup

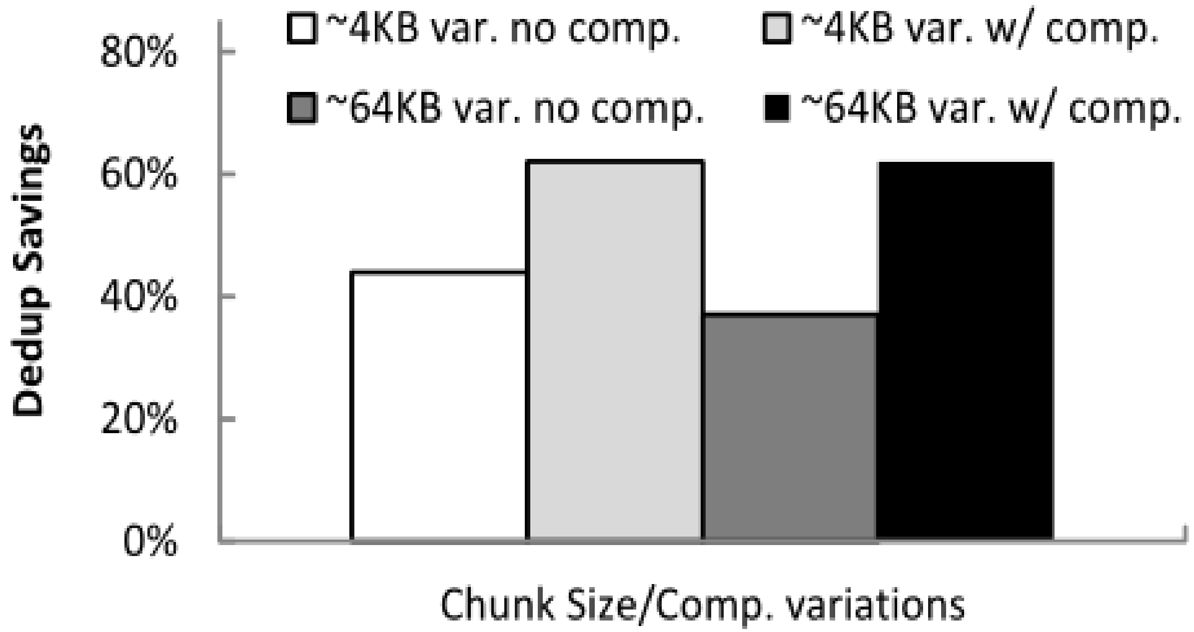
Dataset	Dedup Space Savings		
	File Level	Chunk Level	Gap
VL	0.0%	92.0%	∞
GFS-Japan-1	2.6%	41.1%	15.8x
GFS-Japan-2	13.7%	39.1%	2.9x
HF-Amsterdam	1.9%	15.2%	8x
HF-Dublin	6.7%	16.8%	2.5x
HF-Japan	4.0%	19.6%	4.9x
GFS-US	15.9%	36.7%	2.3x
Sharepoint	3.1%	43.8%	14.1x



- ➢ chunked using a Rabin fingerprint based variable sized chunker, hashed using a SHA1 hash function, compressed using gzip compression
- ➢ **Chunk-level dedup saves more space than file-level.**

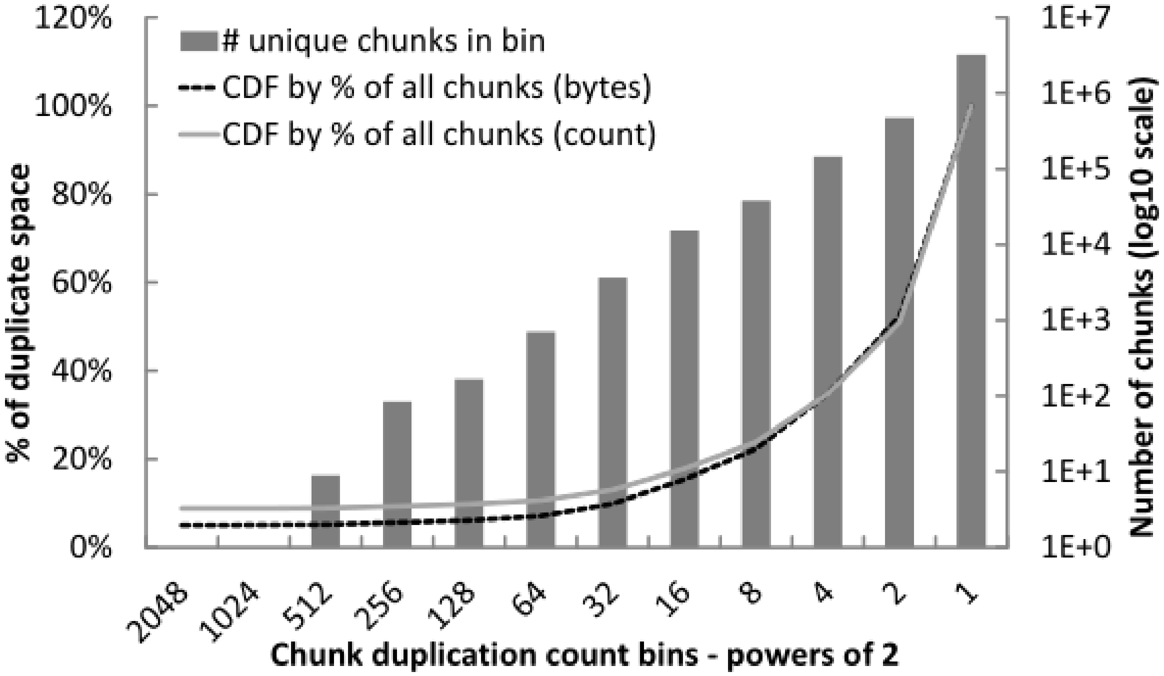
Average Chunk Size

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



GFS-US dataset analysis

Chunk Duplication Analysis



- GFS-Japan-1 dataset
- **X-axis** rightmost: frequency is 1; other bins: chunk frequency in the interval $[2^i, 2^{i+1})$ for i >= 1
- **Y-axis** left: CDF of duplicate space; right: chunk number

- **Nearly 50% chunks are unique.**
  - Strive to reduce the overhead for serving unique data
- **Majority of deduplicate bytes(count) reside in the middle portion of distribution.**
  - Not sufficient to dedup just high frequency chunks

Data Partitioning

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%

- Partition by directory sub-trees (each partition ≤ 10% of total namespace)
- Exclude VL/SDS/sharepoint
- **Dedup after partition by file type as good as global dedup.**

Dataset	Total Size	Per-Server Dedup Savings	Cross-Server Dedup Savings	Cross-Server Dedup Benefit
All Home-Folder Srvrs	2438GB	386GB	424GB	1.56%
All File-Share Srvrs	2897GB	1075GB	1136GB	2.11%
All Japan Srvrs	1436GB	502GB	527GB	1.74%
All US Srvrs	3844GB	1292GB	1354GB	1.61%

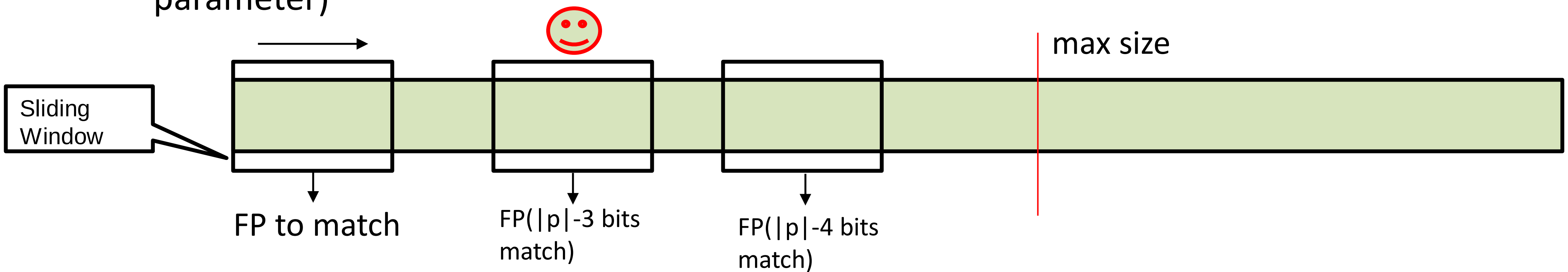
- Aggregate server datasets by location or workload type and dedup per-server/cross server
- **Cross-server dedup benefit is negligible**

Deduplication Technique Choose

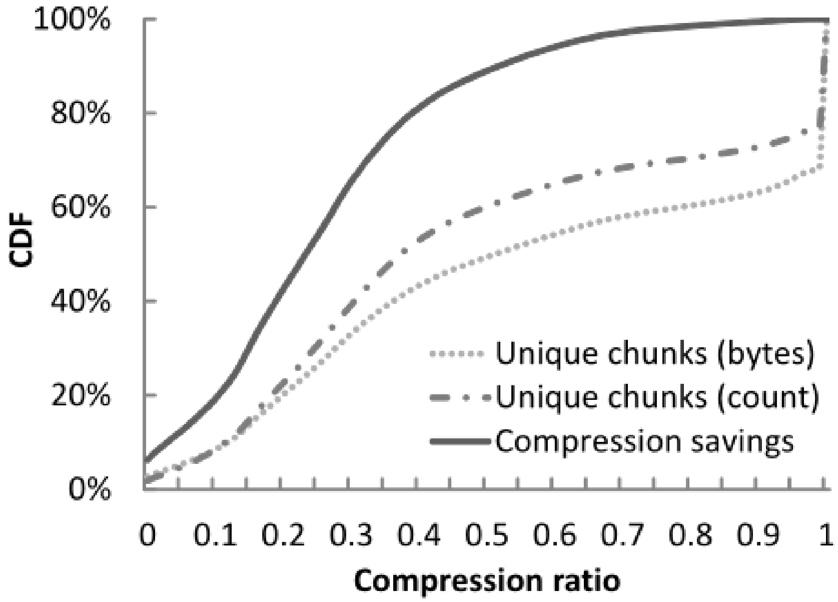
- **Inline deduplication**:
  - process the data synchronously on the write path before the data is written to disk
  - introduce additional write latencies and reduces write throughput.
- **Post-processing(offline) deduplication**:
  - process the data asynchronously after it has been written to disk
  - Dedup on older files may avoid additional latency for most accessed files (most files are not accessed after 24 hours from arrival)
  - flexible to choose idle time to dedup

Data Chunking

- **Basic chunking: Rabin fingerprint based sliding window & S_{min} S_{max} threshold**
  - Only using Rabin fingerprint to identify chunk boundaries causes too large/small chunk size.
  - A boundary making chunk size within S_{min} is simply ignored.
  - Forced boundary at max chunk size (S_{max})
  - Accumulation of peaks around S_{min} and S_{max}, content independent, reducing dedup saving
- **Regression Chunking Algorithm**
  - Goal: obtain uniform chunk size distribution reduce forced chunk boundaries at max size.
  - Basic idea: relax match condition to some suffix of bit pattern P (Rabin fingerprint based chunking)
  - Method: match |P| - i bits of P, with decreasing priority for i = 0, 1, …, k (k is an adjustable parameter)



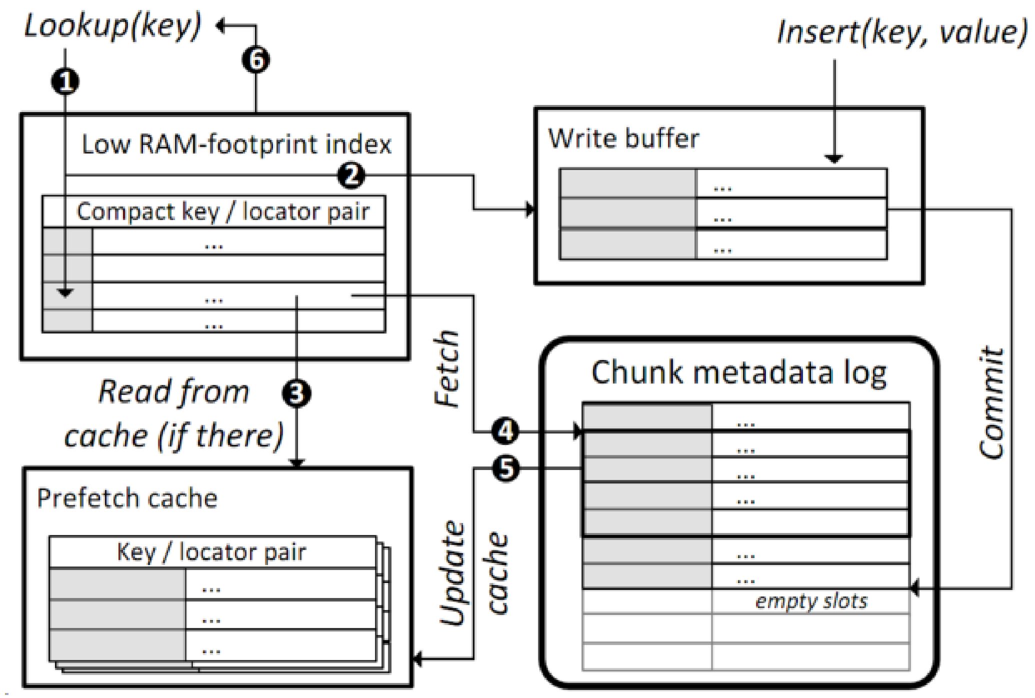
Chunk Compression



- **Compression savings is skewed**
  - 50% unique chunks responsible for 80% compression saving
  - 30% unique chunks do not compress at all
- **Solution: selective compression**
  - compress chunks only with low CR
  - store chunks with high CR primarily
  - improve compression saving while reduce decompression cost

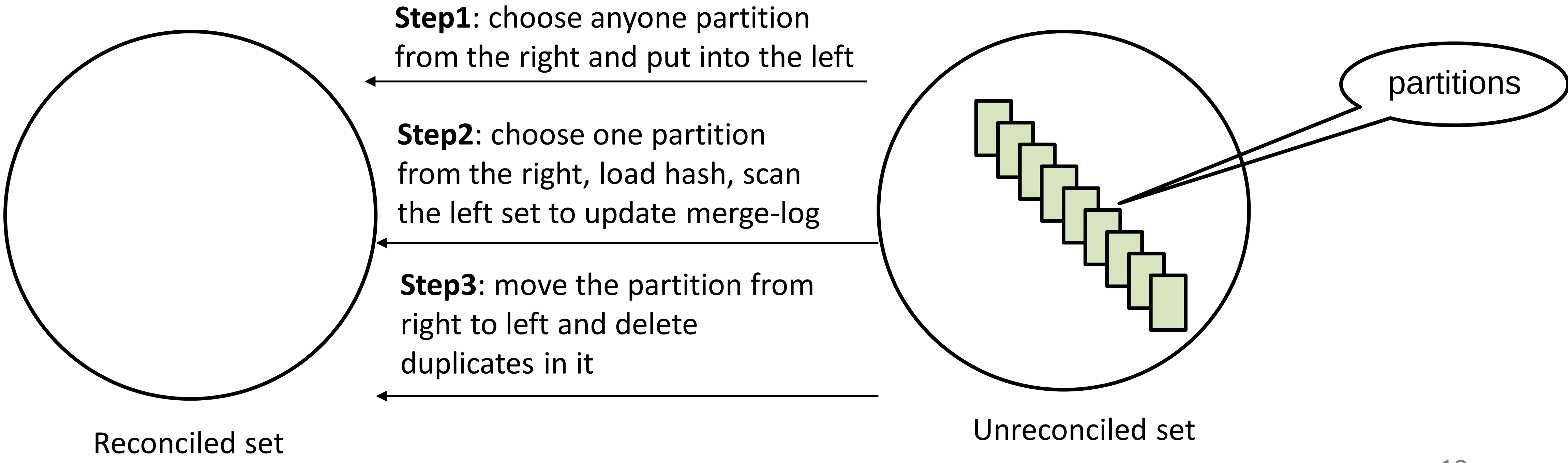
Chunking Index

- **Log-structured organization**
  - Chunk metadata organized in log-structure on disk
  - Insertions aggregated in write buffer in RAM and appended to log in single I/O
- **Low RAM footprint index**
  - A specialized in-memory hash table is used to index chunk metadata records on secondary storage
  - 2-byte signature, 4-byte pointer per entry => 6-byte of RAM per indexed chunk
- **Prefetch cache**
  - Prefetch chunk mappings for next 1000 chunks in a single I/O
  - Prefetch cache size at 100000 entries (5MB of RAM)
  - About 1% of index lookups hitting disk (on all datasets evaluated)



Data Partitioning and Reconciliation

- Partition by file type
- Dedup in each partition rather than the whole(reduce RAM usage again)
- Reconcile duplicates across partitions (perform deduplication across partitions again)
  - Maintain a merge log consist of two chunk ids that are a duplicate of each other



Performance Evaluation

- **RAM frugality**
  - Partitioning reduces RAM usage 3X; Optimized index reduces 9X.
- **Low CPU utilization**
  - lightly higher with optimized index and data partition
- **Low disk usage (not shown in the table)**
  - The median disk queue depth is zero in all cases.
  - At the 75-th percentile, the queue depth increases by 2 or 3 with optimized index and/or data partitioning due to disk-based index lookups and reconciliation's I/O.
- **Sustained deduplication throughput**
  - remain mostly sustained in the range of 26-30 *MB/sec*

	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
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%

- GFS-US dataset, partition by file type, three partitions (partition factor)
- Regular index: full hash(32 bytes) and location information(16 bytes) for each unique chunk stored in RAM
- Optimized index: this paper's index design
- Baseline: no partitions and regular index

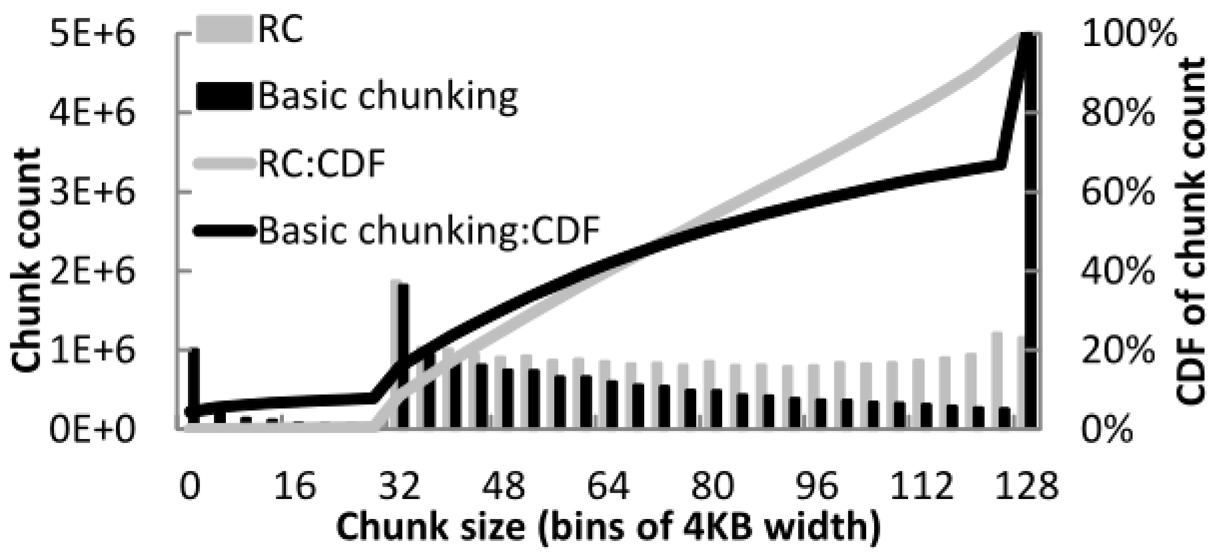
Consuming Time Percentage Evaluation

Deduplication Activity	Optimized index	Regular index w/ partitioning
Index lookup	10.7%	0.4%
Reconciliation	n/a	7.0%
Compression	15.1%	15.3%
SHA hashing	14.3%	14.6%
Chunking	9.7%	9.7%
Storing unique data	11.3%	11.5%
Reading existing data	12.6%	12.8%

- The second column: optimized index with no partition
- The third column: regular index with partition
- The paper's chunking algorithm is used and evaluated in the experiment

- Index lookups' time percentage in *optimized index* is second smallest.
- Reconciliation time percentage in *regular index with partition* is second smallest.
- New chunking algorithm's time-consuming nearly least.

Regression Chunking Evaluation



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%

- RC: regression chunking (new chunking algorithm)
- **Uniform chunk size distribution**

- **Dedup saving improvement with RC chunking.**

Conclusion

- Regression chunking improves deduplication save as to basic chunking.
- Optimized index reduces RAM usage meanwhile maintain CPU/Disk/throughput performance.
- Deduplication & reconciliation technique reduces RAM usage, maintains high dedup ratio even if ignoring cross-server deduplicates.

Rabin Fingerprint

给定一个n位消息$m_0,...,m_{n-1}$，我们将其视为在有限域GF(2)上的n-1次多项式。

$$f(x) = m_0 + m_1 x + \ldots + m_{n-1} x^{n-1}$$

然后，我们随机选择一个在GF(2)上的k次不可约多项式$p(x)$，我们将消息m的指纹定义为在GF（2）上$f(x)$除以$p(x)$的余数$r(x)$，它可以看作是一个k-1次多项式或k位数字。