GFS

**Modificações concorrentes de arquivos:**

Data mutations may be writes or record appends. A write causes data to be written at an application-specified file offset. A record append causes data (the “record”) to be appended atomically at least once even in the presence of concurrent mutations, but at an offset of GFS’s choosing (Section 3.3). (In contrast, a “regular” append is merely a write at an offset that the client believes to be the current end of file.) The offset is returned to the client and marks the beginning of a defined region that contains the record. In addition, GFS may insert padding or record duplicates in between. They occupy regions considered to be inconsistent and are typically dwarfed by the amount of user data.

**Replicação de arquivos:**

By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace. The master maintains all file system metadata.

**Heterogeneidade:**

**Tolerância a Falhas:**

We address fault tolerance by keeping the master state small and fully replicated on other machines. Scalability and high availability (for reads) are currently provided by our shadow master mechanism. Updates to the master state are made persistent by appending to a write-ahead log. Therefore we could adapt a primary-copy scheme like the one in Harp [7] to provide high availability with stronger consistency guarantees than our current scheme. The River model supports m-to-n distributed queues but lacks the fault tolerance that comes with persistent storage, while GFS only supports m-to-1 queues efficiently. Multiple consumers can read the same file, but they must coordinate to partition the incoming load.

**Consistência:**

**Proteção:**

Other metadata includes file ownership and permissions, mapping from files to chunks, and each chunk’s current version. In addition, for each chunk we store the current replica locations and a reference count for implementing copy-on-write.

File creation does not require a write lock on the parent directory because there is no “directory”, or inode-like, data structure to be protected from modification. The read lockon the name is sufficient to protect the parent directory from deletion.

**Eficiência:**

The efficiency drops from 80% to 75% because as the number of readers increases, so does the probability that multiple readers simultaneously read from the same chunkserver.

Small writes at arbitrary positions in a file are supported but do not have to be efficient.

The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file.