Performance Analysis of Hbase

Neseeba P.B, Dr. Zahid Ansari

Department of Computer Science & Engineering, P. A. College of Engineering, Mangalore, 574153, India

***Abstract*— Hbase is a distributed column-oriented database built on top of HDFS. Hbase is the Hadoop application to use when you require real-time random access to very large datasets. Hbase is a scalable data store targeted at random read and writes access of fully structured data. It's invented after Google's big table and targeted to support large tables, on the order of billions of rows and millions of columns. This paper includes step by step information to the HBase, Detailed architecture of HBase. Illustration of differences between apache Hbase and a traditional RDBMS, The drawbacks of Relational Database Systems, Relationship between the Hadoop and Hbase, storage of Hbase in physical memory. This paper also includes review of the Other cloud databases. Various problems, limitations, advantages and applications of HBase. Brief introduction is given in the following section.**

I. INTRODUCTION

Hbase is called the Hadoop database because it is a NoSQL database that runs on top of Hadoop. It combines the scalability of Hadoop by running on the Hadoop Distributed File System [1] (HDFS), with real-time data access as a key/value store and deep analytic capabilities of Map Reduce. Apache Hbase is a NoSQL [2] database that runs on top of Hadoop as a distributed and scalable big data store. This means that Hbase can leverage the distributed processing paradigm of the Hadoop Distributed File System (HDFS) and benefit from Hadoop's Map Reduce programming model [3]. It is meant to host large tables with billions of rows with potentially millions of columns and run across a cluster of commodity Hadoop roots, Hbase is a powerful database in its own right that blends real-time query capabilities with the speed of a key/value store and offline or batch processing via Map Reduce. In short, Hbase allows you to query for individual records as well as derive aggregate analytic reports across a massive amount of data. As a little bit of history; Google was faced with a challenging problem: How could it provide timely search results across the entire Internet? The answer was that it essentially needed to cache the Internet and define a new way to search that enormous cache quickly. It defined the following technologies for this purpose:

1) Google file system: A scalable distributed file system for large distributed data-intensive applications.
2) Big Table: A distributed storage system for managing structured data that is designed to scale to a large size: petabytes of data across thousands of commodity servers.
3) Map Reduce: A programming model and an associated implementation for processing and generating large data set. It was not too long after Google published these documents that we started seeing open source implementations of them, and in 2007, Mike Cafarella released code for an open source Big Table implementation that he called Hbase.

II. DATA MODEL

Hbase actually defines a four-dimensional data model and the following four coordinates define each cell (see Figure 1):

1) Row Key: Each row has a unique row key; the row key does not have a data type and is treated internally as a byte.
2) Array. Column Family: Data inside a row is organized into column families; each row has the same set of column families, but across rows, the same column families do not need the same column qualifiers. Hbase stores column families in their own data files, so they need to be defined upfront, and changes to column families are difficult to make.
3) Column Qualifier: Column families define actual columns, which are called column qualifiers. Column qualifiers are columns themselves.
4) Version: Each column can have a configurable number version, and you can access the data for a specific version of a column qualifier. And then append the time (as a long) to hash. The importance in using a hash is two-fold:

    1) It distributes values so that the data can be distributed across the cluster.
    2) It ensures that the length (in bytes) of the key is consistent and hence easier to use in table scans.

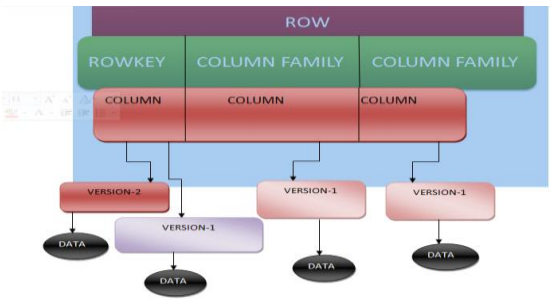

Figure 1.HBase Four-Dimensional Data Model

As shown in Figure 2 an individual row is accessible through its row key and is composed of one or more column families. Each column family has one or more column qualifiers "column" and each column can have one or more versions. To access an individual piece of data, you need to know its row key, column family, column qualifier, and families. Each column family has one or more column qualifiers "column" and each column can have one or more versions. To access an individual piece of data, you need to know its row key, column family, column qualifier, and version. When designing an Hbase data model, it is helpful to think about how the data is going to be accessed. You can access Hbase data in two ways:

1) Through their row key or via a table scan for a range of row keys
2) In a batch manner using map-reduce.
3) This dual-approach to data access is something that makes Hbase particularly powerful. Typically, storing data in Hadoop means that it but not necessarily for real-time access.Let's first look at the real-time access. As a key/value store, the key is the row key, and the value is the collection of column families, as shown Figure 2, the key is the row key we have been talking about, and the value is the collection of column families. You can retrieve the value associated with a key; or in other words, you can "get" the row associated with a row key, or you can retrieve a set of rows by giving the starting row key and ending row key, which is referred to as a table scan. You cannot query for values contained in columns in a real-time query, which leads to an important topic: row key design.

The design of the row key is important for two reasons:

1. Table scans operate against the row key, so the design of the row key controls how much real-time/direct access you can perform against Hbase.
2. When running Hbase in a production environment, it runs on top of the Hadoop Distributed File System (HDFS) and the data is distributed across the HDFS based on the row key. If all your row keys start with "user-" then most likely the majority of your data will be isolated to a single node . Your row keys, therefore, should be different enough to be distributed across the entire deployment/
3. The manner in which you design your row keys depends on how you intend to access those rows. If you store data on a per user basis, then one strategy is to leverage the fact that row keys are ultimately stored as byte arrays in Hbase,

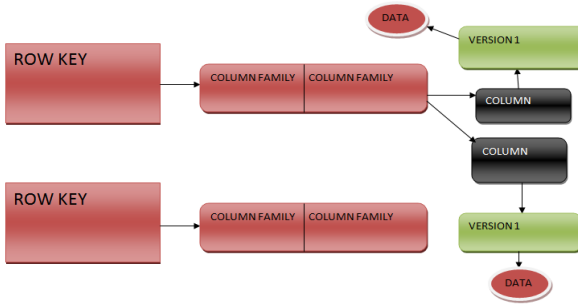

Figure 2.HBase Key-value Data Model

III.COMPARISION BETWEEN HBASE AND RDBMS

Hbase and other column-oriented database are often compared to more traditional and popular relational database or RDBMS.

Table 1.Comparision between HBASE and RDBMS

[1] **HBASE**	[2] **RDBMS**
Hbase mainly on Column-oriented approach	Rdms purely Row-oriented approach
Flexible and dynamic schema,	Only Fixed schema
Works very well with sparse tables.	.Not efficient for sparse tables.
Hbase does not use any query language	Full of query language
Wide tables can be used	only Narrow tables are used
Tight – Integration with MR	Not really
De-normalizes user data.	Normalize as you can
Horizontal scalability-just add hard war.	Hard to share and scale.
Consistent	Consistent
.No transactions can be one .	Transactional database
Works well for semi-structured data as well as structured data	Works well for structured data.

IV. ARCHITECTURE

The general architecture of habse consisting of following Figure 3 shows gives us the architecture of Hbase and Hadoop. The files are primarily handled by the *HRegionServer*'s[4]. But in certain scenarios even the *Hamster* will have to perform low-level file operations. We also notice that the actual files are in fact divided up into smaller blocks when stored within the Hadoop Distributed File system (HDFS). This is also one of the areas where you can configure the system to handle larger or smaller data better.

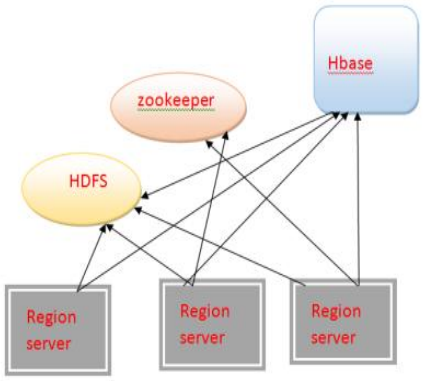

Figure3.Architecture of Hbase

The general flow is that a new client contacts the Zookeeper quorum first to find a particular row key. It does so by retrieving the server name that hosts the -ROOT- region from Zookeeper. With that information it can query that server to get the server that hosts the .META. Table. Responsible to assign the regions to each *HRegionServer* when you start Hbase. This also includes the "special" -ROOT- and Both of these two details are cached and only looked up once. Lastly it can query the .META. Server and retrieve the server that has the row the client is looking for. The *Hamsters* is responsible to assign the regions to each.

HRegionServer when you start Hbase. This also includes the "special" -ROOT- and .META. Tables. When we take a deeper look into the region server, it contain regions and stores as shown below:

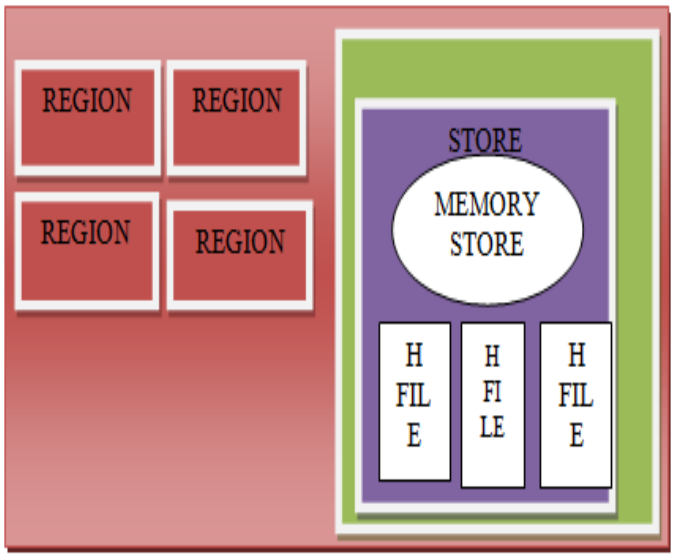

Figure 4. Region server

Table 2 .Open source cloud databases.

Open source cloud database	Developed by	Features
Hbase	Hbase is an open source distributed database. It was first implemented and released by Power Set in 2007. Then it became official subproject of Hadoop from Apache foundation [7].	Hbase employs master-slave architecture. The master keeps track of which slave stores which data and dispatch tasks to relevant slaves. This approach takes risk that if the master node dies, the whole system becomes useless. Hbase is written mostly in Java, and provides APIs through Thrift. It also provides a shell in which user can use HBaseQuey Language (HQL) to manipulate the database.
Cassandra	Cassandra is an open source distributed database. It's first implemented and released by a group in Facebook in 2008.	Cassandra combines features from Google BigTable and Amazon Dynamo, making it a highly scalable key-value store. Casandra is written in JAVA and provides APIs through Thrift which enables different programming languages to operate Cassandra[8]. However, the eventually consistent approach causes data inconsistency over small period of time. Some consistency sensitive application might not tolerate Cassandra. But for most web services and applications, this is a popular trade-off to enhance scalability.
Hyper table	Hyper table is an open source distributed database. It's first developed and released by Zvents in 2007.	Similar to Hbase, Hypertable employs a master- slave architecture in which the master only keeps track of the data among slave nodes. The most famous user of Hypertable is Chinese search engine Baidu. Hypertable is written in C++ and provides APIs via Thrift. As Hbase, users can operate Hypertable[9] by provided shell Hypertext Query Language .

MemcacheDB	Steve Chu - an open source developer released the resulting system as MemcacheDB[10] in 2007.	MemcacheDB employs a master-slave approach for data access, with which clients can read from any node in the system but can only write to the master node. This makes sure a high consistency even though through a multiple entry reading. MemcacheDB[11] uses Berkeley DB for data persistence and replication and is written in C. The memcached library is someplace the Clients can use to access the database. Clients perform queries via the memcachedget_multi function to request multiple keys at once.
MongoDB	MongoDB was designed to provide both the speed and scalability of key-value store and was developed and released as open source in 2008 by 10gen.	MongoDB is a document-oriented database that allows data to be manifested like documents. MongoDB[12] offers three replication styles: master-slave replication, a "replica-pair" style, and a limited form of master-master replication. MongoDB has its own hashtable-liked query language and also provide the scanner interface which is similar to Hbase and Hypertable. MongoDB is deployed exactly the same way as the MemchacheDB[13] deployed it need to be carefully configured using the configuration files.
Voldemort	Voldemort emulates Amazon Dynamo and combines it with caching. It was released as open source in 2009.	Read and write can be performed at any node by clients in Voldemort[14].there is an inconsistent during the view of data across the system. Fetches on a key may result in Voldemort returning multiple values with their version number, this makes a comparison with Cassandra because that Cassandra can only opposed to a single key. Voldemort is written in Java and exposes its API via Thrift; there are native bindings to high-level languages as well that employ serialization via Google Protocol Buffers. A shell is also provided for interactive queries.

V. ADVANTAGES

Advantages of HBASE over other databases

1. Can store large data sets on top of HDFS [15] file storage and will aggregate and analyse billions of rows present in the HBASE tables. In Hbase database can be shared.
2. Operations such as data reading and processing will take small amount of time as compared To traditional relational model.
3. Random read and write operations for online analytical operations HBASE is used extensively.
4. For example: In banking applications such as real-time data updates in ATM machines, HBASE can used.

VI. PROBLEMS WITH HBASE

Some of the problems of hbase are as follows

1. Slow improvements in the security for the different users to access the data from Hbase.
2. In Hbase, we cannot implement any cross data operations and joining operations, of course, we can implement the joining operations using Map Reduce, which would take a lot of time to designing and development. Tables join operations are difficult to perform in Hbase. In some use case, its impossible to create join operations that related to tables that are present in Hbase.
3. Hbase would require new design when we want to migrate data from RDBMS external sources to Hbase servers. However, this process takes a lot of time.
4. It's very difficult to store large size of binary files in Hbase
5. The storage of Hbase will limit real-time queries and sorting
6. Key lookup and Range lookup in terms of searching table contents using key values, it will limit queries that perform on real time.
7. Default indexing is not present in Hbase. Programmers have to define several lines of code or script to perform indexing functionality in Hbase
8. Expensive in terms of Hardware requirements and memory blocks allocations.
9. More servers should be installed for distributed cluster environments (like each server for Name Node, Data Nodes, ZooKeeper, and Region Servers)
10. Performance wise it require high memory machines.
11. Costing and maintenance wise it is also higher.

VII. LIMITATION OF HBASE

1. We cannot expect completely to use Hbase as a replacement for traditional models. Some of the traditional models features cannot support by Hbase
2. Hbase cannot perform functions like SQL. It doesn't supports SQL structure, so it does not contain any query optimizer

3. Hbase is CPU and Memory intensive with large sequential input or output access while as Map Reduce[16] jobs are primarily input or output bound with fixed memory. Hbase integrated with Map-reduce jobs will result in unpredictable latencies

4. In a shared cluster environment, the set up requires fewer task slots per node to allocate for Hbase CPU requirement.

VI. CONCLUSION

This paper contains introduction to Hbase and survey of other existing databases. Paper also includes little of architecture of Hbase .Brief idea of other open source data bases. comparison of Hbase with Relational database.

We are able to conclude that Hbase is highly featured database than any other database .Hbase is faster than sequential execution for larger amounts of data as well as For smaller amounts of data Hbase is scalable as the amount of data increases because it uses technology of Map Reduce programming. The better way would be to go with sequential execution and with lesser number of processors.Idea of different advantages, disadvantages and problems ,limitations with application is mentioned .

REFERENCES

[1]. Huang, Jian, et al. "High-performance design of hbase with rdma over infiniband." Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International.IEEE,2012
[2]. Leavitt, Neal. "Will NoSQL databases live up to their promise?." *Computer* 43.2 (2010).
[3]. Lämmel, Ralf. "Google's MapReduce programming model—Revisited." *Science of computer programming* 70.1 (2008): 1-30.
[4]. Carstoiu, D., A. Cernian, and A. Olteanu. "Hadoop hbase-0.20. 2 performance evaluation." *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*. IEEE, 2010.
[5]. Lineland, HBaseArchitecute – 101 – Storage, Oct 12, 2009, http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html
[6]. Junqueira, Flavio, and Benjamin Reed. *ZooKeeper: distributed process coordination*. " O'Reilly Media, Inc.", 2013.
[7]. Yingjie Shi, XiaofengMeng, Jing Zhao, Xiangmei Hu, Bingbing Liu and Haiping Wang, Benchmarking Cloud-based Data Management Systems, in Proceeding CloudDB '10 Proceedings of the second international workshop on Cloud data management
[8]. Chris Bunch, Jonathan Kupferman and Chandra Krintz, Active Cloud DB: A Database-Agnostic HTTP API to Key-Value Datastores, April 2010 UCSB Tech Report 2010-07
[9]. Hypertable.Eben Hewitt, Cassandra: The Definitive Guide, O'Reilly Media, November 2010, ISBN: 978-1-4493-904 Cassandra. http://cassandra.apache.org/
[10]. Khetrapal, Ankur, and Vinay Ganesh. "HBase and Hypertable for large scale distributed storage systems." *Dept. of Computer Science, Purdue University* (2006): 22-28.
[11]. Wei-ping, Zhu, L. I. Ming-Xin, and Chen Huan. "Using MongoDB to implement textbook management system instead of MySQL." *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE, 2011.
[12]. George, Lars. Hbase: The Definitive Guide: Random Access to Your Planet-Size Data. " O'Reilly Media, Inc.", 2011.
[13]. MemcacheDB.http://memcachedb.org/.
[14]. Voldemort. http://project-voldemort.com/.
[15]. AvinashLakshman and Prashant Malik, Cassandra: a decentralized structured storage system, ACM SIGOPS Operating Systems Review Volume 44 Issue 2, April 2010.
[16]. Blazhievsky, Serge. "Introduction to Hadoop, MapReduce and HDFS for Big Data Applications." *SNIA Education* (2013).
[17]. Nguyen, Phuong, et al. "A hybrid scheduling algorithm for data intensive workloads in a MapReduce environment." Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. IEEE Computer Society, 2012.