Bigtable

Review:
Bigtable is a distributed storage system designed for scale for variety of products at Google. It is kind of similar to a database but does not fully support the relational model and instead provide a data model which consist of row, column and timestamp. Data is indexed using arbitrary string that can be row and column name. Rows->load balancing column->access control and timestamps-> to control multiple version of same data. Bigtable can also be clubbed with Mapreduce as an input or output source. It provides an API for interacting with the table and metadata. Collection of rows are called tablets and is used for range queries. A group of column keys are called column families for access controls. Bigtable has various building blocks such as Google cluster management system for job and resource management, GFS to store and log data files, SSTable to store immutable file format and Chubby file for distributed file locking which helps with the synchronization. The immutable nature helps with synchronization primitives. Bigtable fails if Chubby service is unavailable for extended period of time. It is divided into three components: library to link all the clients, single master and multiple variable number of tablet servers according to the load on the server. Master assigns tablets to tablet servers, manages the number of tablet servers and controls the garbage control in GFS. The tablets handle read and write request and split very large tablets. Here, since the client directly interacts with the tablets, it does not rely on master for the tablet location information hence is lightly loaded. For the tablet location, a three level hierarchy similar to B+- tree is used. First level is in Chubby which has location of the root table. Root table has the location of all the locations in the METADATA tablets which contains all the location of user tablets. The metadata table can be used for recovery in case of failure. Persistent state of tables are stored in GFS. Updates are stored in commit log. The recent commits are stored in RAM in sorted order in a table called memtable. Once the memtable grows in size, the existing memtable is frozen and put into gfs and a new memtable is created. This process is called minor compaction. Each minor compaction creates a SSTable(since frozen means immutable). Once we reach a certain number of SSTable, we merge them and call this process merging compaction. When all SSTable are merge into one SSTable, then its a major compaction. It also performs different kind of optimization such as using locality groups for efficient reads by clients, using the bloom filter(long live the O(1) data structure), commit log, exploit the immutable nature of SSTable for caching, using compression such as Block Cache and and Scan Cache. Because of this features of Google Bigtable, it is able to scale to large fleet of machine and petabytes of data.

Comments:
I feel the paper was a bit unnecessarily long. It felt more like a documentation and

some details about the apis and implementation could have been avoided.

Since it contains a large number of untested code (100000), what are the percentage of bugs/failure discovered later on due to the untested code vs the bugs because of the tested code.

Since BigTable has lot of other google service dependency(such as Chubby), are there any attempts to make it work as a stand alone project? Since it is a complicated project with lots of external dependencies and organizations will naturally find it difficult to adopt.

Since Delta lake provides a layer of ACID table storage for key-value store, does a similar abstraction is available for BigTable since its a distributed column family.