BigTable

Prakhar Srivastav (ps2894)

Summary

BigTable is a distributed non-relational storage system built at Google to help them store, process and query massive amounts of data.

Goals The goals of the project were manifold but the most important were -

- Want asynchronous processes to be continuously updating different pieces of data.
- Need to support very high read / write writes (order of millions per second).
- Additionally, it should support time-series analysis of historical data.
- Efficient scans over all interesting subsets of data which could support efficient joins of large one-to-one and one-to-many datasets.

Although BigTable is built from scratch, it relies on many building blocks that were built within Google on which it rests. The Google File System provides raw storage. A scheduler is used that is responsible for scheduling jobs in BigTable serving. A lock service (Chubby) is used that acts a coordination service, does master election and location bootstrapping. Chubby is also used for configuration storage and service discovery. Lastly, MapReduce is used for reading and writing BigTable data.

Data Model

Data is not stored relationally but rather as a multi-dimensional map where each row and column intersect to form a cell. In each cell, then can have multiple values each of which is mapped to one time stamp. Hence when one cell is updated it is added and the old data is not deleted. Rows are ordered lexicographically and there is transaction support at the row level. A group of rows that are stored together are called **Tablets**. Typically ~100MB to 200MB of data is stored per tablet.

Each machine generally stores around 100 tablets. Having a high number of tablets in each machine ensures easy load balancing and quick recovery in terms of failures. Lastly, in line with the design goals every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row). Apart from rows, another integral parts of the data model are **Column families**. Column families form the basic unit of access control and both disk / memory accounting is performed at the column-family level.

System Structure

Bigtable master that control of metadata and does load-balancing. The tablet servers which are responsible for reads / writes on the data they store. The master is responsible for assigning tablets to tablet servers, detecting the addition and expiration of tablet servers, balancing tablet-server load, and garbage collection of files in GFS. In addition, it handles schema changes such as table and column family creations.

These services sit on top of GFS, Chubby and a scheduling system. The client library connects to the lock service and when the lock is acquired, interacts directly with the tablet servers. For locating tablets, 3 levels

of indirection are required. The first level, called the `META0` table is stored in a tablet - the pointer to which is stored in the lock service. Each row of the `META0` tablet points to a `META1` tablet. Finally,the location of the actual tablet is stored in the `META1` tablet.

Tablet Representation

As soon as a tablet gets a write from a client, the first thing it does it writes it to a append-only log on the GFS. It is important to note that client data does not move through the mas- ter: clients communicate directly with tablet servers for reads and writes. On the disk however, the data is stored in structures called **SSTable** which is just a ordered immutable collection of map from string `->` string.

Reads Reads are served by first reading the data from the `METADATA` table which contains the list of SSTables that comprise a tablet and a set of a redo pointsa - pointers into any commit logs that may contain data for the tablet. The server reads the indices of the SSTables into memory and reconstructs the memtable by applying all of the updates that have committed since the redo points.

Writes First the authorization of the client is checked and then a valid mutation is written to the commit log. Then the contents are written into the write buffer (*MEMTABLE*). When the log entry has been commited, the date then gets written on the write buffer (`MEMTABLE`) in the memory. This is not yet commited to disk.

Other Refinements

There were quite a few other refinements that were added in the system that helped the system more robust and meet the goals that the system was initially set out to meet.

Compaction is the process of freeing up memory in the tablet when the write buffer fills up. This involves either dumping the contents of the buffer to disk (as a SSTable) or merging several SSTables into one and then storing it on GFS. It helps in shrinking the memory usage of the tablet server, and it reduces the amount of data that has to be read from the commit log during recovery if this server dies. Apart from this, there's major compaction and merging compaction which differ in the number of SSTables they compact and the output they produce.

Shared logs helps doing away with managing multiple log files for each SSTable and enables mutations to be appended to a single commit log per tablet server, co-mingling mutations for different tablets in the same physical log. This helps reduce disk space and keeps the number of disk seeks under control.

Locality Groups are just groups of multiple column families. Separate SSTables are generated for each such group. Segregating column families that are not typically accessed together into separate locality groups enables more efficient reads.

Immutability helps in implementing concurrency control over rows. It means that the system does not need to implement any synchronization of accesses to the file-system using SSTables. Finally, the immutability of SSTables also helps in splitting tablets quickly.

Conclusion

In conclusion, the design of the BigTable system shows us how important building blocks are. The use of various systems such as Chubby, GFS made a system like BigTable possible. More importantly, it shows how essential simple designs are building a distributed system at this scale. Even though the data model of BigTable is albietly simple (non-relational, non-transactional across rows) it allows the users to use it successfully in a varety of applications.