iDedup

Latency-aware, inline data deduplication for primary storage
Kiran Srinivasan, Tim Bisson, Garth Goodson, Kaladhar Voruganti
Fast 2012

Ido Hakimi
Shahar Schneor
Outline

• Introduction
• Motivation
• Design
• Architecture
• Components
• Implementation
• Evaluation
• Related work
• Conclusion
Introduction

- Deduplication technologies are increasingly being deployed to reduce cost and increase space-efficiency in corporate data servers.
- However, prior research has not applied deduplication techniques inline to the request path for latency sensitive, primary workloads.
- Inline systems deduplicate requests in the write path before the data is written to disk.

<table>
<thead>
<tr>
<th>Type</th>
<th>Offline</th>
<th>Inline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary, latency sensitive</td>
<td>NetApp ASIS [1], EMC Celerra [11], StorageTank [16], iDedup (This paper)</td>
<td></td>
</tr>
<tr>
<td>Secondary, throughput sensitive</td>
<td>(No motivation for systems in this category)</td>
<td>EMC DDFS [41], EMC Cluster [8], DeepStore [40], NEC HydraStor [9], Venti [31], SiLo [39], Sparse Indexing [21], ChunkStash [7], Foundation [32], Symantec [15], EMC Centera [24], GreenBytes [13]</td>
</tr>
</tbody>
</table>
The Challenge

- To not increase the latency of the already latency sensitive operations
The Obstacles

- **Read Path**
  - Reads are affected by the fragmentation in data layout
  - Fragmentation naturally occurs when deduplicating

- **Write Path**
  - Writes deals with most of the deduplication work
  - Sometimes needs to read on-disk data structures

- **Delete Path**
  - Typically, some metadata records the usage of shared block
  - This metadata must be execute inline to the deletion request
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
The World of Data

NUMBER OF EMAILS SENT EVERY SECOND: 2.9 MILLION
DATA CONSUMED BY HOUSEHOLDS EACH DAY: 375 MEGABYTES
VIDEO Uploaded to YouTube EVERY MINUTE: 20 HOURS
DATA PER DAY PROCESSED BY GOOGLE: 24 PETABYTES
TWEETS PER DAY: 50 MILLION
TOTAL MINUTES SPENT ON FACEBOOK EACH MONTH: 700 BILLION
DATA SENT AND RECEIVED BY MOBILE INTERNET USERS: 1.3 EXABYTES
PRODUCTS ORDERED ON AMAZON PER SECOND: 72.9 ITEMS
Why primary inline deduplication?

- Storage provisioning is easier and more efficient
  - *Offline systems require additional space to absorb the writes prior to deduplication*

- No dependence on system idle time
  - *When the system is busy for long periods of time it cannot perform offline deduplication*

- Disk-bandwidth utilization is lower
  - *Offline systems use extra disk bandwidth when reading in the staged data to perform deduplication and then again to write out the results*
Key Insights

- **Spatial locality** exists in the duplicated data
  - allows to amortize the seeks caused by deduplication by only performing deduplication when a sequence of on-disk blocks are duplicated
  - mitigates fragmentation and addresses the extra read path latency

- **Temporal locality** exists in the accesses of duplicated data
  - enables to maintain an in-memory fingerprint cache to detect duplicates instead of using on-disk structures.
  - removes extra IOs and lowers write path latency
Control Parameters

- The minimum number of sequential duplicate blocks on which to perform deduplication (**Spatial Locality**)
- The size of the in-memory fingerprint cache (**Temporal Locality**)

[Diagram showing fragmentation with random seeks and sequences, with amortized seeks]
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Spatial Locality

**Approach**
- To only deduplicate full sequences of file blocks if and only if the sequence of blocks are:
  - sequential in the file
  - duplicates that are sequential on disk

**Disadvantages**
- Even with this optimization, sequential reads can still incur seek between sequences
- Only a subset of blocks are deduplicated which leads to lower capacity savings
Temporal Locality

Approach
- To maintain the dedup-metadata as a completely memory-resident, LRU cache, thereby, avoiding extra dedup-metadata IOs

Disadvantages
- We might not deduplicate certain blocks due to lack of information (not in cache)
- The memory used by the cache reduces the file system’s buffer cache size, this can lead to a lower buffer cache hit rate, affecting latency
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Overview

- In order to reduce response latency they used NVRAM to buffer client writes.
- These writes are periodically flushed to disk during the **destage** phase.
- The newly written (dirty) blocks need to be deduplicated during the destage phase.
- By performing deduplication during destage, the system benefits by not deduplicating short-lived data that is overwritten or deleted while buffered in NVRAM.
Write Path Flow

- The following steps take place in the write path:
  - For each dirty block, we compute its fingerprint and perform a lookup in the dedup-metadata structure (in RAM)
  - If a duplicate is found, we examine adjacent blocks, using the iDedup algorithm to determine if it is part of a long enough duplicate sequence
  - If a duplicate sequence is found, the file’s metadata is updated. Otherwise, we allocate new disk blocks and add the fingerprint metadata for these blocks
  - Finally, to maintain file system integrity in the face of deletes, we update reference counts of the duplicated blocks in a separate structure on disk
Read Path Flow

- Since iDedup updates the file’s metadata as soon as deduplication occurs, the file system cannot distinguish between a duplicated block and a non-duplicated one.

- This allows file reads to occur in the same manner for all files, regardless of whether they contain deduplicated blocks.
Delete Path Flow

- During deletion, the reference count of deleted blocks is decremented (on disk) and only blocks with no references are freed.
- In addition to updating the reference counts, we also update the in-memory dedup-metadata when a block is deleted.
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
The algorithm should be able to identify sequences of file blocks that are duplicates and whose corresponding DBNs are sequential.

The algorithm should minimize searches in the dedup-metadata to reduce CPU overheads.

The memory and CPU overheads caused by the algorithm should not prevent other file system processes from accomplishing their tasks in a timely manner.
Dedup-metadata cache design

- Each entry maps the fingerprint of a block to its DBN on disk
- They used LRU as the cache replacement policy; other replacement policies did not perform better than the simpler LRU scheme
- Given a block size of 4 KB (typical of many file systems), the cache entries comprise an overhead of 0.8% of the total size
- The memory used by the dedup-metadata cache comes at the expense of a larger buffer cache which affects the hit ratio
Sequence Identification

- The goal is to identify the largest sequence among the list of potential sequences.

- This can be done in multiple ways:
  - **Breadth-first:** Start by scanning blocks in order; concurrently track all possible sequences; and decide on the largest when a sequence terminates.
  - **Depth-first:** Start with a sequence and pursue it across the blocks until it terminates; make multiple passes until all sequences are probed; and then pick the largest. Information gathered during one pass can be utilized to make subsequent passes more efficient.

- Since multiple passes is too expensive, they used the breadth-first approach.
Overlapped Sequences

- Choosing between a set of overlapped sequences can prove problematic
- By accepting $S_1$, we are rejecting the overlapped blocks from $S_2$ or $S_3$; this is the dilemma
Threshold Determination

- For workloads with more random IO, it is possible to set a lower threshold because deduplication should not worsen the fragmentation.
- It is possible to have a real-time, adaptive scheme that sets the threshold based on the randomness of the workload.
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Implementation

- The implementation consists of two subsystems:
  - The dedup metadata management.
  - The iDedup algorithm.
Dedup-metadata management

- Dedup-metadata cache (in RAM):
  
  - Contains a pool of block entries (content-nodes) that contain deduplication metadata organized as a cache.

- Fingerprint hash table (in RAM):
  
  - This table maps a fingerprint to DBN(s).

- DBN hash table (in RAM):
  
  - This table maps a DBN to its content-node; used to delete a block.

- Reference count file (on disk):
  
  - Maintains reference counts of deduplicated file system blocks in a file.
Dedup-metadata cache

- A fixed-size pool of small entries called content nodes, managed as an LRU cache.
- Each content-node represents a single disk block.
  - Content-node contains the block’s DBN and its fingerprint.
  - All the content-nodes are allocated as a single global array.
  - Each content node is indexed by three data structures: the fingerprint hash table, the DBN hash table and the LRU list.
- Fingerprint tool: MD5 checksum of the block’s content.
- Save 24 bytes per entry.
Fingerprint hash table

- Contains content-nodes indexed by their fingerprint.
- Each hash bucket:
  - contains a single pointer to the root of a red-black tree containing the collision list for that bucket.
  - designed to hold 16 entries.
Fingerprint hash table

- Each collision tree content-node represents a unique fingerprint value in the system.

- For thresholds greater than one, it is possible for multiple DBNs to have the same fingerprint, as they can belong to different duplicate sequences.
  - Therefore, all the content-nodes that represent duplicates of a given fingerprint are added to another red-black tree, called the dup-tree.
  - This tree is rooted at the first content-node that maps to that fingerprint.
DBN hash table

- Indexes content-nodes by their DBNs.
- Similar to the fingerprint hash table without the dup-tree.
- Facilitates the deletion of content nodes.
  - *During deletion, blocks can only be identified by their DBNs.*
- The DBN is used to locate the corresponding content-node and delete it from all dedup-metadata.
Reference count file

- Stores the reference counts of all deduplicated blocks on disk.
- Ordered by DBN.
- Maintains a 32-bit counter per block.
iDedup Algorithm

- The iDedup algorithm works in 4 phases for every file:
  - **Phase 1 (per file): Identify blocks for iDedup**
    - Only full, pure data blocks are processed
    - Metadata blocks, special files ignored
  - **Phase 2 (per file): Sequence processing**
    - Uses the dedupe metadata cache
    - Keeps track of multiple sequences
  - **Phase 3 (per sequence): Sequence pruning**
    - Eliminate short sequences below threshold
    - Pick among overlapping sequences via a heuristic
  - **Phase 4 (per sequence): Deduplication of sequence**
Sequence identification

- scanning the blocks in order.
  - *Utilize the fingerprint hash table to identify any duplicates for these blocks.*
- filter the blocks to pick only data blocks that are complete.
  - *Compute the MD5 hash for each block.*
Sequence identification

- $n$ represents the block’s offset within the file – FBN.
- Minimum length of a duplicate sequence is 2.
- Examine blocks in pairs.
- For each pair, e.g., $B(n)$ and $B(n+1)$, we perform a lookup in the fingerprint hash table for $H(n)$ and $H(n+1)$.
- If neither of them is a match, we allocate the blocks on disk normally.
- If there is a match?
Sequence identification

- Four match conditions
  - Both $H(n)$ and $H(n+1)$ match a single content-node: Simplest case, if the DBN of $H(n)$ is $b$, and DBN of $H(n+1)$ is $(b+1)$, then we have a sequence.
  - $H(n)$ matches a single content-node, $H(n+1)$ matches a dup-tree content-node: If the DBN of $H(n)$ is $b$; search for $(b+1)$ in the dup-tree of $H(n+1)$.
  - $H(n)$ matches a dup-tree, $H(n+1)$ matches a single content-node: Similar to the previous case with $H(n)$ and $H(n+1)$ swapped.
  - Both $H(n)$ and $H(n+1)$ match dup-tree content-nodes: This case is the most complex and can lead to multiple sequences.
Sequence identification

Case 4:

- Identify all possible sequences that can start from these two blocks.
- search primitive, nsearch(x):
  - returns ‘x’ if ‘x’ is found.
  - next largest number after ‘x’.
  - error if ‘x’ is already the largest number.
Sequence pruning

- Process the sequences according to their sizes.
- *If a sequence is larger than the threshold, check for overlapping blocks with non-terminated sequences.*
- *Shorter than the threshold, the non-overlapped blocks are allocated by assigning them to new blocks on disk*

Deduplication of blocks

- For each deduplicated block, the file’s metadata is updated with the original DBN at the appropriate FBN location.
- Reference count of the original DBN is incremented.
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Evaluation

- The goal: reasonable tradeoff between performance and deduplication ratio that can be exploited by iDedup for latency sensitive, primary workloads.

- major tunable parameters:
  - Minimum duplicate sequence threshold
  - In-memory dedup-metadata cache size
Out Of The Box

- Evaluation - NetApp FAS 3070 storage system running Data ONTAP R 7.3
- Replay client trace - Intel Xeon
- 2 real world CIFS traces
  - Corporate trace
  - Engineering trace
Comparison Points

- The Baseline values represent the system without the iDedup algorithm enabled (i.e., no deduplication).
- The Threshold-1 values represent the highest deduplication ratio for a given metadata cache size.
Deduplication ratio vs. threshold
Client response time

- Behavior
  - response time as measure of latency.
  - Fragmentation effect on Average latency.
System CPU utilization vs. threshold

- capture CPU utilization samples every 10 seconds.
- compute the CDF for these values (cumulative distribution function).
- Workloads with 1 GB dedup-metadata cache.
- Expecting: Threshold-8 to consume more CPU compared to the Baseline.
System CPU utilization vs. threshold
The size of the dedup metadata cache (and threshold) had no observable effect on the buffer cache hit ratio.

- the dedup-metadata cache size (max of 1 GB) is relatively small compared to the total memory (8 GB).
- the workloads’ working sets fit within the buffer cache.
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Related Work

- Data storage efficiency can be realized via various complementary techniques.
- Deduplication systems can be classified as primary or secondary.
  - Primary storage is usually optimized for IOPs and latency
  - Secondary storage systems are optimized for throughput
- The systems either process duplicates inline, at ingest time, or offline, during idle time.
Outline

- Introduction
- Motivation
- Design
- Architecture
- Components
- Implementation
- Evaluation
- Related work
- Conclusion
Conclusion

- iDedup, an inline deduplication system specifically targeting latency-sensitive, primary storage workloads.

Challenges:
- Fragmentation
- Extra latency

Solutions:
- 2 primary workloads: spatial locality and temporal locality.
Reference

- iDedup: Latency-aware, inline data deduplication for primary storage
  Kiran Srinivasan, Tim Bisson, Garth Goodson, Kaladhar Voruganti. iDedup: Latency-aware, inline data deduplication for primary storage. FAST 2012.