**Bigtable Review**

**18301167 潘岳**

Bigtable is a distributed storage system designed to manage large-scale structured data, which can be extended to PB-level data and thousands of servers. Many Google projects use Bigtable to store data, and these applications present different challenges to Bigtable, such as data size requirements, latency requirements. Bigtable meets these changing requirements and successfully provides flexible, high-performance storage solutions for these products.

Bigtable looks like a database and uses a number of database implementation strategies. Bigtable, however, does not support a full relational data model; rather, it provides a simple data model for the client that dynamically controls the layout and format of the data and takes advantage of the locality of the underlying data store. Bigtable treats data as meaningless strings of bytes, and the client needs to serialize the struct and Unstructured data into Bigtable.

Essentially, Bigtable is a key-value map. Bigtable, according to the authors, is a sparse, distributed, persistent, multidimensional sort map.

Let’s look at dimensions, sorting, mapping. BIGTABLE HAS THREE-DIMENSIONAL KEYS: Row Key, column key, and timestamp. The row key, column key, and timestamp are both byte strings; the timestamp is a 64-bit Integer; the value is a byte string.

The row key can be any byte string, usually 10-100 bytes. Lines are read and written atomically. Bigtable stores data in the Dictionary Order of the row keys. The table of Bigtable is loaded as a unit on a tablet, according to the row key. Initially, there was only a tablet for each table, but as the tables grew, the tablet automatically split and kept the size between 100 and 200 mb. A row is the first level index of a table, and we can reduce the columns, times, and values of that row to a one-dimensional key-value map.

A column is a second-level index, and each row has an unlimited number of columns that can be added or subtracted at any time. For ease of management, columns are divided into families of columns, which are the unit of access control. Columns in a column family generally store the same type of data. The column family of a row is rarely changed, but columns in the column family can be added or deleted at will.

The timestamp is a third-level index. Bigtable allows you to store multiple versions of your data, and versions are distinguished by timestamps. The timestamp can be assigned by Bigtable to indicate the exact time the data entered Bigtable, or by the client. Different versions of the data are stored in descending chronological order, so the latest version of the data is read first. When we added the TIMESTAMP, we got the full data model of Bigtable. When querying, if only the row and column are given, the most recent version of the data is returned; if the row and column timestamp is given, the data that is less than or equal to the timestamp is returned. For example, we look up “AAAAA”/“A: Foo” and return “y” ; we look up “aaaaa”/“A: Foo”/10 And Return “m” ; we look up “aaaaa”/“A: foo/2 and return nothing.

Bigtable relies on several technologies from Google. Store logs and data files with GFS; store data in SSTable File Format; manage Metadata with Chubby.

GFS consists of a master node (Metadata server) , multiple chunkservers (data servers) , and multiple clients (clients running various applications) . In low reliability scenarios, client and Chunkserver can be located at one node. Figure 1 is a diagram of the architecture of GFS, where each node is a regular Linux server, and GFS works by coordinating hundreds of servers to serve various applications.

SSTABLE, which stands for Sorted Strings Table, is an UNMODIFIABLE, ordered key-value mapping that provides querying, traversing, and more. Each SSTable consists of a series of blocks, which Bigtable defaults to 64KB. A block index is stored at the end of a SSTable, and when a SSTable is accessed, the entire index is read into memory. Each tablet is stored in SSTable format in GFS, and each tablet may correspond to multiple SSTable.

Chubby is a highly available distributed lock service. Chubby has five active replicas, while only one master replicas provide service. These replicas are consistent with Paxos algorithm. Chubby provides a namespace (including some directories and files) , each directory and file is a lock, and Chubby’s client has to keep a session with Chubby, losing all the locks if the client session expires.

About the Bigtable cluster: The Bigtable cluster consists of three main parts: a library for the client, a master server, and a number of tablet servers. Bigtable sharpens the table, and the tablet stays in the 100-200 MB range, breaking up into smaller slices or merging into larger ones. Each slice server is responsible for a certain number of slices, processing read and write requests to its slices, as well as splitting or merging slices. The slice server can be added and removed at any time depending on the load. Here the slice server does not actually store the data, but is equivalent to a connection Bigtable and GFS proxy, some of the client’s data operations through the slice server proxy indirect access GFS.

The master server is responsible for allocating slices to slices, monitoring the addition and removal of slices, balancing the load of slices, and processing the creation of tables and columns. Note that the primary server does not store any slices, does not provide any data services, and does not provide location information for the slices.

The client needs to read and write data, direct contact with the chip server. Because the client does not need to get the location information of the chip from the master server, most clients never need to access the master server, which is generally very light.

So how does the client access the tablet? Bigtable uses a data structure similar to a B + Tree to store slice location information. First up is the first floor, Chubby file. On this level is a Chubby file that holds the location of the root tablet. The Chubby file is part of the Chubby Service, and if Chubby isn’t available, that means the root tablet’s location is lost and Bigtable isn’t available.

The second floor is the root tablet. The root tablet is actually the first Shard of the METADATA table, which holds the location of the other slices of the METADATA table. The root tablet is special in that it never splits to keep the depth of the tree constant.

The third layer is the other metadata slices that form the complete metadata table with the root tablet. Each metadata slice contains the location information of many user slices.

The slice data is eventually written to GFS, where the physical form of the slice is a number of SSTable files. When a write request is received by the slice server, the slice server first checks that the request is valid. If it is valid, the write request is first submitted to the log, and then the data is written to memtable in memory. Memtable is the equivalent of SSTable cache. When memtable grows to a certain size, it is frozen. Bigtable then creates a new memtable and converts the frozen memtable to SSTable format and writes it to GFS. This operation is called minor compaction.

When the slice server receives a read request, it also checks whether the request is valid. If it is valid, the read operation looks at the merge view of all SSTable files and memtable, since both SSTable and memtable are themselves sorted, so the merge is fairly fast.

Every time minor compaction creates a new SSTable file, too many SSTable files become less efficient to read, so Bigtable periodically performs merging compaction operations, merging several SSTable and memtable into a new SSTable.

The above is what I regard as the key knowledge points of Bigtable.