Big Table Notes

Bigtable is a distributed storage system for managing structured data.

Data Model

Bigtable is indexed by row key, column key, and timestamp. Row keys and column keys are arbitrary strings. Timestamps are 64-bit integers in microseconds. `(row:string, column:string, time:int64) -> string`

Row Keys

Row keys are typically 10-100 bytes. The max size is 64 KB. Bigtable stores data by the row key in lexicographic order. Each table is dynamically partitioned by a range of rows. These partitions of row ranges are called tablets. In Bigtable, this partitioning approach allows for efficient reads to row ranges.

Column Keys

Column keys are grouped into sets of column families. As column keys within a column family have minimal variability, it allows for optimal data compression within a column family.

Timestamps

Bigtable allows for multiple versions of the same data. These versions are indexed by timestamp. Bigtable can be configured to garbage collect older versions automatically. With garbage collection turned on, only the most recent three versions are stored.

Integrations

Bigtable uses the Sorted String Table (SSTable) file format for its indexes. Refer to Chapter 3: Storage and Retrieval in Designing Data-Intensive Applications for more information on SSTables.

Bigtable uses the Google File System to store log and data files.

Bigtable uses the Chubby Lock Service to ensure a single active master, discover tablet servers, store Bigtable schema information, and store access control lists.

Bigtable depends on Borg, Google's cluster management system, for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine health.

Bigtable can be used as an input source and/or output target for MapReduce jobs.

Design

Bigtable has three major components: a client library, one master server, and many tablet servers.

Master

The Bigtable master is primarily responsible for assigning tablets to tablet servers and balancing tablet server load. It manages garbage collection of GFS files. It handles any schema changes e.g., table and column family updates.

Tablet Servers

Each tablet server will manage between 10 to 1,000 tablets. The tablet server handles all read and write requests to these tablets. The tablet server will also split tablets when it has become too large. Each tablet is about 100 to 200 MB in size. Each tablet is assigned to one tablet server at a time.

Clients

Clients communicate directly with tablet servers through the client library for all their read and write requests. Clients do not rely on the Bigtable master for tablet location information. Clients only communicate with the master on schema changes; therefore, the master is typically lightly loaded.

Tablet Location

Bigtable uses a three-level hierarchy to store tablet location information. This hierarchy can store up to

2^{34} tablets.

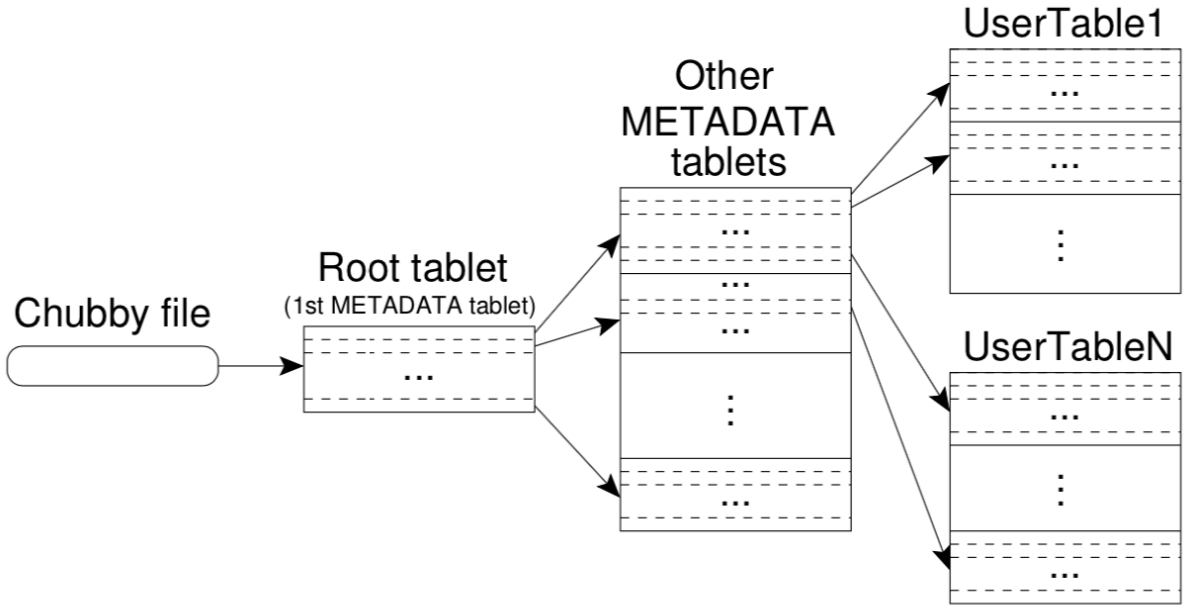

Figure 4: Tablet location hierarchy.

1. The first level is the Root tablet. The Root tablet location is stored in a Chubby file. The Root tablet contains mappings of all the User tablets to their respective METADATA tablet. The Root tablet is never split to maintain a three-level hierarchy.
2. The second level consists of the METADATA tablets. Each METADATA tablet contains the location of a set of User tablets. Each User tablet location is stored under a row key and contains the tablet's table identifier and its end row. The metadata contains the list of SSTables that make up a tablet. Each METADATA row stores about 1 KB in memory. Each METADATA tablet has a max size of 128 MB.
3. The third level consists of all the User tablets. To locate tablets, clients require three network round-trips, including one read from Chubby. The client library caches tablet locations.

Tablet Assignment

The master will detect when a tablet server is no longer serving its tablets. Upon detection, the master will reassign those tablets to new tablet servers immediately.

When a new master starts up, it executes the following steps:

1. The master acquires a unique master lock from Chubby to ensure a single active master.

2. The master scans the servers directory in Chubby to find the live tablet servers.
3. The master requests each tablet server to report on which tablets are assigned to it.
4. The master scans the METADATA tablets to learn of its sets of tablets. When the master encounters tablets that are not assigned to a live tablet server, the master will assign it accordingly.

Transactions

Bigtable will only support single-row transactions. Bigtable does not support general transactions across a range of row keys.

Locality Groups

Clients can group multiple column families into a single locality group. A separate SSTable is generated for each locality group in a tablet. Locality groups allow for more efficient reads of column families that are typically read together.

Compression

Bigtable allows for compression schemes for locality groups. In most cases, clients use a two-pass compression scheme:

- The first pass uses Bentley and McIlroy's scheme that compresses long common strings across a large window.
- The second pass uses a fast compression algorithm that compresses repetitions in a small 16 KB window.

Both compressions are very fast. They encode at 100-200 MB/s and decode at 400-1,000 MB/s.

In an experiment with web page content, the two-pass compression scheme achieved a 10-to-1 reduction in space. In comparison, Gzip compression achieved a 4-to-1 reduction.

Caching

Tablet servers use two levels of caching to improve read performance.

- The Scan Cache is a high-level cache that stores key-value pairs returned by an SSTable.
- The Block Cache is a low-level cache that stores SSTables blocks read from GFS.

The Scan Cache is useful for applications that read the same data repeatedly. The Block Cache is useful for applications that read data close to recently-read data e.g., sequential reads or random reads in the same locality group.

Bloom Filters

Bigtable utilizes bloom filters to reduce the number of disk seeks for read operations. Bloom filters store if an SSTable does not contain data for a row/column pair; therefore, bloom filters prevent disk lookups for non-existent rows or columns.

Examples

- **Google Analytics.** The raw clicks table contains data for each end-user session. The row name is a tuple containing the website's name and the time the session was created. The raw clicks table is compressed to 14% of its original size. The summary table contains predefined summaries for each website. The summary table is compressed to 29% of its original size.
- **Google Earth.** The imagery table is used to store raw imagery. The imagery table is the input source into a MapReduce job. The output target is a table to serve client data.
- **Personalized Search.** The user actions table contains data for all user actions. Each type of action has a separate column family. The user actions table is the input source into a MapReduce job. The output target is a user profiles table. These user profiles are used to personalize live search results. Many other groups have added their own columns to the user profiles table for use in other Google services.