© 2017 IJSRST | Volume 3 | Issue 4 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

A Novel Approach of Design & Implementation of Cloud Big Table

Dr. V. Goutham
Professor and HOD of CSE in Teegla Krishna Reddy Engineering College, Telangana, India

ABSTRACT

Cloud Bigtable, which is sparsely populated table to scale billions of rows and thousands of columns, enabling storing petabytes or terabytes of data across thousands of commodity servers. Google has had to resolve the challenges that many companies bearing the difference is the sheer scale of the problem. They've often had to design entirely new approaches to meet the demand of their businesses. Over the past decade, Google has developed many traditional solutions to carry their own products and services. They've proposed many of these internal solutions in white papers and so many have developed into open source projects that now are the footing of the Hadoop ecosystem. Five foundational Google projects that have changed the era of big data landscape forever. Many projects at Google store data in Bigtable, including Google MapReduce (Apache Hadoop), Google Bigtable (Apache HBase), Google "Borg"(Apache Mesos), Google Chubby(Apache Zookeeper), Google Dremel(Apache Drill). These are just a few examples of the ways Google has set the stage for the Bigdata revolution. Bigtable has successfully furnished a flexible, high-performance solution for all of these Google products. In this paper we represent the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, features of cloud Bigtable, and we describe the design and implementation of Bigtable. This paper describes overview of the client API, the underlying Google infrastructure on which Bigtable depends, fundamentals of the Bigtable implementation; We describe several examples of how Bigtable is used at Google.

Keywords : Cloud, Hadoop ecosystem, Apache Mesos, MapReduce, Apache Zookeeper, Apache HBase, API, Bigtable

I. INTRODUCTION

We have designed, implemented, and installed a distributed storage system for organizing structured data at Google called Bigtable. Bigtable is intended to reliably scale to petabytes of data and thousands of machines. Bigtable has attained several goals: wide-ranging applicability, scalability, high performance, and high availability. In many ways, Bigtable look like a database: it spilts many implementation strategies with databases. Parallel databases and main-memory databases have obtained scalability and high performance, but Bigtable produces a different interface than such systems. Bigtable doesn't support a full relational data model; instead of, it provides clients with a simple data model which supports dynamic control over data layout and format, and it allows clients to reason about the locality properties of the data presented in the underlying storage. Data is listing using row and column names that can be arbitrary strings. Bigtable also considered data as un-interpreted strings, although client frequently serialize various forms of structured data and semi-structured data into these strings. Clients can manage and control the locality of their data through careful choices in their schemas. At last, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

II. CLOUD BIGTABLE FEATURES

A quick, fully controlled, massively scalable NoSQL database service is a Cloud Bigtable. It has several features such as High Performance, Security &

Permissions, Low Latency Storage, Global Availability, Fully Managed, Redundant Autoscaling Storage, Scaling, HBase Compatible, Seamless Cluster Resizing.

Cloud Bigtable has a big performance under high load than alternative products. It means, large applications and workflows are faster, more reliable, and more efficient running on Bigtable. All big data is encrypted both in-flight and at rest. We have rich control over who has access to the data stored in Cloud Bigtable. Cloud Bigtable utilizes a low-latency storage stack. Cloud Bigtable is available in all regions around the world, allowing client to place client's service and data exactly whereever client want it. Cloud Bigtable is provided as a fully managed service, meaning you spend your time developing valuable applications instead of configuring and tuning database for performance and scalability. Furthermore, Google's own Bigtable operations team monitors the service to safeguard issues are addressed quickly. Cloud Bigtable is construct with a redundant internal storage strategy for giant durability. You don't need to configure separate storage or disks, and you only pay for the amount of storage what you are using. During operation without the need for a restart, granting efficient use of resources and helping client applications and workflows stay up and running. Additionally, its native RPC-based API, Cloud Bigtable provides an HBase-compatible interface. This authorizes portability of applications between HBase and Bigtable. Cluster nodes of Cloud bigtable can be dynamically added and removed in Cluster Resizing.

III. BIGTABLE SYSTEM ARCHITECTURE

Bigtable is a compressed, big performance, and proprietary data storage system construct on Google File System, Chubby Lock Service, Sorted Strings Table simply called SStable (log-structured storage like Level DB) and a few other technologies of Google. Bigtable maps two arbitrary string values (row key and column key) and a timestamp (hence three-dimensional mapping) into an associated arbitrary byte array. Cloud Bigtable is not a relational database. It can be defined as a distributed multi-dimensional sorted map. Bigtable is constructed to scale into the petabyte/terabyte range across hundreds or thousands of machines, to make it easy to add more machines to the system and automatically start taking advantage of those resources without any reconfiguration".

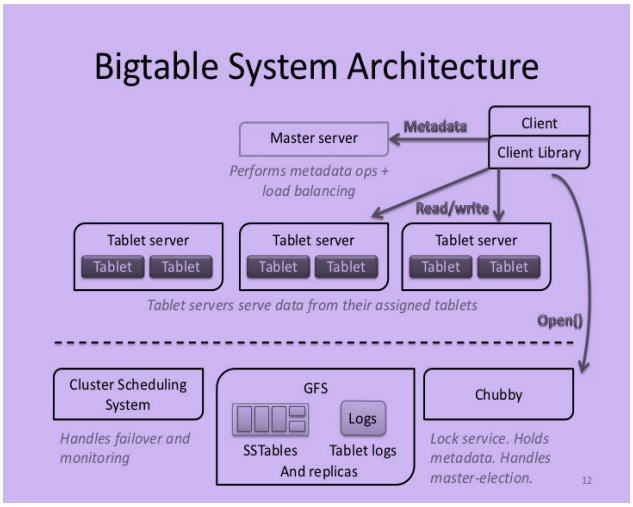

Figure 1.

3.1 Rows

The row keys in a Cloud bigtable are arbitrary strings currently up to 64KB in size, although 10-100 bytes is a typical size for most of our clients. Every read/write of data under a single row key is atomic. By using rowkey Bigtable manages data in lexicographic order. The row range for a bigtable is dynamically segregated, each row range is called a *tablet,* to help balance the workload of queires. Tablets are similar to HBase Regions. As a result, reads of short row ranges are efficient and typically it need of communication with only a small number of machines. Clients can utilize this property by selecting their row keys so that they obtain good locality for their data accesses.

3.2 Column Families

Column keys that are related to one another are typically grouped into sets called *column families.* All data stored in a column family is normally of the same type. A column family must be created before data can be stored under any column key in that column

family; after a family has been created, any column key within the family can be used. It is our main aim that the number of distinct column families in a table be small (in the hundreds at most), and that families scarcely change during operation. Besides, a table may have an unbounded number of columns.

A column key is identified using the following **syntax:** family:qualifier. Column family names must be printable, but qualifiers may be arbitrary strings.

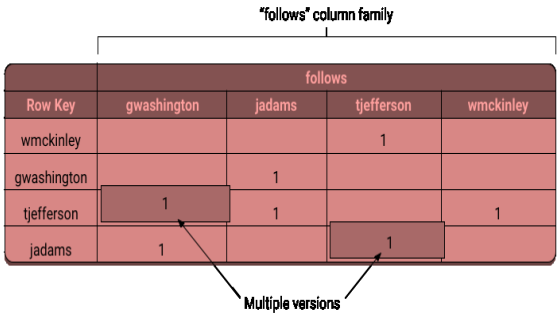

Figure 2.

3.3 Timestamps

Each cell in a Bigtable can contain numerous versions of the same data; these versions are represented by timestamp. Cloud Bigtable timestamps are 64-bit integers. They can be assigned by Bigtable, in which case they represent "realtime" in microseconds, or it can be explicitly assigned by client applications. Applications that required to avoid collisions must produce unique timestamps themselves. Different versions of a cell are stored in decreasing timestamp order, so that the most recent and latest versions can be read first. To make the management of versioned data less onerous, we support two per-column-family settings that inform Bigtable to garbage-collect cell versions automatically. The client can specify only the last *n* versions of a cell be kept, or only new versions be kept (e.g., only keep values that were written in the last seven days). If user storing data for the most recent metrics in a separate table, so there's no need to reverse the timestamps.

Writing to Bigtable:

```
//To  Open the Bigtable
Table*T1=OpenOrDie("/bigtable/web/webtable");
// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

Reading from Bigtable:

```
Scanner scanner(T);
ScanStream *stream;
stream=scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next())
{
printf("%s %s %lld %s\n",
scanner.RowName(),
stream->ColumnName(),
stream->MicroTimestamp(),
stream->Value());
}
```

3.4 Load balancing

A Cloud Bigtable table is clubbed into contiguous blocks of rows, called tablets, to help balance the workload of queries. These tablets are shared among different machines, called Cloud Bigtable nodes. In the original Bigtable whitepaper, these cloud bigtable nodes are called "tablet servers." The nodes are well organized into a Cloud Bigtable cluster, which is related to a Cloud Bigtable instance, a container for the cluster. Each node in the cluster holds a subset of the requests to the cluster. By adding more number of nodes to a cluster, client can increase the number of concurrent requests that the cluster can manage, as well as the max throughput for the entire cluster.Each Cloud Bigtable zone is controlled by a master process. Master Process balances workload and data volume within clusters. The master splits busier/larger tablets into two halfs and merges less-accessed/smaller tablets together, redistributing them between nodes as required. If a certain tablet gains a spike of traffic, the master will split the tablet in two half, then move one

of the new tablets to another node. Cloud Bigtable controls all of the splitting, merging, and rebalancing automatically, saving clients the effort of manually managing their tablets.

To get the best write performance from Cloud Bigtable, it's important to distribute writes simultaneously as possible as across nodes. One of the best way to attain this goal is by using row keys they do not follow a predictable order. For example, use the hash of a string rather than the actual string, as long as you avoid hash collisions.

IV. API

The Cloud Bigtable API produces functions for creating and deleting bigtables and column families. It also provides functions for changing cluster nodes, table, and column family metadata, such as access control rights. User applications can write or delete values in cloud Bigtable, improve values from individual rows, or iterate over a subset of the data in a table. The code for writing a table shows that uses a RowMutation abstraction to perform a series of updates. The call to Apply operates an atomic mutation to the Webtable: it adds one anchor to www.cnn.com and deletes a different anchor. The C++ code reading from big table shows that uses a Scanner abstraction to iterate over all anchors in a particular row. Clients can repear over multiple column families. There are several mechanisms for limiting the rows, columns, and timestamps provided by a scan. For example, we could restrict the scan above to only produce anchors whose timestamps fall within ten days of the current time.

V. BUILDING BLOCKS

Bigtable is construct on several other pieces of Google infrastructure. Bigtable utilizes the distributed Google File System (GFS) to store data and log files. A Bigtable cluster operates in a shared pool of machines that can run a broad variety of other distributed applications. Bigtable processes often share the same machines with processes from other applications. Bigtable be depended on a cluster management system for scheduling jobs, managing resources on shared machines, which can deal with machine failures, and monitoring status of machine. The Google *SSTable* file format is used internally to store Bigtable data. An SSTable produces a persistent, ordered immutable map from keys to values, where both keys and values are inconsistant byte strings. Operations are provided to look up the value associated with a specified key, and to iterate over all key or value pairs in a specified key range. Internally, each SSTable contains a sequence of blocks (each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to situate blocks; the block index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we will first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate and suitable block from disk. Optionally, an SSTable can be completely mapped into memory, which permits us to perform lookups and scans without touching disk.

VI. IMPLEMENTATION

The Bigtable implementation has three vital components: a library that is linked into every client, one is master server, and many tablet servers. So many Tablet servers may be dynamically added (or deleted) from a cluster to accommodate changes in workloads. The master server is in-charge for assigning tablets to tablet servers, detecting the addition and validation of tablet servers, balancing tablet-server load. Master server is also responsible for garbage collection of files in GFS. Additionally, it manages schema changes such as cloud bigtable and column family creations. Each tablet server handles a set of tablets. The tablet server manages read and write requests to the tablets which has loaded, and also distributes tablets that have grown too large. As with many single-master

distributed storage systems, client data does not move through the master: clients can interact directly with tablet servers for reads and writes. Because Bigtable clients do not depend or rely on the master for tablet location information, most clients never interact with the master. so, the master is lightly loaded in practice. A Bigtable cluster stores a number of tables. Each table consists of a group of tablets, and each tablet consists all data affiliated with a row range. Initially, each table consists of just one tablet. As a table increase in size, it is automatically distributed into multiple tablets, each approximately 100-200 MB in size by default.

VII. CONCLUSION

In this process of designing, implementing, maintaining, and supporting Bigtable, we acquired useful experience and learned various interesting lessons. we learned that large distributed systems are vulnerable to many types of pitfalls and failures assumed in many distributed protocols. Most recent projects have tackled the problem of providing distributed storage or higher-level services over broad area networks, often at Internet scale. This includes work on distributed hash tables that tackle projects such as CAN, Chord, Tapestry, and Pastry. These systems address concerns that do not arise for Bigtable, such as inflated variable bandwidth, untrusted clients, or frequent reconfiguration; decentralized control and Byzantine fault tolerance are not the goals of Bigtable. We are in the process of performing several additional Bigtable features, such as sustain for secondary indices and infrastructure for building cross-data-center reproduced Bigtables with multiple master replicas. We also begun deploying Bigtable as a service to product groups, so individual groups do not need to support their own clusters. As our service clusters scale, we will need to deal with more resource-sharing issues within Bigtable itself. Finally, we have found that there are significant advantages to building our own storage solution at Google. We have gotten a considerable amount of flexibility from designing our own data model for Bigtable. Additionally, our control over implementation of Bigtable, and the other Google infrastructure upon which the Cloud Bigtable depends on, means that we can remove bottlenecks and inefficiencies as they arise.

VIII. AUTHORS

Dr V. GOUTHAM is a Professor and Head of the Department of Computer Science and Engineering at Teegala Krishna Reddy Engineering College affiliated to J.N.T.U Hyderabad. He received Ph.D. from Acharya Nagarjuna University M.Tech from Andhra University.His research interests are Software Reliability Engineering, software testing, software Metrics, and cloud computing.

IX. REFERENCES

[1]. Kumar, Aswini, Whitchcock, Andrew, ed., Google's BigTable, First an overview. BigTable has been in development since early 2004 and has been in active use for about eight months (about February 2005)..

[2]. Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Michael 'Mike'; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E (2006), "Bigtable: A Distributed Storage System for Structured Data",(download ebook) (PDF), Google.

[3]. Chang et al. 2006, p. 3: 'Bigtable can be used with MapReduce, a framework for running large-scale parallel computations developed at Google. We have written a set of wrappers that allow a Bigtable to be used both as an input source and as an output target for MapReduce jobs'

[4]. Google File System and BigTable", Radar (World Wide Web log), Database War Stories (7), O'Reilly, May 2006.

[5]. "Google Bigtable, Compression, Zippy and BMDiff". 2008-10-12. Archived fromthe original on 1 May 2013. Retrieved 14 April 2015.