symantec™

Confidence in a connected world.

Rethinking Deduplication Scalability

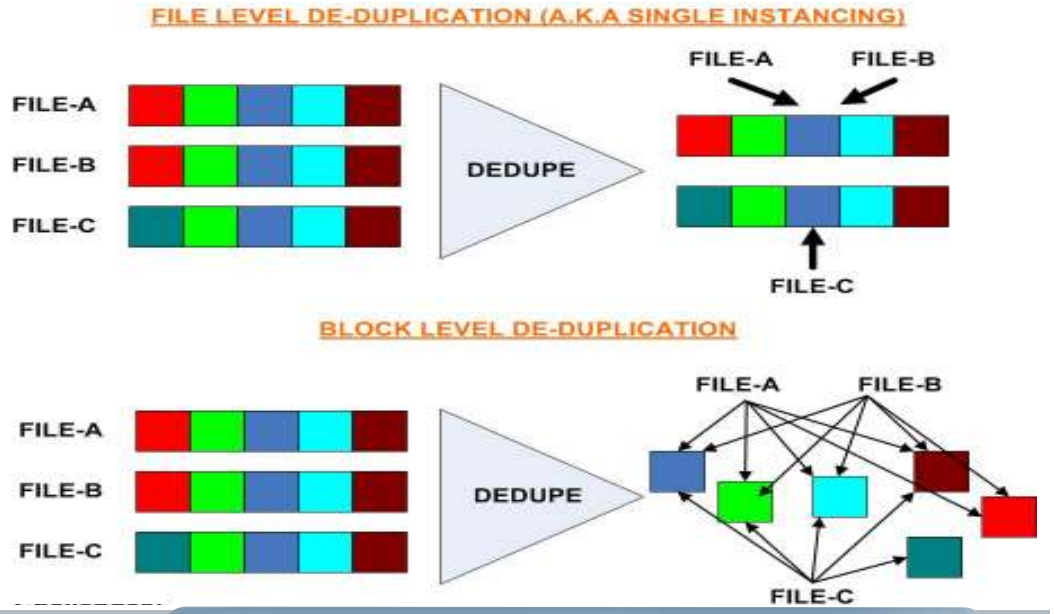
Petros Efstathopoulos, Fanglu Guo

Symantec Research Labs

June 2010

State of Deduplication

- Deduplication is maturing
  - Standard feature of backup/archive systems
  - Many approaches, algorithms, techniques
    - Inline, offline, file level, block level, variable/fixed sized blocks, global/local deduplication,…
  - High performance deduplication solutions available
  - Near-optimal usage of raw storage

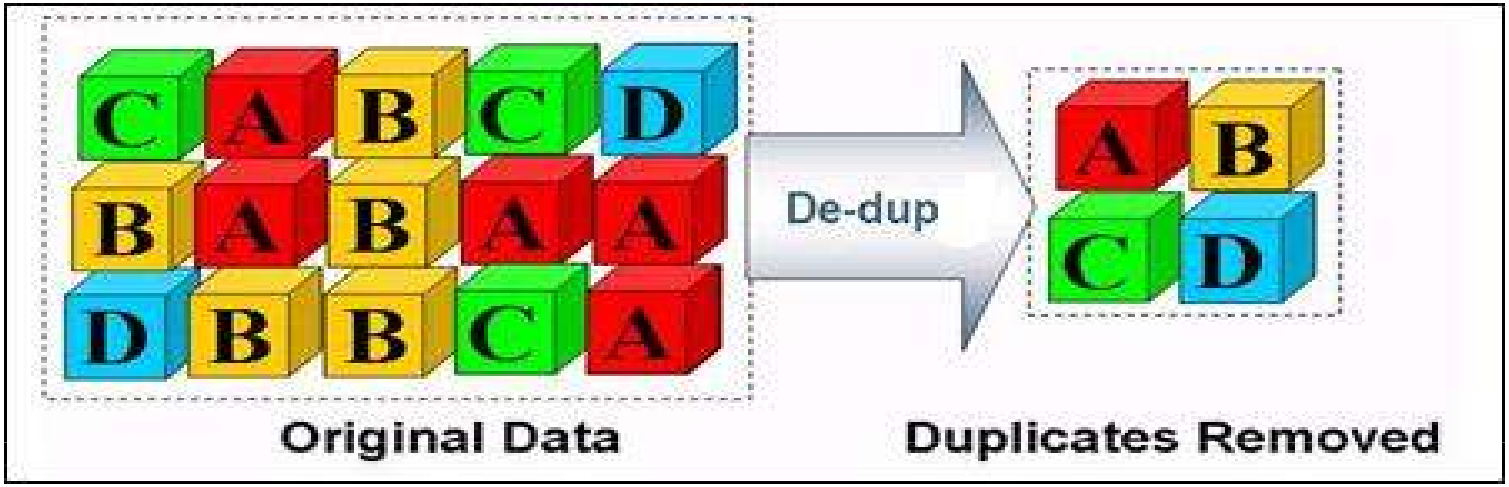


State of Deduplication

- Deduplication is maturing
  - Standard feature of backup/archive systems
  - Many approaches, algorithms, techniques
    - Inline, offline, file level, block level, variable/fixed sized blocks, global/local deduplication,…
  - High performance deduplication solutions available
  - Near-optimal usage of raw storage
- However: scalability still a problem
  - Only a few tens of TBs supported per node
    - Not enough for next-generation storage systems

Deduplication systems optimized for deduplication efficiency but scalability (i.e., raw supported capacity) requirements are increasing dramatically

Challenges Scaling Up



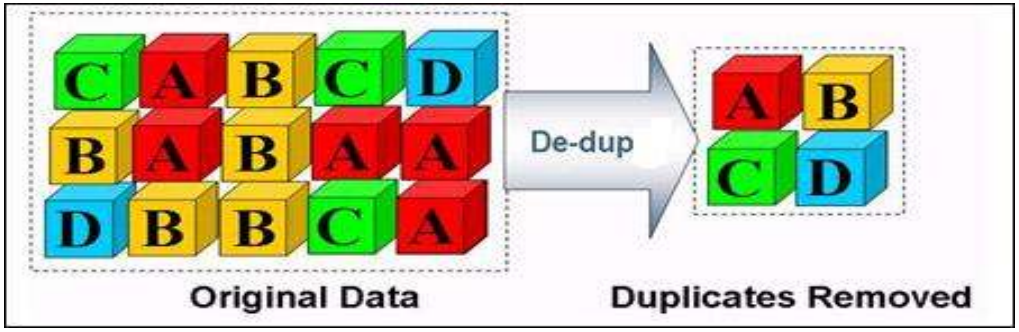
+

Deduplication Metadata

"Have I seen this block before?"

(if "yes", then where is it?)

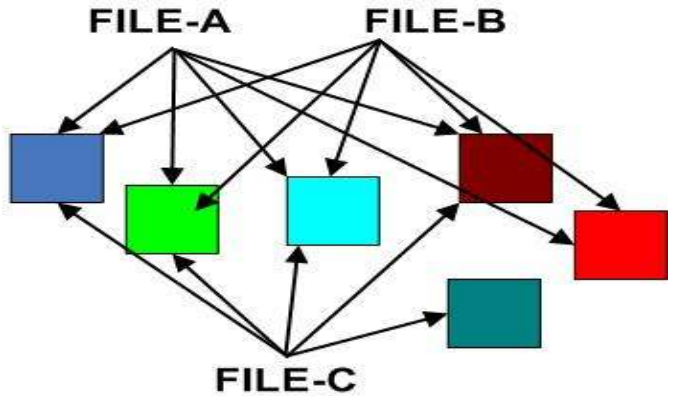
Challenges Scaling Up



"Have I seen this block before?"

- **Indexing Scalability Challenges**
  - System capacity bound by indexing capacity
  - Resorting to larger segment sizes
  - >75 K ops/sec
- **Reference Management Challenges**
  - Scalability bound by resource reference management and reclamation mechanism
  - Deletion errors not acceptable
  - Crashes possible
  - >75 K reference updates

Item	Scale	Remarks
Raw Capacity	C = 400 TB	
Segment Size	S = 4 KB	
Num of Segments	N = 100 * 10^9	N = C / S
Fingerprint size	F >= 20 B	
Block Index	M = 2000 GB	M = N * F
Disk Speed	Z = 300 MB/sec	
Lookup speed requirement	75 Kops/sec	Z / S



Goals

- Scalability
  - 100+ billion objects with high throughput
  - Willing to sacrifice *some* deduplication efficiency
- Performance
  - Near raw disk throughput for backup and restore
  - Near raw disk throughput for data deletion
  - Reasonable deduplication efficiency

Next Generation Dedup - Indexing

- **Increase supported capacity → Bigger segment size**
  - Lowers dedup efficiency/granularity – too coarse!
  - Supported capacity still low (a few tens of TBs)
- **"Progressive Sampled Indexing"**
  - Can't fit all FPs in memory → keep **1 out of *N* segment FPs**
- ***Sampling rate R*** = function (RAM, storage capacity, desired segment size)
- Estimate of sampling rate: **R = (S * M) / (E * C) = 1 / N**
  - R = sampling rate
  - E = bytes/entry
  - C = total raw storage in TBs
  - S = segment size in KBs
  - M = GBs of main memory used for indexing

Sampling Rate

- Example: 32 bytes/entry, 4KB segments and 32GB of RAM
  - No sampling (R=1) → 4 TB storage
  - 400TBs of storage → R ~= **1/100** – i.e., "keep 1 out of 100 FPs"
- Supports "infinite storage" → capacity VS dedup efficiency
- ***Progressive sampling:*** used storage VS available storage
  - Sampling rate = function(***used storage***, available RAM)
    - Start with no sampling (R=1), progressively lower R, down to R_{min} = (S * M) / (E * C)
- Straight sampling → poor deduplication efficiency
  - 1 out of N segments deduplicatable (i.e., "hit" on index)
- **Fingerprint cache: take advantage of spatial locality**
  - Small part of index memory: index hit → pre-fetch & cache all FPs in container

Dedup Indexing: Hitting the Disk

- SRL prototype → Sampled index in memory
- Index recovery after reboot/crash → **Disk Index**
  - Checkpointed index state – minimum necessary to resume indexing
- What if we store the index directly on disk?
  - More indexing capacity → Full(er) disk index → Better deduplication
- Disk operations too slow, bad performance
- SSDs provide interesting alternative

SSD Indexing

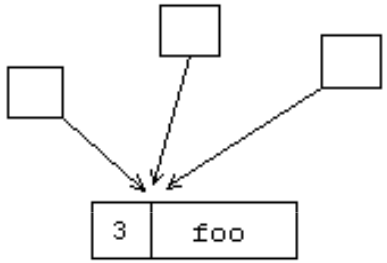
- Example: 32GB RAM → 128GB of SSD → x4 indexing capacity
  - 32 bytes/entry ~= 4 Gentries → 4KB segment → only 16 TB storage
  - With a much larger segment size of 128KB → ~512 TB
    - SSD holding full index could help scalability in the short term
- SSD-based sampled indexing:
  - Better sampling rates based on SSD capacity
    - x4 capacity → ***R' = 4 * R = 1 / 25***
  - Sorted index on SSD
  - All available memory used for speedup mechanisms:
    - Memory "catalogue" summarizing SSD ***guarantees*** that every lookup will take ***at most one*** SSD access (~4.8 MB of memory / GB of storage, for 20-byte FPs, 4KB SSD blocks)
    - Container caching/pre-fetching
    - Bloom filter

Reference Management & Resource Reclamation:

Grouped Mark-and-Sweep

The Problem and Challenges

- Problem: Can I delete a segment/object? Is it still in use/pending to be used?

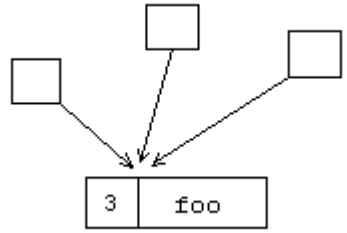


Challenges:

Scale:	**100 billion**
Reliability:	**No deletion error in a distributed system - crashes possible**
Speed:	**75k/s reference updates**

Next generation Dedup - Deletion

- **Reference count**
  - Simple but hard to make it crash-resilient (especially in a distributed system)
    - E.g., lost or repeated count update
  - Corrupted Object -> Who is using it??

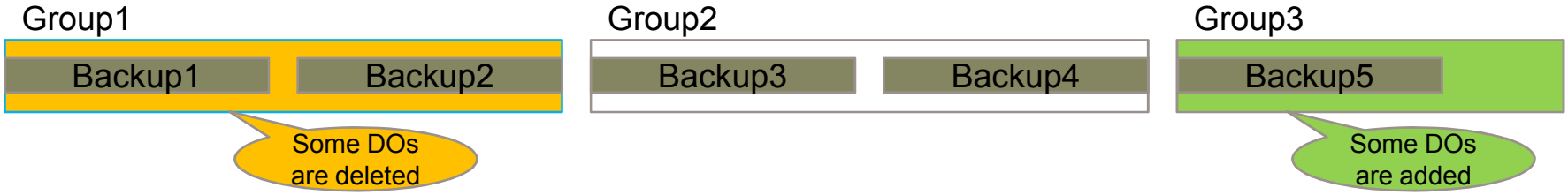


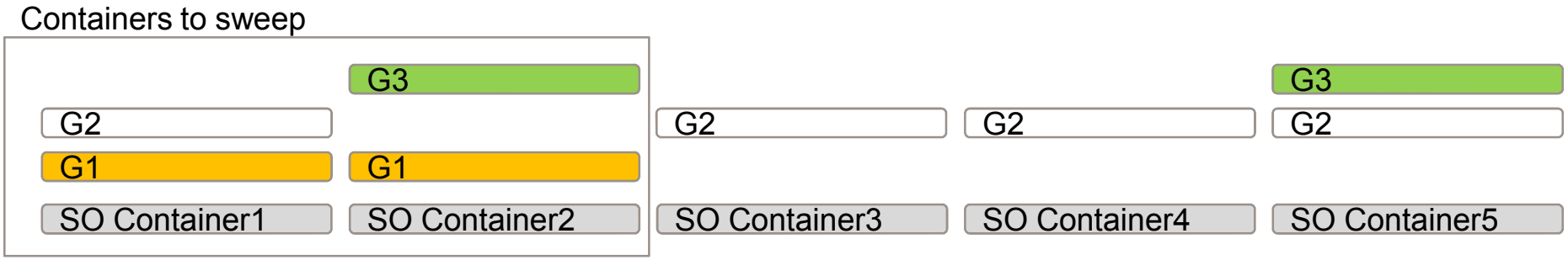
- **Reference list**
  - Resilient to “repeated updates” but ***not*** “lost updates”
  - Poor scalability & performance, variable size list needs to be stored on disk

- **Mark-and-sweep (aka garbage collection)**
  - **Workload proportional to system capacity**
    - 400 TB system, 4 KB segment, 20 byte FP → read 2 TB of data during mark phase
      → 1.85 hours to go through once at 300 MB/second
    - x10 deduplication factor ~= 20 TB → 18.5 hours

Next Generation Dedup – Grouped Mark-and-Sweep

- **Mark: Divide and save**
  - Divide backups to groups
  - Track changes to the groups
  - Only re-mark changed groups
  - Mark results are saved and reused if not changed
- **Sweep: Track affected containers**
  - Only sweep containers that may have deleted objects
- **Net result:**
  - **Workload = f (nature of work)**
    - Instead of system capacity





Preliminary Prototype Evaluation

Prototype Implementation & Preliminary Throughput Results

- Full prototype implemented in C++
  - Sampled indexing, Grouped M&S, Backup management logic, etc
  - Multithreaded implementation for max throughput:
    - Multiple hash calculation threads, multiple backup/disk writer threads
- 3GHz Intel Xeon multicore system, 8 TB Clariion FC storage
  - Hardware read/write baseline: ~380/330 MB/sec
- Single thread backup: ~310 MB/sec (94% of baseline)
  - CPU-bound hash calculation
- Single thread restore throughput: ~ 350 MB/sec (92% of baseline)
- Deduplication performance: up to 1180 MB/sec
  - Limited by disk read performance (reading pre-fetched container indexes)

Preliminary Index Results

- Example: drop rate 2%, 4 KB segments, 1KB buckets, 149 Mentries capacity

X	Load	Drop %	Insert	Lookup	Remove	Throughput (4KB segments, 3GHz Xeon)
147 M	97 %	~ 2 %	13,101	7,620	16,836	**916 / 1,575 / 713 MB/sec**

- 4 VMWare image versions (10.2 GB, 2621440 * 4 KB segments):

Unique segs	Total unique	Optimal space	Space used	Achievement	Diff
518326	518326	2073	2782	75%	
733267	921522	3686	4508	82%	MS Updates
904579	1189230	4756	5639	84%	NIS 2010
1145029	1616585	6466	7396	87%	Apps

- Preliminary SSD index implementation test results:

	SATA SSD	ioDrive
Seek time	0,24 msec	0,06 msec
Read Throughput	90 MB/sec	454 MB/sec
Cycles/lookup	138,871	55,590
Ops/sec	17,323	**53,646**

GMS Scalability Test

- Measurements: ~empty and ~full (8 TB, 2 billion objects, 4 KB segments)
- Preliminary evaluation:
  - Process time is proportional to workload
  - Process throughput is stable and fast regardless of capacity
- Results vary depending on underlying filesystem

Data	Ext3 Add (time/throughput)	Ext3 Remove (time/throughput)
30 GB	4.29 sec (6.99 GB/sec)	21.77 sec (1.38 GB/sec)
300 GB	39.33 sec (7.63 GB/sec)	218.77 sec (1.37 GB/sec)
990 GB	163.02 sec (6.07 GB/sec)	690.29 sec (1.43 GB/sec)

Conclusions & Future Work

- Scalable deduplication is possible
  - And necessary for system management/administration
- Sampled indexing + pre-fetching works
  - Scalability and throughput goals achieved
- Grouped Mark-and-Sweep: scalable reference management

- Work in progress and next goals:
  - Comprehensive end-to-end testing with real workloads
  - Testing of concurrent backups
  - Distributed operation
  - Complete SSD index implementation and testing

Thank You!

Petros Efstathopoulos

Petros_Efstathopoulos@symantec.com