Bigtable: A Distributed Storage System for Structured Data

Google Bigtable is a storage system designed to support multiple Google products. Bigtable needed a system that would be highly available, highly scalable, and have good performance. Google achieved this goal with some careful decisions about how to structure the data model and interactions with the tablet servers. The following questions cover some of those decisions and how they impact the performance of Bigtable.

1. “The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.” While a table is stored in the form of KV(Key-value) items, what is the key?

The key is a combination of the row key, column key, and the timestamp.

2. “Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.” How would clients select keys to get good locality? What possible advantages could a client obtain by having the locality?

Since Bigtable maintains the row keys in lexicographic (alphabetic) order, clients can select keys that are alphabetically close to each other to get good locality. When reading data, reading a short range of rows will be more efficient and require less machines to communicate to get the values. Therefore, having good locality for related data will allow for more efficient and faster results.

3. “Bigtable uses the distributed Google File System (GFS) to store log and data files.” To ensure high data reliability, does BigTable need to maintain multiple replicas for each of its data items?

While multiple replicas are necessary, Bigtable doesn’t need to maintain replicas itself. GFS already has inbuilt mechanisms to handle file replication, so Bigtable can trust GFS to ensure high data reliability.

4. “The Google SSTable file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings.” What does it mean by “immutable”? Why is this feature required?

“Immutable” means the SSTable cannot be modified once it is created. Immutability is required because the cost of trying to modify SSTables as write requests come in is very high. Instead, it is faster to let the SSTables be immutable and store the changes in the memtable elsewhere.

5. “A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: ...” Describe how a KV item is retrieved from an SSTable and why only one disk access is required for a lookup? [Hint: assume each block in an SSTable is 4KB, the disk access unit.]
Because the index is loaded into memory when the SSTable is loaded, to perform a lookup, the system can do a binary search on the index in memory to locate the appropriate block in the SSTable. Once the block is located, it can be read from the disk and iterated over to find the KV item.

(6) “Of these updates, the recently committed ones are stored in memory in a sorted buffer called a memtable; the older updates are stored in a sequence of SSTables.” Why do older updates exist and possibly exist in a sequence of SSTables?

It’s not convenient or cost-effective to delete older versions as soon as they’re not needed; it would impact the performance. Instead, older versions are stored temporarily until the system finds a break to go back and delete them.

(7) “A merging compaction that rewrites all SSTables into exactly one SSTable is called a major compaction.” What is minor compaction, and what is major compaction? Why is major compaction needed? How is a KV item deleted?

Minor compaction is converting a memtable into an SSTable. Merging compaction reads the contents of a few SSTables and the memtable and produces a new SSTable. The SSTables and memtable we used can now be deleted after merging compaction. This compaction keeps the number of SSTables to a manageable level. Both minor compaction and merging compaction do not delete data. Instead, special deletion entries are left in the SSTables that tell which data to ignore in older SSTables. Major compaction rewrites all SSTables into one SSTable that contains no deleted data or deletion entries. Major compaction is needed so that the level of SSTables can be reduced to a smaller amount. Without major compaction, the number of levels would continue to grow and make read requests take a long time to process.

To delete a KV item, a delete operation is sent to Bigtable. That request is stored in the memtable as a deletion entry. When the deletion entry is read, it suppresses the KV item that will be deleted so it can’t be read. Eventually, minor compaction turns the memtable into an SSTable. The deletion entry remains the same. The SSTable may go through some merging compaction as time goes on. Eventually, during a major compaction, the KV item is finally deleted so it does not appear in the new SSTable.