Scientific Journal Impact Factor (SJIF) : 3.518
International Journal of Modern Trends in Engineering and Research
IJMTER
ISSN (ONLINE) : 2349 - 9745
ISSN (PRINT) : 2393 - 8161

Distributed Database Case Study on Google's Big Tables

Anjali diwakar dwivedi[1], Usha sadanand patil[2] and Vinayak D.Shinde[3]
[1,2,3] *Computer Engineering, Shree l.r.tiwari college of engineering*

Abstract- Big table is a distributed storage system which is compressed, high-performance, column-oriented data storage system. It is used to handle very large amount of data in size of peta-bytes among thousands of servers. Google BigTable serves as the database for applications such as the Google Personalized Search, Google App Engine Data store, web indexing, Google Earth and Google Analytics. BigTable successfully serves as a high-performance solution for all theses Google product. BigTable has a large impact on NoSQL database design. In this paper we describe the simple data model provided by Big table, which gives clients dynamic control over the layout of data and format, and we describe the design and implementation of Big table.

I. INTRODUCTION

In case of distributed databases data storage is diverse at different remote locations. These databases were stored in dedicated servers. Companies were in need of many servers to store their huge databases. The RDMS - Relational Database Management Systems was used by organizations to handle their data traditionally. In order to maintain large amount of data scalability and distribution came into focus. In the 1970's, it surfaced that it is difficult to distribute a RDMS while keeping all the features and guarantees. New solutions for the distribution of databases were needed, with the rise of the large Internet systems, with their huge amount of data and requests. For this reason data management systems called NoSQL DMS (or NoSQL data stores), have been created and are became more important. The term NoSQL introduced in 2009 abbreviated as "Not Only SQL". Although at this time, popular Apache Cassandra distributed database (NoSQL) existed in market or were in internal company use (like Google BigTable), a major boost in development has occurred since then.

A distributed database is a database in which portions of the database are stored on multiple computers within a network. Users have access to the portion of the database at their location so that they can access the data related to their tasks without interfering with the work of others. A centralized distributed database management system (DDBMS) manages the database as if it were all stored on the same computer. The DDBMS synchronizes all the data periodically and, in cases where same data is accessed by multiple users, guarantees that updates and deletes performed on the data at one location will be automatically reflected in the data stored somewhere else[1]. The Functions, Processing logic, Control and Data is distributed.

Bigtable development started in 2004. Nowadays a number of Google applications use Bigtable for example MapReduce(which is often used for generating and modifying data stored in Bigtable), Google Maps, web indexing, Blogger.com, My Search History, Google Earth, Gmail, Google Code hosting, Google Book Search and YouTube. Google's motive for developing its own database includes better control of performance characteristics and scalability [2].

II.BIG TABLE: DATA MODEL

A Bigtable is a distributed, sparse, persistent multidimensional sorted map. The map is indexed by a column key, row key and a timestamp. Map values are an array of uninterrupted bytes.
(Row: string, column: string, time: int64) → String

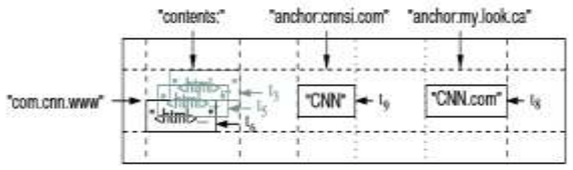

Figure1: BigTable Data Model

The above example illustrates table that stores Web pages. Row name is Reversed URL. The column family "contents" contains the page contents and the column family "anchor" contains the text of any anchors which reference the page. Home page of CNN is referenced by both MY-look home pages and Sports Illustrated so the row contains columns named anchor: my.look.ca and anchor: cnnsi.com. Each anchor cell has one version; the contents column has three versions, at timestamps t3, t5, and t6. A huge collection of web pages copy and related information that could be used by many different projects is maintained by a table and his table is called as the WebTable.
The URLs are used as row keys, various forms of WebPages as column names, and the contents of the WebPages are stored in the contents: column under the timestamps when they were fetched in WebTable, as depicted in Figure1.

A. Rows

Each and every row in a table linked with row key that is an arbitrary string of up to 64 kilobytes in size, although most keys are much smaller. Every read or write of data covered by a single row key is atomic. Bigtable uses row key to maintain data in lexicographic order. Bigtable maps row key to the address of each row. The row area for a table is dynamically divided. Each row area is called a tablet, which is the element of distribution and load balancing. Reads with short row ranges need communication with only a small number of machines and are efficient. Clients can take advantage of this property by selecting their row keys so that they get good locality for their data accesses. In Web table, pages in the similar domain are grouped collectively into contiguous rows by reversing the hostname components of the URLs. Much stronger locality will be provided by URLs with the domain portion reversed for data accesses which is helpful for domain analysis. For example, the key com.google.maps/index.html stores data for maps.google.com/index.html. Domain and host analyses made more efficient by storing pages from the same domain.

B. Column Families

Grouping of Column keys into sets called *column families*, which figures the basic unit of access control. Usually similar type of data is stored in a column family. It's required to create column family before storing data under any column key in that family. After creating family, we can use any column key within the family. It is our wish that the number of distinct column families in a table be small (in the hundreds at most), and that families hardly change during operation. In contrast, a table may have an unbounded number of columns.

The *"family: qualifier"* is the syntax to name a column key. Names of column family must be printable whereas equalizers may be arbitrary strings. An example column family for the WebTable is language, which stores the language in which a webpage was written. Column key stores each web page's language ID and there will be only one column key in the language family.

The "anchor" is another useful column family for this table. Each column key in this family represents a single anchor, as shown in Figure 1. The name of the referring site is the qualifier and the cell contents is the link text. Column-family handles both disk and memory accounting and access control. These controls in our Web table example will allow us to manage many different types of applications: a few that add new base data, a few that read the base data and create derived column families, and few that are only allowed to view existing data (and possibly not even to view all existing families for privacy reasons).

C. Timestamps

Every cell in a Bigtable can contain multiple versions of the similar data; these versions are indexed by timestamp. Timestamps of Bigtable are of 64-bit integers. Either they can be assigned by Bigtable, in which case they represent "real time" in microseconds, or be explicitly assigned by client applications. Unique timestamps need to be generated by applications themselves that avoid collisions.

In-order to read most recent versions of cell, different versions of a cell are stored in decreasing timestamp order. Burden of programmer in managing the large datasets and associated versions can be reduced by setting tables which apply garbage collection on older versions automatically. The client can specify either that only the last n versions of a cell be kept or that only new-enough versions be kept (e.g., only keep values that were written in the last seven days). In our WebTable example, we set the timestamps of the crawled pages stored in the contents: column to the times at which these page versions were actually crawled. The garbage-collection mechanism described above lets us keep only the most recent three versions of every page[3].

III. BIG TABLE: OVERALL ARCHITECTURE

A Bigtable is fragmented into tablets, with a given tablet being approximately 100–200 megabytes in size. Managing tablets and supporting operations for accessing and changing the associated structured data are main tasks of the Bigtable infrastructure. Mapping the tablet structure onto the file system (GFS) and ensuring effective load balancing across the system are task of implementation. Bigtable makes heavy use of both GFS and Chubby for the storage of data and distributed coordination. A single instance of a Bigtable implementation is known as a cluster, and a number of tables can be stored in each cluster.

The architecture of a Bigtable cluster is similar to that of GFS, consisting of three major components (as shown in Figure 2):

- A library component on the client side;
- A master server;
- A potentially large number of tablet servers.

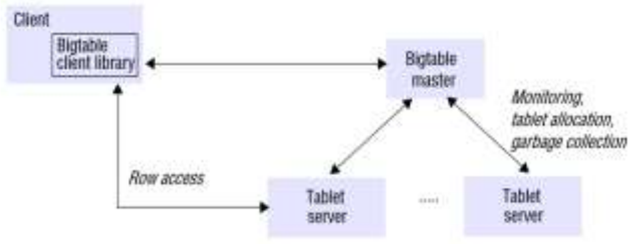

Fig2: Architecture of Bigtable

In terms of scale, as of 2008, 388 production clusters ran across multiple Google machine clusters, with an average of around 63 tablet servers per cluster but with many being significantly larger (some with more than 500 tablet servers per cluster). The number of tablet servers per cluster is also dynamic, and to increase throughput, new tablet servers are added to the system at runtime.

There are two key design decisions in Bigtable which are similar to those made for GFS. At First, single master approach is adopted by Bigtable, for reason - to maintain a centralized view of the state of the system thus supporting load-balancing decisions and optimal placement and because of the inherent simplicity in implementing this approach. Secondly, a strict separation maintained by the implementation between data and control with a lightweight control regime maintained by the master and data access entirely through appropriate tablet servers, with the master not involved at this stage (maximum throughput is ensured when accessing huge datasets having direct interaction with the tablet servers).

Following are the control tasks associated with the master:

• monitoring the status of tablet servers and reacting to both the availability of new tablet servers and the failure of existing ones;
• assigning tablets to tablet servers and ensuring effective load balancing;
• Garbage collection of the underlying files stored in GFS.

Bigtable goes further than GFS in that the master is not involved in the core task of mapping tablets onto the underlying persistent data. This means that Bigtable clients do not have to communicate with the master at all, a design decision that significantly reduces the load on the master and the possibility of the master becoming a bottleneck. Bigtable uses GFS for storing its data and uses Chubby for implementing monitoring and load balancing [4].

Chubby is used by BigTable to track tablet servers: When starting up each tablet server creates and acquires a lock on a uniquely named file. The directory was monitored by master where these files are created and can react to certain conditions:

If a new file is created (and a lock acquired) this means that a new tablet server is starting up and data can be assigned to this node. If a lock is released (because of a timeout), the master knows that a node lost connection and should be considered down. When starting up the Master itself scans the lock directory in Chubby and then ask every tablet server for their tablet assignments. The tablet servers hold parts of their tablets which they store in memory while the whole tablet is stored on the distributed GFS. Write requests to a tablet are handled by the matching (single) tablet server. This server appends the write requests to its commit-log (stored on GFS) and then alters it data-set (in memory and on GFS). Read requests are answered by the tablet server either from memory or from the data-files on GFS. Periodically the node also dumps the current dataset onto the GFS and thus makes the commit-log shorter and easier to handle [5].

IV. IMPLEMENTATION

The Bigtable implementation consists of three important components: **a library**-that is linked into every client, **one master server-**it assign tablet to tablet server and also balance the tablet server load ,and add garbage collection of files in gfs, and **many tablet servers-** Tablet servers can be added or removed from a cluster to accommodate changes in workloads. Each tablet server manages a set of tablet server (typically we have somewhere between ten to a thousand tablets per). It also perform read and write operations on the tablets that it has loaded and break tablets that have grown large. In Bigtable clients do not rely on the master for tablet location information. Therefore, the master is lightly loaded.

A. Tablet Location

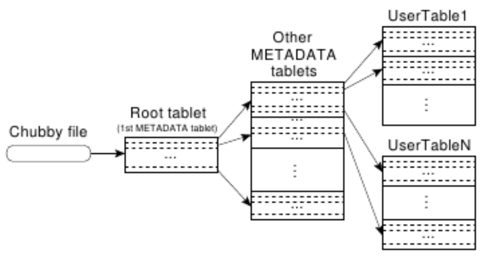

Figure 4: Tablet location hierarchy.

We have use three level hierarchy to store tablet location information. The first level is file stored in chubby that contain information about the root tablet. The root tabletcontainsthe location of all tablets in a special metadata table.A metadata tablets contains the location ofuser tablets.The metadata table stores the location of a tabletunder a row key .each metadata row stores around 1kb of data in memory our three-level location hierarchy is capable to store 261 bytes in 128mb tablets.

B. Tablet Assignment

One tablet server contains each tablet. The master keeps track of live tablet, including which tablets are assigned or unassigned. When a tablet is unassigned, and a tablet server has a sufficient space available then the master will send a tablet load request to the tablet server. Chubby is used to keep track of tablet servers .When a tablet server is initialize, it acquire and create an exclusive lock on unique filename in a special Chubby directory. The master keeps look on this directory to discover tablet servers. A tablet server stops serving if it loses its exclusive lock: e.g. due to a network partition the server can lose its chubby session. (Chubby provides a mechanism that allows a tablet server to check whether it holds its lock or not without interrupting the network traffic.).A tablet server can gain reacquire exclusive lock on its file only if that file still exist its file. If file does not exist, then the tablet server can never serve again, so it kills itself. Whenever a tablet server terminates it releases its lock. The master is responsible for detecting terminated tablet server and reassigning those tablets to the tablet server, for detection purpose the master will periodically ask the tablet server about the status of their lock, if master cannot reach the server during its termination stage then master will acquire exclusive lock on that server file. If the master is able to gain the lock, then the Chubby is live. Once a server's file has been deleted, the master can move all the tablets into the set of unassigned Tablets. To ensure that a Bigtable cluster is not vulnerable the master kills itself if session of chubby expires. However ,as stated above, if master fails it will not change the assignment of tablets. When a master is initiated by the cluster management, it needs to discover the assignment of current tablet before it can change them. The master runs the following steps at startup. (1) The master acquires a unique lock in Chubby, which prevents concurrent master instantiations. (2) The master scans the server’s directory in Chubby to and the live servers. (3) The master communicates with every live tablet server to discover the assignment of tablets to each server. (4) The master scans the metadata table to learn the set of tablets. Whenever this scan encounters a tablet that is not assigned, the master adds that tablet to the set of unassigned tablets, which makes that tablet eligible for tablet assignment. One complexity is that the scan of the metadata table is not completed until the metadata tablets have been assigned to the server. Therefore, before running this scan (step4), if an assignment of the root tablet was not discovered during step 3, the master adds the root tablet to the set of unassigned tablets. The root tablet contains the names of all meta data tablets. This ensures that the root tablet will be assigned because the master knows about all the tablet after it has scanned the root tablet. The set of existing tablets only changes when a table is deleted or created, two existing tablets are merged to create one larger tablet, or an existing tablet is divided into two smaller tablets. The master is able to keep track of these changes. Tablet splits are treated special since they are started by a tablet server. The tablet server perform the split by recording information for the new tablet in the metadata table. When the split is performed, it gives notification to the master. In case the split notification is lost (because either the tablet server or the master is not alive), the master detects the new tablet when it asks a tablet server to load the tablet which is divided now. The tablet server will give notification to the master about the split of server, because the entries in the metadata table will specify only a part of the tablet that the master has asked it to load.

C. Tablet Serving

The committed state is stored in GFS, as shown in figure 5. The newly updated log is store in the memory in sorted form known as memtable. The previously updates are stored in a SSTables.

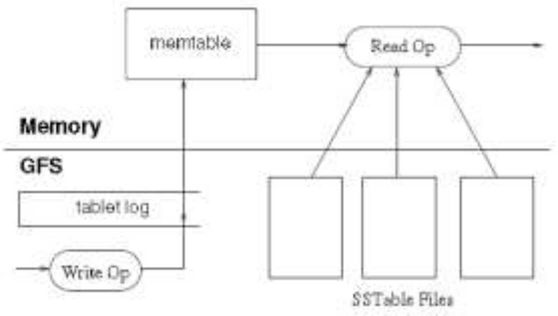

Fig 5: Tablet Representation

To reacquire a tablet, a tablet server checks its metadata. The metadata consist of SSTables that contain a redo point and tablets. These reads the index of the SSTables to the memory and redevelops the memtable by adding all the updates from the redo points. When a write operation occurs at a tablet server, the server checks that sender is authorized. Authorization is checked by reading the list of authorizes writers from a chubby file. After the write operation is performed, its contents are added into the memtable. When a read operation occurs at a tablet server, it is similarly checked for well-formed and proper A valid read operation is performed on a merged sequence of memtable and SSTables. Incoming write and read operations can be performed while tablets are merged and split.

D. Compactions

As write operations are performed, the memtable size increases. When the memtable size exceeds the limit then a new memtable is created. This minor compaction has two goals :it increases the memory usage of the tablet server, and it decreases the amount of data from the log if this server dies. Each minor compaction creates a new SSTables. If this behavior is unchecked, read operations might need to combine updates from a numbers of SSTables. Instead, we will periodically executing a file by merging compaction in the background. A merging compaction reads all the contents of some memtable and SSTables, and writes out anew SSTables. A merging compaction that writes all SSTables into single SSTables it is called a major compaction. A major compaction, develop an SSTables that contains no deleted data. This compaction allow Bigtable to re-acquire resources that was used by deleted data, and also allow it to ensure that deleted data should be disappears from the system on timely basis.

V. REAL APPLICATIONS

A. Google Analytics

Google Analytics (analytics.google.com) is a service that is used by the webmaster that analysis traffic pattern, it provide overall statistics like no. Of unique visitor per day. It provides aggregate statistics; webmasters code a JavaScript program in their web pages. This program is invoked whenever user visits this page. Google Analytics collect all this data and makes it available to webmasters. We describe two of the tables which are used by Google Analytics. The raw click table (size approximate200 TB) that maintains a row for every end-user session. The row name is an attribute that contain the website name and the time when session time is created. This table is compressed to 14%of its original size. The summary table (size approximate 20 TB) contains various summaries for each website. The raw click table is responsible for generating summary table. This table compresses to 29% of its original size.

B. Google Earth

Google earth is a collection of services that provide user to access satellite imagery of the world surface using Google earth customer client software and web-based Google map. This product helps user to navigate throughout the world wide surface. This system uses two tables one table for pre-process data and another table for serving client data. The pre-processing pipeline uses one table to store raw Imagery. During pre-processing, the imagery is cleaned and processed into final serving data. This table contains approximately 70 terabytes of data so it is served from disk. The images are compressed already, so Bigtable compression is disabled. The serving system uses one table to address data stored in GFS. This table is relatively small (size approximate 500 GB), but it serves tens thousands of queries per second with low latency. As a result, this table is hosted on several hundreds of tablet servers and contains in memory column families.

C. Personalized Search

Personalized Search is a service that records user queries and variety of Google application such as web search, news, images. Users can revisit their old queries by browsing the history .Personalized Search stores every user's data in Bigtable. Each user is assigned a row based on their unique id. All user activity are stored in a table. A column is stored for each type of action (for example, there is a column that stores all e-commerce sites).The Search data is replicated across many Bigtable clusters to increase availability of resources or information to the clients. The current system now a day's uses application subsystem that is built in servers. The design of the Personalized Search storage system is also used by other groups to add new user information in personalized search columns, and the system is now used other Google application that need to store user configuration and settings.

VI. ADVANTAGES OF BIGTABLE

Big table supports access control at a column level. Bigtable has its own DSL (digital subscriber line) for processing stored data. Correction safety, via CRC checksums. It support inter-cluster replication .Bigtable supports transactions, even cross table transaction. Bigtable index building is very powerful .Bigtable is technically only available through PaaS , like Google App Engine. Bigtable is also highly scalable and has been used to host multiple petabytes in a single 'cell'.

VII. DISADVANTAGES OF BIGTABLE

Big table offer a much simpler API than MySQL and Postgre SQL, so there is no built-in support for secondary indexing and SQL. Applications have to build these features for themselves. As for MongoDB, Bigtable-based projects tend to be more focused on and availability. BigTable still offers fewer features for accessing data, like its indexing capability. Developers just have to download binary and start writing JSON (which become BSON) documents right away using a variety of clients for different languages, whereas Bigtable implementations require setting up multiple components including Hadoop's HDFS, and Zoo Keeper, which are not trivial. Bigtable is highly consistent for single-row updates, but offers no consistency for multiple row updates or cross table update. Applications have to manage any consistency requirements across rows or tables themselves. It has no ACID properties. It depends on your app if you can live without consistency.

VIII. CONCLUSION

We have described about Bigtable, a distributed system forstoring structured data at Google. Application that uses Bigtable have great performance, scalability and availability. Several Google properties is using Bigtable successfully. Google has many advantages by building its own storage solution by having full control and by removing bottleneck as they rise.

REFRENCES

[1] http://searchoracle.techtarget.com/definition/distributed-database
[2] https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&vd=0ahUKEwjspuP8qPLAhWRj44KHZrsAhwQFggbMAA&url=http%3A%2F%2Fwww.springercom%2Fcda%2Fcontent%2Fdocument%2Fcda

_downloaddocument%2F9781447156000c2.pdf3FSGWID%3D00451430025p175494435&usg=AFQjCNExs0mQw5tT7X2XQMlwLbgowmgQ&bvm=bv.117868183,d.c2E
[3] https://en.wikipedia.org/wiki/Bigtable
[4] https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiz647DpuPLAhVMGo4KHcADBHYQFggbMAA&url=http%3A%2F%2Fresearcgoogle.com%2Farchive%2Fbigtableosdi06.pdf&usg=AFQjCNHWt7UiCw7feq5ygj2Y5pWeQLPZ9Q&bvm=bv.117868183,d.c2E
[5] https://www.google.co.in/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-