**The Google File System**

The main focus of this paper is on Google File System (**GFS**) and discussed the file system designs, system interactions, master operations, fault tolerance and diagnosis. **GFS** is a scalable, fault-tolerance distributed file system custom-designed to handle the rapidly growing demands of Google’s data processing needs. **GFS** provides fault tolerance while running on commodity hardware and delivers high aggregate performance to large number of clients.

**GFS** development is guided by assumptions.

1. The system is built from many inexpensive commodity components that often fail and this component failure is the norm rather than the exception so it must be constantly monitored.
2. The system stores a modest number of large files and usually Multi-GB files are the common case however small sized files are also supported but need not to optimize for them.
3. The system Workloads consist of large streaming reads and small random reads and the workloads have usually large sequential writes which append data to files.
4. The system must efficiently implement parallelism for clients for read and write processes.
5. High sustained bandwidth is more important than low latency because most of the target applications place a premium on processing data in bulk at a high rate.

**GFS** is on top of local file systems in individual machines to create a meta-file system, where each of the participating machine is called a chunk-server. **GFS** cluster consists of a single master and multiple chunk-servers and is accessed by multiple clients. Files are divided into fixed-size chunks identified by an immutable and globally unique 64 bit chunk handle and this chunk division helps to reduces client master interaction, reduces network overhead and size of meta-data that master stores.

To reduce the overhead of master clients never read and write file data through the master. Client contact the master to get address of appropriate chunk-server it should contact and caches this information for a limited time and interacts with the chunk-server directly for many subsequent operations.

**GFS** has snapshot and record append operations. Snapshot creates a copy of a file or a directory tree at low cost while the Record append allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity of each individual client’s append.

The master communicates with chunk-servers in Heart Beat messages to give it instructions and collect their state. Master checkpoints its state when logs size become very large. In case of failure, master loads the last checkpoint and then replay the subsequent logs. This will help to reduce the startup time.

**GFS** maintain data consistency across all replicas of data. Clients will always see the same data, regardless of which replicas they read from. Clusters are used to compare the results using one cluster designated for research and development by over a hundred engineers and other cluster dedicated for production data processing. Micro-benchmarks are used to determine reads, writes, record appends, storage and meta-data performance.

**Questions:**

1. Each client request have to go through master, isn't it a limiting factor ?
2. Master keep track of chunk-server state. With increase of chunk-servers the overhead for master wouldn't increase to large extent ?
3. **GFS** is based on Linux operating system. Why isn't it Window or Mac-Os based ?
4. What will be performance of read and writes when the number of clients grows ?