Google File System

(The below post provides a quick overview of the Google File System.)
(Original article at the end of the post is recommended for a comprehensive understanding.)

1. GFS is a distributed file system designed to store terabytes of data with replication and reliability.

2. Components can fail anytime. Some can fail so badly that they cannot be recovered at all.

3. Designed for huge files - multi-GB in size.

4. Update operations are not recommended. Most operations are either read or append.

5. Largest GFS clusters have around 1000 storage nodes with 300 TB of data. Multiple clusters are in use to support different applications.

6. GFS should constantly monitor itself and replace lost components quickly.

7. Atomic appends should be guaranteed but with minimal synchronization overhead.

8. Latency is not of paramount importance as goal is to support batch systems.

9. GFS supports all regular file operations like create, delete, open, close, read, write and file hierarchies.
   It also supports snapshot and atomic appends.

Architecture

1. One master and multiple chunkservers.

2. Files are divided into fixed-size chunks.
   Each chunk is identified by immutable and globally unique 64-bit chunk-ID assigned by master at the time of chunk creation.
   This allows for 2^{64} = $18x10^{18}$unique chunks. If each chunk is one MB also, total

storage space would become 10^9 terabytes.

3. Usually chunksize is 64 MB. Such large chunksize reduces number of entries in the master node.
   Also, clients can comfortably performa several operations in the vicinity of a hot or current file-index without opening too many network connections.

4. Each chunk is replicated (usually 3 times) for reliability. Replication factor is configurable.

5. Master manages all meta-data like:
   a. Namespace
   b. Mapping from files to chunks
   c. Garbage collection of orphaned chunks
   d. Chunk migration between servers
   e. Heartbeat management to all chunkservers for health checking.

6. There is no caching of file chunks - mostly due to enormous size of chunks and very little re-use of same portions of a file.

7. Clients do cache metadata however.

8. Master node does not participate in any read/write of actual file contents.
   It only hands-over replica information to clients who interact directly with replicas.

9. For very frequently accessed files, throughput of queries per second can be increased by adding more replicas.

10. The master stores all the metadata in memory. Large filenames are compressed using prefix compression.
    This helps the master to be very fast which in turn reduces the load on the master when serving queries.

11. Master does not persist chunkserver mapping to files but gathers it by periodically refreshing it from chunkservers.
    This is more efficient and reliable when 100s of nodes are involved.

12. Operation Log (also called T-log (T=Transaction) in some systems) contains a historical record of critical metadata changes.
    Write calls do not succeed unless an entry has been made in the Operation Log.
    Since its so critical, this log is replicated across several machines.
    Master can recover from a crash/shutdown by replaying its state from the T-log.
    To avoid full replay, a checkpoint is also stored such that master does not need to

replay state before that checkpoint.

13. A very simple strategy for efficient checkpointing is to roll over the logfile for T-log when a particular number of records have been processed.

14. Master guarantees consistency by ensuring the order of mutations on all replicas and using chunk version numbers.
    If a replica has incorrect version, it is garbage collected at the earliest.

15. Since clients cache chunk locations, they could read stale data.
    So clients are expected to refresh their cache frequently.

16. Checksums are used to check chunks' validity. If a chunk is corrupted, it is restored from other replicas as soon as possible.

17. GFS guarantees atleast once writes for writers.
    This means that records could be written more than once as well (although rarely).
    It is the responsibility of the readers to deal with these duplicate chunks.
    This is achieved by having checksums and serial numbers in the chunks which help readers to filter and discard duplicate data.

18. Application forwards write chunks to all replicas including the primary node.
    Once all replicas have received the chunks, a write request is sent to the primary node.
    This is done so that primary can assign a serial number to the chunks and send the same to all other replicas.
    This helps in ordering of multiple chunks for same file with multiple writers.

19. If any replica fails, above step is repeated again resulting in the possibility of duplicate writes.
    Hence the guarantee of writing atleast once.

20. Snapshots are created almost instantaneously by following the copy-on-write method.
    This means that when a snapshot request comes, only the metadata is copied.
    Chunk references in the metadata point to older chunks only.
    When a write request comes to older or the newer chunk, only then the copy is actually made.
    Also, the copy is always made locally on the same node to save network traffic.

Master-Node Operations

1. GFS does not have i-node like tree structure for directories and files.
   Rather it has a hash-map that maps a filename to its metadata and read-write locks are applied on each node of the hash table for synchronization.
   In any operation involving /d1/d2/d3/.../dn/leaf, read locks are applied on /d1, /d1/d2/, /d1/d2/d3/, /d1/d2/.../dn/ and read or write lock on the leaf.
   Due to this, concurrent writes on the same leaf are prevented right away.
   Advantage of this system is that it allows concurrent modifications in the same directory.

2. Replicas should be as distributed as possible - for example not being in the same rack for sure.
   This gives better reliability and faster read responses at the cost of slower writes.

3. Stale replicas are detected by version number strategy.
   During any append/write operation, master increments the version number of all nodes participating in the write operation.
   If a node is left out in this process, it will report a lower version number when it comes online.
   At this time, master detects that node as having a stale copy and schedules it for deletion.

Reference:http://research.google.com/archive/gfs.html