[2]. Ghemawat S, Gobioff H, Leung ST. The Google file system. InProceedings of the nineteenth ACM symposium on Operating systems principles 2003 Oct 19 (pp. 29-43).

The problem being addressed?

The work done by authors Ghemawat et al. resulted in the creation of a novel distributed file system, which was specially designed for rising workloads at Google. For this, they used commodity hardware to build a working model, discussed its internal functionality, and explained its features, such as fault tolerance and replication strategies. In addition, they compared and contrasted workload breakdown between the research and development cluster and the production cluster. Finally, they also documented their experiences in building the above product.

Limitations of Existing Work?

As per the authors, existing distributed file systems like MGFS (Minnesota Google File System), GPFS (General Parallel File System), NASD (Network Attached Storage Device), and AFS (Andrew File System) use different design techniques and algorithms to store the data in an optimized way. As a result, every system has its own performance limitations due to these design choices. Therefore, the authors state that a standardized distributed system with high reliability and performance is much needed.

Summary of Proposed Architecture?

The authors proposed the Google File System that contains 3 main components: a) master: a commodity server that stores metadata and files mapping information in it, b) clients: the end user applications that dump data into a distributed file system and c) chunkservers: a component of distributed file system that organizes and stores data in the form of chunks that holds data as 64MB blocks.

Characteristics of each component:

Master:

1) The job of the master is to redirect all the requests from the clients to their respective chunkservers.

2) It chooses a primary chunkserver initiating lease by following the system protocol.

3) The master maintains 2 tables in memory for quick access.

4) One maps the **file names** to an array of chunk ids, while the other monitors the list of chunkservers, version numbers, and lease expiration parameters.

5) A log file is also kept on the master local disk which maintains the checkpoints of all the operations including state management, version tracking, and chunk handle management.

6) Chunk handle management involves assigning a unique 64-bit identification number to every block.

Client:

1) The only task of a client is to generate a file write request and to partition the incoming file into 64 MB chunks when a file needs to be saved on disk.

Chunkservers:

1) The chunkservers do most of work such as communicating with clients after a lease agreement is initiated by the master.

2) Writing the partitioned chunks to its disks as 64 MB blocks. In addition each block is replicated by a standard replication factor of three to ensure high availability of a particular file.

Summary of Internal working and result of evaluation?

The authors proposed that when an update is made to an existing chunk, first the client enquires with the master regarding which chunkserver holds the current lease for the requested chunk.

a) Then the master checks, if the lease is acquired or not.

b) If not it gets one and sends information regarding primary and secondary chunkservers to the client.

c) Thereafter, the client caches this information for future needs and contacts the master again only when the primary chunkserver becomes unreachable.

d) The client pushes the data to its primary and secondary chunkservers, which store data in **LRU** buffer cache until it is utilized or expired.

e) Once all the replicas send an acknowledgement saying that data is received, now the client generates a write signal to the primary chunkserver.

f) Later the primary assigns serial numbers to all the changes it receives, and asks secondary chunkservers to perform a write request in a specified order.

g) Thenafter, once the write operation is performed the secondary server notifies the primary server saying that it has completed the write, and the primary responds to the client, stating it has successfully executed its task.

h) In this way, if there are any write issues at the downstream servers the operation is retried until it is completed.

The authors evaluated the cluster by deploying it at a data center. They collected metrics regarding the real-time usage of the system with 19 servers (which include chunkservers, master, master replica) connected to one switch and 16 clients connected to another switch with a capacity of 1 GBPS. Overall for research and development cluster they proposed that the theoretical and pragmatic efficiency of a read operation, for single and multiple clients are commendable with 80% and 75%. This exemplifies that the read operation supports the cluster to operate effectively at peak loads with single and multiple clients. Similarly, the practical working write operation has almost 50% efficiency in concurrent clients when compared to theoretical a measure due to the chance of high collision while writing data to the chunkservers at the same time. For record appends, for a single client it starts at 6.0 MB/s and drops to 4.8 MB/s for concurrent clients due to congestion and variation of transfer rates by multiple clients in the network.

Any Research Directions Provided?

The authors provided multiple research directions from the standpoint of distributed computing. These include building a fault-tolerant and highly available system, by citing early sources like AFS (Andrew file System), XFS, Intermezzo, and Minnesota GFS. Also, they provided more information regarding the differences between the existing architectures and laid a solid foundation by providing novel algorithms that regards to fault tolerance and handling huge volumes of data reliably.