An Introduction to Bigtable : A Storage Model for Handling Massive Volumes of Structured Data

Chetan Mhatre
Department of Information Technology
Saraswati College of Engineering, University of Mumbai, India

Fareen Shaikh
Department of Information Technology
Saraswati College of Engineering, University of Mumbai, India

Ritesh Kamble
Department of Information Technology
Saraswati College of Engineering, University of Mumbai, India

Shubham Singh
Department of Information Technology
Saraswati College of Engineering, University of Mumbai, India

Shilpa Kolte
Department of Information Technology
Saraswati College of Engineering, University of Mumbai, India

ABSTRACT
Bigtable is a storage model designed for handling massive amount of data mainly over distributed systems. It stores data in tabular format but it is not a relational database. It is the proprietary system of Google Inc. It is fault tolerant, persistent and highly scalable. It is written in java, python, go and ruby. It has been at the heart of many google systems including google web search, gmail etc. In this paper the basic data model of the Bigtable is explained along with some concepts related to Bitable.

Keywords
Big data, Bigtable, Distributed Storage System.

1. INTRODUCTION
Bigtable is a data storage system built by google as its own proprietary system. Google started initial development of Bigtable in 2004. Bigtable not a relational database, it is a distributed, multilevel map. It is highly compressed and also high performance.

Bigtable is being designed with in-built fault tolerance and high scalability. It takes care of hardware failure and it is scalable to thousands of servers. It can handle millions of reads/writes per second also it can manage several terabytes of in-memory data and several petabytes of data on disk spread across multiple locations. This data can be of huge variety ranging from URLs, web searches, numeric data and satellite imagery. Bigtable is capable of managing its servers dynamically. It can add or remove servers from clusters dynamically also in case of excessive workload on a particular server it does dynamically spread its workload on other servers which currently are not under pressure.

Bigtable is a data storage facility but it does not support full relational data model. It provides clients with a data model where they can have a dynamic control over data layout and formatting. Data is stored using row, column and timestamp triplet.

The rest of the paper is arranged in the following manner . Section 2 explains data model along with row column and timestamp. Section 3 explains the concept of compaction in Bigtable.. Section 4 explains Tablet representation. Section 5 explains the whole of Bigtable system structure in detail and its underlying components. Section 6 will explain how to locate a particular tablet on the Bigtable .Section 7 gives a brief overview about some of the API's available with the Bigtable. In section 8 the conclusion has been made.

2. DATA MODEL
Bigtable is a sparse, distributed, multilevel map which indexed by a combination of row ,column and timestamp values.

(row:string, column:string,Timestamp:int64)→string

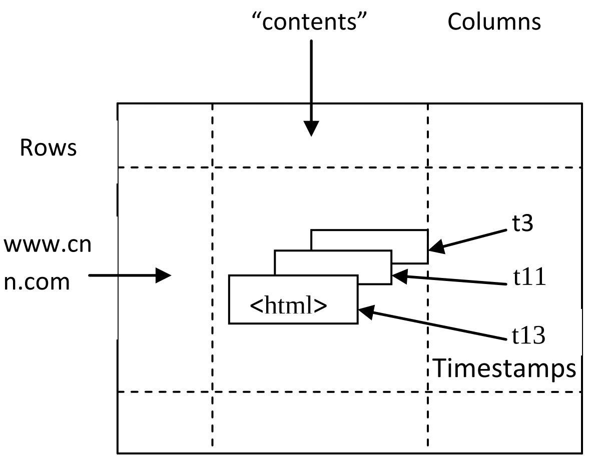

Fig 1: Bigtable Data storage model

Fig 1 demonstrates how Bigtable can store multiple values (instances) of data in a single cell using timestamp, it shows multiple versions of html pages stored at different values of timestamp. This feature makes Bigtable a truly multidimensional map. Google settled on this data model after a lot of research with similar data models.

2.1 Row
Row keys are arbitrary strings up to 64KB of size. Bigtable maintains data in lexicographical order of row key. Atomic access is provided to any of the columns in a row. Multiple contiguous rows grouped together is called a tablet. Rows can be dynamically partitioned into a tablet for better system performance. This tablets can move from one machine to another. One tablet can be handled by one machine at a given point of time .One tablet can hold up to 100 – 200 MB of data. If data in a tablet goes beyond this range or if the load on one tablet is more than other tablet then the tablet is dynamically partitioned into multiple tablets by selecting an appropriate

point from where the tablet can be evenly partitioned.

2.2 Column

Column keys are also strings having a two level naming structure which has column family and optional qualifiers.

Column name→ **family : optional_qualifier**

Column family are the basic unit of access control. They have the associated type information for various utilities that can dump data in various formats Column families must be created before storing data under column key in that family. Optional qualifiers on the other hand do provide valuable information for various operations on sorting, grouping etc.

2.3 Timestamp

Since Bigtable can store multiple versions of data in a cell it uses timestamp values to index these multiple versions. Timestamp is a 64-bit integer value. This timestamp value can be set by the Bigtable or be explicitly set by the clients. Timestamps make it possible for clients to access most recent values or use values among a specific range of timestamp.

Lookup options include returning most recent 'k' values, return most recent values, return all values etc.

Column families can be marked with certain attributes which may be considered by the system during performing compactions. One of the advantage of this attributes can be that users no longer will have to write garbage collection by themselves since this attributes will take care of that.

3. COMPACTIONS

Tablet state represents a set of immutable compacted SSTable[7] (Sorted Strings Table) files along with a log file.

Minor compaction happens when memory state fills up ,pick the tablet with most data and write contents to SSTable stored in GFS. Separate file for each locality group is created.

Major compaction happen periodically and they compact all the SSTable for the tablet into a new base SSTable on the GFS.

4. TABLET REPRESENTATION

As mentioned earlier a tablet is contiguous row grouped together. Now we focus on the internal details of the tablet as to how the tablet is represented internally. It consists of an append only log, an in memory buffer and a couple of SSTable all stored on GFS.

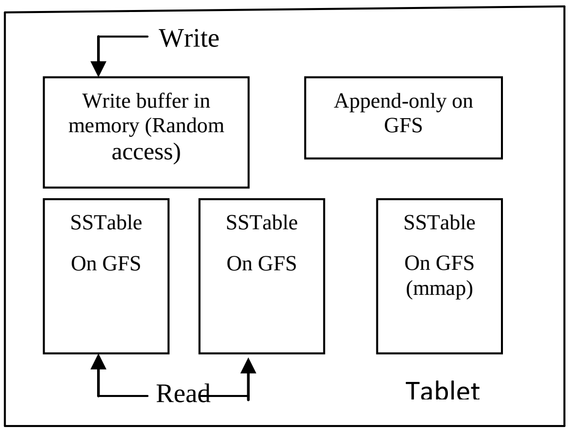

Fig 2 : Tablet Representation of Bigtable

The SSTable is a file of key/value string pairs, sorted by keys. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings[7].One machine holds roughly a hundred servers. Append only log is stored on GFS. It just holds the entry of all the mutations .This is mostly used for recovery purpose. All the mutations are appended to a log. There is one append log per tablet server. When a write request comes in that request is first put into a queue, from that queue a thread is invoked which then that thread writes the data to GFS. Once write is committed to GFS the same log is then mirrored to in memory buffer. This buffer holds all the writes that have not been committed to the disk file. Data on disk is conceptually merged in SSTable. Some reads may find the required in memory itself and some may have to search the data on SSTable if they don't find it in the memory. When this memory fills up the data on this memory is dumped into a SSTable file. Eventually it may be decided to combine all this small table into one large SSTable on disk. Bloom filters are used to find whether data can be present for a particular row column combination in SSTable.

5. BIGTABLE SYSTEM STRUCTURE

The Bigtable system structure consists of Tablet servers, Master servers, Bigtable client library and the lower level building blocks namely the Cluster scheduling system, GFS and the Lock service.

Tablet servers manage multiple tablets by serving data and accepting writes to the data stored in various tablets across the system. Master server on the other hand manages all this tablet servers by managing the metadata operations, it also creates new tablets and looks after the load balancing .The clients application opens the cells and then opens the table, this creates a small data structure in the client library. It's this same client processes which can issue read and write operation directly to the tablet server. Bigtable client library has all the API's and client side routines to use the system functionalities.

It can read or write to a tablet server, perform metadata operations on the master server etc. The lower level building block are the blocks on which the entire Bigtable system works on and this system makes use of this low level system blocks to get their work done. The cluster scheduling system handles system failure and monitors the entire system. GFS is yet another proprietary system designed at google which is actually used by the Bigtable to store tablet data and mutation logs. It is enhanced for Google's core data storage and usage needs (primarily the search engine), which can generate enormous amounts of data that needs to be retained. Lock service also holds some of the metadata and also elects master server among servers competing to be master servers, at a given point only one server can act as master server the rest other just wait to acquire the master lock.

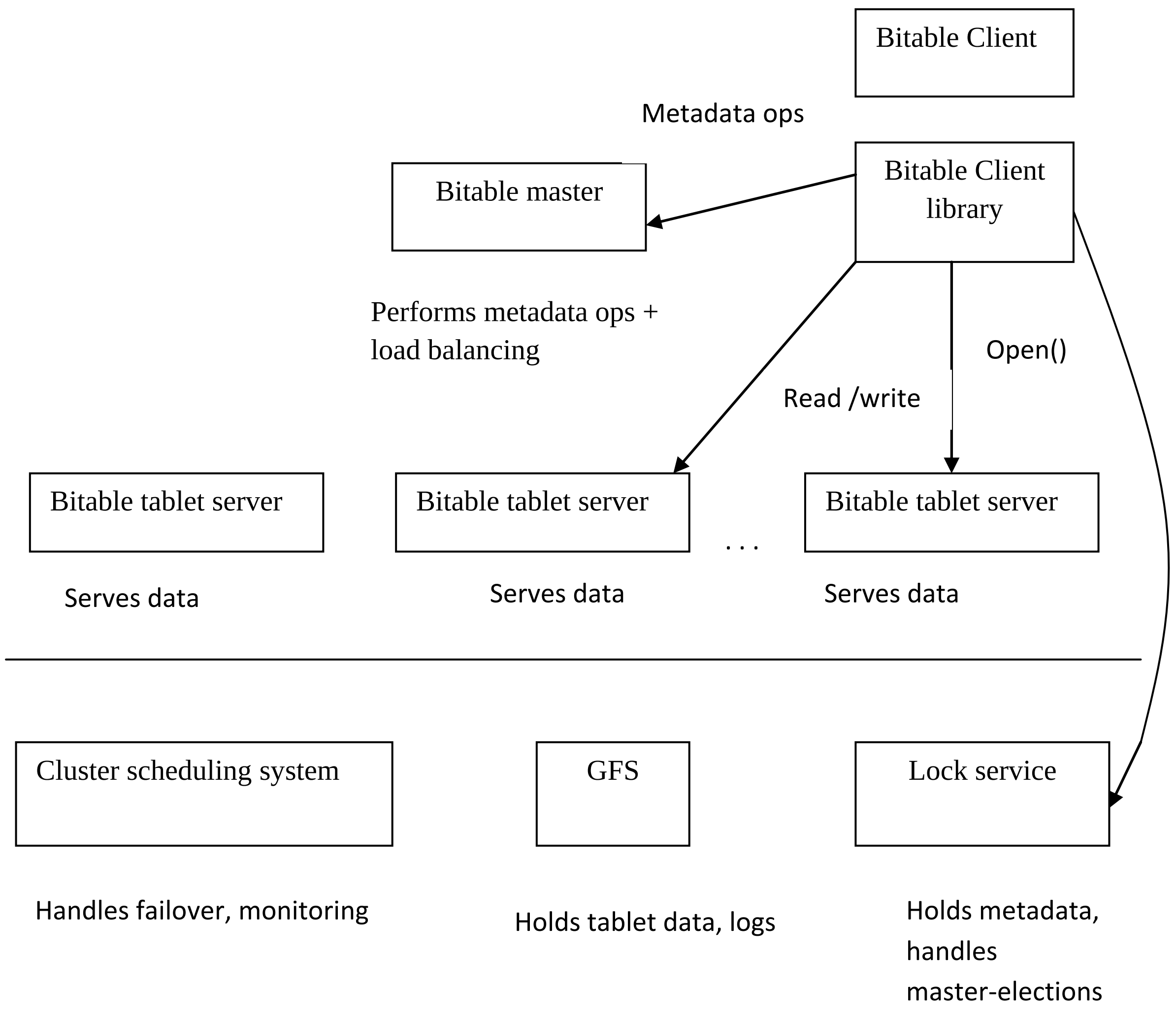

Fig 3 : Bigtable System Structure

6. LOCATING TABLETS

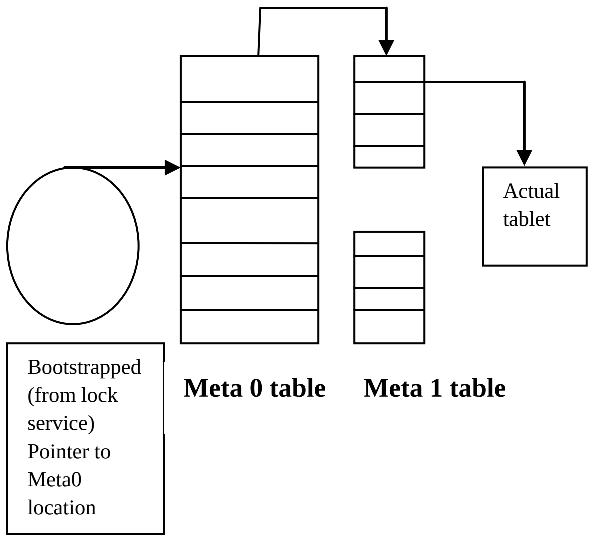

Fig 4 : Tablet location hierarchy

In the Tablet location hierarchy the first step is the file that stores location of Meta 0 Table and that file is in the lock service. It is a chubby file[1] which holds a pointer to the location of Meta 0 table. A Meta 0 table holds all the required information to locate the appropriate row from the Meta 1 table. The Meta 0 tablet it is never split to ensure that the tablet location hierarchy has no more than three levels. It is kept as a whole on a single machine. The row entry in the Meta 1 table has the actual location of the tablet that is required by the client. Each table row stores roughly 1KB of data in memory. There is a limit of 128 MB tables, so the three-level location scheme is sufficient to address 234 tablets which roughly is equal to 261 bytes in 128 MB tablets[2].

The clients library caches the tablet locations. Tablets are stored in memory so no costly GFS access is required also the aggressive prefetching of tablets locations further improve the overall response time of the system.

7. API

Bigtable API provides provisions to perform various operations on Bigtable. It provides function to create or delete tables and column families. Write operation is atomic in Bigtable.
Few functions are explained below

7.1 Write operation

Set() **:** Writes to cells in a row.
Delete() **:** Delete cells in a row.
DeleteRow() **:** Delete all cells in a row.

7.2 Read operation

Scanner ()**:** Reads arbitrary cells in Bigtable.

There is only one kind of operation for the read. Each row read is atomic in nature.

8. CONCLUSION

This paper presents an overview Google's Bigtable and its components. Bigtable can work with almost all types of data because it treats all its data to be arbitrary strings. A brief overview of the data model of the Bigtable and the Bigtable system structure is done. Compactions has been explained along with the entire Tablet location hierarchy. The API support with the Bigdata is simple and very easy to use. Since it's first use in 2005 big table has powered many systems at google. The systems using Bigtable include projects like Google's web index, Google Earth, Google finance and many more. It is a powerful model to manage data over huge clusters and has proved to be very efficient in its use over time and with various different types of data. In May 2015 Google launched a public version of the Bigtable called the Cloud Bigtable as a part of Google's cloud services.

Future Scope involves a study as to how consistency can be achieved in multi-row updates, improved built-in support for SQL like operations when and where required, handling hardware issues more efficiently when dealing with Meta0 table as Meta0 hardware is the site of single point of failure, a better programming support for handling Bigtable operations more efficiently.

9. REFERENCES

[1] Mike Burrows, The Chubby lock service for loosely-coupled distributed systems (OSDI 06).

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable : A Distributed Storage System for Structured Data (OSDI 06).

[3] Washington.eduhttp://www.cs.washington.edu/events/colloquia/archive?id=437

[4] Official documentation of Google cloud Bigtable: https://cloud.google.com/bigtable/docs

[5] Google I/O 2009 – Mercurial on Bigtable https://www.youtube.com/watch?v=ri796Hx8las

[6] Google I/O 2008 , Underneath the Covers at Google: Current Systems and Future Directions Jeff Dean (Google), https://www.youtube.com/watch?v=qsan-GQaeyk

[7] stackoverflow.com/questions/2576012/what-is-an-sstable

[8] Whitchcock, Andrew, Google's Bitable:http://andrewhitchcock.org/?post=214

[9] http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

[10] Bill howe on Bigtable at the University of Washington https://class.coursera.org/datasci-002/lecture/107

[11] https://en.wikipedia.org/wiki/Bigtable#cite_ref-o.27reilly_12-0

[12] http://www.slideshare.net/zafargilani/bigtable-15039321?qid=b7358981-5dfe-4a59-ae29-d933d0da9a46&v=qf1&b=&from_search=11

[13] http://read.seas.harvard.edu/cs261/2011/bigtable.html