Summary 2

> *Gabriela Gutiérrez Valverde - 2019024089*

> *Book: Bigtable: A Distributed Storage System for Structured Data*

Introduction to Bigtable

- Designed to reliably scale to petabytes of data and thousands of machines.
- Wide applicability, scalability, high performance, and high availability.
- Shares many implementation strategies with databases.
- Data is indexed using row and column names that can be arbitrary strings.
- Schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Data Model

It is a sparse, distributed, persistent multidimensional sorted map, indexed by a row key, column key, and a timestamp.

Rows:

- Row keys in a table are arbitrary strings.
- Every read or write of data under a single row key is atomic.
- Each row range is called a tablet.

Column Families:

- Group of column keys.
- All data stored in a column family is usually of the same type.
- A column family must be created before data can be stored under any column key in that family.

Timestamps:

- Multiple versions of the same data are indexed by timestamp.
- Keep N versions and garbage-collect cell the other versions automatically.

API

Provides functions for creating and deleting tables and column families. Also, changing cluster, table, and column family metadata, such as access control rights. Client applications can write or delete values in Bigtable, look up values from individual rows, or iterate over a subset of the data in a table. Supports single-row transactions and allows cells to be used as integer counters.

Building Blocks

Uses the distributed Google File System (GFS) [17] to store log and data files. Depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine status.

SSTable: is used internally to store Bigtable data. Provides a persistent, ordered immutable map from keys to values.

Chubby: highly-available and persistent distributed lock service. Consists of five active replicas, one of which is elected to be the master and actively serve requests.Uses the Paxos algorithm to keep its replicas consistent in the face of failure. Provides a namespace that consists of directories and small files. Maintains a *session*.

Implementation

Mayor components:

- Library that is linked into every client.
- One master server.
- Many tablet servers.

Tablet location: three-level hierarchy analogousto that of a B+ tree to store tablet location information

- First level: a file stored in Chubby that contains the location of the root tablet.
- Second level: contains the location of all tablets in a special METADATA table.
- Third level: contains the location of a set of user tablets.

Tablet Assignment: Each tablet is assigned to one tablet server at a time. The *master* keeps track of the set of live tablet servers, and the current assignment of tablets to tablet servers, it periodically asks each tablet server for the status of its lock. Whenever a tablet server terminates, it attempts to release its lock so that the master will reassign its tablets more quickly. Master steps for startup:

1. Grabs a unique master lock in Chubby.
2. Scans the servers directory in Chubby to find the live servers.
3. Communicates with every live tablet server to discover what tablets are already assigned to each server.
4. Scans the METADATA table to learn the set of tablets.

When a table is created or deleted two existing tablets are merged to form one larger tablet, or an existing tablet is split into two smaller tablets. Incoming read and write operations can continue while tablets are split and merged.

Tablet Serving: To recover a tablet, a tablet serve reads its metadata from the METADATA table. This contains the list of SSTables that comprise a tablet and a set of a redo points.

Compactions: When the memtable size reaches a threshold, the memtable is frozen, a new memtable is created, and the frozen memtable is converted to an SSTable and written to GFS.

Refinements

Locality groups: A group pf multiple column families. Enables more efficient reads.

Compression: User-specified compression format is applied to each SSTable block. Small portions of an SSTable can be read without decompressing the entire file.

Caching for read performance: Scan Cache is a higher-level cache that caches the key-value pairs returned by the SSTable interface to the tablet server code. Block Cache is a lower-level cache that caches SSTables blocks that were read from GFS.

Bloom filters: Allows us to ask whether an SSTable might contain any data for a specified row/column pair.

Commit-log implementation: Append mutations to a single commit log per tablet server, co-mingling mutations for different tablets in the same physical log file. Avoid duplicating log reads by first sorting the commit log entries in order of the keys.

Speeding up tablet recovery: Compaction reduces recovery time by reducing the amount of uncompacted state in the tablet server's commit log. Then, other minor compaction to eliminate any remaining uncompacted state in the tablet server's log that arrived while the first minor compaction was being performed.

Exploiting immutability: only mutable data structure is the memtable. To reduce contention during reads of the memtable, we mak each memtable row copy-on-write and allow reads and writes to proceed in parallel.

Performance Evaluation

set up a Bigtable cluster with N tablet servers to measure the performance and scalability of Bigtable as N is varied. R is the distinct number of Bigtable row keys involved in the test. Helpes mitigate the effects of performance variations caused by other processes running on the client machines.

- *Sequential read benchmark:* generated row keys in exactly the same way as the sequential write benchmark, but instead of writing under the row key, it read the string stored under the row key.
- *Scan benchmark:* similar to the sequential read benchmark, but uses support provided by the Bigtable API for scanning over all values in a row range.
- *Random reads (mem) benchmark:* similar to the random read benchmark, but the locality group that contains the benchmark data is marked as in-memory.

Single tablet-server performance: Random reads are slower than all other operations by an order of magnitude or more.

Scaling: Bottleneck on performance. Rebalancing is throttled to reduce the number of tablet movements , and the load generated by our benchmarks shifts around as the benchmark progresses.

Real Aplications

- **Google Analytics:** helps webmasters analyze traffic patterns at their web sites.
- **Google Earth:** operates a collection of services that provide users with access to high-resolution satellite imagery of the world's surface.
- **Personalized Search:** opt-in service that records user queries and clicks across a variety of Google properties such as web search, images, and news.

Lessons:

- Large distributed systems are vulnerable to many types of failures.
- It is important to delay adding new features until it is clear how the new features will be used.
- Importance of proper system-level monitoring.
- Value of simple designs.