**Bigtable Summary**

Bigtable is a distributed storage system for managing data structure that is designed to scale to a very large size. Bigtable can scale to perabytes of data and thousands of machines. It has achieved several goals like wide applicability, scalability, high performance and high availability. Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by row key, column key and a timestamp. Each entry in the map is an uninterpreted array of bytes.

The row keys in the table are arbitrary string where read or write of data under a single row is atomic. Bigtable maintain data in lexicographic order by row key. Each row range of a table is called a *tablet*, which is the unit for load balancing and distribution. Reads of short row range are efficient. Column keys are grouped into sets called *column families*.Similar data is stored in a column family. Column key is named using the syntax: *family:qualifier*. Access control and disk and memory accounting are performed at the column-family level. Cells in Bigtable can contain multiple versions of the same data; these versions are indexed by a 64-bit integer *timestamp*. Applications must generate unique timestamps to avoid collisions.

The Bigtable API provides functions to create and delete tables and column families. It also provides functions to change cluster, table, and column family metadata, such as access control rights. Bigtable support CRUD operations on single-row transactions. It also allows cells to be used as integer counters. And finally it supports execution of client-supplied scripts in the address space of the server. Bigtable uses Google File System to store log and data files. It relies on the distributed lock service *Chubby.*

Bigtable implementation has three components: a library that is linked into every client, one master server, and many tablet servers. The Master keeps track of the tablet servers and the tablets assigned to it. A *locality group* is created with multiple column families. A user-specified compression is applied to each SSTable block. Tablet server have two level of caching to enhance performance.

Google Analytics makes use of Bigtable by storing two tables; the raw click table and the summary table. Google Earth uses one table to preprocess data, and a different set of table for serving client data. Personalized search stores each user’s data in a Bigtable.

**Chubby Lock Service Summary**

Chubby is a lock service that provides an interface like the distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance. Chubby is intended to be used within a loosely-coupled distributed system. Its purpose is to allow its clients to synchronize their activities. The Google File System and Bigtable both use Chubby. A lock server makes it easier to maintain existing program structure and communication pattern. Chubby also serves as a name server.

Chubby is a coarse-grained lock service that imposes far less load on the lock server. The locks are acquired rarely. and such locks allow many clients to be adequately served by a modest number of lock servers with somewhat lower availability. Chubby has two major components: a server and a client library which communicate via RPC. The Chubby cells consist of small set of servers known as *replicas*. The replicas use a distributed consensus protocol to elect a *master*; replicas will not elect a different master for an interval of a few seconds known as the *master lease*. Master initiates reads and writes to the database that is shared across all replicas.

Chubby export a file system interface similar to UNIX. The Chubby name space contains files and directories called *nodes.* Each node acts as a read-write lock. Chubby provides a means by which *sequence* numbers can be introduced into only those interactions that make use of locks. Chubby clients can subscribe to events such as file content modified, child node added, removed or modified, Chubby master failed over, a handle has become invalid, lock acquired, conflicting lock requested from another client.

Clients see a Chubby handle as a pointer to an opaque structure that supports various operations. Handles are created by Open() and destroyed by Close(). There are other operations like Poison(), GetContentsAndStat(), GetStat(), ReadDir(), SetContents(), SetACL(), Delete), Acquire() etc. To reduce read traffic, Chubby clients cache file data and node meta-data. A Chubby session is a relationship between a Chubby cell and a Chubby client; it exists for some interval of time, and is maintained by periodic handshakes called KeepAlives. Chubby uses the replicated version of Berkeley DB as its database.

Chubby has become Google’s internal name service. It is used with MapReduce, Bigtables and Google File Systems as a standard repository for files that require high availability, such as access control lists.

**Google File Systems Summary**

Google File System is a scalable distributed file system for large distributed data-intensive systems. GFS provides *snapshot* and *record append* operations. Snapshot creates a copy of a file or a directory tree at low cost. Record append allows multiple clients to append data to the same file concurrently. A GFS cluster consists of a single master and multiple chunkservers which is accessed by multiple clients. The system metadata is maintained at the master. GFS has a chunk size of 64MB.

The master stores three major types of metadata: the file and chunk namespaces, the mapping from files to chunks, and the locations of each chunk’s replicas. All metadata is kept in the master’s memory. The Operation Log is central to the GFS which provides the historical record of the critical metadata changes information. GFS minimizes the involvement of the master. Mutation changes the content or metadata of a chunk, they are performed at all the chunk’s replicas.

Data flow in GFS is pushed linearly along a carefully picked chain of chunkservers in a pipelined fashion. A *record append* is an atomic append operation. Client specifies on the data and using the record append, GFS appends it to the file at an offset. Snapshot operation makes a copy of the file or directory instantaneously. It can be used to create branch copies of huge datasets.

The master executes all namespace operations. In addition, it manages chunk replicas throughout the system. GFS maintains high availability by using *fast recovery* and *replication* techniques. GFS uses checksumming at each chunkserver to detect corruption of stored data hence maintaining data integrity. GFS provides diagnostic logs that maintain events such as chunkserver going up or down and all RPC requests and replies.

GFS meets storage needs at Google and is widely used the storage platform for research and development as well as production data processing.

**Map Reduce Summary**

MapReduce is a programming model for processing large datasets. It makes use of *a key/value pair* and applies a *reduce* function. MapReduce jobs run terabytes of data on thousands of machines on Google’s cluster every day. The *Map* function takes input pairs and creates the intermediate key/value pairs. The *Reduce* function uses these intermediate key/value pairs and merges them to a smaller set of values. The Map invocations are distributed over multiple machines, whereas the Reduce invocations are distributed using a partitioning function *hash(key)* ***mod*** *R.* The following sequence is followed when a MapReduce function in invoked:

1. Map Reduce library splits the input file into M pieces
2. Master program assigns the other worker programs the Map and Reduce tasks
3. The worker with Map task processes the key/value pair and passes it to the user defined Map function
4. The buffered pairs are written to the local disk
5. The Reduce worker reads this buffered data and sorts it by intermediate keys
6. The output of the Reduce function is written to the final output file.
7. MapReduce call is returned back to the user code

The Master maintains several *data structures*. It also provides *Fault Tolerance* in case of a worker failure, master failure, semantics in the presence of failure. MapReduce uses no network bandwidth as input data is read locally. The partitioning function results in a well-balanced partition. The intermediate key/value pairs are processes in guaranteed increasing order. MapReduce can skip unwanted records which could cause failures. MapReduce helps local execution and easier debugging.

MapReduce program written in 2003, has been used successfully in large-scale machine learning problems, Google News, large-scale graph computations etc. One of the most significant use of MapReduce is the complete rewrite of the production indexing system that produces the data structures used for the Google web search service. With MapReduce the indexing code is simpler, smaller and easier. The performance of MapReduce library is good as distribution and parallelization is hidden within it.

The MapReduce model is easy to use and a large variety of problems are easily expressible as MapReduce computations.