**Google File System (GFS)**

Google File System (GFS) shares many of the goals such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. Constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system. Each file typically contains many application objects such as web documents. The files are only read, and often only sequentially. A variety of data share these characteristics. Some may constitute large repositories that data analysis programs scan. The system is built from many inexpensive commodity components that often fail. It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis. A GFS cluster consists of a single master and multiple chunkservers and is accessed by multiple clients. The Google File System demonstrates the qualities essential for supporting large-scale data processing workloads on commodity hardware. While some design decisions are specific to our unique setting, many may apply to data processing tasks of a similar magnitude and cost consciousness.

**MapReduce: Simplified Data Processing on Large Clusters**

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. The input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues. The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user’s reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

**Bigtable: A Distributed Storage System for Structured Data**

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Bigtable has successfully provided a flexible, high-performance solution for all of these Google products

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes. Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamp. Bigtable timestamps are 64-bit integers. They can be assigned by Bigtable, in which case they represent “real time” in microseconds, or be explicitly assigned by client applications. The Bigtable API provides functions for creating and deleting tables and column families. It also provides functions for changing cluster, table, and column family metadata, such as access control rights. Bigtable is built on several other pieces of Google infrastructure. Bigtable uses the distributed Google File System (GFS). The Google SSTable file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings.

**The Chubby lock service for loosely-coupled distributed systems**

Chubby provides an interface much like a distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance. Many instances of the service have been used for over a year, with several of them each handling a few tens of thousands of clients concurrently. Chubby has two main components that communicate via RPC: a server, and a library that client applications link against. All communication between Chubby clients and the servers is mediated by the client library. A Chubby cell consists of a small set of servers (typically five) known as replicas, placed so as to reduce the likelihood of correlated failure (for example, in different racks). The replicas use a distributed consensus protocol to elect a master; the master must obtain votes from a majority of the replicas, plus promises that those replicas will not elect a different master for an interval of a few seconds known as the master lease. The master lease is periodically renewed by the replicas provided the master continues to win a majority of the vote. Chubby exports a file system interface similar to, but simpler than that of UNIX. Each Chubby file and directory can act as a reader-writer lock: either one client handle may hold the lock in exclusive (writer) mode, or any number of client handles may hold the lock in shared (reader) mode. Clients see a Chubby handle as a pointer to an opaque structure that supports various operations.