Scalable Deduplication for Archival Systems

Deepavali Bhagwat
SSRC

Darrell Long
SSRC

Kave Eshghi
HP Labs, Palo Alto
Archival Systems

- Archival data will increase tenfold from 2005 to 2010
  - Total worldwide digital archive capacity in the commercial and government sectors will grow to 27 exabytes
  - Bulk of archived data will consist of unstructured content
  - Compliance regulations such as Sarbanes Oxley and HIPAA require organizations to retain data for long periods
- Internet Archive has incoming data at 25TB/month
  - [Schwarz et al., WIP MSST 2006]
Deduplication in Large Scale Archival Systems

- Index lookup: an integral part of the deduplication process in archival systems
  - A lookup index can consist of chunk signatures, shingles
- In large scale archival systems index lookup becomes a performance bottleneck
  - Index size increases
  - Query, update performance suffers
  - Network bandwidth underutilized
  - Turnaround time not satisfactory

- Approaches used today:
  - Out-of-band deduplication
  - In-band deduplication
Deduplication Approaches

- **Out-of-band deduplication**
  - Done asynchronously — archival data first written to temporary storage, then deduplicated ‘out-of-band’
  - Advantage: Full use of network bandwidth
  - Disadvantages:
    - Need to write data twice — on temporary media and archival media
    - Need to have extra storage space available
  - Network Appliance, Tivoli Storage Manager, Sepaton Inc.

- **In-band deduplication**
  - Done while data written to archival storage systems (Virtual Tape Library)
  - Advantage: Data needs to be handled only once
  - Disadvantage: Slows down the archival process
  - Data Domain, Diligent Corp, Symantec Corp
Efficient deduplication

- **Requirements:**
  - Parallel execution with multiple deduplication engines
  - Fast turnaround
  - Bandwidth utilization

- **Ideal approach:**
  - Partition lookup indices
  - Access a small fraction of partitions to identify and eliminate redundancies
Our Solution

- Apply a document routing and index partitioning algorithm for detection of redundancies between documents
  
Scalable Similarity Based Retrieval in a Large Corpus

- **Similarity Based Retrieval:**
  
  Given a file $f_q$, find all documents in a given repository similar to $f_q$

- **For similarity based retrieval:**
  - Extract features from every repository file and query file
  - Compute similarity using features
  - The degree of similarity $\Rightarrow$ The number of features shared
  - The results ranked on the basis of the degree of similarity

- The document routing algorithm has been shown to preserve the recall and precision of similarity based queries

- Similarity based retrieval used in archival systems to identify content overlaps/redundancies and conserve storage space

- Features of a document in the case of a chunk-based archive are chunk signatures
Partitioning the Chunk Index

- Goal: Partition chunk index $I$ into multiple partitions $I_0, \ldots, I_{K-1}$
- When adding a new file $f_n$ to the archive:
  - Extract chunk signatures
  - Choose $m$ partitions using the document routing algorithm ($m$: routing factor)
  - Route *all* the chunks to $m$ partitions, $m \ll K$
  - Add the chunk signatures to each of the $m$ partitions
- When identifying redundancies within a file $f_q$
  - Extract chunk signatures
  - Choose $m$ partitions using the document routing algorithm
  - Route all the features to $m$ partitions, $m \ll K$
  - Query each of the $m$ partitions to identify redundant chunks
- Goal: Minimal loss of compression while $m$ being much smaller than $K$
Document Routing Algorithm Adapted for Archival Systems

- **Input:**
  - $C(f)$: The set of chunk signatures of $f$
  - $K$: the number of partitions
  - $m$: the routing factor ($m \ll K$)

- **Output:**
  - Set of integers $R = \{r_0, \ldots, r_{m-1}\}$, the partitions to which $f$ will be routed

- **Algorithm:**
  - Compute $bot_m(C(f))$
  - For every signature $c$ in $bot_m(C(f))$ compute $(c \mod K) \Rightarrow R$
Broder’s Theorem

- Consider:
  - Two sets $S_1$ and $S_2$
  - $H(S_1)$ and $H(S_2)$: sets of hash values of every element in $S_1$ and $S_2$

$$P(min(H(S_1)) = min(H(S_2))) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

The probability that the two sets $S_1$ and $S_2$ have the same minimum hash element is the same as their similarity measure.

- If two sets are highly similar the minimum element of $H(S_1)$ and $H(S_2)$ is the same with high probability.
- Two files $f_n$ (in the repository) and $f_q$ (query file) are highly similar they will be routed to the same partition with high probability.
- Use $m > 1$ least elements to increase the probability of finding similar files.
Deduplication for a Chunk-Based Archive

File → extract → Chunk signatures → $bot_m$ → m bottom chunks → mod K → target Index partitions

Chunks NOT found: $\{c_4, c_{11}, \cdots, c_{99}\}$, $\{c_4, c_{99}\}$, $\{c_4, c_{12}, c_{15}, \cdots, c_{99}\}$

Only store chunks: $\{c_4, c_{99}\}$

Chunk Store
Experimental Setup

- Experimental Data

<table>
<thead>
<tr>
<th></th>
<th>Internet Archive</th>
<th>Santa Cruz Sentinel</th>
<th>Linux Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of files</td>
<td>2045460</td>
<td>82523</td>
<td>172584</td>
</tr>
<tr>
<td>Average file size</td>
<td>29 kB</td>
<td>1.2 MB</td>
<td>21 kB</td>
</tr>
<tr>
<td>Total file space</td>
<td>58 GB</td>
<td>43 GB</td>
<td>3.5 GB</td>
</tr>
<tr>
<td>Average chunk size</td>
<td>872 B</td>
<td>945 B</td>
<td>924 B</td>
</tr>
<tr>
<td>Average chunks/file</td>
<td>31</td>
<td>540</td>
<td>22</td>
</tr>
</tbody>
</table>

- Experiment methodology
  - Extract chunks from every file (duplicate free data set)
  - Extract chunk signatures by calculating the MD5 value across the chunk’s contents
  - Use one index to store chunks and measure obtained compression
  - Partition the index into multiple partitions
  - Route documents to a selected set of partitions using the document routing algorithm
  - Record the compression obtained with multiple partitions
Compression for Linux sources (128 partitions)
Compression for Linux Sources, 128 partitions using SHA1, Rabin’s Fingerprint and MD5 for Chunk Signatures
Compression for Internet Archive, MD5 signatures
Compression for Santa Cruz Sentinel, MD5 chunk signatures
Dynamic Re-partitioning

File → extract → Chunk signatures → $bot_m$ → m bottom chunks → mod K → target Index partitions

0 → … → … → P → … → K-1
Dynamic Re-partitioning of Indices

File → extract → Chunk signatures → $bot_m$ → m bottom chunks → mod K → target Index partitions

$0$ → \ldots → \ldots → \ldots → K-1

Partition

mod $(2^K)$ (l=1)

P → K+P

Re-partition P using File recipes

$f : c_1, c_5, \ldots, c_{77}$
Dynamic Re-partitioning of Indices

File

extract

Chunk signatures

bot

m bottom chunks

mod K

target Index partitions

0

…..

…..

……….…..

mod (2^K) (l=1)

mod (2^K) (l=2)

Partition

Partition

Partition

K+P

K+P

2K+P

P

Re-partition P using File recipes
Conclusion

- The document routing algorithm is an effective, scalable solution for deduplication in archival systems
- We were able to achieve more than 95% of the original compression using 128 partitions while routing documents to only 4 partitions
Scalable Deduplication for Archival Systems

Deepavali Bhagwat
SSRC

Darrell Long
SSRC

Kave Eshghi
HP Labs, Palo Alto