Master Operations

Let's learn the different operations performed by the master.

We'll cover the following

The master executes all namespace operations. Furthermore, it manages chunk replicas throughout the system. It is responsible for:

- Making replica placement decisions
- Creating new chunks and hence replicas
- Making sure that chunks are fully replicated according to the replication factor
- Balancing the load across all the ChunkServers
- Reclaim unused storage

Namespace management and locking#

The master acquires locks over a namespace region to ensure proper serialization and to allow multiple operations at the master. GFS does not

have an i-node like tree structure for directories and files. Instead, it has a

hash-map that maps a filename to its metadata, and reader-writer locks are applied on each node of the hash table for synchronization.

- Each absolute file name or absolute directory name has an associated read-write lock.
- Each master operation acquires a set of locks before it runs.
- To make operation on `/dir1/dir2/leaf`, it first needs the following locks:
  - Reader lock on `/dir1`
  - Reader lock on `/dir1/dir2`
  - Reader or Writer lock on `/dir1/dir2/leaf`
- Following this scheme, concurrent writes on the same leaf are prevented right away. However, at the same time, concurrent modifications in the same directory are allowed.
- File creation does not require write-lock on the parent directory; a read-lock on its name is sufficient to protect the parent directory from deletion, rename, or snapshot.
- Write-lock on a file name stops attempts to create multiple files with the same name.
- Locks are acquired in a consistent order to prevent deadlock:
  - First ordered by level in the namespace tree
  - Lexicographically ordered within the same level

Replica placement#

To ensure maximum data availability and integrity, the master distributes replicas on different racks, so that clients can still read or write in case of a rack failure. As the in and out bandwidth of a rack may be less than the sum of the bandwidths of individual machines, placing the data in various racks can help clients exploit reads from multiple racks. For ‘write’ operations, multiple racks are actually disadvantageous as data has to

travel longer distances. It is an intentional tradeoff that GFS made.

Replica creation and re-replication#

The goals of a master are to place replicas on servers with less-than-average disk utilization, spread replicas across racks, and reduce the number of 'recent' creations on each ChunkServer (even though writes are cheap, they are followed by heavy write traffic) which might create additional load.

Chunks need to be re-replicated as soon as the number of available replicas falls (due to data corruption on a server or a replica being unavailable) below the user-specified replication factor. Instead of re-replicating all of such chunks at once, the master prioritizes re-replication to prevent these cloning operations from becoming bottlenecks. Restrictions are placed on the bandwidth of each server for re-replication so that client requests are not compromised.

How are chunks prioritized for re-replication?

- A chunk is prioritized based on how far it is from its replication goal. For example, a chunk that has lost two replicas will be given priority on a chuck that has lost only one replica.
- GFS prioritizes chunks of live files as opposed to chunks that belong to recently deleted files (more on this when we discuss Garbage Collection (https://www.educative.io/collection/page/5668639101419520/5559029852536832/5335676130689024)). Deleted files are not removed immediately; instead, they are renamed temporarily and garbage-collected after a few days. Replicas of deleted files can exist for a few days as well.

Replica rebalancing#

Master rebalances replicas regularly to achieve load balancing and better

disk space usage. It may move replicas from one ChunkServer to another to bring disk usage in a server closer to the average. Any new ChunkServer added to the cluster is filled up gradually by the master rather than flooding it with a heavy traffic of write operations.

Stale replica detection#

Chunk replicas may become stale if a ChunkServer fails and misses mutations to the chunk while it is down. For each chunk, the master maintains a chunk Version Number to distinguish between up-to-date and stale replicas. The master increments the chunk version every time it grants a lease (more on this later) and informs all up-to-date replicas. The master and these replicas all record the new version number in their persistent state. If the ChunkServer hosting a chunk replica is down during a mutation, the chunk replica will become stale and will have an older version number. The master will detect this when the ChunkServer restarts and reports its set of chunks and their associated version numbers. Master removes stale replicas during regular garbage collection.

Stale replicas are not given to clients when they ask the master for a chunk location, and they are not involved in mutations either. However, because a client caches a chunk's location, it may read from a stale replica before the data is resynced. The impact of this is low due to the fact that most operations to a chunk are append-only. This means that a stale replica usually returns a premature end of a chunk rather than outdated data for a value.

← Back

Metadata

Next →

Anatomy of a Read Operation

Mark as Completed


Report an Issue