Scientific Journal of Impact Factor (SJIF): 4.72

e-ISSN (O): 2348-4470

International Journal of Advance Engineering and Research Development

Emerging Trends and Innovations in Electronics and Communication Engineering - ETIECE-2017

Volume 5, Special Issue 01, Jan.-2018 (UGC Approved)

Distributed Big Table for Relational Database

Taseem Nazir[1], Khalil Ahmed[2], Aaqib Hassan Bulla[3], Junaid Farooq War[4], [5]Haider Mehraj

[1,2]*Department of Computer Science and Engineering,SoET,BGSBU*
[3, 4,5] *Department of Electronics and Communication Engineering,SoET,BGSBU*

Abstract—*In order to change the underlying file system storage and distribute the data among numerous independent machines,big tables are widely used. The main aim of the paper is to provide dynamic control to clients over data layout and design and to provide efficient storage and retrieval of data without disturbing the availability and performance.*

Keywords: *Big table, Dynamic Control, Availability, Performance,Storage*

I. INTRODUCTION

Big table is a storage system to manage the data that is widely distributed amongst multiple servers. The tasks at MNC'S that work with data store data in Big table, including web indexing. These applications place very different demands on Big table, both in terms of data size and latency requirements. Despite these varied demands, Big table has successfully provided high-performance solution for all. Big table works on the principle of partition and replication of data .In case of partition, data is bifurcated on the basis of a key and stored on multiple servers and in case of replication, data is replicated among multiple servers.Big table is different from relational database management system where in spite of rows, columns are stored in sequential storage locations. A Big table holds numerous tables with each table comprising of further set of tablets, and each tablet contains all data associated with a row range. At first, each table consists of just one tablet. When the size of the table increases, it is automatically split into further more tablets of fixed size. In contrast to database management system, big table depends on cluster management system to meet all the functionalities of resource management and failures.

II. LOCKS AND IMPLEMENTATION

Big table depends upon the availability and consistency of lock services. Locks are used to ensure –there is at most one transaction writing any data in a tablet, reads and writes in a tablet are atomic .The Big table implementation has three major components: a library that is associated with every client, A master server and numerous tablet servers. In order to accommodate changes in workload,we can add or remove the tablet servers. Balancing the server load, changes in the schema, garbage collection in the file system are done by the master server. Bifurcation of large tables, handling read and write requests and management of set of tables is done by the tablet server. The communication overhead between the client and master is quite low leading to less overhead of master servers and thus helps in efficiency.

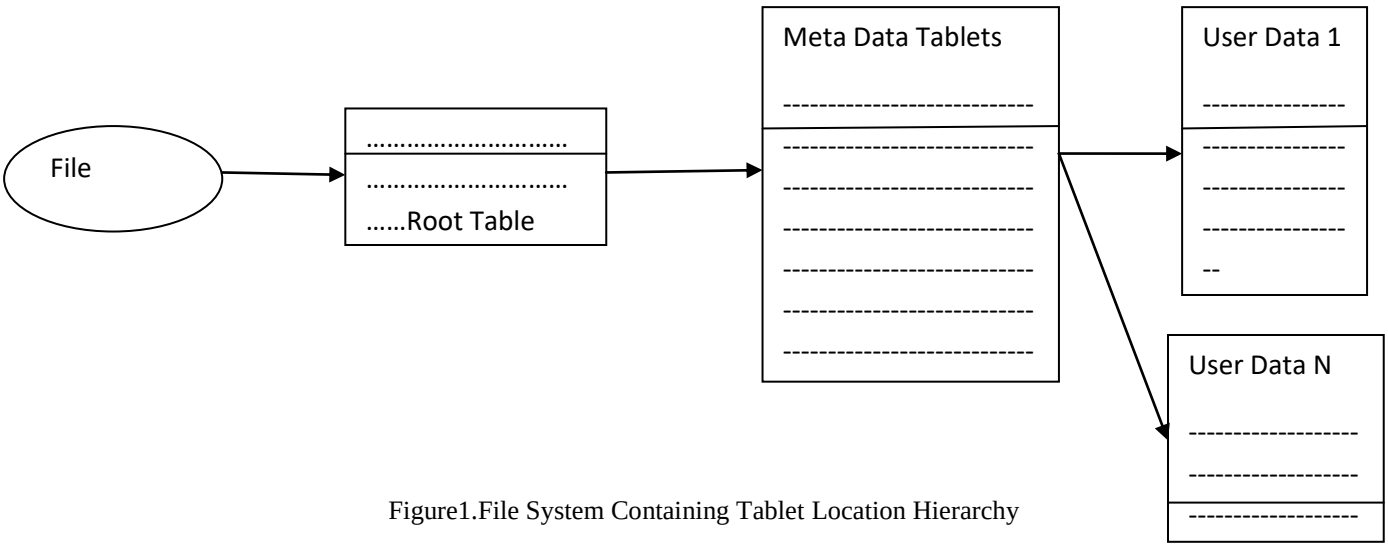

Figure1.File System Containing Tablet Location Hierarchy

III. REFINEMENTS

The implementation described in the previous sectionrequired a number of refinements to achieve the highperformance, availability, and reliability required by users. The refinements could be done on various basis like:

A.Club The Data Locally

Clients can group multiple column families together intoa *group*. A separate SSTable is generated foreach locality group in each tablet. Segregating columnfamilies that are not typically accessed together into separatelocality groups enables more ef_cient reads. Forexample, page metadata in Webtablelike that of a language and checksums) can be in one locality group, and thecontents of the page can be in a different group

B.Compression

Clients can control whether or not the SSTables for alocality group are compressed, and if so, which compression format is used. The user-specified compressionformat is applied to each SSTable block where size is controlled by local group parameters.Although we lose some space by compressingeach block separately, we benefit in that small portionsof an SSTable can be read without decompressingthe entire file. Many clients use a two-pass custom compression scheme. The first pass uses Bentley andMcIlroy's scheme [3], which compresses long common strings across a large window. The second pass uses afast compression algorithm that looks for repetitions ina small 16 KB window of the data. Both compressionpasses are very fast.they encode at 100.200 MB/s, anddecode at 400.1000 MB/s on modern machines.Even though we emphasized speed instead of space reductionwhen choosing our compression algorithms, thistwo-pass compression scheme does surprisingly well.

C.Caching for read performance

To improve read performance, tablet servers use two levelsof caching. The Scan Cache is a higher-level cachethat caches the key-value pairs returned by the SSTableinterface to the tablet server code. The Block Cache is alower-level cache that caches SSTables blocks that wereread from GFS. The Scan Cache is most useful for applicationsthat tend to read the same data repeatedly. TheBlock Cache is useful for applications that tend to readdata that is close to the data they recently read

IV. PERFORMANCE

N-Tablet servers are used to measure the performance.The configuration of servers made use of 1GB of memory ,1500 machines,The same number of client machines were used in order to ensure that there is no starvation in terms of client tablets. The tablet servers and master, test clients, and GFSservers all ran on the same set of machines. Every machineran a GFS server. Some of the machines also raneither a tablet server, or a client process, or processesfrom other jobs that were using the pool at the same timeas these experiments.R is the distinct number of Bigtable row keys involvedin the test. R was chosen so that each benchmark read orwrote approximately 1 GB of data per tablet server.The *sequential write* benchmark used row keys withnames 0 to R 1. This space of row keys was partitionedinto 10N equal-sized ranges. These ranges wereassigned to the N clients by a central scheduler that as signed the next available range to a client as soon as the client finished processing the previous range assigned to it. This dynamic assignment helped mitigate the effects of performance variations caused by other processes running on the client machines. A string under single row was written ,each string was generated with the help of random function.and was therefore uncompressible., In order to avoid any cross-row compression, , strings under different row key were distinct. However, performance does not increase linearly. Formost benchmarks, there is a significant drop in per-serverthroughput when going from 1 to 50 tablet servers. Thisdrop is caused by imbalance in load in multiple server
configurations, often due to other processes contendingfor CPU and network. Our load balancing algorithm attemptsto deal with this imbalance, but cannot do a perfectjob for two main reasons: rebalancing is throttled toreduce the number of tablet movements (a tablet is unavailablefor a short time, typically less than one second,when it is moved), and the load generated by our benchmarksshifts around as the benchmark progresses.

V.CONCLUSION

C-Store and Bigtable share many characteristics: both systems use a shared-nothing architecture and have two different data structures, one for recent writes, and one for storing long-lived data, with a mechanism for moving data from one form to the

other. The systems differ significantly in their API: C-Store behaves like a relational database, whereas Bigtable provides a lower level read and write interface and is designed to support many thousands of such operations per second per server. C-Store is also a .read-optimized relational DBMS., whereas Bigtable provides good performance on both read-intensive and write-intensive applications. Big table's load balancer has to solve some of the same kinds of load and memory balancing problems faced by shared-nothing databases[1][2]. Our problem issomewhat simpler:

A. There is no consideration for the possibility of multiple copies of the same data, possibly in alternateforms due to views or indices;
B. Let the user describe what data belongs in memory and what data should stayon disk, rather than trying to determine this dynamically;
C. There is no complex queries to execute or optimize.

REFERENCES

[1] STONEBRAKER, M., AOKI, P. M., DEVINE, R., LITWIN, W., AND OLSON, M. A. Mariposa: A new architecturefor distributed data. In *Proc. of the Tenth ICDE*(1994), IEEE Computer Society, pp. 54.65.
[2]. COPELAND, G. P., ALEXANDER, W., BOUGHTER, E. E., AND KELLER, T. W. Data placement in Bubba. In Proc. of SIGMOD (1988), pp. 99.108.
[3]. BENTLEY, J. L., AND MCILROY, M. D. Data compression using long common strings. In Data CompressionConference (1999), pp. 287.295.
[4]. ABADI, D. J., MADDEN, S. R., AND FERREIRA,M. C. Integrating compression and execution in column oriented database systems. *Proc. of SIGMOD* (2006).
[5]. Bigtable: A Distributed Storage System for Structured Data Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
[6] Surabhi Mittal and Gaurav Tripathi, "Role of Mobile Phone Technology in Improving SmallFarm Productivity", Agricultural Economics Research Review, Vol. 22, pp. 451-459, 2009.

