Abdullah Gharaibeh*, Cornel Constantinescu, Maohua Lu, Ramani Routray, Anurag Sharma, Prasenjit Sarkar, David Pease and Matei Ripeanu*

IBM Research – Almaden, * University of British Columbia

IBM

DedupT: Deduplication for Tape Systems

Document number

Outline:

- The Problem
  - Background, Motivation &Challenges
- The Solution
  - Cross-tape Chunk Placement & Evaluation
  - On-tape Chunk Placement & Evaluation
- Summary
  - Main Results

Why dedup for tapes:

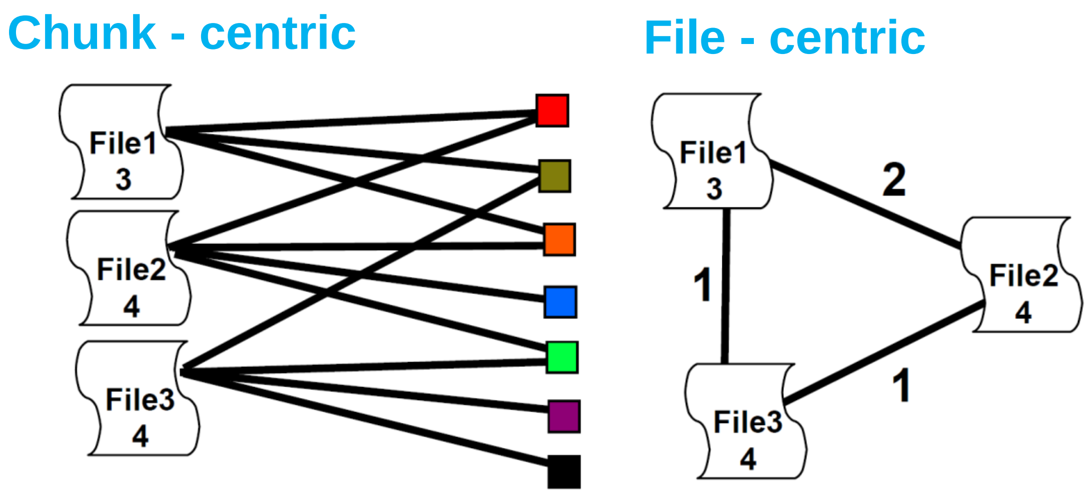
- Tapes will continue to play a large part in the storage landscape.
  - Great features: *longevity, reliability, power* and recently filesystem-like access.
- Storage tasks tapes are good at (archival, backup, database snapshots, virtual images) is where data is highly *deduplicable*.

Challenges for dedup on tapes:

- High *tape mount* overhead:
  - If the chunks of a file end up on more than one tape then the retrieval time significantly increases (due to the multiple tape mounts).
- High tape *seek time*:
  - chunks of a file that are placed out-of-order will increase retrieval time (an end-to-end seek takes ~90 seconds).

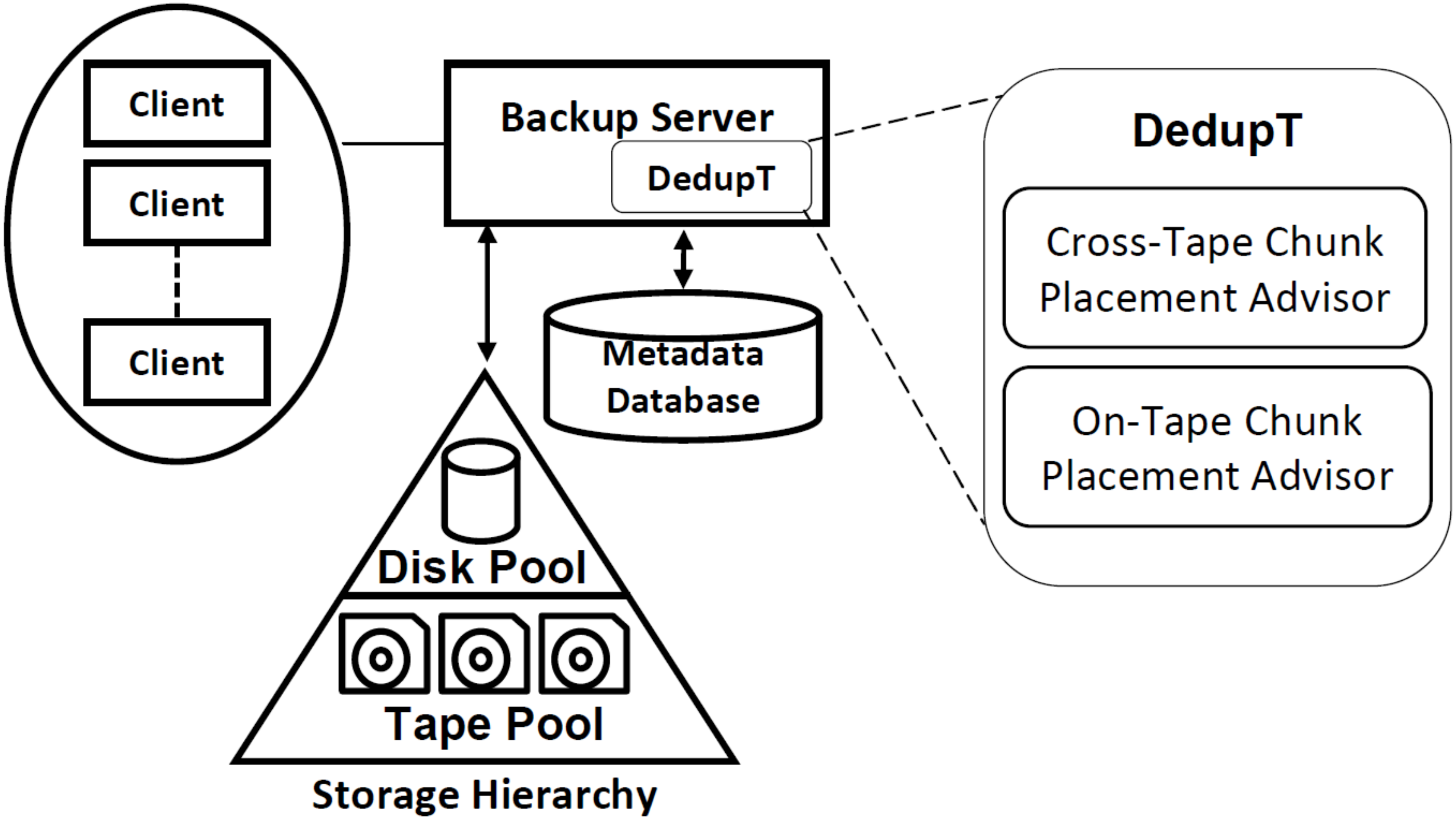
DedupT Solution:

- A *sparse*, *low memory* graph model for representing deduplicated data.
  - Exposes the degree of similarity (amount of content sharing) between objects (ex. files).
  - Enables efficient partitioning of a large set of deduplicated files, into tape size partitions.
  - Allows for fast computation of deduplicated partition sizes.



- A simple *on-tape chunk placement* algorithm – that reduces seek time overhead due to chunk fragmentation.

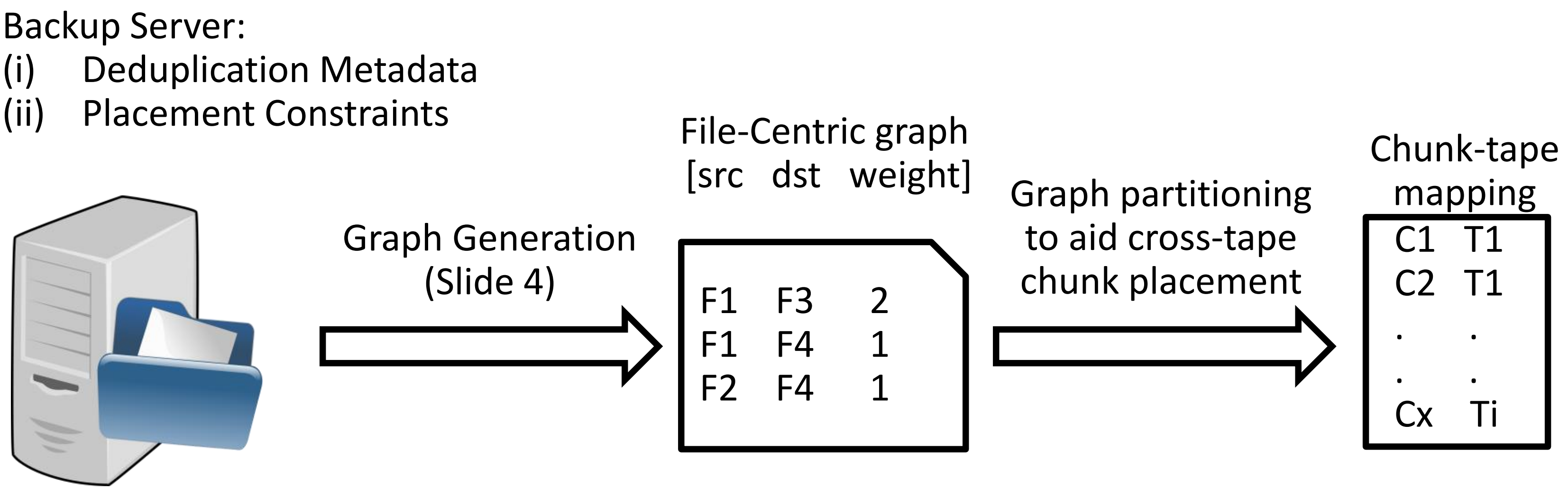
DedupT Placement in a typical Data Protection Solution



Outline:

- The Problem
  - Background & Motivation
- The Solution
  - Cross-tape Chunk Placement & Evaluation
  - On-tape Chunk Placement & Evaluation
- Summary
  - Main Results

Cross-Tape Chunk Placement



- All chunks of a file must be available on a single tape.
- Scalable (to many petabyte systems) and fast.
- Cater to specific placement policies (ex. all the files of a user placed together).

Building File-centric Graph from Deduplication Metadata

Metadata in database:
file's –chunk-maps.
(Hash(C1) = H1)

F1 H1
F1 H2
F2 H4
F2 H3
F3 H1
F3 H2
F4 H3
F4 H1

Graph generation process from deduplication metadata

1. Group by Hash

F1 H1
F3 H1
F4 H1
F1 H2
F3 H2
F2 H3
F4 H3
F2 H4

e.g. generate F1-F3 edge

2. Create edges

F1 F3 1
F1 F4 1
F1 F3 1
F2 F4 1

3. Group by edge

F1 F3 1
F1 F3 1
F1 F4 1
F2 F4 1

4. Reduce edge weights

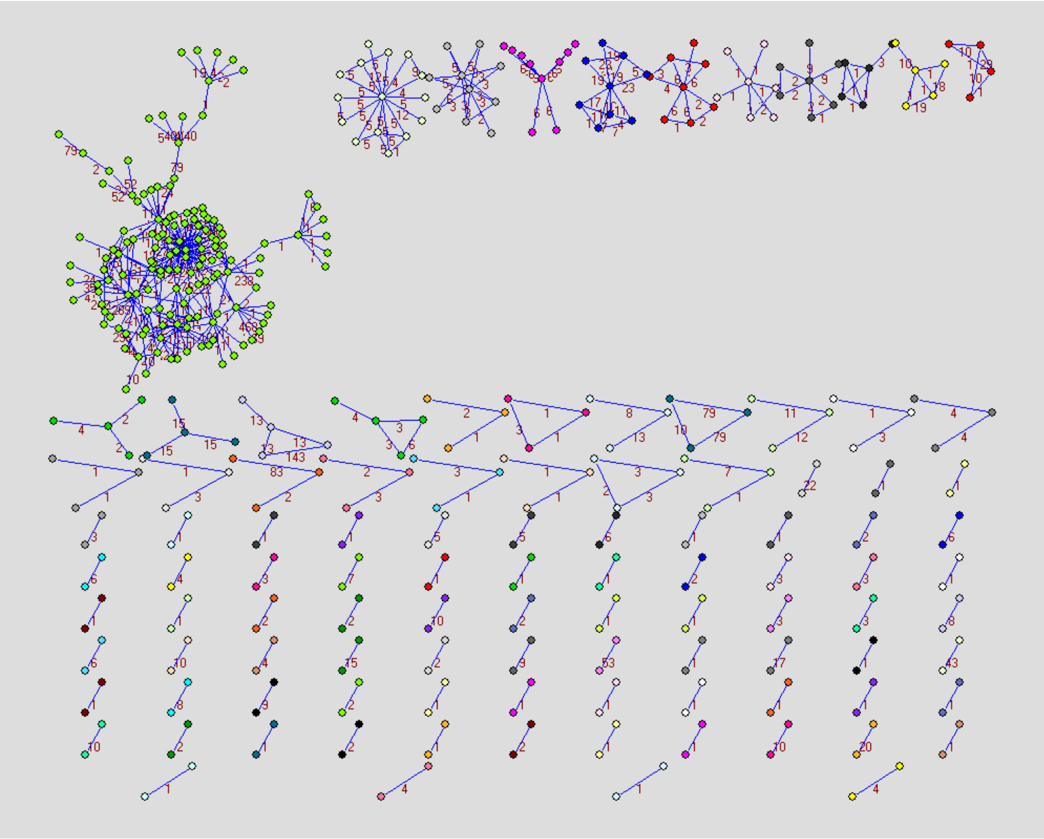
Graph Edge List
[src dst weight]

F1	F3	2
F1	F4	1
F2	F4	1

Partitioning the Graph into Tape-size Partitions

File-centric graphs usually show many isolated clusters with one or few large.

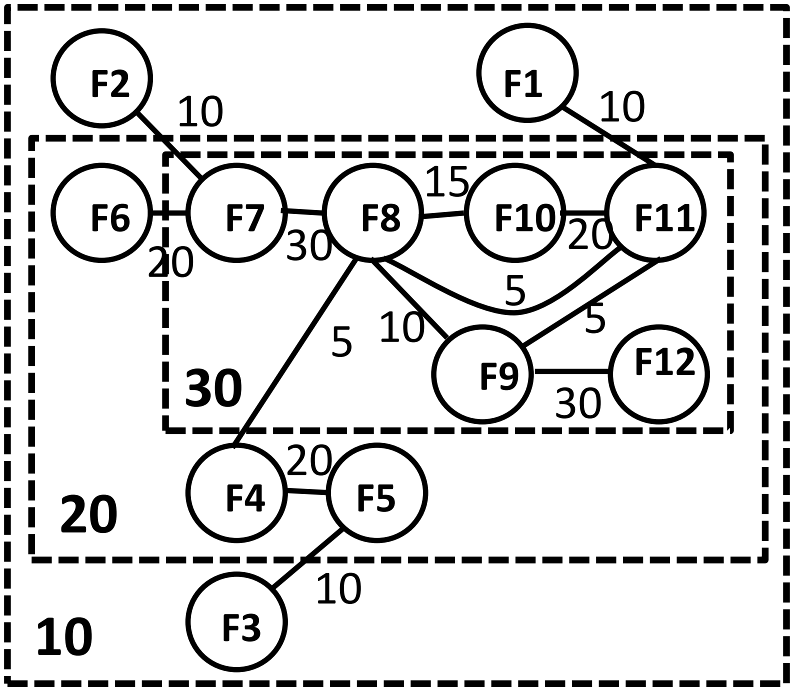
Next is a file-centric graph representation of a folder with 432 files (that share at least one chunk):



- ❑ All chunks of a file must be available on a single tape.
- ❑ Reduce the number of chunks replicated across the resulting partitions (edge cuts).

Partitioning the Graph into Tape-size Partitions – cont.

- Phase1: We identify the clusters (connected components) of the graph.
  - Separate components do not share data so they can be placed on different tapes (as needed) without “cutting” any edges (chunks replicated).
- Phase2: Partition the large-sized components using the k-core decomposition of them.



- The core (k+1) is always a subgraph of core k. The coreness of a vertex is the maximum core it belongs to.
- In a bottom-up strategy, the first partition includes files with coreness between [0, x), where x is chosen such that the partition size fills the tape; the second partition is between [x, y) and so on… .

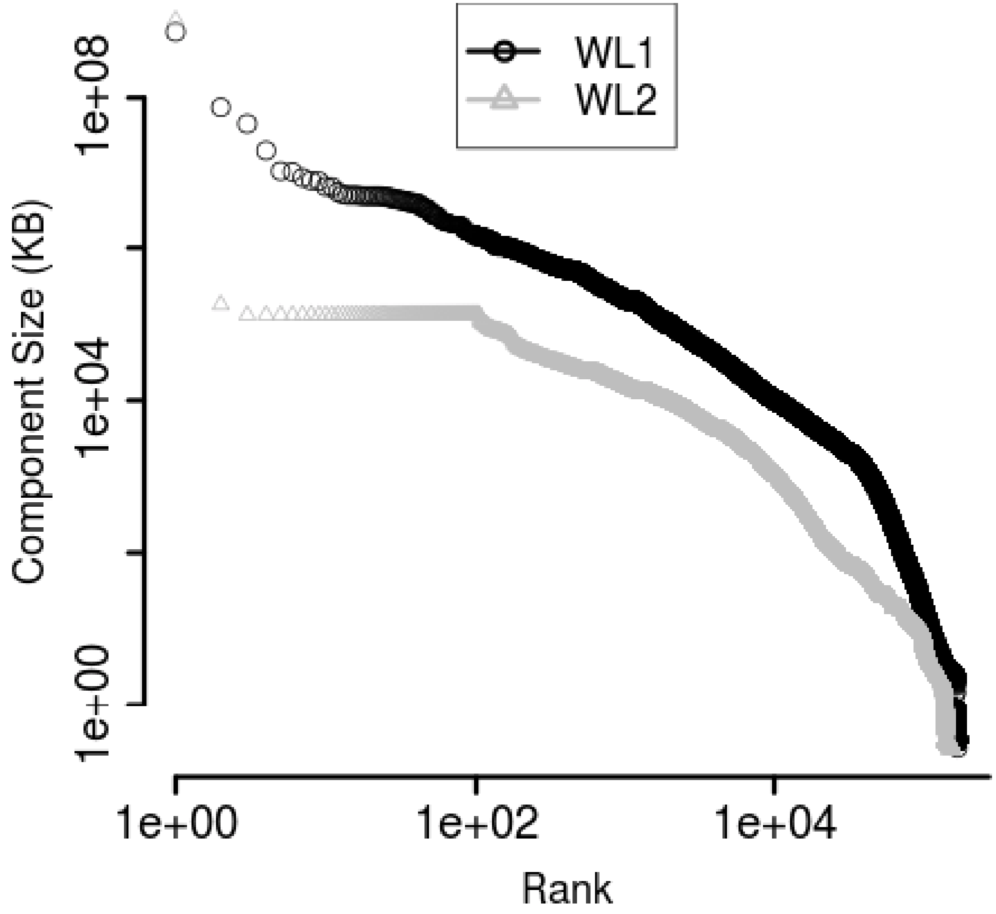
Workload summary (Table 1):

Workload	WL1	WL2
Total Size	3,052 GB	1,532GB
Duplicates Size	978GB	460GB
Duplicates (%)	32%	30%
Num. of Files	289,295	201,406
Avg. File Size	10MB	7.79MB
Median File Size	82KB	18KB
Num. of Chunks	17,509,025	12,021,126
Avg. Chunk Size	182KB	102KB
Median Chunk Size	71KB	52KB

Graph characteristics (Table 2):

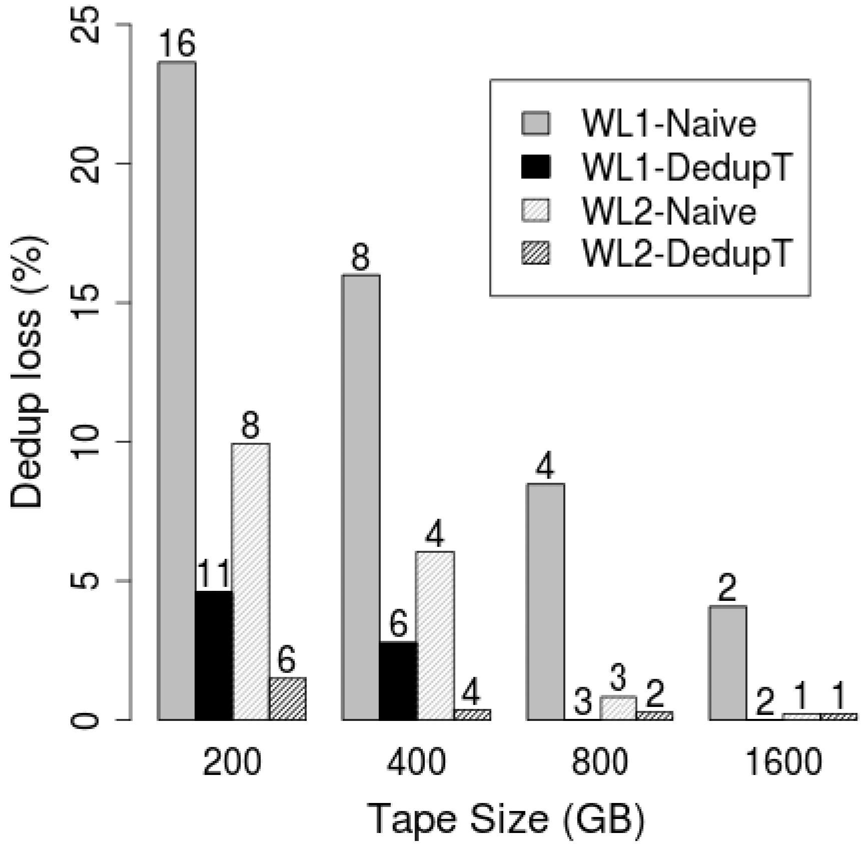
Workload	WL1	WL2
Time to Generate	5.6 min	3.8 min
Number of Vertices	289,295	201,406
Number of Edges	327,472	246,244
Graph Density	8.00E-06	1.20E-05
Number of Components	166,089	149,083
Size of the Largest Comp.	695 GB	987 GB

Component (cluster) sizes distribution:



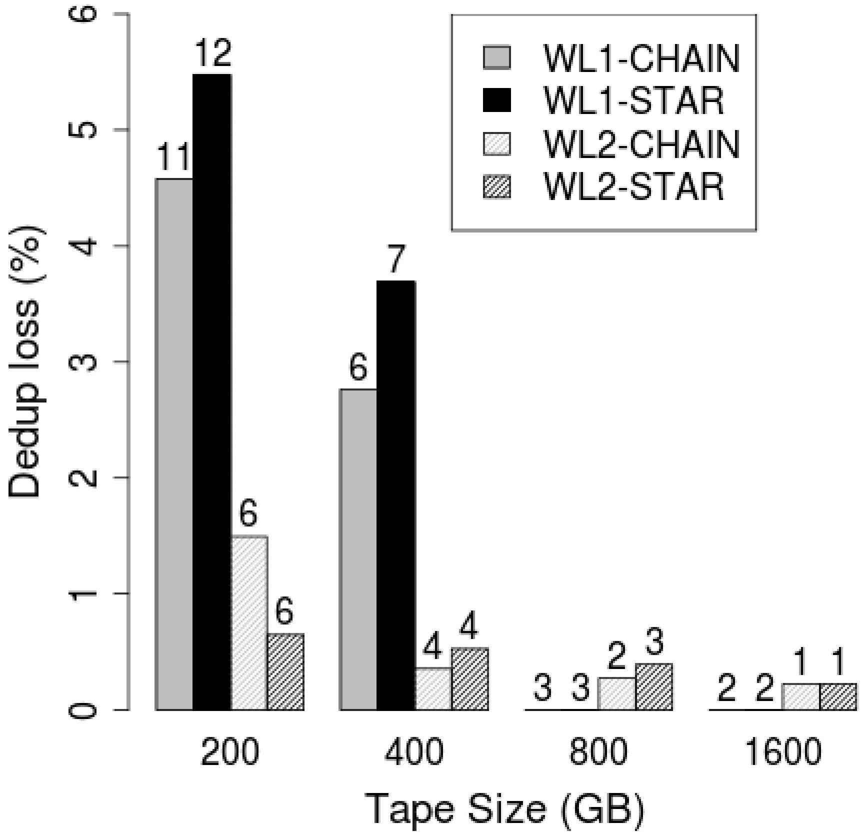
Evaluation – cont.

Comparing DedupT with Naïve Placement:



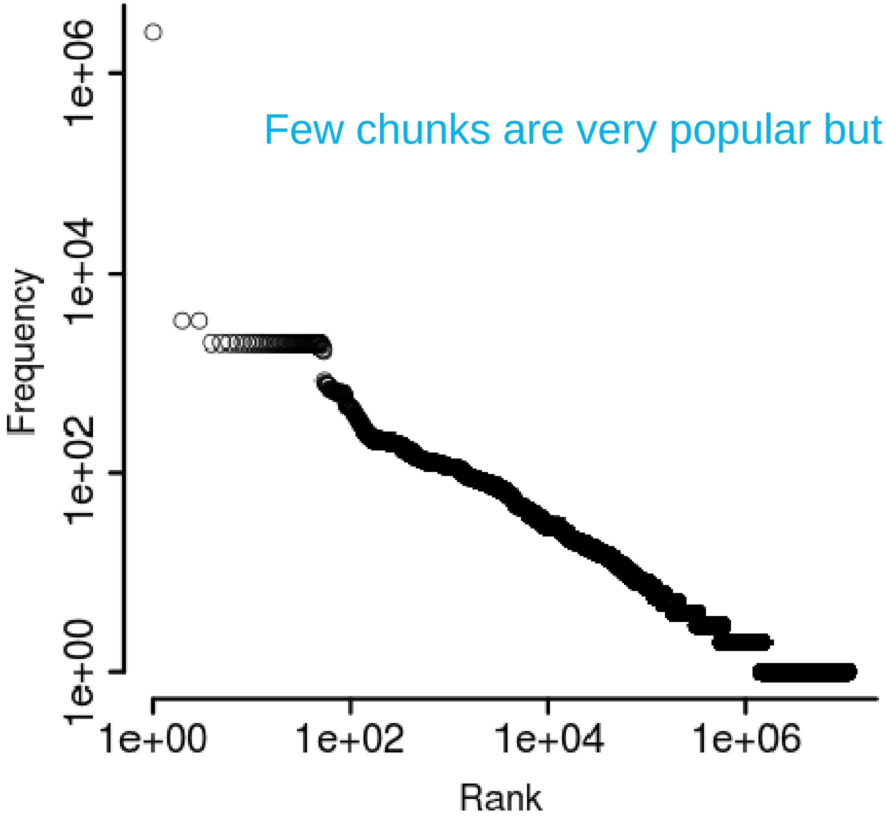
Dedup loss = amount of replicated chunks due to partitioning / duplicates size

Comparing two linking strategies: CHAIN and STAR:



Chunk Popularity Distribution

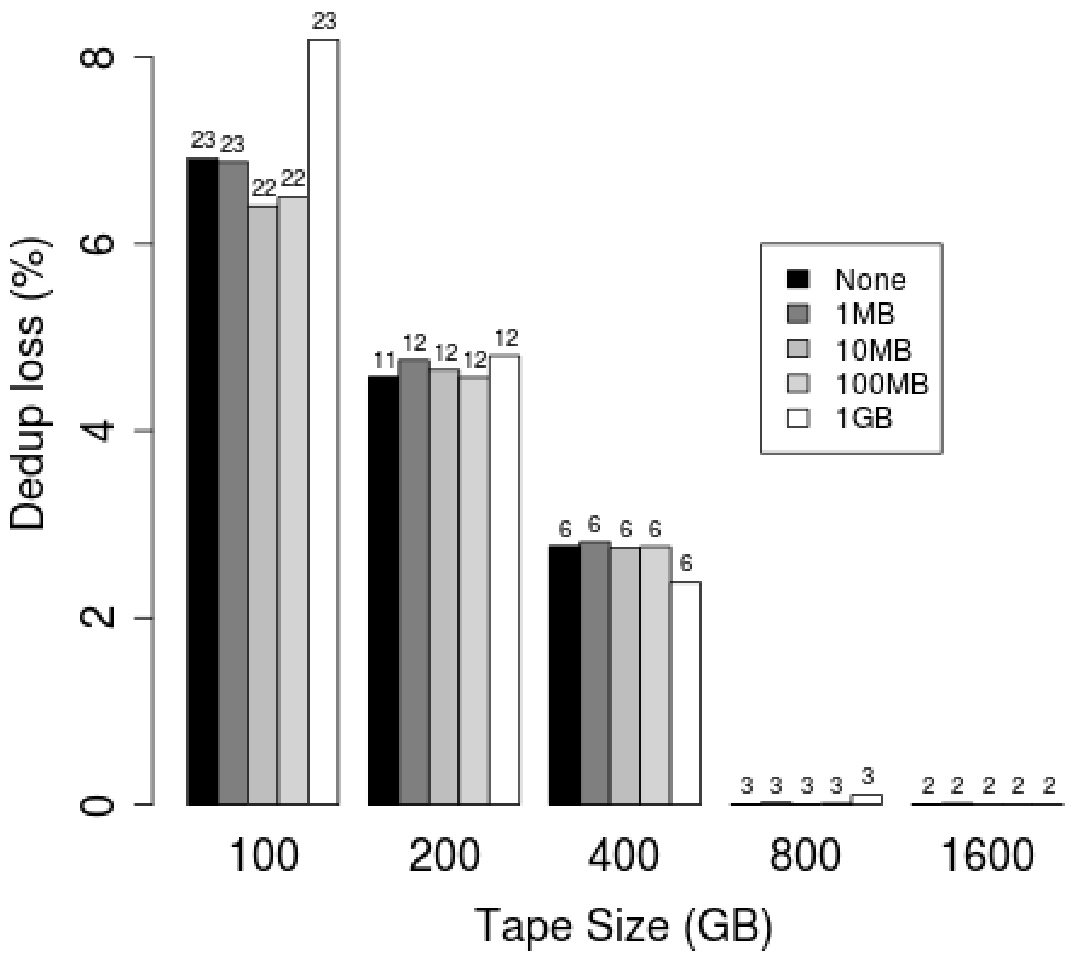
Few chunks are very popular but most show-up only few times.



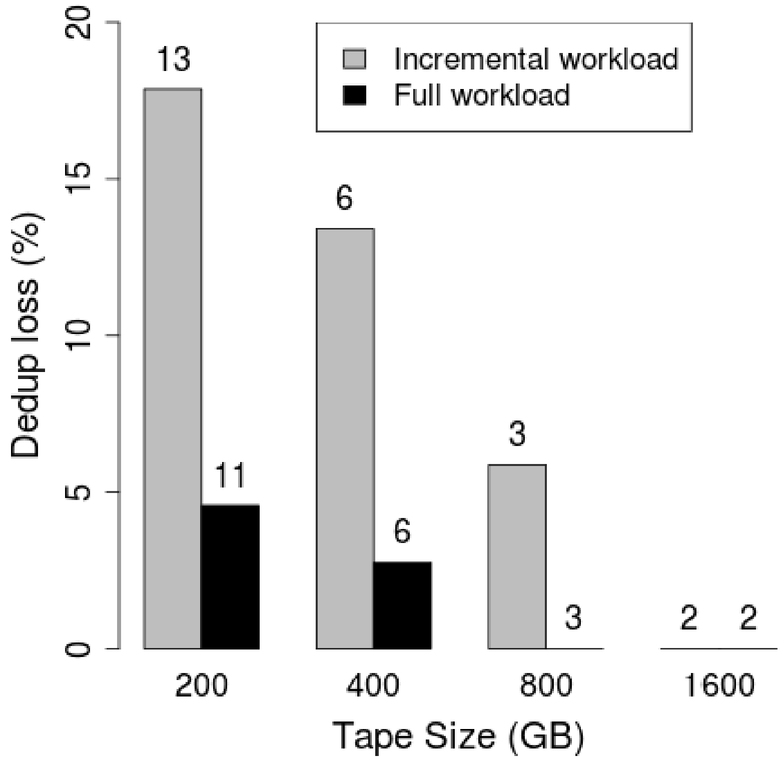
Log-log plot of chunk frequency distribution for WL1

Evaluation – cont.

Results of Replicating the Popular Chunks on All Tapes:



Adding Data Incrementally (WL1 workload):



- WL1 was divided into 10 batches (representing 10 periods of time from the files metadata) and pushed one after the other.
- All the files already placed on a tape are aggregated into a single vertex in the new graph model for the next batch insertion (vertex weight = sum of deduplicated file weights, edges also aggregated).

Outline:

- The Problem
  - Background & Motivation
- The Solution
  - Cross-tape Chunk Placement & Evaluation
  - On-tape Chunk Placement & Evaluation
- Summary
  - Main Results

On Tape Chunk Placement

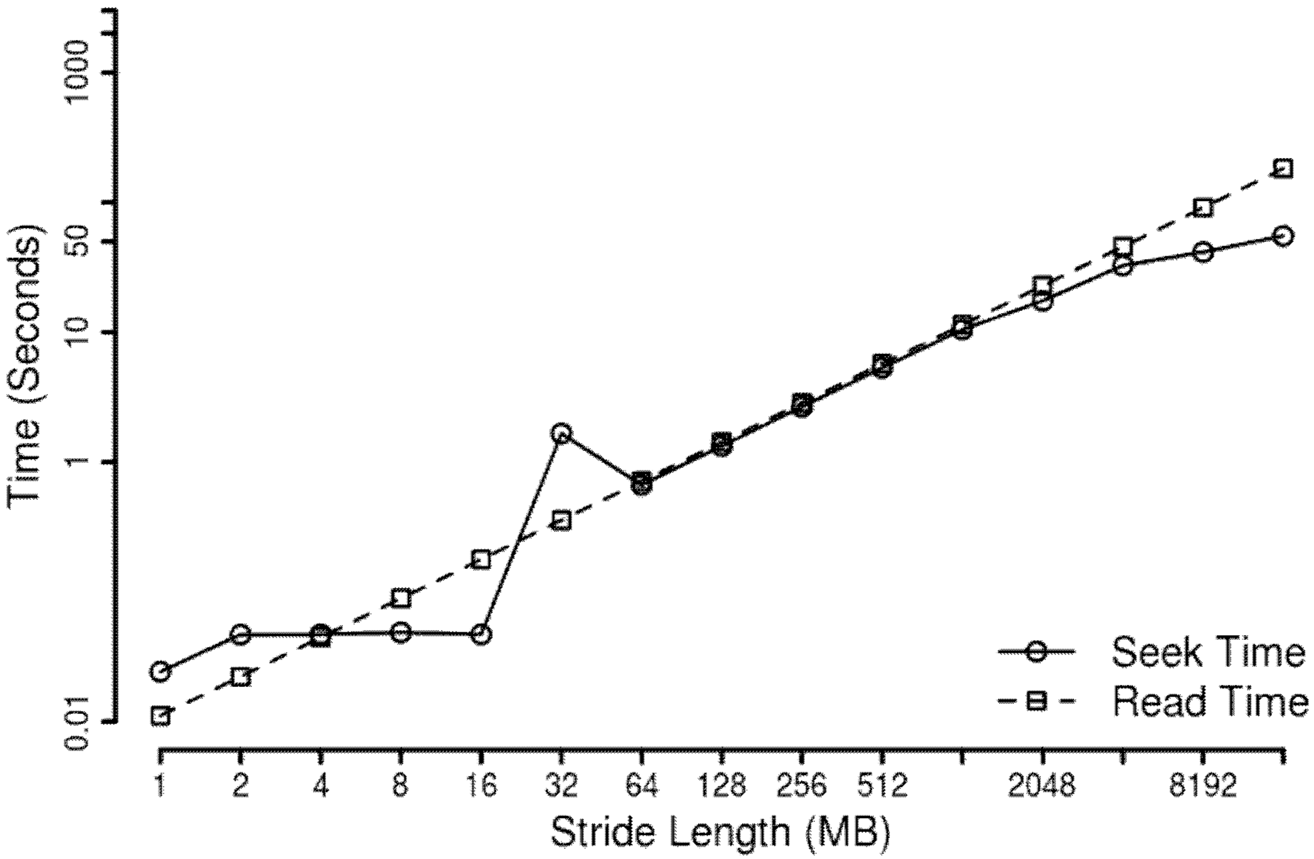
Motivation:

Tape high *seek* time and combined with data *fragmentation* due to deduplication can lead to high restore times if chunks are not “carefully” placed on tape.

Restore scenarios:

- **Restore entire tape(s)** – the traditional way of using tapes
  - On-tape placement is straightforward – chunks can be read sequentially at high speed in any order, and buffered in a disk-based scratch storage for the tape.
  - Our cross-tape placement solution ensures that all chunks needed to reconstruct the files are on one tape.
- **Restore a subset of files** – gaining traction
  - Our Simple Placement algorithm turns out to work pretty well in practice.

Tape Performance Characterization



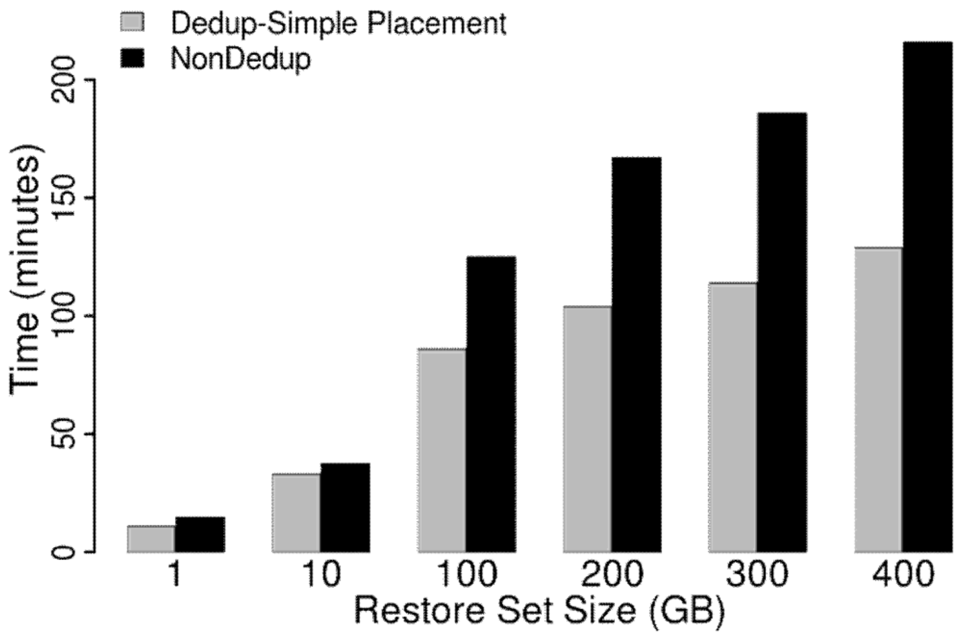
On Tape Chunk Placement – cont.

Simple Placement algorithm (for restoring subsets of files):

- Place the files (their chunks) on the tape in increasing order of file sizes. It uses the file to chunk map for each tape (from the cross-tape placement).

- For restoring (a subset of files), a *read plan* is created that reads all necessary chunks in the order of increasing tape offset (so all seeks are in the direction of data layout).
  - The read plan also uses the reed vs. seek threshold of 4 MB (as shown on previous slide).

Restore Performance while varying Restore Set Size



- The files in a Restore Set were picked randomly and same sets were used for both methods.
- Although there were more seeks for Dedup than for NonDedup, the seeks for Dedup were shorter.

- As Restore Set Size increases the difference is mostly due to less data (due to deduplication).
  (For 1GB restore set size deduplication was only 2.5% while for 300GB it was 30%.)

Summary:

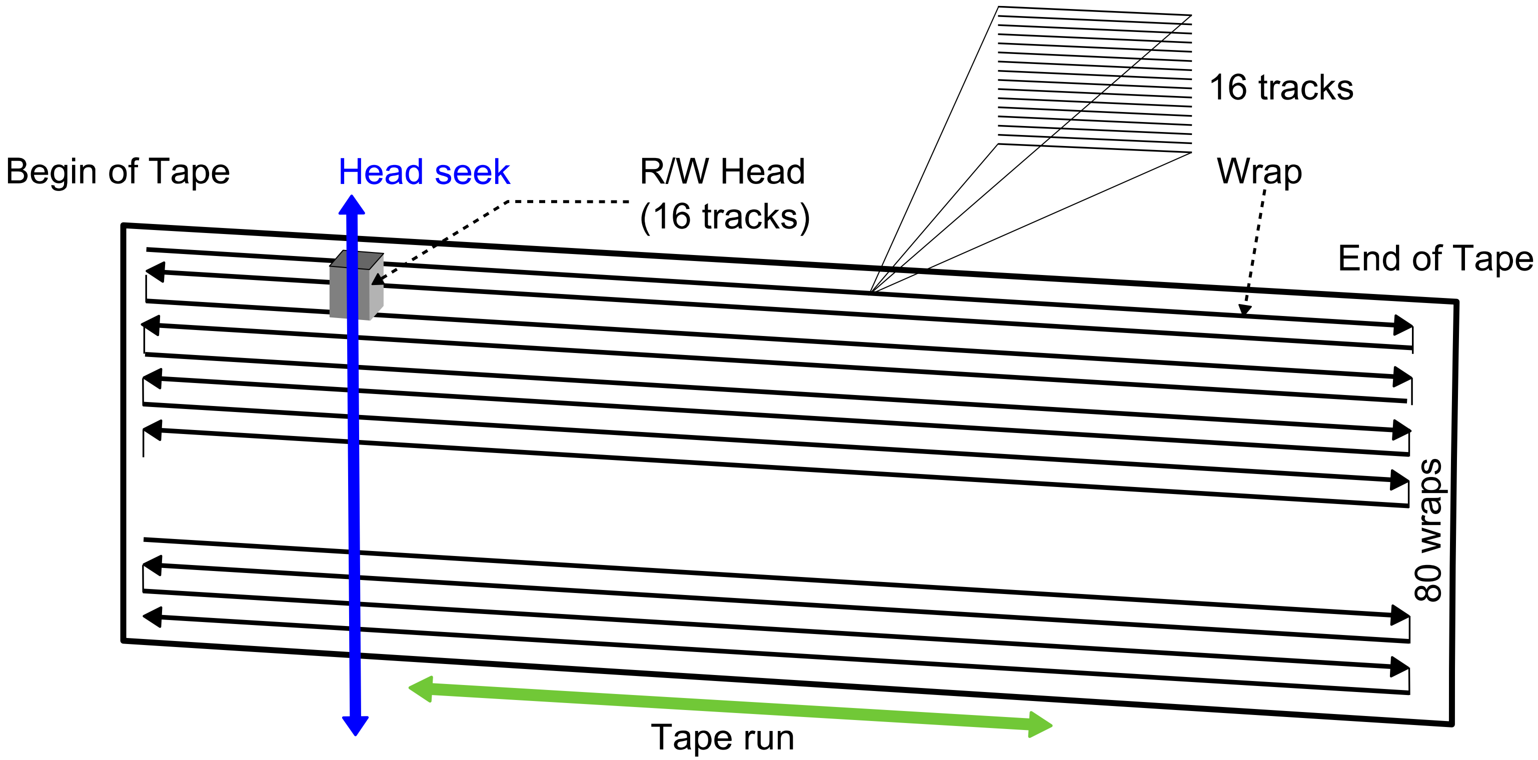
- This is the first work to demonstrate that tape based systems can fully benefit from the gains offered by deduplication without major penalties in terms of data retrieval.

- We addressed the main challenges for efficient data dedup on tapes:
  - High tape mount overhead
  - Seek time

- Our *chunk placement algorithms* are able to preserve up to 95% of dedup efficiency while:
  - completely eliminating the above major recovery time overheads.
  - improving performance of migrating data to tape pools (proportional with dedup efficiency).
  - reducing tape wear
  - offering restore performance 30% – 40% better than that of non-deduplicated tape.

Backups

Data layout on LTO-5 Tape



Zooming into the largest component:

Partition by file popularity: yellow (min degree=1), green(2), red(3) and blue(4)