A filesystem is a way of storing and organizing computer files and data. It also makes access to this data possible. They may be plugged to a data storage device that could be a Hard Disk, or a cd-room. There are also virtual filesystems, which do not concern themselves on how the data is actually stored, but how to access it.

The data storage device in the google file system are basically cheap components. File systems must keep track of which sectors belong to which file and which are not being used.Most file systems address data in fixed-sized units.. in the google fileystem it is a chunk

The google filesystem is a distributed file system. File systems are distributed then they need protocols to send data from different servers. The distributed file systems do not concern themselves with how the data is actually stored.

Inherent to distributed systems:

Concurrency – Figure out how to do locks, latency in locking or concurrent writes.

How can the system proceed with this failures.

NFS server will show how this files are exposed in the outside world

Caching mechanism to read more quickly.

GFS shares many of the same goals as previous distributed file systems such as

* performance,
* scalability,
* reliability,
* availability.

They are however not focused in servicing users, they let the applications do that, instead they service the google applications.

What Motivated the creation of GFS.

1. Component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines that are built from commodity parts.
2. Files are huge by traditional standards. The reason for this is that when working with files that are information in the order of TBs it’s unmanageable to use KB-sized files. Stores a modest number of large files, Multi-GB files are common, so ways are sought to manage them efficiently. Small files are supported but no need to optimize for them.
3. Most files are mutated by appending new data rather than overwriting.
   1. large, sequential writes that append data to files
   2. Small writes at arbitrary positions in a file are supported but do not have to be efficient.
4. Co-designing the applications and the file-system API benefits the overall system. Primitives can be created specifically for the google apps.
5. High sustained bandwidth is more important than low latency.

Interface.

GFS does not implement a standard API such as POSIX.

They support the usual operations to create delete open close read and write files.

However they add their own operations as well such as snapshot and record append.

Record append allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity of each individual client’s append.

Architecture.

The files are divided into fixed size chunks. Each chunk is identified by an immutable & globally unique 65 bit chunk handle assigned by the master at the time of chunk creations.

Each chunk is stored on local disks as linux files and read or write chunk data specified by a chunk handle and byte range. For reliability the chunks are replicated in multiple chunkservers.

The GFS cluster consists of a single master which holds all the file system metadata, this includes the namespace, access ctrl info, and the mapping from files to chunks & the current locations of chunks.

The master periodically communicates with the chunkservers to obtain their state. This operation happens very quickly.

A GFS client app will implement its API and communicate with the master and the chunkservers to read or write data on behalf of the applications.

Neither clients nor chunkserver caches file data. Caching offers little benefit because most apps stream through huge files or have working sets that are too large to be cached. Good in that they don’t have to worry about. Chunkservers will use the linux’s buffer cache which already keeps the frequently accessed data in memory.



Clients do not read or write data through the master, they do so directly with the chunk servers. The client uses the master to ask which chunkservers it must contact in order to get a file.

The master does not keep a persistent record of which chunkservers have a replica of a given chunk. It simply polls chunkservers for that information at startup.

The operation log contains a historical record of critical metadata changes. It is central to GFS. Not only is it the only persistent record of metadata, but it also serves as a logical time line that defines the order of concurrent operations.

Stale

replicas will never be involved in a mutation or given to clients asking the master for chunk locations. They are garbage collected at the earliest opportunity. Since clients cache chunkl ocations, they may read from a stale replica before that information is refreshed. This window is limited by the cache entry’s timeout and the next open of the file, which purges from the cache all chunki nformation for that file.

Questions?

What’s a chunk handle?