Google Bigtable Introduction and Performance Analysis

Yuanzhen Du
University of Rochester
206-388-9153
ydu20@ur.rochester.edu

Xinyu Ren
University of Rochester
360-440-6997
roger.ren@rochester.edu

ABSTRACT

In this paper we will introduce how bigtable works, its basic structure, implementation algorithm, advantages and comparison with traditional SQL.

1. INTRODUCTION

People might be unfamiliar with term 'Bigtable', however we use Google Map, Google Earth, Gmail and YouTube from time to time and they are all powered by Bigtables. Unlike traditional relation database. 1. It can efficiently handle large scale of data: petabytes and across thousands of commodities. 2. No schema database. 3. Suitable for handling semi structure data. 4. Self-managed and handle massive workload with consistent low latency and high throughput.

2. Data Model

Bigtable basically is a sparse, distributed, persistent multidimensional sorted map, three important elements account for constructing index for sorting and searching records. Rows, columns and timestamp.

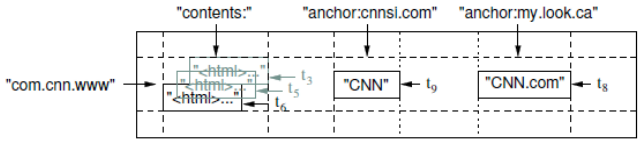

Figure 1 Webtable

2.1 Rows

Arbitrary string is used as row keys which size up to 64kb. Its' atomically type value make it easy to update and to reason system's behavior. In the Webtable example above, reverse domain name is used as row key to avoid conflictions. Index are cut according to ranges as single unit called tablet. Client operating queries within small range index is efficient.

2.2 Column Family

Column family follow a form of family: qualifier. Same types of content grouped into one column key. Family key are created before data loaded in. While traditional database allows unbounded amount of columns, Bigtable restricted the number around 100 and it barely changes over time. Control access with disk and memory operations are performed at column family level. Operations such as add new, view or create.

2.3 Timestamp

For each column, bigtables use timestamp (64-bit integer) to track update of data cell. For sequential data, when new data wrote in, it adds an additional layer of data with new timestamp. Data structure can be viewed as 3D with third dimension. Timestamp is also convenient for defining garbage-collect which means certain data or only last versions of data will be kept.

3. API

Multiple clients can be used as functions to perform operations on database. Hbase java, go client, python client and BigData tool such as Apache, Hadoop can run map/reduce jobs that read from write to cloud bigtable [1].

4. Bottom up Structures of Bigtable

Bigtable uses Google File System (GFS) [2] to save log and data. It depends on cluster management system and scheduling tablets assignment, compactions on shared machines. Next we introduce elements of the structure.

4.1 SSTable

SSTable is a file format to stored persistent, ordered immutable map from key to values. It contains block index (block of 64kb size) and can be applied binary search to locate certain block. Client loaded SSTable loaded into memory to do search without touching disk.

4.2 Master and Tablets Server

There are two kinds of severs in Bigtable, master server and tablet server. Master server mainly responsible for 1. Assigning tablets to tablet server. 2. Detecting addition or expiration in tablets sever. 3. Balancing workload between servers and avoid server not vulnerable to Internet failure. 4. Garbage collections and change of schema.

Tablets server major jobs: Interacting with clients and writing, reading and splitting tables. In that situation there is only light load for master server. It also keeps log with redo records as data are wrote and updated. Recently commits will be loaded into memory and older records stored into SSTable [3].

5. Implementation

Bigtable use a highly available and persistent distributed lock services called Chubby. It ensures only one active master at any time; and store all the location of Bigtable data. As a table grows, it automatically split into multiple tablets and each of them 100-200 MB by default.

5.1 Tablet location

Three level of hierarchy structure is used to store table location information. First level is the chubby with root tablet, it stores locations of all other user tablets and never split. In that way it maintains the three level structures. This structure can save up to 2^{61} tablets. Client can access table location through cache and the worst situation of client's cache is stale, it takes up to six round trips through the structure. Table also contain log information to help debugging and performance analysis.

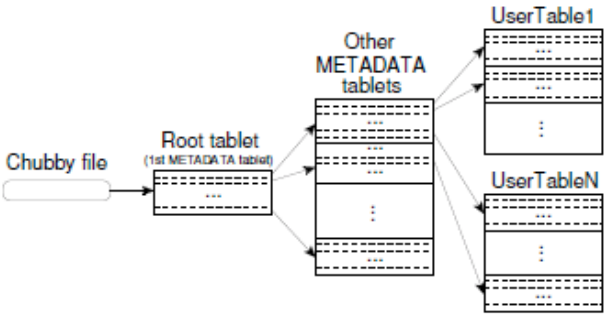

Figure 2 Three level hierarchy structure

5.2 Tablet assignment

When a tablet server is started, it will create a lock on a uniquely named file in the Chubby's [4] "servers" directory. 1. It master monitors those directory and look for new tablet servers. 2. Tablet will stop serving and attempt to regain the lock and exclusive lock is lost. When existing tablets changes a table is created or deleted by merging the old ones. Master can track these changes and when split happened, it will also notify master.

6. Compaction

As write operation keep executing, size of metastable might exceed limit. In that situation old table was frozen and been wrote to SSTable in GFS. Merging compactions will be executed periodically and few SSTable will be read and wrote on a new SSTable.

There are two kinds of compactions: major and non-major. Non major will help to reserve deletion information or deleted data which allows machine to reclaim data from deleted data and let them disappears in a time fashion.

7. Compare with SQL Database

For SQL database, when data grow into large scale maintenance and operation become impractical. In SQL server, data has to fit into tables and if it can't, new database structure need to be designed which can be complex again and difficult to handle [5]. For Bigtables, it supports features like automatic repair, easier data distribution and simpler data models. Since it is schema less so that data could be inserted without any predefined schema [6].

8. Spanner in Bigtable

Spanner is a NewSQL distributed relational database and it support ACID properties and also SQL queries. Spanner data model is semi relational model. It can also provide consistent replication of data which is similar to megastore [7].

8.1 Spanner Configuration

Above is the Spanner server configuration, universe is a set of spanner consists several zones. Universe master maintain all zones and interactive debugging while placement driver can move data between zones automatically. It acts like tablet server and consist with location proxy and many other span servers.

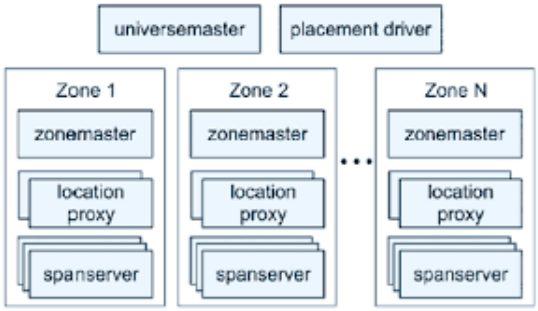

Figure 3 Spanner server configuration

8.2 Spanserver Stack

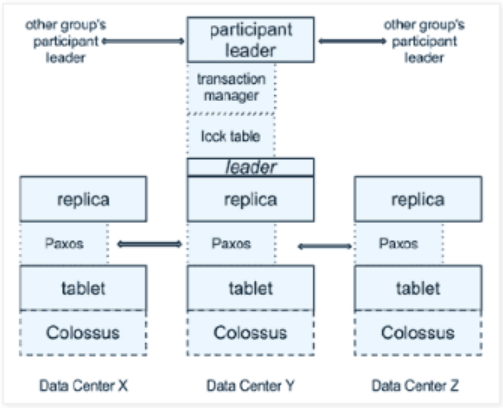

Figure 4 Spanner server stack

The Paxos state machine are used for replication, the key-value mapping state of the replica is stored in the relative tablet. We called the replicas Paxo group. There is a leader in every replica and the lock table contains two-phase locking: the maps key's range and its' lock state. If transaction only involve one Paxos group it will bypass the transaction manager however when multiple groups are involved, groups' leaders coordinate to perform two-phase commit.

9. PERFORMANCE OPTIMIZATION

The fundamental working mechanism of Bigtable was introduced in the above sections. Further tuning and optimization are performed to improve the Bigtable performance and reliability.

9.1 Data Compression

Data compression is a common approach that is utilized in database optimization. The objective is to save disk storage space. Data that was compressed, once be queried, needs to be first decompressed before becoming retrievable. Hence there is a trade of between the degree of space reduction and accessing speed. Namely, the higher the degree of space reduction, the slower the assessing speed.

In Google Bigtable, the priority objective is set to data accessing speed. End users have full control over the compression method and selection of the to-be-compressed data sections. The most popular approach is to implement a two-pass custom compression scheme [1]. In the first step, Bentley and McIlroy's scheme is used to compress long common strings. The second step on the

other hand, aims at small data pieces (16KB) with repetitions. Because both compression algorithms are fast pass and the two steps compression can well separate the dataset, the overall speed oriented design objective can be satisfied. Surprisingly, the space reduction performance did not encounter too much negative impact from implementing the two-pass method. As high as 10-to-1 reduction ratio can be achieved, whereas a one-step common Gzip reduction have ratio from 3-to-1 or 4-to-1 [1]. This is due to the intrinsic naming scheme of Bigtable, such that users usually choose row names that allow similar data to be stored in adjacency.

9.2 Data Reading

Despite Bigtable is mostly used with very large volume of data, not all data are equally important or get equally queried. Therefore, it is intuitive to store and cache data according to the access frequency and pattern. Bigtable achieve this optimization through allocating data to different locality groups and caching data by reading frequency.

In locality group method, users can aggregate, according to their specific needs, several columns to form a locality group. One common way to do so is to cluster the high frequent queried data together as a group and the less queried data together as the other group. The database reading efficiency hence can be greatly enhanced as a result of the reduction of full database scan. If some small pieces of data are very frequently accessed, users can also directly store those data as in-memory locality group, such that the reading frequency can be further improved.

Bigtable implement two levels of caching: Scan Cache and Block Cache. The Scan Cache stores only the key-value pairs and it is suitable for applications where some data is read repeatedly. Block Cache directly cache the entire SSTable from Google File System (GFS) and it is suitable to read data that are in closely located locality groups. For example, sequential reading is a good candidate for Block Cache method.

Finally, Bigtable can reduce disk access rate by apply a Bloom filter, which can determine the existence of a specific row/column pair without reading the disk. For certain applications (e.g. heterogeneous data are stored separately), Bloom filter can greatly reduce the initial indexing process time. However, if the application contains data that are evenly distributed, Bloom filter can hinder the reading speed.

9.3 Commit-log Version Control

Transactions are widely used in database system to assure stability and provide recovery options. For a distributed database system, the commit-log must be carefully controlled. It is both redundant and low efficient to keep an individual log for each machine.

Bigtable keeps only one physical single log file, through append mutations to a single commit log server. Normal database performance can be improved significantly from the merge of logs, except for the recovery operation. To solve this issue, any machine needs recovery will be distributed into a number of servers to perform the recovery task. Commit-Log sorting is used as well to further improve the recovery process by avoiding duplicating log files.

10. PERFORMANCE EVALUATION

Bigtable is designed to process very large volume of data through parallel computing. The key feature to test about the performance if Bigtable is the scalability.

Reading and writing 1000-byte values to Bigtable was tested from a single server to 500 servers [1]. 6 types of operation were tested: scans, random reads from memory, random writes, sequential reads, sequential writes and random reads from disk. All test operations implement systematic key generation algorithm to mitigate the performance variations caused by other programs running on the machine.

The test results are shown in Figure . From a single server to 500 servers, we can see a throughput increase by more than a factor of 100[1] in general.

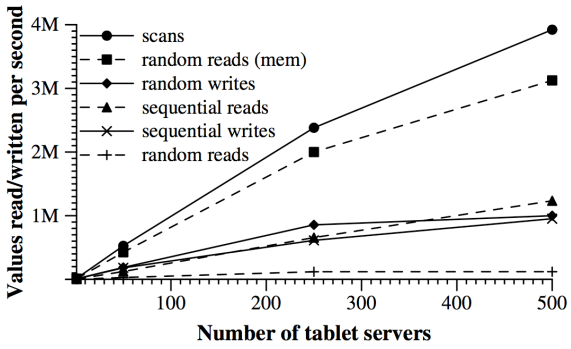

Figure 5 Scalability of Bigtable [1]

The operation benefits the most from the increase of server is the ones whose performance is limited by the individual CPU. Such as a scan or random reads from memory. As many as a factor of 300 was observed from the performance of scan or random reads from memory. For reads and writes that touches disks, the improvement was still significant but the rate of improvement decays quickly with the increase of servers. Among all the tests performed, random read from disk shows the worst scaling ability, as shown in Figure with '+' sign. This occurs because the network can be easily saturated through distribution of query data. The throughput therefore is limited.

The throughput per server under different server configurations can be seen in Figure . The performance is not linearly increase at all. Especially from a single server to fifty servers, there is a significant drop of through put. The Google team explained this is probably caused by the imbalance load in multiple server configurations, where other programs running on the machine can contend for CPU and network resources.

	# of Tablet Servers			
Experiment	**1**	**50**	**250**	**500**
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

Figure 6 Throughput of each server [1]

11. BIGTABLE APPLICATIONS

More than 60 Google products use Bigtable, including Google Analytics, Google Finance, Google Earth and Personalized search. Currently Bigtable have more than 400 Bigtable clusters, with a

combined more than 25,000 servers. Figure shows the distribution of servers in Bigtable clusters.

# of tablet servers			# of clusters
0	..	19	259
20	..	49	47
50	..	99	20
100	..	499	50
> 500			12

Figure 7 Cluster and server distribution [1]

The Bigtable can serve to end users as well as to internal data batch processing. Tables varies widely from size, cell, schema, compression ratio and etc. Here we briefly introduce three Google products that use Bigtable.

11.1 Google Earth

Google Earth enable users to navigate across the globe surface by pan, move, and view and annotate satellite images. Google Earth uses two tables: one to preprocess the image data and the other to serve client data.

The image table size for Google Earth is 70 TB, not compressed and 0% stored in memory. This is because the table is not latency sensitive. Through MapReduce, the overall processing speed is about 1MB/s. In this table, each row corresponds to a segment of geographic picture. Rows are named in a fashion that adjacent geographic segments are stored close to each other, to improve sequential reading performance.

The serving table is much smaller, only around 500GB, with 64% compression ratio and about 33% are stored in memory. This is because this tables deals with more than tens of thousands queries per second. This table is also hosted across hundreds of servers.

11.2 Google Analytics

Google analytics provides website hosts the ability to analyze their website traffic patterns through statistics including unique visitors' counts, page views per day and site tracking reports. It is implemented through embedding a JavaScript.

Two basic tables are used in Google analytics: the raw click table and the summary table. Both are not in memory storage due to the lack of latency sensitivity. The raw click table is in about 200TB in total size with compression rate of 14%. Each row name corresponds to an end-user session. The summary table is about 20TB with 29% compression rate. It contains predetermined summaries for each website. The table is populated by periodically scheduled MapReduce job from the raw click table.

11.3 Personalized Search

Google Personalized Search enables users to store their click and search histories across all Google products to achieve personalized experience through usage pattern recognition.

Typical table size per user is 4TB with compression ratio of 47%. Each user has a unique userId and is assigned a row using that userId. MapReduce job was created to generate user profile and the user profile was used to create personalized live search results.

12. CONCLUSIONS

Bigtable is a large-scale distributed storage and management system for structured data developed by Google. Bigtable was conceived due to traditional database systems, when face datasets with very large scale, often encounter feasibility, performance or prohibitive high cost challenges. Bigtable features robust scalability to petabytes of data and thousands of machines [1]. Numerous conventional database implementation strategies are utilized in Bigtable, such as parallel database strategy [2] and main-memory database strategy. More than sixty Google products are built upon Bigtable: Google Earth, Google Analytics and etc. Besides, Bigtable is also commercially available for external users. Data in Bigtable is indexed by a time stamp together with row and column names which can be any arbitrary strings. Bigtable is equipped with a simple and elegant data model to enable user to control the layout and format of data dynamically. The schema of Bigtable also allows end user to fully control the locality of their data.

13. Reference:

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *7th Symp. Oper. Syst. Des. Implement. (OSDI '06), Novemb. 6-8, Seattle, WA, USA*, pp. 205–218, 2006.

[2] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," *OSDI '06 Proc. 7th Symp. Oper. Syst. Des. Implement. SE - OSDI '06*, pp. 335–350, 2006.

[3] A. Khetrapal and V. Ganesh, "HBase and Hypertable for large scale distributed storage systems A Performance evaluation for Open Source BigTable Implementations."

[4] M. Database, "Google ' s Bigtable Oriented Datab ases," vol. 3, no. 8, pp. 60–63, 2016.

[5] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," 2003.

[6] A. Salehnia, "Relational Database Management Systems (RDBMDS)," pp. 1–8.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.