Building a High-performance Deduplication System

Fanglu Guo Petros Efstathopoulos

Symantec Research Labs

Symantec Corporation, Culver City, CA, USA

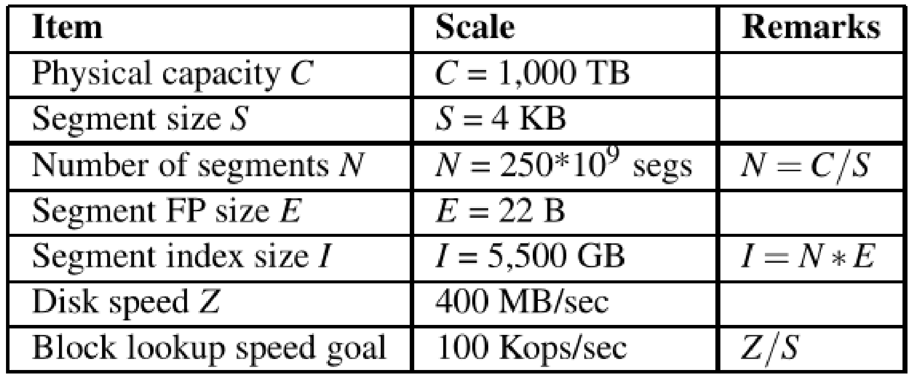
ATC 2011

Background

- Deduplication system aim to solve problems:
  - **Deduplication efficiency**: how well the system can detect and share duplicate data units
  - **Scalability**: the ability to support large amounts of raw storage with consistent performance.
  - **Throughput**: the rate at which data can be transferred in and out of the system
  - **Reference management**: who use what, when to reclaim unused segment.

Challenges and motivations

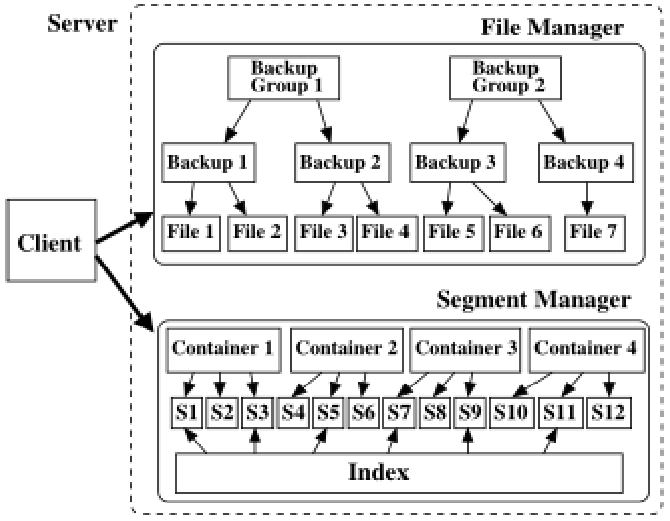
- **Indexing challenge**
  - Not support scalability/ dedupe efficiency together
- **Reference management**
  - Sweep-and-mark (aka Garbage Collection)
  - Low performance and time-consuming
- **Client-server interaction**
  - Reading files from disk is I/O-bound
  - Calculating cryptographic is CPU-intensive
  - Bottleneck: high latency and low communication throughput



Item	**Scale**	**Remarks**
Physical capacity C	C = 1,000 TB	
Segment size S	S = 4 KB	
Number of segments N	$N = 250*10^9$ segs	$N = C/S$
Segment FP size E	E = 22 B	
Segment index size I	I = 5,500 GB	$I = N * E$
Disk speed Z	400 MB/sec	
Block lookup speed goal	100 Kops/sec	Z/S

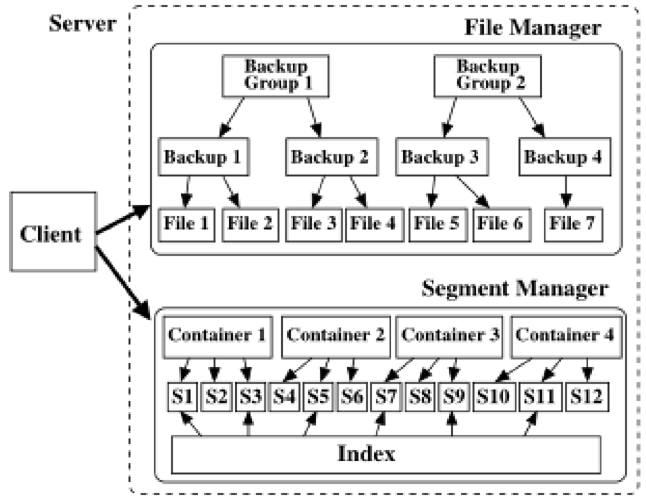
Note: SSD cannot achieve 60 Kops/sec

Overall architecture



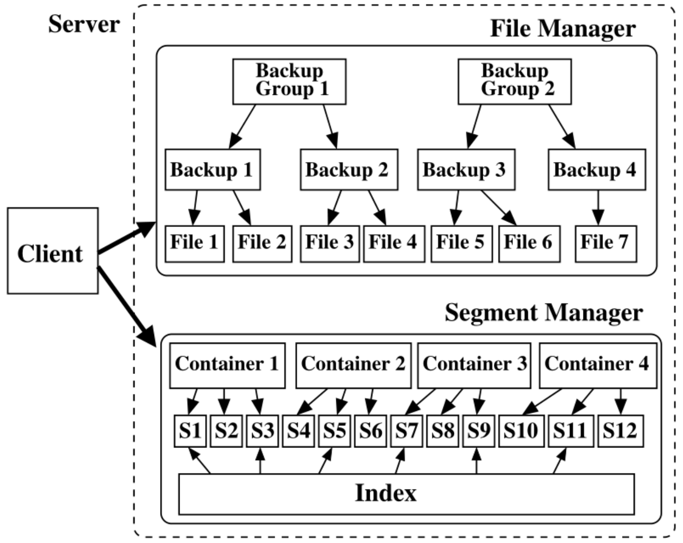
- ➢File manager
  - a three level hierarchy: file/backup/backup group
  - File is represented by a list 'A 1 B 2 C 3' (A: segment FP, 1: location)
  - Backup-represents a list of files
  - Organize backups into backup group for coarse-granularity tracking
- ➢Segment Manager
  - Storage and access unit: container, consist of raw data and catalog which lists all FPs stored in the container.
  - Dedupe index, update when segments are added/removed
- ➢Client
  - Read file context/perform segmentation/calculation FP
  - Initiate data transfers for new FPs

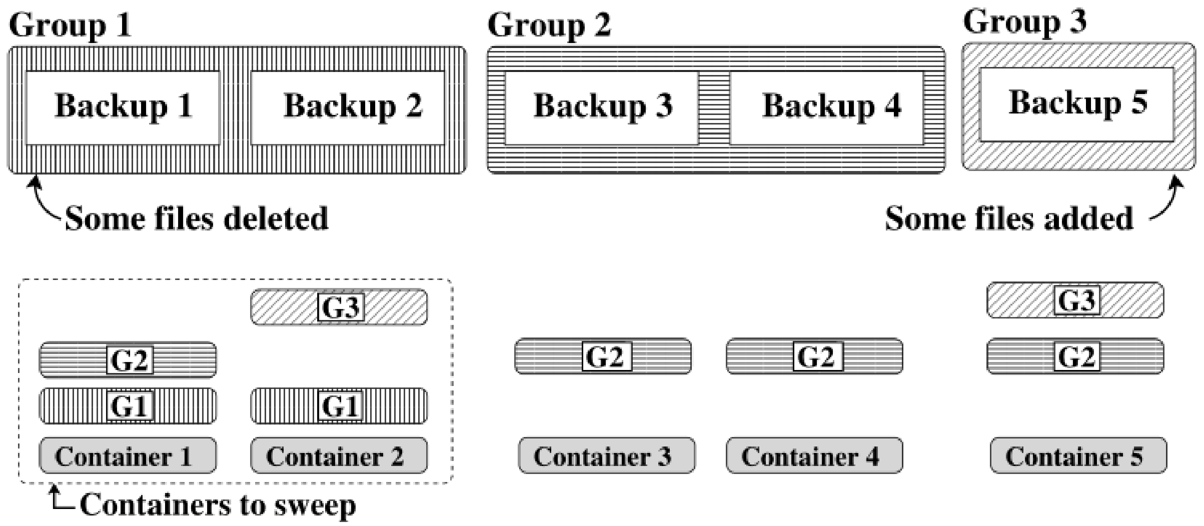
Progressive Sampled Indexing



- ➢Sampling index
  - Observation: RAM is not big enough to store total index.
  - A sampling period T: maintaining ‘1 out T’ FP in memory, others in disk.
  - How to calculate T:
    1/T = *number of index entry (FP) in RAM /number of segment in disk*
- ➢Pre-fetching and caching
  - A sampled FP is hit, then pre-fetch all FPs from that container’s catalog into a memory cache.
- ➢Progressive sampling
  - As number of segment in disk grows, T increases;
  - Making T smaller decrease pre-fetch operation.
  - Low-bound: each container has at least one FP in RAM index

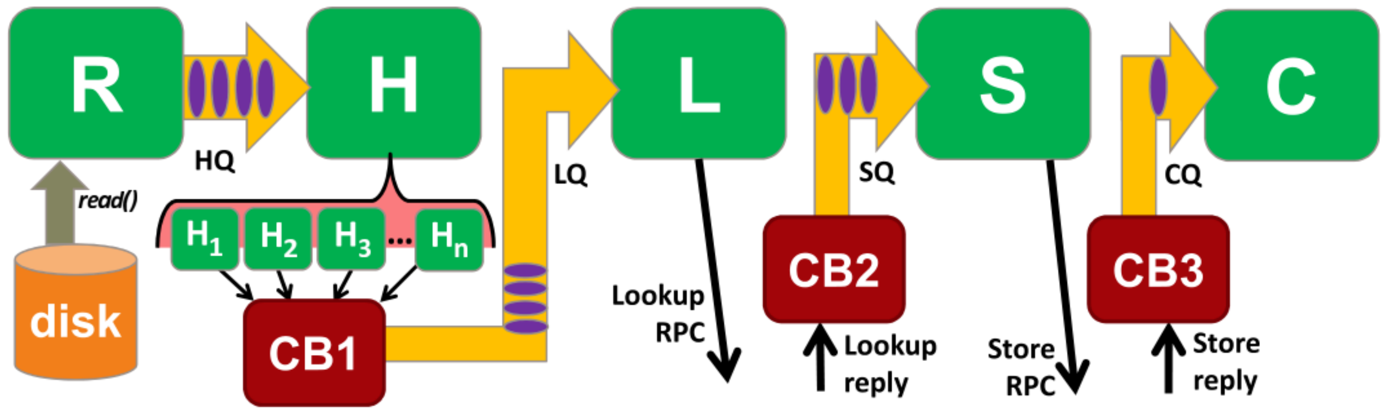
Grouped Mark-and-Sweep





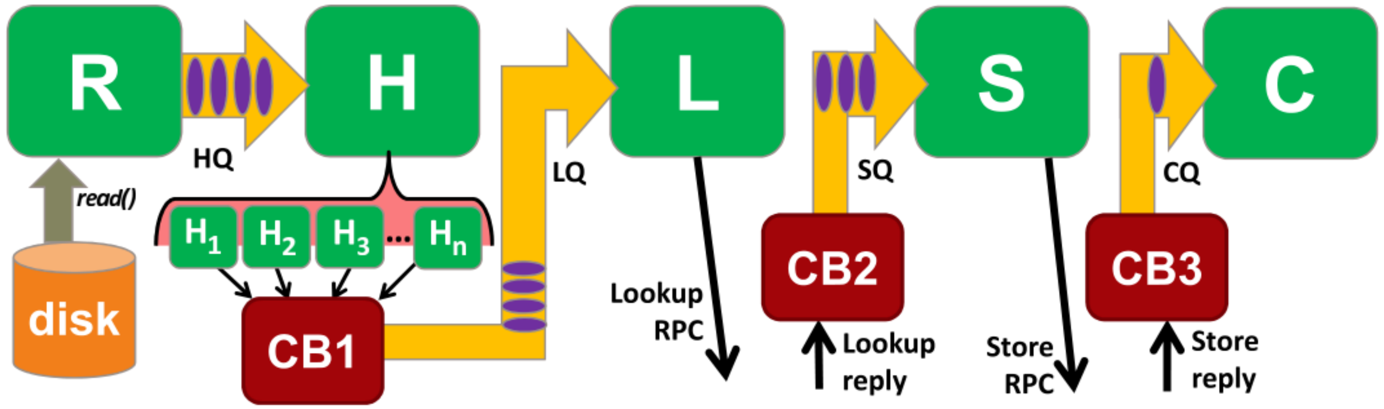
- ➢Step1: mark changed groups
- ➢Step2: Add affected containers to the sweep list.
- ➢Step3: sweep and reclaim freed segments

Client-Server Interaction



1. **Reader thread R** reads data and enqueues requests to **hash queue HQ**
2. **Hashing thread H** dequeue requests, perform segmentation and calculate FPs (multi-thread*(H_1,H_2, . . . ,H_n)*).
3. **Callback function CB1** enqueues the updated request to the **lookup queue LQ**.

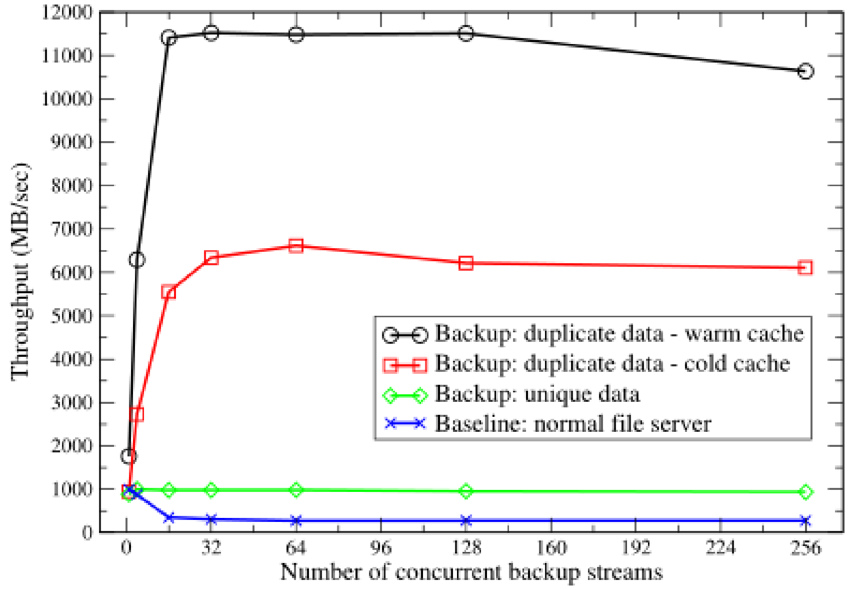
Client-Server Interaction



4. **Lookup thread L** receives requests from LQ and issues one single, batched, asynchronous lookup RPC to the server
5. **Callback function CB2** delivers the RPC reply and creates references to the containers of the FPs that were found on the server. If one or more FPs were not found, CB2 enqueues the updated request in the **store queue SQ**.
6. **Store thread S** sends raw data blocks to the back-end through one single, batched, asynchronous RPC.
7. **Callback function CB3** ensures that the write operation was successful, and forwards the last request for each file to the **close queue CQ**.

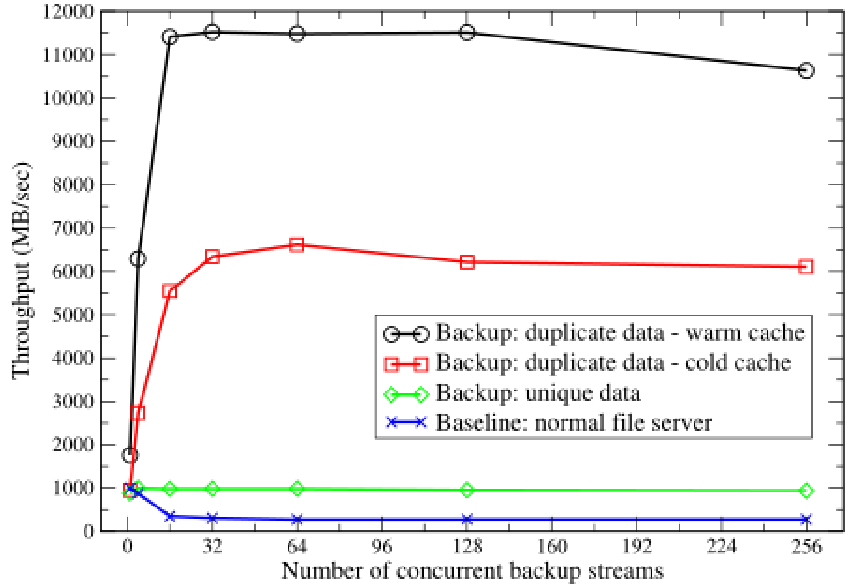
Backup throughput Evaluation

- **Baseline**: backing up unique data using normal file system, disk contention increases with the number of concurrent backups.
- **Unique data**: backing up unique data using this prototype, our prototype performs segmentation on all incoming data (regardless of content source), no disk contention

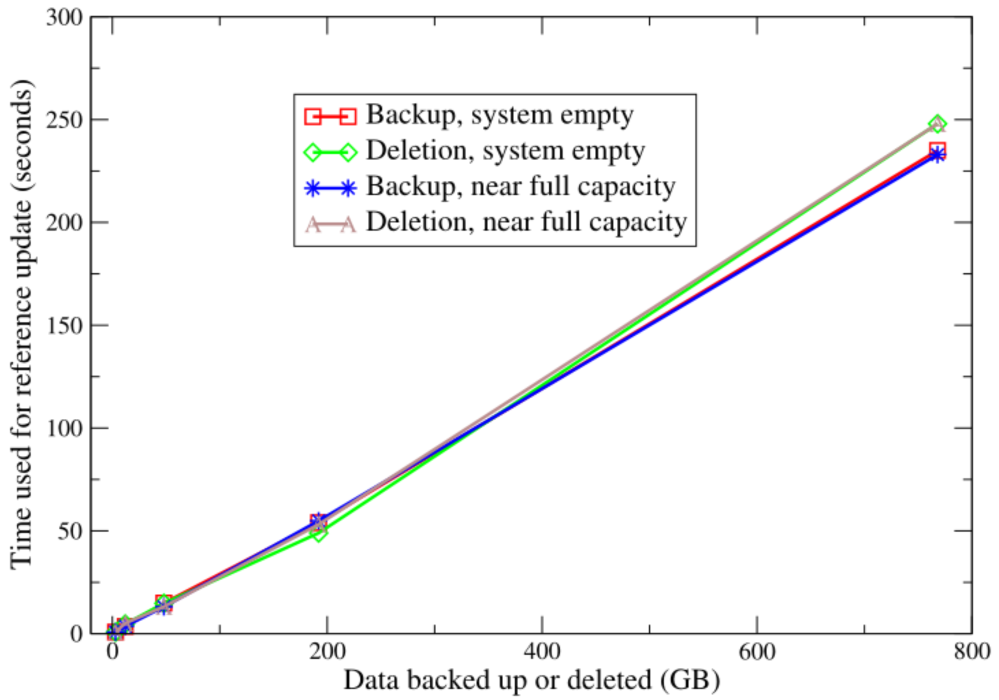


Backup throughput Evaluation

- **Duplicate data - cold cache**: after upper operation, backup the same dataset again, increase as concurrent number, decrease as pre-fetching is saturated and concurrency overhead.
- **Duplicate data-warm cache**: backup same data third time, overall higher due to many FPs in cache (no need pre-fetching).

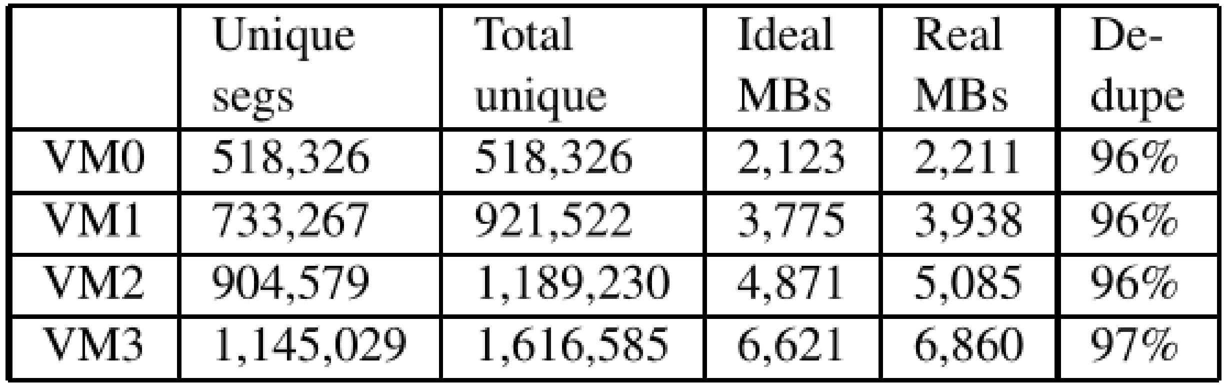


Reference Update Throughput



- ➢ Deletion is slightly slower because when segments get deleted, they also need to be removed from the index.
- ➢ **Group sweep-and-mark** operation is proportional to change.

Deduplication Efficiency

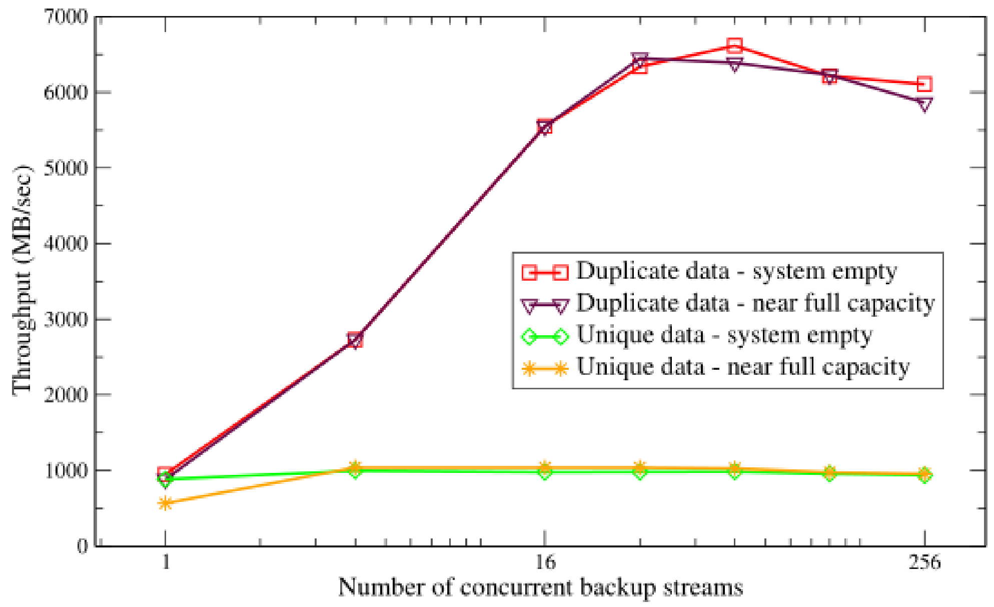


	Unique segs	Total unique	Ideal MBs	Real MBs	De-dupe
VM0	518,326	518,326	2,123	2,211	96%
VM1	733,267	921,522	3,775	3,938	96%
VM2	904,579	1,189,230	4,871	5,085	96%
VM3	1,145,029	1,616,585	6,621	6,860	97%

- VM0: a VMware “gold” disk image
- VM1: VM0 + all Microsoft updates and service packs
- VM2: VM1 + a large anti-virus suite installed
- VM3: VM2 + the installation of various utilities

- More than 96% dedupe efficiency (not 100%)

Scalability



Throughput scalability tests show that there is no significant throughput drop when we get close to full capacity.

Conclusion

- **Deduplication efficiency:** little sacrifice
- **Scalability:** nearly consistent performance close to full capacity
- **Throughput:** much higher than normal file server
- **Reference management:** group sweep-and-mark operation is proportional to file change, instead of total capacity.