3.5 Cloud file systems: GFS and HDFS, Comparisons among GFS and HDFS.

- The Google File System (GFS) is designed to manage relatively large files using a very large distributed cluster of commodity servers connected by a high-speed network.
- It is therefore designed to

  (a) expect and tolerate hardware failures, even during the reading or writing of an individual file (since files are expected to be very large) and

  (b) support parallel reads, writes and appends by multiple client programs. A common use case that is efficiently supported is that of many 'producers' appending to the same file in parallel, which is also being simultaneously read by many parallel 'consumers'.

- As a result they also do not scale as well as data organizations built on GFS-like platforms such as the Google Datastore.
- The Hadoop Dis- tributed File System (HDFS) is an open source implementation of the GFS architecture that is also available on the Amazon EC2 cloud platform; we refer to both GFS and HDFS as 'cloud file systems.'
- The architecture of cloud file systems is illustrated in Figure.

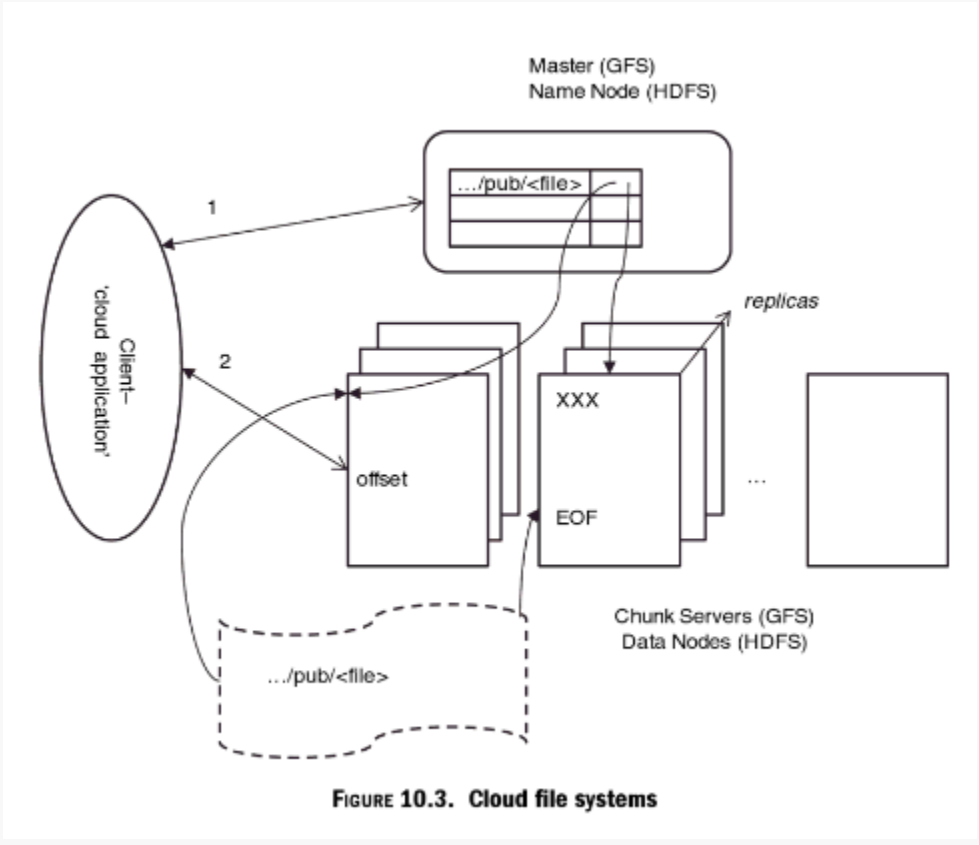

FIGURE 10.3. Cloud file systems

- Large files are broken up into 'chunks' (GFS) or 'blocks' (HDFS), which are themselves large (64MB being typical). These chunks are stored on commodity (Linux) servers called Chunk Servers (GFS) or Data Nodes (HDFS); further each chunk is replicated at least three times, both on a different physical rack as well as a different network segment in anticipation of possible failures of these components apart from server failures.
- When a client program ('Cloud Application') needs to read/write a file. It sends the full path and offer to the Master(GFS) which sends back meta-data for one (in the case of read) or all (in the case of write) of the replicas of the chunk where this data is to be found.

- The client caches such meta-data so that it need not contact the Master each time. Thereafter the client directly reads data from the designated chunk server; this data is not cached since most reads are large and caching would only complicate writes.
- In case of a write, in particular an append, the client sends only the data to be appended to all the chunk servers; when they all acknowledge receiving this data it informs a designated 'primary' chunk server, whose identity it receives (and also caches) from the Master.
- The primary chunk server appends its copy of data into the chunk at an offset of its choice; note that this may be beyond the EOF to account for multiple writers who may be appending to this file simultaneously. The primary then forwards the request to all other replicas which in turn write the data at the same offset if possible or return a failure. In case of a failure the primary rewrites the data at possibly another offset and retries the process.
- The Master maintains regular contact with each chunk server through heartbeat messages and in case it detects a failure its meta-data is updated to reflect this, and if required assigns a new primary for the chunks being served by a failed chunk server. Since clients cache meta-data, occasionally they will try to connect to failed chunk servers, in which case they update their meta-data from the master and retry.
- It is shown that this architecture efficiently supports multiple parallel readers and writers. It also supports writing (appending) and reading the same file by parallel sets of writers and readers while maintaining a consistent view, i.e. each reader always sees the same data regardless of the replica it happens to read from.
- Finally, note that computational processes (the 'client' applications above) run on the same set of servers that files are stored on. As a result, distributed programming systems, such as MapReduce, can often schedule tasks so that their data is found locally as far as possible, as illustrated by the Cluster system.

GFS Architecture:

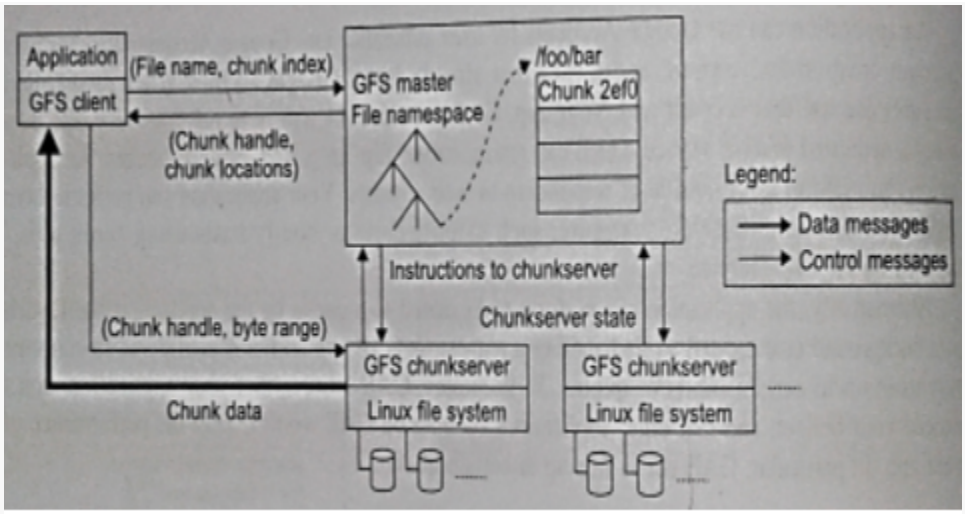

Fig. Architecture of Google File System (GFS)

1. There is single master in the whole cluster which stores metadata.
2. Other nodes act as the chunk servers for storing data.
3. The file system namespace and locking facilities are managed by master.
4. The master periodically communicates with the chunk servers to collect management information and give instruction to chunk servers to do work such as load balancing or fail recovery.
5. With a single master, many complicated distributed algorithms can be avoided and the design of the system can be simplified.
6. The single GFS master could be the performance bottleneck and single point of failure.
7. To mitigate this, Google uses a shadow master to replicate all the data on the master and the design guarantees that all data operations are transferred between the master and the clients and they can be cached for future use.

8. With the current quality of commodity servers, the single master can handle a cluster more than 1000 nodes.

The features of Google file system are as follows:

1. GFS was designed for high fault tolerance.
2. Master and chunk servers can be restarted in a few seconds and with such a fast recovery capability, the window of time in which data is unavailable can be greatly reduced.
3. Each chunk is replicated at least three places and can tolerate at least two data crashes for a single chunk of data.
4. The shadow master handles the failure of the GFS master.
5. For data integrity, GFS makes checksums on every 64KB block in each chunk.
6. GFS can achieve the goals of high availability, high performance and implementation.
7. It demonstrates how to support large scale processing workloads on commodity hardware designed to tolerate frequent component failures optimized for huge files that are mostly appended and read.

HDFS Architecture:

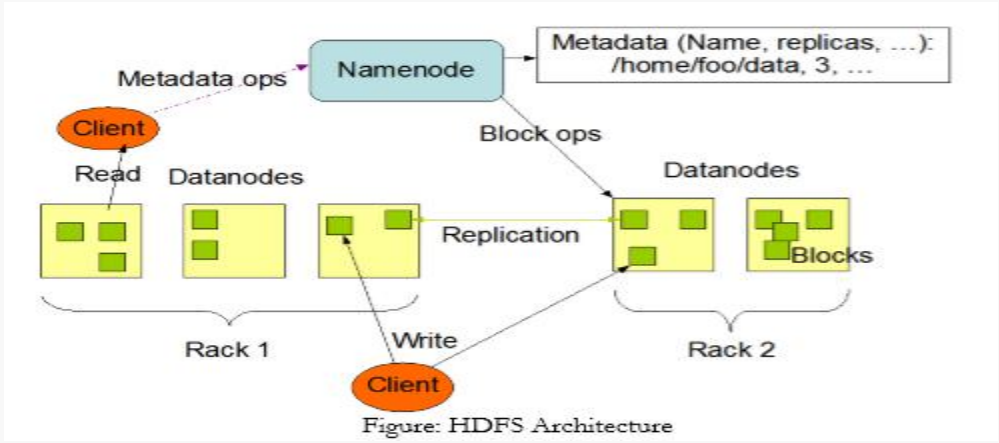

Figure: HDFS Architecture

1. The Hadoop Distributed File System (HDFS) is designed to provide a fault-tolerant file system designed to run on commodity hardware. The primary objective of HDFS is to store data reliably even in the presence of failures including Name Node failures, Data Node failures and network partitions.
2. HDFS uses a master/slave architecture in which one device (the master) controls one or more other devices (the slaves). The HDFS cluster consists of a single Name Node and a master server manages the file system namespace and regulates access to files.
3. **NameNode and DataNodes:** The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.
4. **The File System Namespace:** The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas.The NameNode maintains the file system namespace.
5. **Data Replication:**HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
6. Block:Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Features of HDFS

1. Manages the files system manespare. Regulation clients access to files, It also execute file system operations such as naming, and opening files and directories data node.
2. Fault detection and recovery : Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanism for quick and automatic fault detection and recovery.
3. Huge data sets : HDFS should have hundreds of nodes per cluster to move the applications having huge data sets.
4. Hardware at data : A requested tasks can be done efficiently, when the computation takes place near the data. Especially where huge data base are involved it reduces the network traffic and increase the through PD.

Comparisons among GFS and HDFS:

Key Point	HDFS Framework	GFS Framework
Objective	Main objective of HDFS to handle the Big-Data	Main objective of HDFS to handle the Big-Data
Language used to Develop	Java Language	C, CPP Language
Implemented by	Open source community, Yahoo, Facebook, IBM	Google
Platform	Work on Cross-platform	Work on Linux
License by	Apache	Proprietary or design by google for its own used.
Files Management	HDFS supports a traditional hierarchical directories data structure [9].	GFS supports a hierarchical directories data structure and access by path names [9].
Types of Nodes used	NameNode and DataNode	Chunk-server and MasterNode
Hardware used	Commodity Hardware or Server	Commodity Hardware or Server
Append Opration	Only supports append operation	supports append operation and we can also append base on offset.
Database Files	Hbase	Bigtable is the database
Delete Opration and Garbage Collection	First, deleted files are renamed and store in particular folder then finally remove using garbage collection method.	GFS has unique garbage collection method in which we cannot reclaiminstantly. It will rename the namespace It will delete after the 3 days during the second scaned.
Default size	HDFS has by default DataNode size 128 MB but it can be change by the user	GFS has by default chunk size 64 MB but it can be change by the user
Snapshots	HDFS allowed upto 65536 snapshots for each directory in HDFS 2.	In GFS Each directories and files can be snapshotted.
Meta-Data	Meta-Data information managed by NameNode.	Meta-Data information managed by MasterNode.
Data Integrity	Data Integrity maintain in between NameNode and DataNode.	Data Integrity maintain in between MasterNode and Chunk-Server.
Replication	There are two time replicas created by default in GFS [10].	There are three time replicas created by default in GFS [10].
Communication	Pipelining is used to data transfer over the TCP protocol.	RPC based protocol used on top of TCP\IP.
Cache management	HDFS provide the distributed cache facility using Mapreduse framework	GFS does not provide the cache facility