Sorted Deduplication: How to Process Thousands of Backup Streams

Jürgen Kaiser
Zentrum für Datenverarbeitung (ZDV)
Johannes Gutenberg-Universität Mainz
Motivation

• Deduplication: popular compression technique in (backup) storage systems
• First systems around 2000
• Today: changed requirements
  • Before: few (big) streams
  • Now: many streams (example: cloud backup)
• Traditional performance optimizations are less efficient for many streams
• Our Contribution: deduplication approach tailored for many streams
Reminder: Deduplication

chunking

fingerprinting

chunk identification

recipe generation

container

recipe

storage system

input data

FP1, FP2, FP3, FP2, FP4, FP1, FP2
Reminder: Deduplication

- **input data**
- **chunking**
- **fingerprinting**
- **chunk identification**
- **recipe generation**
- **container**
- **recipe**
- **storage system**

FP1, FP2, FP3, FP2, FP4, FP1, FP2
Reminder: Deduplication

input data

chunking
finger printing
chunk identification
recipe generation

container

storage system

FP1, FP2, FP3, FP2, FP4, FP1, FP2

recipe

FP1, FP2, FP3, FP2, FP4, FP1, FP2
Reminder: Deduplication

chunking

chunk identification

recipe generation

input data

FP1, FP2, FP3, FP2, FP4, FP1, FP2

container

recipe

storage system
Reminder: Deduplication

input data

chunking

fingerprinting

chunk identification

recipe generation

container

recipe

storage system
Reminder: Deduplication

chunking

fingerprinting

chunk identification

recipe generation

input data

FP1, FP2, FP3, FP2, FP4, FP1, FP2

recipe

container

storage system

FP1, FP2, FP3, FP2, FP4, FP1, FP2

FP1, FP2, FP3, FP2, FP4, FP1, FP2

FP1, FP2, FP3, FP2, FP4, FP1, FP2

FP1, FP2, FP3, FP2, FP4, FP1, FP2
Reminder: Deduplication

input data

chunking

chunk identification

recipe generation

fingerprinting

container

FP1, FP2, FP3, FP2, FP4, FP1, FP2

recipe

storage system
Reminder: Deduplication

chunking

chunking

fingerprinting

chunk identification

recipe generation

storage system

container

recipe

FP1, FP2, FP3, FP2, FP4, FP1, FP2

input data

FP1

FP2

FP3

FP2

FP4

FP1

FP2
Scaling up to 1000+ streams...

- Low memory per stream
  - Reduced cache efficiency → more I/O operations
- Each stream generates own I/O pattern
- At this scale: stream-locality utilization becomes less effective
Patterns for 1 stream

DDFS

Sparse Indexing
Patterns for 128 streams

DDFS

Sparse Indexing
Scaling up to 1000+ streams...
Scaling up to 1000+ streams...

With a single client:
- Sort index entries according to stream
- Almost sequential access
Scaling up to 1000+ streams...
Scaling up to 1000+ streams...

Step 1: establish same fingerprint arrival order
- Sort the fingerprints in clients
Scaling up to 1000+ streams...

Step 1: establish same fingerprint arrival order
- Sort the fingerprints in clients

Step 2: hold index entries in the same ordering
- Given by index implementation (LSM-Tree)
Scaling up to 1000+ streams...

Step 1: establish same fingerprint arrival order
- Sort the fingerprints in clients

Step 2: hold index entries in the same ordering
- Given by index implementation (LSM-Tree)

Step 3: synchronize stream processing
- Simple merge
Scaling up to 1000+ streams...

Result:
- Sequential I/O, independently from #clients
- High index locality: Max 1 I/O per index region (= index page)
- No interference among the streams
New Problem

- How to restore data?
  - Before: process and restore in same order
  - Now: process order != restore order
- Clients chunk & fingerprint data
- Send fingerprints first, send chunks in orig. order

Client

Server

[Index lookup]

[update recipes]

[recipes + raw chunk data]

[0x01, 0x03, 0x07, 0x08, 0x09,…]

[nil, 4, 4, nil, 27,…]
More Problems

• What to do with weak clients?
• What to do with weak interconnects?
• How to scale to multiple servers?
Evaluation

1. Comparison with other systems
2. Scaling properties a prototype implementation (SCI)
Comparison with DDFS and SI

- **DDFS**: Data Domain Deduplication File System
  - Exact deduplication
- **SI**: Sparse Indexing (HP)
  - Approximate deduplication

- **Data Sets:**
  - **HOME**: 597 diff. streams, 7TB total, home directories
  - **Microsoft**: 140 diff. streams, 49TB total, workstations

- **Metric**: Average number of generated I/Os per 1K chunks
Comparison with DDFS and SI

![Graphs showing comparison with DDFS and SI for Microsoft and HOME]
Comparison with DDFS and SI

Microsoft

HOME
Patterns for Microsoft, 28th backup gen.

DDFS

Sparse Indexing

SCI
Memory Consumption

- In Simulations: used 8GB
- Today: much more main memory
- Question: How do the systems compare for more memory?
Prototype

• Questions:
  • How fast is the chunk identification?
  • What is the bottleneck?
• Backup volume (per client)
  • Bigger: more checks p. index page $\rightarrow$ higher CPU load
• Index size
  • Bigger: more index pages $\rightarrow$ higher HDD load
Prototype

- **Hardware:**
  - Intel Xeon, 4 cores @3.3GHz
  - 16GB RAM
  - 10Gbit ethernet
  - Single HDD

- **Used artificial data**

- **Assumptions:**
  - 8KB chunks
  - 90% deduplication ratio
  - 50% compressability of raw data
Scaling properties of the prototype

- Backup size: variable
- Fingerprint index size: fixed

![Graph 1: Backup Vol p. client](chart1)

- Backup size: fixed
- Fingerprint index size: variable

![Graph 2: Vol. pref. chunks](chart2)
Conclusion

• SCI: Sorted chunk indexing

• Core idea: enforce stream processing to
  • access same index region
  • at the same time
  • in the same order

• Works best with many streams + big backups
  • Low I/Os per 1K chunks
  • Sequential disk access
  • Eliminates the chunk identification as bottleneck
Thank you!
Questions?
Diff. Memory usage, HOME

Average I/Os per 1K chunks vs. Max. Memory

- DDFS
- SI
- SCI

Rel. num. of stored duplicates

May 5, 2016 | Jürgen Kaiser
SCI per generation

HOME

Microsoft
Scaling properties of the prototype

- Backup size: variable
- Fingerprint index size: fixed

Backup Vol p. client:
- 1GB
- 16GB
- 32GB

Vol. pref. chunks:
- 1TB
- 4TB
- 16TB

Throughput (GB/s):

Rel. Time Ident.:

Total Time (s):

Number of Clients:
IO Patterns