Google File System

Motivation

four major differences from the distributed file system at that time:

* component failures are common (SW + HW)
* files are huge (100ish MB in 2003)
* appends and seq-reads are common, random writes are rare.
* co-designing the applications and FS benefits the overall system

NFS: no failure recovery, files are much smaller, needs to support random write well, stand-alone FS API.

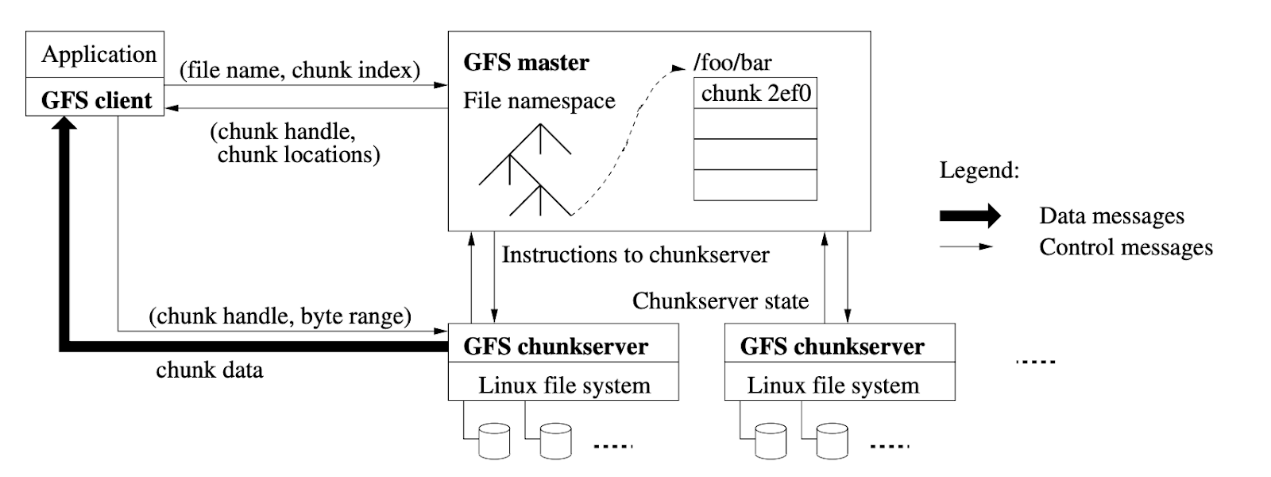
Design overview  --- assumptions

* commodity hardwares often fail
* files are large (multi-GB is common)
* reads are mostly large seq-reads (> 1MB), small random reads will be sorted by clients.
* large, sequential writes (aka append)
* atomic semantics for concurrent append write (example: producer-consumer)
* optimize for bandwidth, as opposed to latency

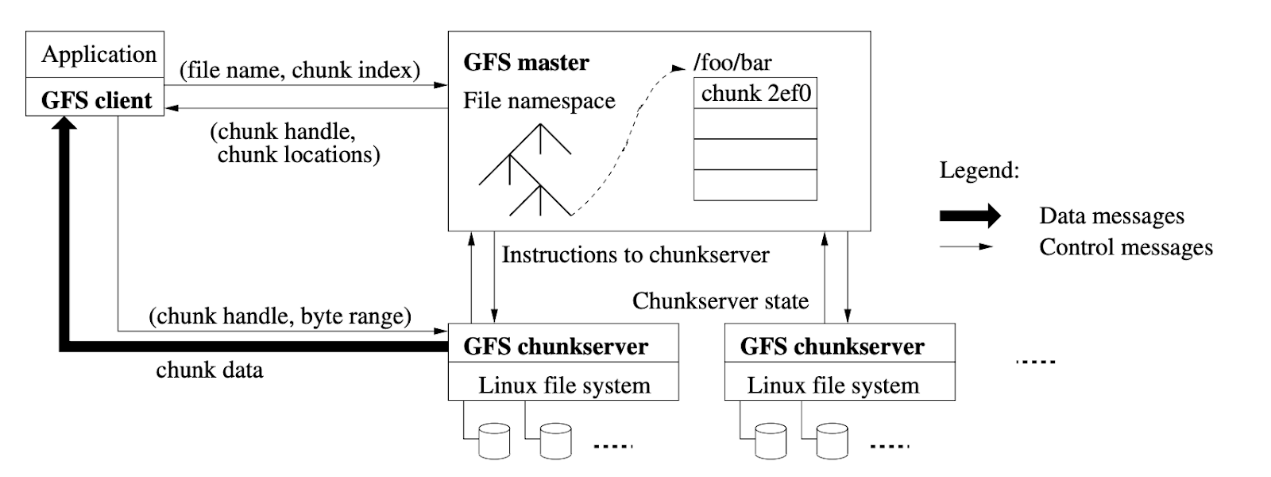
Design overview  --- interface

* create
* delete
* open
* close
* read
* write
* **snapshot: create a copy of a file or a directory tree**
* **record append: concurrent append**

Design Review



* files divided into chunks
* globally unique id for each chunk (chunk handle)
* replication at the chunk level
* master handling metadata (naming, access control etc)
* master handling system-wide activities: lease management, GC, chunk migration, health monitor
* separation of control and data
* no cache: most reads are wide-range scan

Design overview  --- read request overview 

* client (filename, offset) -> (filename, chunk\_id)
* ask master the chunk handle and location of the replicas
* client caches this information
* send request to one replica, supplying chunk handle and offset

no client-master interaction until the cache expires.

Design overview  --- master

single master could easily be a system bottleneck

* must minimize the involvement of the master in read/write
* sepration of control and data panel
* master election is supported by Chubby [Chubby paper OSDI2006]

Design overview  --- Chunk size

GFS chose 64MB, far above the block size of typical FS (in KB)

* reduce interaction between client and master
* reduce the network overhead between client and chunkserver
* reduce the size of metadata

disadvantage: fragmentation

load balancing (hot, small files results in hot spots) with more replicas