Review on Bigtable, A Distributed Storage System for Structured Data

Bigtable is a distributed storage system designed to manage structured and semi-structured data at a very scale. The data size can scale up to petabytes across multiple servers. This design was first proposed and published by Google, Inc. in 2006[1]. The motivation behind such storage architecture was to achieve high availability, high performance, scalability and wide applicability. Successful adoption of Bigtable by many Google projects like Google Analytics, Personalized Search, Google Earth, Orkut, and many more proves the effectiveness of such a design. In many ways, Bigtable is like a database. It has a data model and client API to perform the read and write operations. This review gives a brief overview of the data model, the client API's and infrastructure on which the data store is built. Additionally, we will also discuss some of the advantages of using such a data store and also some disadvantages of using this system.

As the name suggests, BigTable can be visualized in a tabular store consisting of rows and columns, but very much different from the relational database. At the physical level its implemented as a persistent, distributed, multi-dimensional, sorted map. A String of size 64KB is used as a key for each row and column.[1] These keys along with timestamp index the map, and the value in the map is an interpreted array of bytes. This helps in storing semi-structure data. Since Bigtable is meant to store a large volume of data, the complete data is chunked according to key ranges of rows and each one is called a tablet. These tablets are then distributed using load balancing to different servers for enabling parallel computing of queries. Each of these tablets can have large number of columns, but these columns are grouped together into column families. These column families form the basic unit for implementing access control mechanisms. It is assumed that the number of column families will be limited typically in the order of hundreds. The columns are grouped together into families based on relationship between the contents stored in those columns. Since the data in the same family is normally accessed together, data in columns belonging to same family is compressed and stored together. Thus at time of retrieval the related data can be retrieved in minimal disk lookups. Also to further increase the efficiency, families with frequent requests can be configured to be stored in-memory for better performance. Lastly the timestamp in the index protects the existing data from being updated, which means updates are processed as new entries. This property encourages massive parallelism, as the data, which is being read, cannot be modified by another parallel process. This also demands for regular garbage collection, as we need to clean the data store by deleting obsolete data. Figure1 is an example of how information from the web crawler can be stored.

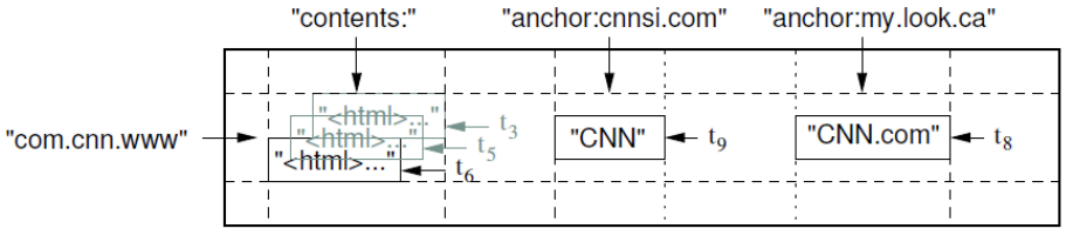

Figure 1: Storing a single row with multiple timestamps containing information about a web page. "com.cnn.www" is row key; "anchor" is column family and "connsi.com" is column name under the family.. [1]

Bigtable provides API's to add and delete tables, column families and change cluster and family metadata like access control rights. Bigtable currently supports single-row transactions only but a client can submit a batch script written in Google's internal data processing language for batch processing of queries involving multiple rows. Since the index already contains timestamp, Bigtable also allows filtering of queries using timestamp. Bigtable doesn't provide a full fledge query language like SQL, since there are no complex queries like Joins supported. Hence additional modules like query parser and query optimizer are not required. Only way to run complex queries is via scripts and it allows cells to be used as counters for aggregation based queries. Bigtable can also be used with MapReduce framework to perform large-scale parallel computations across distributed tablets.

Bigtable is built on other pieces of Google infrastructure. These include Google's distributed File System (GFS) to store data and log files, which internally uses Google's Sorted String Table (SSTable) file format. Most of the features in the data model are ready supported by this file format. Bigtable is not

allocated a dedicated server and memory pool. Hence it also uses a cluster management system for scheduling jobs, managing resources, dealing with failures, and monitoring system health. Bigtable heavily relies on highly available Google's distributed lock service called Chubby. Bigtable is based on the classic master slave architecture and chubby is used to synchronize between the nodes of the cluster. At a time there is one master server and several tablet servers. Master node ensures high availability of all the tablets, garbage collection and load balancing on each of the tablet servers. Client directly contacts the tablet server for data. The location information of each tablet is stored in a root metadata file on Chubby. Master node also uses this file to keep track of all the tablets allocated to the servers and any unallocated tablet is allocated and the location is updated in this file. A commit redo log file is maintained on each tablet server for recovery. All the tablets on that server share the same commit log. A regular checkpoint is done on commit files to reduce the time of recovery and memory footprint. A lot of refinements are done on top of this to increase the efficiency of the system. Some of these refinements include compression of data on disk, ensuring locality of columns in same family to minimize Disk IOs and maintaining single commit log with 2 write heads for high availability and to reduce overhead of multiple commit files.

The system performance is evaluated by evaluating the performance of random reads, sequential reads, random writes, sequential writes and scans. Random reads involves transfer of 64KB of block while only 1KB of data is read. This is not quite efficient since each row requires transfer of complete block. Sequential reads are much more effective than random reads as one fetch effective retrieves a range of rows from the tablet and also since the tablet is distributed in such a way that column families stay together. Cache is also implemented to improve the performance in each of the cases. Sequential write and random write are both done via a single commit log file hence their performance is almost the same. The writes happen to the commit file in memory and then when a certain limit is reached a new SSTable is created for the same and the memory gets flushed. Scans are also as faster as the tablet servers can return a large block of data at a time. The client also maintains a cache of the blocks retrieved and hence is very much effective in case of scans and sequential reads. In case of random reads normally there will be a lot of cache misses and will require a frequent transfer of data over the network.

The main advantage of using a Bigtable is that even though it gives a tabular representation of data, it does not require a special query language and expensive join operations to retrieve data. The sequential reads, sequential and random writes and scans are further optimized for greater efficiency. Being a distributed system, which can easily scale out, there are no restrictions on the number of rows and columns. The system also ensures high availability. The system also has some limitations that need to be addressed. The system only has a single index and retrievals happen based this index. The system should be extended to store secondary indexes. These indexes can also be stored as a two level hierarchy where the metadata file is used to locate the tablet server and then the tablet server maintains a separate file which points to actual record depending on the index. This was implemented later.[4] The system needs to support features with respect to data security. These include features like data encryption. Finally the network can form a bottleneck in the system when serving over the Internet. Hence the system can be evaluated to see if maintaining multiple copies of the data will improve the performance and also protect from data loses. [2][3]

One of the applications built on Bigtable is Google Analytics, which uses Bigtable to store raw clicks on a particular web page and uses this information for analytics. Google analytics actually maintains 2 tables one for raw clicks and another for summary information and it regularly runs MapReduce jobs to analyze and cleans the raw clicks table and updates the summary table. Other applications are Google Earth, which uses Bigtable to store preprocessed imagery data, Personalized Search and Orkut to store social connections. Thus, in conclusion Bigtable is a richer key-value store aimed to support semi-structured data for efficient read/write performance at a very large scale. [3]

References.

1) http://static.googleusercontent.com/media/research.google.com/en/us/archive/bigtable-osdi06.pdf
2) http://en.wikipedia.org/wiki/BigTable
3) http://mas.cmpe.boun.edu.tr/wiki/lib/exe/fetch.php?media=courses:cmpe473-fall-2012:bigtable.pdf
4) https://cloud.google.com/appengine/articles/storage_breakdown