Haystack and F4

Haystack

The Haystack system is based on a clear goal: to reduce the metadata required for each image, thereby enabling full access to memory, and reducing or even avoiding disk access when capturing image metadata, and improving the speed of access to long tail images.

To achieve this goal, Haystack consists of three main parts: Cache, Store, Directory. Haystack Store is responsible for the persistent storage of images and is the only component of Filesystem Metadata that manages images. The Filesystem Metadata refers to the metadata that instructs how to read image data from a physical disk. There is also Application Metadata, which specifies how to build a URL for a browser to access an image. Haystack Directory is responsible for generating the URL for the image and returning to the Web Server. The Web Server returns the image URL to the browser. When the browser gets the image URL, it will request the image data from the CDN or HayStack Cache according to the URL's address. Haystack Cache is similar to an internal CDN, which provides the Cache of popular picture for Store and provides shelter when the upstream CDN node fails, accepting direct requests from the client to re-read the content.

Haystack treats each photo file as a Needle containing metadata about the actual data and logical photo files. Part of the metadata needs to be loaded into memory for photo lookup, including Key, Alternate Key, Offset of photo in physical volumes, and Size. In theory, a machine can rebuild its in-memory mapping by reading all the physical volumes, which is time consuming because all the data must be read from disk, so an index file is maintained for each volumes to keep each Needle looking for relevant metadata. Haystack Store adopts the recycling strategy of delayed deletion. To delete photos, only add a Needle with deletion mark to the volume, and periodically execute Compaction task to recycle the deleted space. The Compaction operation scans the data in all the old data files to delete the last photo and generate a new data file.

F4

F4 was developed to improve on the performance of Facebook’s existing storage system Haystack. It provides low latency, and storage resilience, but uses a lower replication factor of files. The data stored in the system are called Binary Large OBjects (BLOBs), which is an immutable binary data format; they are read many times, never modified, and sometimes deleted. The data stored by the system can vary greatly in terms of access; Haystack was build for relatively frequently accessed information, or “hot” data, while “warm” data is a separate use case and optimizations have been created in the form of of f4

Storage efficiency was one of the main design goals of f4. One of the hurdles to achieve this was maintaining the ability to be fault tolerant. Reed-Solomon coding has been used to reduce the storage overhead of data. The tradeoff of lower replication factor this technique provides is that the data has lower maximum read throughput and recovery times after failure increase.

BLOBS are stored in f4 cells, which are groups of racks. These cells are locked, meaning that the data is read-only. Files are divided into contiguous sequences of blocks, with additional parity blocks being generated for error handling and detection. Each f4 cell has 5 different types of nodes: name, storage, backoff, coordinator, and rebuilder. The name nodes store that data that maps data, parity, and the storage nodes. The storage nodes store and handle the read and delete operations for BLOBS. Each BLOB is encrypted, with decryption happening in parallel with the read. The backoff nodes handle the reconstruction of data when failures occur in the cell. They are cpu heavy and only rebuild requested BLOBs. Rebuilder nodes reconstruct blocks stored on failed components by using the parity blocks. The reconstructed data can then be migrated. Coordinator nodes handle maintenance tasks such as validating the block layout and rebalancing blocks.