Resumen 2 y Resumen 3
Estudiante: Celina Madrigal Murillo
Carné: 2020059364

Bigtable: A Distributed Storage System for Structured Data

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.

1 Introduction

Bigtable is designed to reliably scale to petabytes ofdata and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability.

2 Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

Rows

The row keys in a table are arbitrary strings. Every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row), a design decision that makes it easier for clients to reason about the system’s behavior in the presence of concurrent updates to the same row.

Column Families

Column keys are grouped into sets called column families, which form the basic unit of access control. All datastored 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; after a family has been created, any column key within the family can be used.

Timestamps

Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamp. Bigtable timestamps are 64-bit integers. They can be assigned by Bigtable, in which case they represent “real time” in microseconds, or be explicitly assigned by client applications. Applications that need to avoid collisions must generate unique timestamps themselves. Different versions of a cell are stored in decreasing timestamp order, so that the most recent versions can be read first.

3 API

The Bigtable API provides functions for creating and deleting tables and column families. It also provides functions for 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.

4 Building Blocks

Bigtable is built on several other pieces of Google infrastructure. Bigtable uses the distributed Google FileSystem (GFS) [17] to store log and data files. A Bigtable cluster typically operates in a shared pool of machines that run a wide variety of other distributed applications, and Bigtable processes often share the same machines with processes from other applications. Bigtable depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring machine status.

5 Implementation

The Bigtable implementation has three major components: a library that is linked into every client, one master server, and many tablet servers. Tablet servers can be dynamically added (or removed) from a cluster to accomodate changes in workloads.

5.1 Tablet Location

We use a three-level hierarchy analogous to that of a B+- tree [10] to store tablet location information. The first level is a file stored in Chubby that contains the location of the root tablet. The root tablet contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets.

5.2 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, including which tablets are unassigned. When a tablet is unassigned, and a tablet server with sufficient room for the tablet is available, the master assigns the tablet by sending a tablet load request to the tablet server.

5.3 Tablet Serving

The persistent state of a tablet is stored in GFS. Updates are committed to a commit log that stores redo records. 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. To recover a tablet, a tablet server reads its metadata from the METADATA table.

5.4 Compactions

As write operations execute, the size of the memtable increases. 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. This minor compaction process has two goals: it shrinks the memory usage of the tablet server, and it reduces the amount of data that has to be read from the commit log during recovery if this server dies. Incoming read and write operations can continue while compactions occur.

6 Refinements

The implementation described in the previous section required a number of refinements to achieve the high performance, availability, and reliability required by our users. This section describes portions of the implementation in more detail in order to highlight these refinements.

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. Segregating column families that are not typically accessed together into separate locality groups enables more efficient reads.

Compression

Clients can control whether or not the SSTables for a locality group are compressed, and if so, which compression format is used. The user-specified compression format is applied to each SSTable block. Although we lose some space by compressing each block separately, we benefit in that small portions of an SSTable can be read without decompressing the entire file.

Caching for read performance

To improve read performance, tablet servers use two levels of caching. The Scan Cache is a higher-level cache that caches the key-value pairs returned by the SSTable interface to the tablet server code. The Block Cache is a lower-level cache that caches SSTables blocks that were read from GFS. The Scan Cache is most useful for applications that tend to read the same data repeatedly. The Block Cache is useful for applications that tend to read data that is close to the data they recently read.

Bloom filters

A Bloom filter allows us to ask whether an SSTable might contain any data for a specified row/column pair. For certain applications, a small amount of tablet server memory used for storing Bloom filters drastically reduces the number of disk seeks required for read operations. Our use of Bloom filters also implies that most lookups for non-existent rows or columns do not need to touch disk.

Commit-log implementation

If we kept the commit log for each tablet in a separate log file, a very large number of files would be written concurrently in GFS. Depending on the underlying file system implementation on each GFS server, these writes could cause a large number of disk seeks to write to the different physical log files. In addition, having separate log files per tablet also reduces the effectiveness of the group commit optimization, since groups would tend to be smaller. To fix these issues, we append mutations to a single commit log per tablet server, co-mingling mutations for different tablets in the same physical log file [18, 20].

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. This compaction reduces recovery time by reducing the amount of uncompacted state inthe tablet server’s commit log. After finishing this compaction, the tablet server stops serving the tablet. Before it actually unloads the tablet, the tablet server does another (usually very fast) minor compaction to eliminate any remaining uncompacted state in the tablet server’s log that arrived while the first

minor compaction was being performed. After this second minor compaction is complete, the tablet can be loaded on another tablet server without requiring any recovery of log entries.

Exploiting immutability

Besides the SSTable caches, various other parts of the Bigtable system have been simplified by the fact that all of the SSTables that we generate are immutable. Since SSTables are immutable, the problem of permanently removing deleted data is transformed to garbage collecting obsolete SSTables.

7 Performance Evaluation

We set up a Bigtable cluster with N tablet servers to measure the performance and scalability of Bigtable as N is varied. The tablet servers were configured to use 1 GB of memory and to write to a GFS cell consisting of 1786 machines with two 400 GB IDE hard drives each.

Single tablet-server performance

Random reads are slower than all other operations by an order of magnitude or more. Each random read involves the transfer of a 64 KB SSTable block over the network from GFS to a tablet server, out of which only a single 1000-byte value is used. The tabletserver executes approximately 1200 reads per second, which translates into approximately 75 MB/s of data read from GFS. This bandwidth is enough to saturate the tablet server CPUs because of overheads in our networking stack, SSTable parsing, and Bigtable code, and is also almost enough to saturate the network links used in our system.

Scaling

Aggregate throughput increases dramatically, by over a factor of a hundred, as we increase the number of tablet servers in the system from 1 to 500.

8 Real Applications

As of August 2006, there are 388 non-test Bigtable clusters running in various Google machine clusters, with a combined total of about 24,500 tablet servers.

8.1 Google Analytics

Google Analytics is a service that helps webmasters analyze traffic patterns at their web sites. It provides aggregate statistics, such as the number of unique visitors per day and the page views per URL per day, as well as site-tracking reports, such as the percentage of users that made a purchase, given that they earlier viewed a specific page.

8.2 Google Earth

Google operates a collection of services that provide users with access to high-resolution satellite imagery of the world’s surface, both through the web-based Google Maps interface and through the Google Earth custom client software. These products allow users to navigate across the world’s surface: they can pan, view, and annotate satellite imagery at many different levels of resolution. This system uses one table to preprocess data, and a different set of tables for serving client data.

8.3 Personalized Search

Personalized Search is an opt-in service that records user queries and clicks across a variety of Google properties such as web search, images, and news. Users can browse their search histories to revisit their old queries and clicks, and they can ask for personalized search results based on their historical Google usage patterns.

9 Lessons

One lesson we learned is that large distributed systems are vulnerable to many types of failures, not just the standard network partitions and fail-stop failures assumed in many distributed protocols. Another lesson we learned is that it is important to delay adding new features until it is clear how the new features will be used. A practical lesson that we learned from supporting Bigtable is the importance of proper system-level monitoring. The most important lesson we learned is the value of simple designs.

10 Related Work

The Boxwood project [24] has components that overlap in some ways with Chubby, GFS, and Bigtable, since it provides for distributed agreement, locking, distributed chunk storage, and distributed B-tree storage. In each case where there is overlap, it appears that the Boxwood's component is targeted at a somewhat lower level than the corresponding Google service. The Boxwood project's goal is to provide infrastructure for building higher-level services such as file systems or databases, while the goal of Bigtable is to directly support client applications that wish to store data.