Single Master and Large Chunk Size

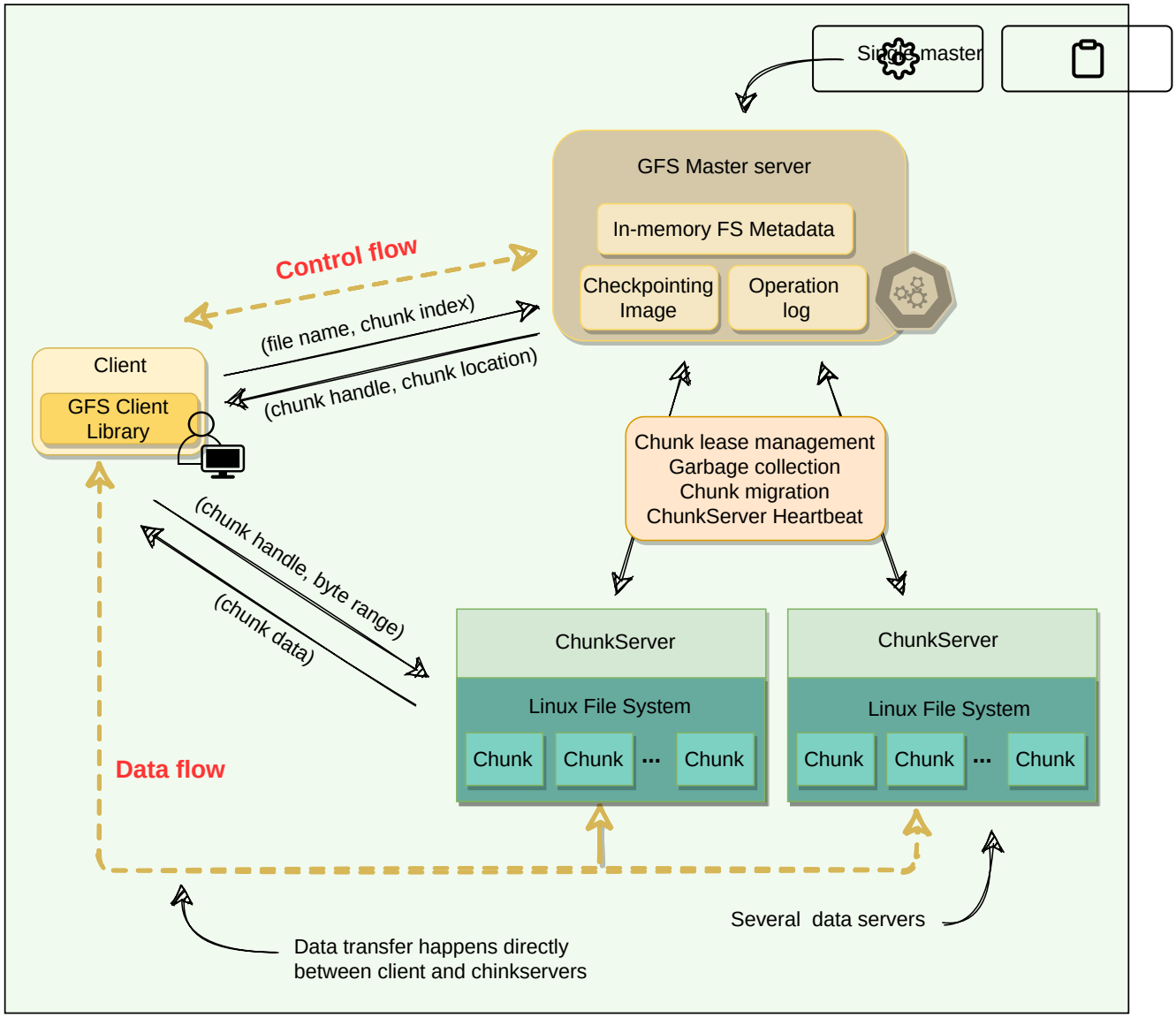
This lesson will explain why GFS has a single master and a large chunk size.

We'll cover the following

- Single master
- Chunk size
  - Lazy space allocation

Single master#

Having a single master vastly simplifies GFS design and enables the master to make sophisticated chunk placement and replication decisions using global knowledge. However, GFS minimizes the master's involvement in reads and writes so that it does not become a bottleneck. Clients never read or write file data through the master. Instead, a client asks the master which ChunkServers it should contact. The client caches this information for a limited time and interacts with the ChunkServers directly for many subsequent operations.



GFS's high-level architecture

Chunk size#

Chunk size is one of the key design parameters. GFS has chosen 64 MB, which is much larger than typical filesystem block sizes (which are often around 4KB). Here are the advantages of using a large chunk size:

1. Since GFS was designed to handle huge files, small chunk sizes would not make much sense, as each file would have a map of a huge number of chunks.
2. As the master holds the metadata and manages file distribution, it is involved whenever chunks are read, modified, or deleted. This also means that a small chunk size would significantly increase the

amount of data a master would need to manage and increase the amount of data that would need to be communicated to a client, resulting in extra network traffic.

3. A large chunk size reduces the size of the metadata stored on the master, which enables the master to keep all the metadata in memory, thus significantly decreasing the latency for control operations.
4. By using a large chunk size, GFS reduces the need for frequent communication with the master to get chunk location information. It becomes feasible for a client to cache all information related to chunk locations of a large file. Client metadata caches have timeouts to reduce the risk of caching stale data.
5. A large chunk size also makes it possible to keep a TCP connection open to a ChunkServer for an extended time, amortizing the time of setting up a TCP connection.
6. A large chunk size simplifies ChunkServer management, i.e., to check which ChunkServers are near capacity or which are overloaded.
7. Large chunk size provides highly efficient sequential reads and appends of large amounts of data.

Lazy space allocation#

Each chunk replica is stored as a plain Linux file on a ChunkServer. GFS does not allocate the whole 64MB of disk space when creating a chunk. Instead, as the client appends data, the ChunkServer lazily extends the chunk. This lazy space allocation avoids wasting space due to internal fragmentation. Internal fragmentation refers to having unused portions of the 64 MB chunk. For example, if we allocate a 64 MB chunk and only fill up 20MB, the remaining space is unused.

One disadvantage of having a large chunk size is the handling of small files. Since a small file will have one or a few chunks, the ChunkServers storing those chunks can become hotspots if a lot of clients access the

same file. To handle this scenario, GFS stores such files with a higher replication factor and also adds a random delay in the start times of the applications accessing these files.

← Back

High-level Architecture

Next →

Metadata

Mark as Completed

Report an Issue