Deduplication

Deduplication is a form of compression. At a high level, deduplication systems

1. identify duplicate objects, and
2. eliminate redundant copies of information

How the system defines and "object" and how the system defines a "redundant copy" is system specific.

Deduplication systems can be defined along several axes.

On-line vs. offline

In on-line dedup systems, the deduplication process happens at the time that data is written. A typical online deduplication system works as follows:

1. When a write request is received (before data is actually written), the system is checked for duplicate copies.

- If the object is found to be unique, then the object is written as normal.
- If a duplicate object is found in the system, the new object is not written; instead, a reference to the original is written in its place.

In off-line dedup systems, the deduplication process happens after data is already persisted. A typical offline deduplication system will have a background process that scans data, and replaces duplicate objects with references to a common copy.

Fingerprinting

Fingerprints are unique content identifies. Scanning a system and comparing all objects byte-by-byte is impractical, so a cryptographic hash is used as a unique ID. Then, to compare objects (regardless of their size), you only need to compare their fingerprints.

- For this to work, the probability of collision must be close-to-zero. A collision is a correctness error
- We often choose hash functions and hash sizes so that the probability of a collision is less likely than the probability of a hardware error.

Chunking

Chunking is the process of breaking data into objects. Chunks can be whole-file objects, fixed-size chunks, or variable-sized chunks.

- Whole-file deduplication is simple and often low-overhead. If two files are exact copies, only one is written. Subsequent copies store references to the original.
  - The amount of metadata required to keep track of all objects in the system is quite low.
  - Any modification to a file requires unique copies to be made, which means that the system has a lower compression ratio.
- Fixed-size chunks are often defined in sizes that are multiples of system hardware parameters, like memory pages or disk sectors.
  - The process of defining chunks is easy
  - If a single chunk is modified, common chunks can still be shared
  - If data *shifts*, for example after an insertion of a byte at the head of a file, then all subsequent chunks will shift. Thus, local changes *can* falsely cause duplicates to be treated as unique
- Variable sized chunks are often defined by the contents of a file.
  - If a single chunk is modified, common chunks can still be shared
  - If data *shifts*, then it is unlikely that nearby chunk boundaries are affected, since the boundaries are determined by the contents

Chunk Size

- Large chunks require less metadata overheads, but usually result in lower deduplication ratios.
- Small chunks require more metadata overhead to index, but usually have higher deduplication ratios

Indexing

The chunk index can be a bottleneck in large deduplication systems, since it likely will not fit into RAM. Hashes are randomly distributed, so fingerprint index lookups often have no locality.

- Bloom filters can be used to detect whether a fingerprint exists, eliminating the need for some unnecessary lookups.
- Some systems group fingerprints into groups on disk. If groups are defined by temporal locality, then caching and evicting based groups may improve cache efficiency
  - Question: how would you define the appropriate "group" for a very common chunk (e.g., the chunk is common to many unrelated files, each with their own fingerprint group)