# Reading Summary – Assignment 3

* Reading 1: The Google File System

Google File System is a distributed file system developed by Google to provide access to data using commodity hardware. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients. Google File System was designed to meet growing demands of Google’s data processing needs.

GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of Google’s application workloads and technological environment that reflect a marked departure from some earlier file system design assumptions. First, component failures are the norm rather than the exception. Second, files are huge by traditional standards. Multi-GB files are common. Third, most files are mutated by appending new data rather than overwriting existing data.

A GFS cluster consists of a single master and multiple chunkservers and is accessed by multiple clients. Files are divided into fixed-size chunks. Each chunk is identified by an immutable and globally unique 64-bit chunk handle assigned by the master at the time of chunk creation. Chunk servers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers.

* Reading 2: 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 major contributions of MapReduce are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Map function takes an input pair and produces a set of intermediate key/value pairs. The Reduce function accepts an intermediate key I and a set of values for that key. The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function. The number of partitions (R) and the partitioning function are specified by the user. The master keeps several data structures. For each map task and reduce task, it stores the state (idle, in-progress, or completed), and the identity of the worker machine (for non-idle tasks).

* Reading 3: 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, often petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

Bigtable provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp. The row keys in a table are arbitrary strings (currently up to 64KB in size). Every read or write of data under a single row key is atomic. Bigtable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing. Column keys are grouped into sets called column families, which form the basic unit of access control. All data stored in a column family is usually of the same type. 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.

* Reading 4: The Chubby lock service for loosely-coupled distributed systems

The Chubby lock service allows its clients to synchronize activities and to agree on basic information about their environment. It is intended to provide coarse-grained locking as well as reliable storage for a loosely-coupled distributed system. The primary goals included reliability, availability to a moderately large set of clients, and easy-to-understand semantics.

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. There is an optional third component which is a proxy server.

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. 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.

Clients find the master by sending master location requests to the replicas listed in the DNS. Non-master replicas respond to such requests by returning the identity of the master. Once a client has located the master, the client directs all requests to it either until it ceases to respond, or until it indicates that it is no longer the master. Write requests are propagated via the consensus protocol to all replicas; such requests are acknowledged when the write has reached a majority of the replicas in the cell. Read requests are satisfied by the master alone; this is safe provided the master lease has not expired, as no other master can possibly exist. If a master fails, the other replicas run the election protocol when their master leases expire; a new master will typically be elected in a few seconds.