The note of *The Google File System* :

Google File Storage System (GFS) is a large distributed system built on cheap servers. It treats server failure as a normal phenomenon, and automatically faults through software, which greatly reduces system costs while ensuring system availability and reliability.

GFS is the cornerstone of Google's entire distributed system. Other storage systems such as Google BigTable and Google Megastore are directly or indirectly built on GFS. In addition, Google's large-scale batch processing system MapReduce also uses the GFS system as the input and output of massive data.

1. Single Master node:

How to avoid a single Master node from becoming a system performance bottleneck:

① The client only requests metadata information from the Master, and does not read and write chunk data through the Master node. ChunkServer is directly responsible for the specific data read and write operations.

② When the client requests metadata of a certain chunk from the Master, the Master will return the information of several chunks immediately after the chunk at a time, effectively reducing the number of requests made by the client.

2. Chunk size:

Choose 64MB as the chunk size, which is much larger than the block size of a general system. This is determined by the specific business characteristics: that is, large files are usually operated.

3. Master metadata:

Master monitors the state of ChunkServer with heartbeat information. Among them, the copy location information of the chunk is polled (heartbeat information) ChunkServer when the master is started, and then periodically polled to obtain the copy location information of the chunk, instead of directly persisting the information in the master. This simplifies the data synchronization problem between the Master and ChunkServer.

The note of *Bigtable: A Distributed Storage System for Structured Data* :

The core data model of Bigtable is a sparse multi-dimensional Map data structure, indexed by (row-row, column-column, timestamp-timestamp), and stores user data that is not transparent to Bigtable in each indexed unit.

Row: Logically, Bigtable data is stored in row units, sorted by Row Key as index. There is no limit to the number of units that can be stored in a row. Physically, it is also stored in row key order, so basically user data tables design should make full use of this, and try to store related data with the same or similar Row Key.

In addition, Bigtable data is also divided into data segments (tablets) in the range of rows. Each tablet can be managed by a different TabletServer, so that the data in different segments is served by different Server nodes, which can ensure the data throughput rate (by comparing Reasonable design of Row Key to distribute hot data and balance load).

Columns: The columns of Bigtable are grouped by Family and presented in the format of Family:qualifier. On the one hand, the purpose of grouping is to control permissions in the unit of Family, and more importantly, to physically group and store different related but different types of data to facilitate better compression. Taking into account the sparseness of the data and the uncertain schema, the number of specific columns is unlimited, and there is no need to declare in advance. But which families must be determined in advance.

Timestamp: Timestamp is used to save the data of different versions of Cell. You can use the time stamp of the system or any series of data designated by the user to identify the version number. The megastore built on the Big Table is used timestamp to implement ACID based on MVCC.

The note of *Bigtable: MapReduce: Simplified Data Processing on Large Clusters* :

MapReduce is a programming model and its related implementation for processing and generating large data sets. Its operation can be briefly summarized as the following steps: a large input is divided into many small input blocks, and at the same time, multiple computers in a distributed system form a large computer cluster. These small inputs are separated The blocks will be executed by the computers in the cluster, and a master machine will distribute the characters. These computing machines become workers. The master allocates this small input block to the worker, and then the worker performs calculations. We call this calculation process Map. Map will process the input data and generate a set of results. The entire Map process will produce a lot of calculation results, we call these calculation results the intermediate value (intermediate value). Next, these intermediate values ​​will be allocated to the workers to perform the Reduce task, and this process is also allocated by the master. Regarding the Reduce task, its main goal is to merge the intermediate value pairs with associated intermediate values ​​(key/value pairs) and calculate the result.

The above process is the basic step of a MapReduce task. Of course, the Map function and Reduce function of the process can be specified by the user. The program code designed under the MapReduce architecture will automatically run in parallel and run in a distributed cluster. The next thing we need to care about is: the division of input data, the scheduling of machines in program operation, the processing errors of machine calculations, and the management of data exchange between machines. In this note, record and think about the methods of dealing with these problems in the paper.