HBase - Hadoop Database

Ankit Goyal
Global MISM, Heinz College
Carnegie Mellon University
Adelaide, Australia
agoyal3@andrew.cmu.edu

***Abstract*— In this digital era, we are inundated with data from sources like mobile, cloud services, social media platforms, microblogging sites, etc. According to some sources, 90% of the world's data has been generated in last two years. Most of this data is unstructured and is growing at twice the rate of structured data. The task of storing and analyzing this data in an efficient and economical way has become more important than ever. It has become beyond the scope of conventional relational databases to manage this ever changing data. Researchers, developers, web pioneers and companies around the world observed the need for alternate solutions to manage this large quantity of data, which led to the emergence of NoSQL databases. [1]**

The number of companies started with the projects to develop NoSQL databases to cater their specific needs. There are around 225+ NoSQL databases available till date, which are designed for specific operations and tasks. Apache's Cassandra, LinkedIn's project Voldemort, Apache Hadoop-based project HBase, Amazon's Dynamo, MongoDB and Redis are few which are most widely used. In this paper, we will discuss about the HBase NoSQL database, which is built on top of the Hadoop Distributed File System (HDFS). The non-textual data like graphics, music, videos, etc. is stored in HDFS and the location of the data is present in HBase. This allows quick storage, access and retrieval of the data [2]. The paper also discusses about HBase architecture, unique features and benefits, use cases, comparison with other NoSQL databases and its future prospect.

Keywords—HBase; NoSQL database; Hadoop; column-family

I. INTRODUCTION

In its early days, Google was facing a challenging problem on how to provide quick search results through its search engine from the entire internet. The solution required to cache the internet by web crawling and then search through this vast cache, using the most efficient and quickest method possible. In order to implement this, Google proposed technologies like Google File System – a distributed file system which is scalable and capable of handling large volumes of distributed data, BigTable – a distributed storage system which can manage petabytes of structured data and MapReduce – a technique to generate and process huge data sets. In 2006, Google published papers, illustrating the data storage and processing frameworks used in these technologies. Soon after these papers were published, many open source implementations started worldwide. In 2007, Mike Cafarella, an open source developer, released the code for BigTable implementation, which he called HBase. It soon gained popularity in projects where Hadoop was being used. Today, HBase has continuously growing large user and developer's community, making it one of the top-level Apache projects. It is being used in organizations like Salesforce, Twitter, Facebook, Adobe, StumleUpon and many more [3].

II. HBASE

Apache HBase is an open-source column-oriented NoSQL database built on top of Hadoop Distributed File system (HDFS) and is written in Java language. It is implemented on Google's BigTable paper. It offers distributed fault-tolerance, horizontal scalability, automatic sharding, automatic failover support, random read/write access to huge data sets, support for Java API and MapReduce jobs. The hybrid architecture enables real time querying capabilities with the speed of key-value store using HDFS and offline or batch processing using Hadoop's MapReduce programming model. Due to this native integration with Hadoop, it is also referred as Hadoop database [3].

HBase can be executed in three different modes [4]:

- *Standalone* – running using just one Java process, primarily for local development and study purposes.
- *Pseudo-distributed* – running on a single machine using many Java processes.
- *Fully-distributed* – running on a cluster of machines distributed across the network.

A. Logical Architecture

The architecture of HBase servers follows the Master-Slave relationship. Each cluster in HBase has one master node known as HMaster, which is responsible for administration tasks, cluster management, assigning regions to region servers and controlling fail over and load balancing. The data is stored in tables which are split across regions known as HRegions. The server assigned to multiple HRegions is known as HRegionServer, which is responsible for regions management, processing read/write requests, splitting of regions and direct interaction with the client [5].

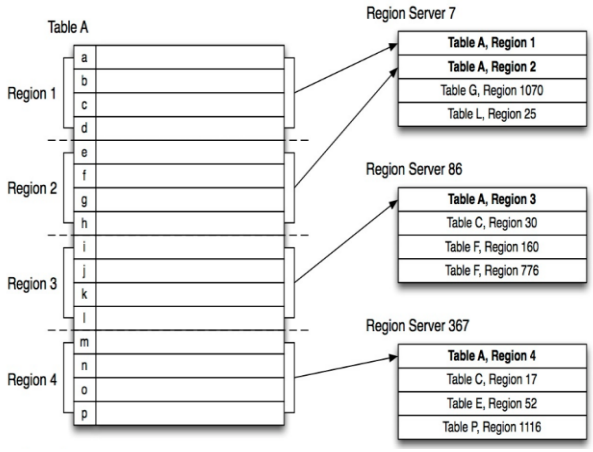

Figure 1. HBase Logical Architecture [5]

B. HBase Data Model

HBase data model is designed to handle data of varying data types, size, columns and fields. The data layout makes it easier to partition it across the clusters in a distributed system. The logical components of HBase data model are [5]:

- *Tables*: HBase organized the data into tables, which are logical collection of the rows stored in different HRegions.
- *Rows*: The table consists of rows which has a unique row key for data access. The rowkeys are treated as a byte array and don't have any data type associated with them.
- *Column Families*: The data is organized within a row in column families. Each column family can have one or more columns associated. These need to be defined beforehand and any changes at later stages are difficult. Every row in the table has the same set of column families, but it's not necessary to store all data in the columns.
- *Column Qualifier*: The columns defined by column families are called column qualifiers. The column qualifiers need to be defined up front.
- *Cell*: A cell is uniquely identified by a combination of a row, column family and column qualifier. The cell values do not have any data type and are treated as byte arrays.
- *Version/Timestamps*: All the values within a cell are versioned and can be accessed later on for any specific version. In case any timestamp is not specified during write operation, then current timestamp is used.

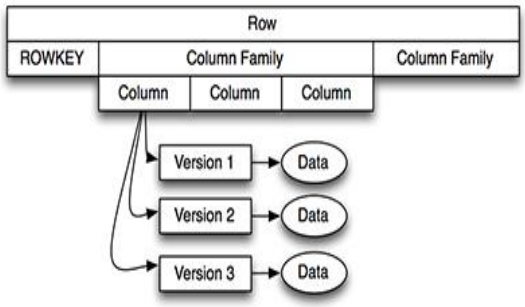

Figure 2. HBase Data Model [3]

The data access in HBase can be done using two different methods – a) random access of rows using the rowkeys. b) In batch or offline manner through map-reduce queries. The HBase allows four operations on the data present in tables - Get, Put, Delete and Scan. Given the role of rowkeys in the process of data access, it becomes utmost important to appropriately select rowkeys for easy and efficient data access [3].

C. Features and Functionalities

- *Sorted rowkeys* –The data in HBase is stored in a lexicographically sorted way according to the rowkeys. This feature facilitates fast read performance, optimizes the scan operation based on the range of rowkeys and permits data retrieval in a single database call. For example, the application focusing on retrieval of recent records can use timestamp as rowkey [7].
- *Control on data sharding* – HBase provides the application developer with an option to have a control on how the data is physically distributed across the cluster. The rowkeys are used as the main criteria for even distribution of the data across Hadoop cluster. Since data in the table is sorted based on rowkeys, every region plays role in storing the part of row key space. Therefore, application developer needs to choose the rowkey skillfully as it has strong influence on how the data sharding is done physically. [7].
- *Strong consistency* – In the CAP theorem, we studied how NoSQL databases make a tradeoff between consistency and availability. From design perspective, HBase values more on consistency than availability. It also supports certain ACID transaction properties at row level. This impacts the write performance compared to other eventually consistent databases but that's the tradeoff which developers have to make as HBase is designed to achieve high read performance. Overall, it guarantees strong consistency and offers right value of the data, thus giving more freedom to the application developers to focus on other aspects of the application like business logic, user experience etc. [7].
- *Scalability* – It provides scalability in both modular and linear form. It can host large data sets with billions of rows and millions of columns irrespective of the data type. It facilitates querying on individual records as well as complex aggregated analytic reports on large data sets [3].
- *Automatic failover support* – HBase offers high availability by means of replication and automatic failover to backup clusters. HBase master automatically detects for any node failures. In case of failure, the regions on failed node are distributed across the surviving nodes [8].
- *API Support* – Java API is supported by HBase allowing easy access to the clients [9].
- *MapReduce Support* – MapReduce support enables parallelized processing of large volumes of data in batches or offline mode. [9]

D. CAP Theorem

The CAP theorem states that for any distributed computer system it is impossible to provide all of the three guarantees simultaneously i.e. Consistency, Availability and Partition Tolerance. For any given distributed system, since partition tolerance is must, therefore it has to make a tradeoff between consistency or availability.

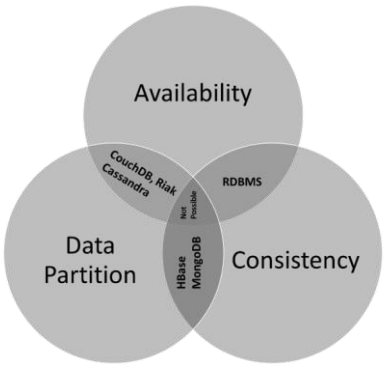

Figure 3. HBase and CAP Theorem

HBase achieves strong consistency and partition tolerance in CAP theorem. It trades off availability up to some extent to achieve strong consistency. This means that even at the time of failure, the clients will read the same version of the row, thus ensuring high consistency. The tradeoff with availability can be explained based on how the replication takes place in HBase. Since HBase is built on HDFS, the replication is handled at storage level. In case, if one of the HRegionServer goes down, all of its regions will have to be reassigned to another HRegionServer. During this period of reassignment, the clients interacting with those regions will encounter degraded availability for short period of time. [6]

E. When to choose HBase

With the myriad of NoSQL databases available to choose from, selecting the right database for application development becomes a tough decision. Below are some of the factors which should be considered while choosing HBase as a database:

- *Data Volumes*: HBase should be considered when unstructured data in the order of peta-bytes need to be stored and analyzed. In case of small volumes of data, there will be inefficient utilization of the resources as the data will be stored and processed using one node only [12].
- *Application types*: HBase is preferred when application needs a variable schema and access to data is done based on a key. Also, when there are lot of versions associated with the data which need to be stored and retrieved later [13].
- *Hardware Environment*: Since HBase is built on top of HDFS, its performance is determined by the underlying hardware resources. For HBase implementation good hardware support is required as HDFS needs minimum of 5 nodes to work efficiently [13].
- *No Requirement of relational features*: The application should not have any dependency or requirement of the relational database features like triggers, complex joins or queries etc. The porting of application data from relational database to HBase database is not an easy task, so requirements should be analyzed thoroughly [13].
- *Quick Access to data*: HBase is best candidate when application requires real time random read/write data access. The sorted rowkeys feature in HBase allows quick retrieval of data in one call based on the rowkey value. HBase databases are customized for high read performance.

III. USE CASES

HBase is backed up by its large open source community, strong users and developers base, and strong connections with Hadoop. There are numerous use cases involved with HBase and tech giants across the globe like Facebook, Mozilla, Flurry, Adobe, Yahoo, Twitter, Stumbleupon, Meetup, OpenLogic etc. are running numerous projects using HBase. [10]

Some of the common use cases which have been implemented using HBase are:

- *Canonical web-search problem*: This use case is the primary reason for existence of HBase. The search engines like Google, need to build indexes which will help them in searching documents from the entire internet and returning search results in quick time. The row-level access offered by HBase allows the web crawlers to insert and update the searched documents. MapReduce further helps in generation of efficient and optimized search index. This index can be used to retrieve search results as HBase are specifically optimized for high read performance. [11]
- *Capturing Incremental Data*: Most widely use case of HBase is to store the incremental data coming from various data sources. Few examples of data sources include web crawls, the advertisement information containing which user saw the advertisement and for how much time, time series data etc. Social media websites like Facebook and Stumbleupon need to store counter which would increment every time a user clicks ‘Like’ or ‘+1’ button on the websites respectively. Given the amount of users and data traffic generated by these websites, they required a solution which was highly scalable and flexible. Stumbleupon, who was using MySQL at that time, had to switch to HBase for this task and implemented an atomic increment which was later incorporated in the Apache-Hadoop project. Similarly, Facebook also uses HBase to store the user ‘Likes’. Mozilla and Trend Micro also use HBase to capture and store the crash reports from softwares installed on user’s systems, user activity logs and later on analyze them using big data analytics. This has been possible because of the flexible schema offered by HBase. [11]

- *Content Serving*: HBase is used by content management systems (CMS) at the back end to store and serve the large volumes of data generated by the users in form of tweets, micro blogs, Facebook posts etc. Salesforce uses HBase for its CRM product. The strong consistency and ability to handle millions of transactions per day were the main reasons behind Salesforce adopting HBase. Moreover, Hadoop was already being used by Salesforce for large offline batch processing and strong integration of HBase with Hadoop provided additional value added benefits. URL shorteners like Su.pr, a product of Stumbleupon, uses HBase to store millions of shortened URLs and their mapping to original URLs. [11]

- *Information Exchange*: One of the most discussed use case in this category is that of Facebook messages. Facebook uses HBase for its existing messaging infrastructure. According to the information revealed by Facebook engineers, it receives billions of messages per day, adding up to 250 TB of data every month to its clusters. After considering other available options like Cassandra, Facebook finally settled with HBase for messaging infrastructure due to features like strong consistency, high read and counter throughput, automatic sharding, capacity to handle large data, active user community and experience with Hadoop implementation. This is considered to be one of the biggest HBase deployments in terms of size and number of servers involved. [11]

IV. Comparison

Some of the most popular NoSQL databases available today include – Redis, based on key-value store, MongoDB – document oriented and Cassandra – column oriented database similar to HBase. Redis is key-value store database used mainly for caching data and is not suitable for handling large workloads [1]. Amongst these databases, MongoDB has the highest market share and is favored most by developers because of its ease to learn and it offers very simple implementation. It has become a preferred choice for building many mobile and web applications due to its interface with data exchange formats like JSON. MongoDB has its limitations when it comes to data-intensive applications requiring complex data operations. In such cases column-oriented databases such as Cassandra and HBase, are first choice of preference. [7]

Facebook originally developed Cassandra but in 2011, they opted for HBase to implement their messaging infrastructure. According to latest db-engines.com rankings, between two column-oriented databases, Cassandra is more widely used than HBase. This can be explained based on how the development of both these projects is being managed. Cassandra which started off as an open source project, is now primarily being headed by DataStax Inc. All the documentation and development work is being looked over by DataSpax and they have been putting in lot of efforts to promote the database. Whereas HBase is a complete open source project driven by community. There are 12 different companies involved in its development so it loses the spark as that of being managed by one. However, both the databases offer unique features and are preferred by different companies based on the requirements. [7]

The below table summarizes some of the major differences between Cassandra and HBase databases:

Point	HBase	Cassandra
Foundations	Based on Google's BigTable	Based on DynamoDB (Amazon)
CAP Theorem	HBase achieves strong consistency and partition tolerance	Cassandra achieves high availability and partition tolerance.
Optimization	HBase is optimized for reads, supported by single-write master and offers strong consistency.	Cassandra is optimized for write operations. Cassandra offers eventual consistency and does quorum reads which are slower than HBase reads.
Partitioning	HBase only supports ordered partitioning.	Cassandra officially supports both random and ordered partitioning but random partitioning is most preferred due to hot spots issue with ordered partitioning.
Aggregation	HBase offers simple aggregations like SUM, MIN, MAX, AVG and STD.	Aggregations are not supported in Cassandra nodes and must be provided by clients.
Development	Lacks friendly language for development.	SQL like language for development allows easy transition of developers from SQL background.

Figure 4. HBase vs. Cassandra [14]

V. Conclusion

HBase is an Apache top-level project and has a diverse and large community of users, commercial vendors and developers, which are continuously working on the improving the HBase experience. Over the years, HBase has proved its presence in terms of scalability, availability and flexibility but still it has certain architectural flaws and engineering problems which has limited its adoption rate compared to other NoSQL databases like Cassandra and MongoDB. Some of the limitations include complex installation and implementation due to dependency on other Hadoop services, susceptibility to single point of failures due to master-slave replication method and high reliability on the underlying hardware which is sometimes responsible for its poor performance [15].

On the other hand, as a Hadoop database, it is set to gain from the advancements made in the area of Hadoop and MapReduce. The success of the Hadoop plays a major role in deciding the future of the HBase. Due to its strong integration with Hadoop platform, it is often considered as first level of choice for vendors who have already adopted Hadoop as their Big Data platform solution. The recent inclusion of support for SQL in HBase using Apache Phoenix and further integration with Spark are focused on understanding the needs and offering services in align with the business requirements of clients [7].

Moreover, HBase is considered as one of the strong potential database solution for upcoming use cases like management of time series data associated with Internet-of-Things. The recent HBase service offerings like Microsoft on Azure, Amazon on Amazon web services (AWS) and Facebook for choosing HBase for its messaging infrastructure over Cassandra, its own implementation show signs of potential growth of HBase. [16]

Summarizing, the HBase is expected to grow steadily in the coming years and the latest innovations and developments will help in overcoming the drawbacks in comparison to other NoSQL databases. Hadoop distribution and HBase community will have an important role to play in this future success by increasing the promotion efforts as done by Cassandra and MongoDB communities. They need to come up with a roadmap which will make it more favorable option for the enterprise customers. [16]

REFERENCES

[1] M. Asay, "MongoDB, Cassandra, and HBase -- the three NoSQL databases to watch," InfoWorld, 19 November 2014. [Online]. Available: http://www.infoworld.com/article/2848722/nosql/mongodb-cassandra-hbase-three-nosql-databases-to-watch.html. [Accessed 7 April 2016].

[2] M. N. Vora, "Hadoop-HBase for Large-Scale Data," IEEE, Mumbai, 2011.

[3] S. Haines, "Introduction to HBase, the NoSQL Database for Hadoop," InformIT, 27 October 2014. [Online]. Available: http://www.informit.com/articles/article.aspx?p=2253412. [Accessed 5 April 2016].

[4] A. K. Nick Dimiduk, "Hello HBase," in HBase in Action, New York, Manning Publications Co., 2013, pp. 15-16.

[5] StratApps, "Introduction to HBase," StratApps, [Online]. Available: http://www.stratapps.net/intro-hbase.php. [Accessed 6 April 2016].

[6] E. Soztutar, D. Das and C. Shanklin, "Apache HBase High Availability at the Next Level," HortonWorks, 22 January 2015. [Online]. Available: http://hortonworks.com/blog/apache-hbase-high-availability-next-level/. [Accessed 6 April 2016].

[7] G. Mazars, "An introduction to HBase, the 'Hadoop database'," Jaxenter, 28 May 2015. [Online]. Available: https://jaxenter.com/hbase-hadoop-database-114373.html. [Accessed 5 April 2016].

[8] W. Qiang, "Facebook Messages & HBase," 22 April 2011. [Online]. Available: http://www.slideshare.net/brizzzdotcom/facebook-messages-hbase. [Accessed 6 April 2016].

[9] M. Chimmiri, "Key Features in Hbase," Hadooptpoint, 29 September 2014. [Online]. Available: http://www.hadooptpoint.com/key-features-in-hbase/. [Accessed 5 April 2016].

[10] Apache HBase, "Powered By Apache HBase," Apache, 2016. [Online]. Available: http://hbase.apache.org/poweredbyhbase.html. [Accessed 6 April 2016].

[11] A. K. Nick Dimiduk, "HBase use cases and success stories," in HBase in Action, New York, Manning Publications Co., 2013, pp. 8-15.

[12] J. Anderson, "A Look At HBase, the NoSQL Database Built on Hadoop," 2015. [Online]. Available: http://thenewstack.io/a-look-at-hbase/. [Accessed 6 April 2016].

[13] Abhishek, "How and when should you use HBase NoSQL DB?," Eduonix, 1 April 2016. [Online]. Available: https://www.eduonix.com/blog/bigdata-and-hadoop/use-hbase-nosql-db/. [Accessed 6 April 2016].

[14] A. Gupta, "HBase vs Cassandra," 1 December 2012. [Online]. Available: http://bigdatanoob.blogspot.com.au/2012/11/hbase-vs-cassandra.html. [Accessed 6 April 2016].

[15] H. Li, "The Future of Hadoop is Misty," Wordpress, 3 March 2016. [Online]. Available: https://haifengl.wordpress.com/2016/03/03/the-future-of-hadoop-is-misty/. [Accessed 7 April 2016].

[16] D. Henschen, "Big Data Debate: Will HBase Dominate NoSQL?," InformationWeek, 8 April 2013. [Online]. Available: http://www.informationweek.com/big-data/software-platforms/big-data-debate-will-hbase-dominate-nosql/d/d-id/1111048. [Accessed 6 April 2016].