Instituto Tecnológico de Costa Rica

Bases de Datos - IC4302

Resumen 2 y 3

Estudiante:

Andrea María Li Hernández - 2021028783

Profesor:

Gerardo Nereo Campos Araya

Fecha de entrega: 29 de agosto del 2022

Segundo Semestre, 2022

Bigtable: A Distributed Storage System for Structured Data

Introduction

Bigtable is a **distributed storage system** for managing structured data that is designed to scale to petabytes of data across multiple commodity servers. It shares many implementation strategies with databases, but it provides clients with a simple data model that allows clients to dynamically control whether to serve data out of memory or from disk. The data is indexed using row and column names that can be arbitrary strings. Bigtable has achieved several goals such as:

- Wide applicability and scalability.
- High performance and availability.

Bigtable is used by more than 60 Google products and projects, including Google Analytics and Google Earth.

In this summary you'll find the description of the data model, an overview of the client API, examples of use, along with more useful information about this system.

Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

- The map is indexed by a row key, column key and timestamp.
- Each value in the map is an uninterpreted array of bytes.

Rows:

- The keys are arbitrary strings.
- Every read or write of data under a single row key is atomic.
- The **row range** for a table is dynamically partitioned.
  - Each row range is called a ***tablet***, this is the unit of distribution and load balancing.

Column Families:

- **Column keys** are grouped into sets called ***column families***.
- They form the basic unit of access control.
- All data stored in a column family is usually of the same type.
- A column key name syntax: ***family:qualifier***.

Timestamps:

- Each cell in a Bigtable can contain **multiple versions** of the same data and these versions are **indexed** by timestamp.
- They are 64-bit integers.
- They can be assigned by Bigtable (microseconds) or be assigned by client applications.
- The most recent versions can be read first

API

The Bigtable API provides:

- Functions for creating and deleting tables and column families.
- Functions for changing cluster, table, and column family metadata.

- Single-row transactions, which can be used to perform atomic read-modify-write sequences on data stored under a single row key.

Building Blocks

Bigtable is built on several other pieces of Google infrastructure.

- It uses the distributed **Google File System (GFS)** to store log and data files.
- Bigtable processes often share the same machines with processes from other applications.
- It depends on a **cluster management system** for scheduling jobs, managing resources on shared machines, dealing with failures and monitoring machine status.
- The **Google SSTable** file format is used internally to store Bigtable data.
    - An SSTable provides a persistent, ordered and immutable map from keys to values.
- Bigtable relies on a highly-available and persistent distributed lock service called **Chubby**.
    - Chubby consists of 5 active replicas, one of which is the master that actively serves requests.
    - The service is live when most replicas are running and can communicate with each other.
    - It uses the Paxos algorithm to keep its replicas consistent in the face of failure.
    - Some tasks: To ensure that there is at most one active master at any time, store Bigtable schema information and store access control lists.
    - If Chubby becomes unavailable for an extended period, Bigtable becomes unavailable.

Implementation

It has 3 major components:

1. A library that is linked into every client.
2. One master server: Responsible for assigning tablets to tablet servers, balancing tablet-server load, and garbage collection of files in GFS.
3. Many tablet servers.

- Clients communicate directly with tablet servers for reads and writes.
- A Bigtable cluster stores several tables, each table consists of a set of tablets, and each tablet contains all data associated with a row range.
- **Tablet location**: It uses 3 levels to store tablet location information. Hierarchy diagram:

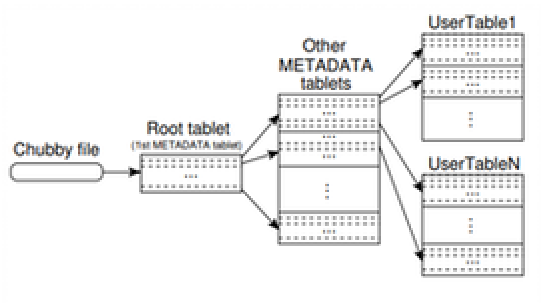

Figure 4: Tablet location hierarchy.

- **Tablet assignment**: Each tablet is assigned to one tablet server at a time and Chubby keeps track of tablet servers. Whenever a tablet server terminates, it attempts to release its lock so that the master will reassign its tablets more quickly. The master kills itself if its Chubby session expires.
- **Tablet Serving**: The persistent state of a table is stored in GFS. Updates are committed to a commit log that stored redo records and the recently committed are stored in memory in a sorted buffer called a ***memtable***. When a write operation arrives at a tablet server, the server checks that it is well-formed, and that the sender is authorized, and a read operation arrives, it is similarly checked as described before.
- **Compactions**: As write operations execute, the size of the memtable reaches a threshold and the frozen memtable is converted to an SSTable and written to GFS.

Refinements

This section describes portions of the implementation in more detail to achieve the high performance required by the users.

- **Locality groups**: Clients can group multiple column families together into a locality group. A separate SSTable is generated for each locality group in each tablet.
- **Compression**: Clients and control whether the SSTables for a locality group are compressed and the format that is used.
- **Caching for read performance**: To improve read performance, tablet servers use 2 levels of caching: The Scan Cache and the Block Cache.
- **Speeding up tablet recovery**: If the master moves a tablet from one tablet server to another, the source tablet server first does a minor compaction on that tablet, and this reduces recovery time.
- **Exploiting immutability**: The immutability of SSTables enables us to split tablets quickly.

Performance Evaluation

To test the Bigtable performance, the authors made some experiments to evaluate the performance of the system. Here are some of the results and conclusions:

- **Single tablet-server performance**: Random and sequential writes perform better than random reads. Sequential reads perform better than random reads.
- **Scaling**: Aggregate throughput increases dramatically as we increase the number of tablet servers in the system from 1 to 500.

Real Applications

In this section there’s a description of how three Google products use Bigtable.

1. **Google Analytics**: Google Analytics is a service that helps webmasters analyze traffic patterns at their websites. GA uses 2 tables: The raw click table that maintains a row for each end-user session, and the Summary table which contains various predefined summaries for each website.
2. **Google Earth**: It allows users to navigate across the world’s surface. This system uses one table to preprocess data and another set of tables to serve client data.
3. **Personalized Search**: This is an opt-in service that records user queries and clicks across a variety of Google properties like web search, images, and news.

Lessons

In this section, the authors mention some lessons that they learned in the process of designing, implementing, maintaining, and supporting Bigtable. Some lessons are the following:

- 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.
- It is very important to have proper **system-level monitoring**, this could make it possible to detect and fix problems quickly.
- The most important lesson is the value of **simple designs**, code and design clarity are of immense help in code maintenance and debugging.

Related Work

The Boxwood project was made to provide infrastructure for building higher-level services such as file systems or databases; it has components that overlap in some ways with Chubby, GFS, and Bigtable, because it provides for distributed agreement, locking, distributed chunk and B-tree storage.

C-Store and Bigtable share many characteristics such as using a shared-nothing architecture and having 2 different data structures, one for recent writes and another for storing long-lived data. Nevertheless, the main difference is their API, C-Store behaves like a relational database, while Bigtable provides a lower level read and write interface.

Conclusions

After describing Bigtable, a distributed system for storing structured data at Google, here are some of the conclusions made by the authors:

- The users like the **performance and high availability** provided by the system’s implementation; besides that, they can scale the capacity of their clusters by adding more machines to the system.
- New users are sometimes uncertain of how to best use the Bigtable interface. Nevertheless, the fact that many Google products successfully use Bigtable proves that the **design works well in practice**.
- There are many advantages to building their storage solution at Google because of the amount of **flexibility** from designing their own data model for Bigtable.