edureka.co

HBase Tutorial | What is HBase? | HBase Facebook Case Study | Edureka

Shubham Sinha

16-20 minutes

As we mentioned in our *Hadoop Ecosytem* blog, HBase is an essential part of our Hadoop ecosystem. So now, I would like to take you through HBase tutorial, where I will introduce you to Apache HBase, and then, we will go through the Facebook Messenger case-study. We are going to cover following topics in this HBase tutorial blog:

Apache HBase Tutorial: History

Let us start with the history of HBase and know how HBase has evolved over a period of time.

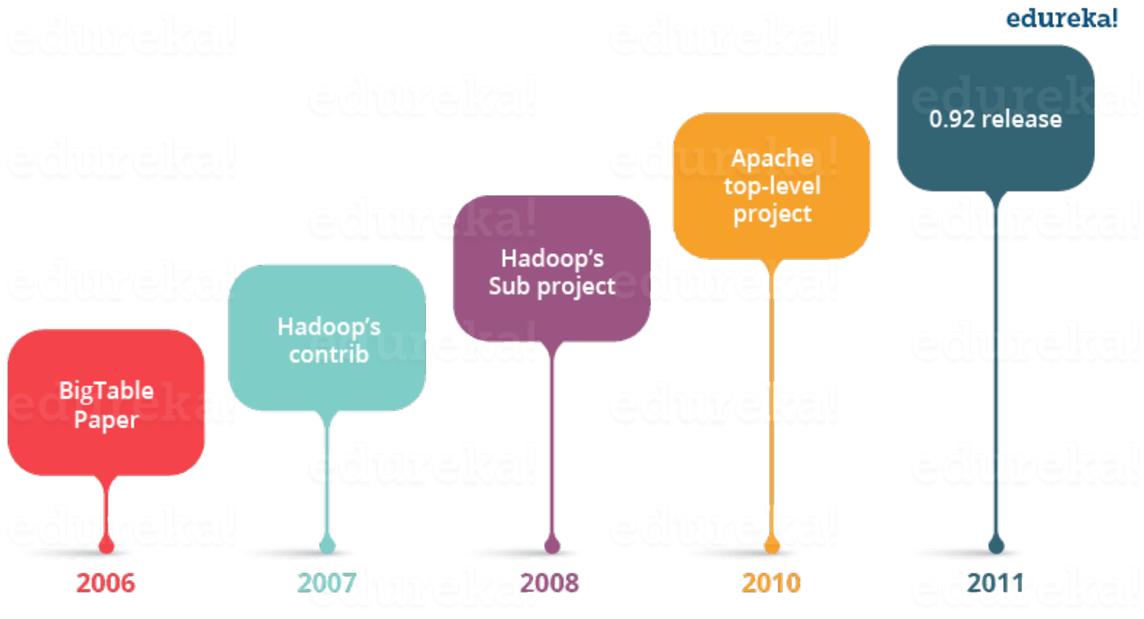

Figure: History of HBase

- Apache HBase is modelled after Google's BigTable, which is used to collect data and serve request for various Google services like Maps, Finance, Earth etc.
- Apache HBase began as a project by the company Powerset for Natural Language Search, which was handling massive and sparse data sets.
- Apache HBase was first released in February 2007. Later in January 2008, HBase became a sub project of Apache Hadoop.
- In 2010, HBase became Apache's top level project.

HBase Tutorial | NoSQL Databases | Edureka

After knowing about the history of Apache HBase, you would be curious to know what is Apache HBase? Let us move further and take a look.

Apache HBase Tutorial: Introduction to HBase

HBase is an open source, multidimensional, distributed, scalable and a ***NoSQL database*** written in Java. HBase runs on top of *HDFS* (Hadoop Distributed File System) and provides BigTable like capabilities to Hadoop. It is designed to provide a fault tolerant way of storing large collection of sparse data sets.

Since, HBase achieves high throughput and low latency by providing faster Read/Write Access on huge data sets. Therefore, HBase is the choice for the applications which require fast & random access to large amount of data.

It provides compression, in-memory operations and Bloom filters (data structure which tells whether a value is present in a set or not) to fulfill the requirement of fast and random read-writes.

Let's understand it through an example: A jet engine generates various types of data from different sensors like pressure sensor, temperature sensor, speed sensor, etc. which indicates the health of the engine. This is very useful to understand the problems and status of the flight. Continuous Engine Operations generates 500 GB data per flight and there are 300 thousand flights per day approximately. So, Engine Analytics applied to such data in near real time can be used to proactively diagnose problems and reduce unplanned downtime. This requires a distributed environment to store large amount of data with ***fast random reads and writes*** for real time processing. Here, HBase comes for the rescue. I will talk about HBase Read and Write in detail in my next blog on ***HBase Architecture***.

As we know, HBase is a NoSQL database. So, before understanding more about HBase, lets first discuss about the NoSQL databases and its types.

Apache HBase Tutorial: NoSQL Databases

NoSQL means *Not only SQL*. NoSQL databases is modeled in a way that it can represent data other than tabular formats, unkile relational databases. It uses different formats to represent data in databases and thus, there are different types of NoSQL databases based on their representation format. Most of NoSQL databases leverages availability and speed over consistency. Now, let us move ahead and understand about the different types of NoSQL databases and their representation formats.

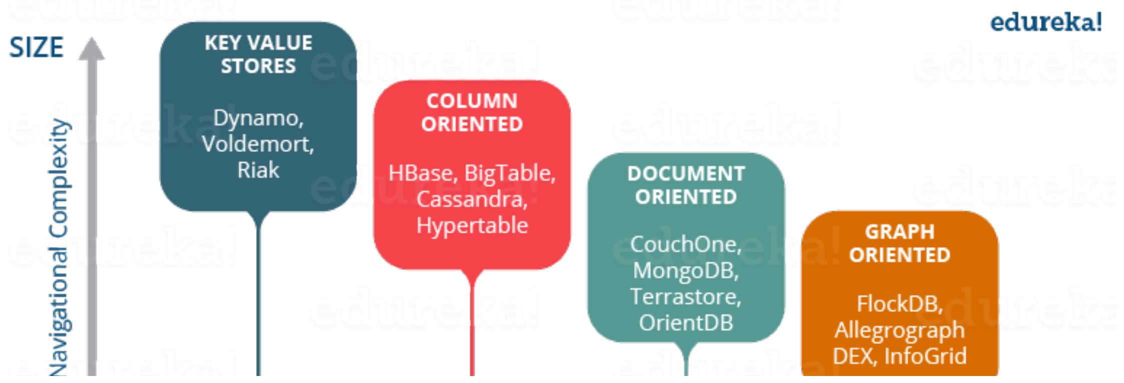

Figure: Comparison of different types of NoSQL databases

Key-Value stores:

It is a schema-less database which contains keys and values. Each key, points to a value which is an array of bytes, can be a string, BLOB, XML, etc. e.g. Lamborghini is a key and can point to a value Gallardo, Aventador, Murciélago, Reventón, Diablo, Huracán, Veneno, Centenario etc.

Key-Value stores databases: Aerospike, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB.

Use-case

Key-value stores handle size well and are good at processing a constant stream of read/write operations with low latency. This makes them perfect for User preference and profile stores, Product recommendations; latest items viewed on a retailer website for driving future customer product recommendations, Ad servicing; customer shopping habits result in customized ads, coupons, etc. for each customer in real-time.

Document Oriented:

It follows the same key value pair, but it is semi structured like XML, JSON, BSON. These structures are considered as documents.

Document Based databases: Apache CouchDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB.

Use-Case

As document supports flexible schema, fast read write and partitioning makes it suitable for creating user databases in various services like twitter, e-commerce websites etc.

Column Oriented:

In this database, data is stored in cell grouped in column rather than rows. Columns are logically grouped into column families which can be either created during schema definition or at runtime.

These types of databases store all the cell corresponding to a column as continuous disk entry, thus making the access and search much faster.

Column Based Databases: HBase, Accumulo, Cassandra, Druid, Vertica.

Use-Case

It supports the huge storage and allow faster read write access over it. This makes column oriented databases suitable for storing customer behaviors in e-commerce website, financial systems like Google Finance and stock market data, Google maps etc.

Graph Oriented:

It is a perfect flexible graphical representation, used unlike SQL. These types of databases easily solve address scalability problems as it contains edges and node which can be extended according to the requirements.

Graph based databases: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso, Stardog.

Use-case

This is basically used in Fraud detection, Real-time recommendation engines (in most cases e-commerce), Master data management (MDM), Network and IT operations, Identity and access management (IAM), etc.

HBase and Cassandra are the two famous column oriented databases. So, now talking it to a higher level, let us compare and understand the architectural and working differences between HBase and Cassandra.

HBase Tutorial: HBase VS Cassandra

- HBase is modelled on BigTable (Google) while Cassandra is based on DynamoDB (Amazon) initially developed by Facebook.
- HBase leverages Hadoop infrastructure (HDFS, ZooKeeper) while Cassandra evolved separately but you can combine Hadoop and Cassandra as per your needs.
- HBase has several components which communicate together like HBase HMaster, ZooKeeper, NameNode, Region Severs. While Cassandra is a single node type, in which all nodes are equal and performs all functions. Any node can be the coordinator; this

removes Single Point of failure.

- HBase is optimized for read and supports single writes, which leads to strict consistency. HBase supports Range based scans, which makes scanning process faster. Whereas Cassandra supports single row reads which maintains eventual consistency.
- Cassandra does not support range based row scans, which slows the scanning process as compared to HBase.
- HBase supports ordered partitioning, in which rows of a Column Family are stored in RowKey order, whereas in Casandra ordered partitioning is a challenge. Due to RowKey partitioning the scanning process is faster in HBase as compared to Cassandra.
- HBase does not support read load balancing, one Region Server serves the read request and the replicas are only used in case of failure. While Cassandra supports read load balancing and can read the same data from various nodes. This can compromise the consistency.
- In CAP (Consistency, Availability & Partition -Tolerance) theorem HBase maintains Consistency and Availability while Cassandra focuses on Availability and Partition -Tolerance.

Now let's take a deep dive and understand the features of Apache HBase which makes it so popular.

Apache HBase Tutorial: Features of HBase

edureka!
Sorted Map
Consistent

Figure: Features of HBase

- **Atomic read and write:** On a row level, HBase provides atomic read and write. It can be explained as, during one read or write process, all other processes are prevented from performing any read or write operations.
- **Consistent reads and writes:** HBase provides consistent reads and writes due to above feature.
- **Linear and modular scalability:** As data sets are distributed over HDFS, thus it is linearly scalable across various nodes, as well as modularly scalable, as it is divided across various nodes.
- **Automatic and configurable sharding of tables:** HBase tables are distributed across clusters and these clusters are distributed across regions. These regions and clusters split, and are redistributed as the data grows.
- **Easy to use Java API for client access:** It provides easy to use Java API for programmatic access.
- **Thrift gateway and a REST-ful Web services:** It also supports Thrift and REST API for non-Java front-ends.
- **Block Cache and Bloom Filters:** HBase supports a Block Cache and Bloom Filters for high volume query optimization .
- **Automatic failure support:** HBase with HDFS provides WAL (Write Ahead Log) across clusters which provides automatic failure support.
- **Sorted rowkeys:** As searching is done on range of rows, HBase

stores rowkeys in a lexicographical order. Using these sorted rowkeys and timestamp, we can build an optimized request.

Now moving ahead in this HBase tutorial, let me tell you what are the use-cases and scenarios where HBase can be used and then, I will compare HDFS and HBase.

I would like draw your attention toward the scenarios in which the HBase is the best fit.

HBase Tutorial: Where we can use HBase?

- We should use HBase where we have large data sets (millions or billions or rows and columns) and we require fast, random and real time, read and write access over the data.
- The data sets are distributed across various clusters and we need high scalability to handle data.
- The data is gathered from various data sources and it is either semi structured or unstructured data or a combination of all. It could be handled easily with HBase.
- You want to store column oriented data.
- You have lots of versions of the data sets and you need to store all of them.

Before I jump to Facebook messenger case study, let me tell you what are the differences between HBase and HDFS.

HBase Tutorial: HBase VS HDFS

HDFS is a Java based distributed file system that allows you to store large data across multiple nodes in a Hadoop cluster. So, HDFS is an underlying storage system for storing the data in the distributed environment. HDFS is a file system, whereas HBase is a database (similar as NTFS and MySQL).

Big Data Training

As Both HDFS and HBase stores any kind of data (i.e. structured, semi-structured and unstructured) in a distributed environment so lets look at the differences between HDFS file system and HBase, a NoSQL database.

- HBase provides low latency access to small amounts of data within large data sets while HDFS provides high latency operations.
- HBase supports random read and writes while HDFS supports WORM (Write once Read Many or Multiple times).
- HDFS is basically or primarily accessed through MapReduce jobs while HBase is accessed through shell commands, Java API, REST, Avro or Thrift API.

HDFS stores large data sets in a distributed environment and leverages batch processing on that data. E.g. it would help an e-commerce website to store millions of customer’s data in a distributed environment which grew over a long period of time(might be 4-5 years or more). Then it leverages batch processing over that data and analyze customer behaviors, pattern, requirements. Then the company could find out what type of product, customer purchase in which months. It helps to store archived data and execute batch processing over it.

While HBase stores data in a column oriented manner where each column is stored together so that, reading becomes faster leveraging real time processing. E.g. in a similar e-commerce environment, it stores millions of product data. So if you search for a product among millions of products, it optimizes the request and search process, producing the result immediately (or you can say in real time). The detailed *HBase architectural explanation*, I will be covering in my next blog.

As we know HBase is distributed over HDFS, so a combination of both gives us a great opportunity to use the benefits of both, in a tailored solution, as we are going to see in the below Facebook messenger case study.

Facebook Messaging Platform shifted from Apache Cassandra to HBase in November 2010.

Facebook Messenger combines Messages, email, chat and SMS into a real-time conversation. Facebook was trying to build a scalable and robust infrastructure to handle set of these services.

At that time the message infrastructure handled over 350 million users sending over 15 billion person-to-person messages per month. The chat service supports over 300 million users who send over 120 billion messages per month.

By monitoring the usage, they found out that, two general data patterns emerged:

- A short set of temporal data that tends to be volatile
- An ever-growing set of data that rarely gets accessed

Facebook wanted to find a storage solution for these two usage patterns and they started investigating to find a replacement for the existing Messages infrastructure.

Earlier in 2008, they used open-source database, i.e. Cassandra, which is an eventual-consistency key-value store that was already in production serving traffic for Inbox Search. Their teams had a great knowledge in using and managing a MySQL database, so switching either of the technologies was a serious concern for them.

They spent a few weeks testing different frameworks, to evaluate the clusters of MySQL, Apache Cassandra, Apache HBase and other systems. They ultimately selected HBase.

As MySQL failed to handle the large data sets efficiently, as the indexes and data sets grew large, the performance suffered. They found Cassandra unable to handle difficult pattern to reconcile their new Messages infrastructure.

The major problems were:

- Storing the large sets of continuously growing data from various Facebook services.
- Requires Database which can leverage high processing on it.
- High performance needed to serve millions of requests.
- Maintaining consistency in storage and performance.

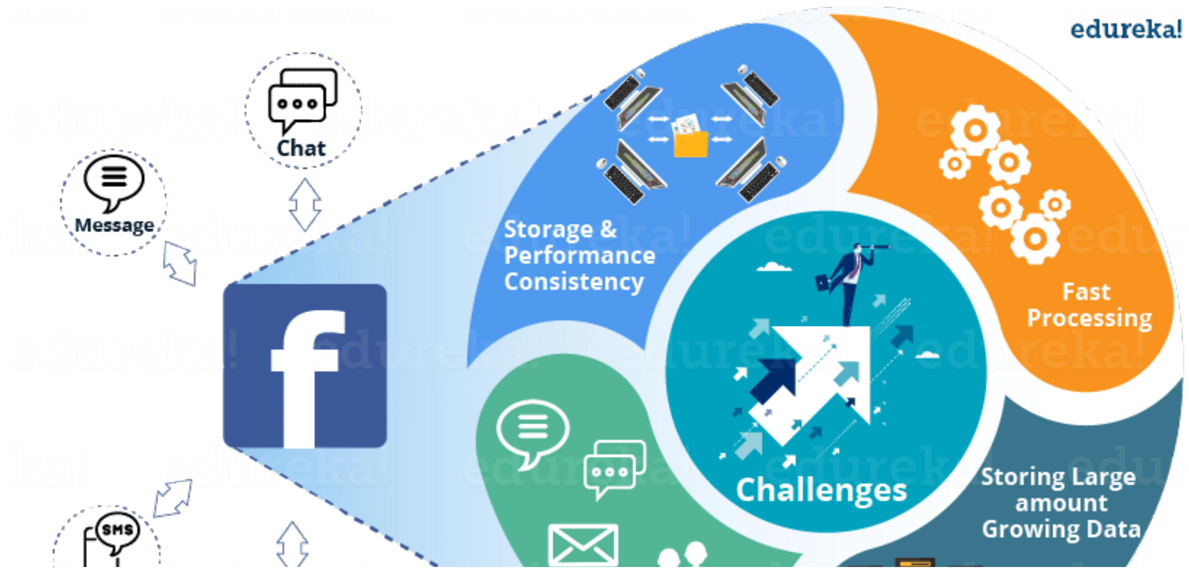
edureka!
Chat
Message
SMS
Storage & Performance Consistency
Fast Processing
Challenges
Storing Large amount Growing Data

Figure: Challenges faced by Facebook messenger

For all these problems, Facebook came up with a solution i.e. HBase. Facebook adopted HBase for serving Facebook messenger, chat, email, etc. due to its various features.

HBase comes with very good scalability and performance for this workload with a simpler consistency model than Cassandra. While they found HBase to be the most suitable in terms of their requirements like auto load balancing and failover, compression support, multiple shards per server, etc.

HDFS, which is the underlying file system used by HBase also provided them several needed features such as end-to-end checksums, replication and automatic load re-balancing.

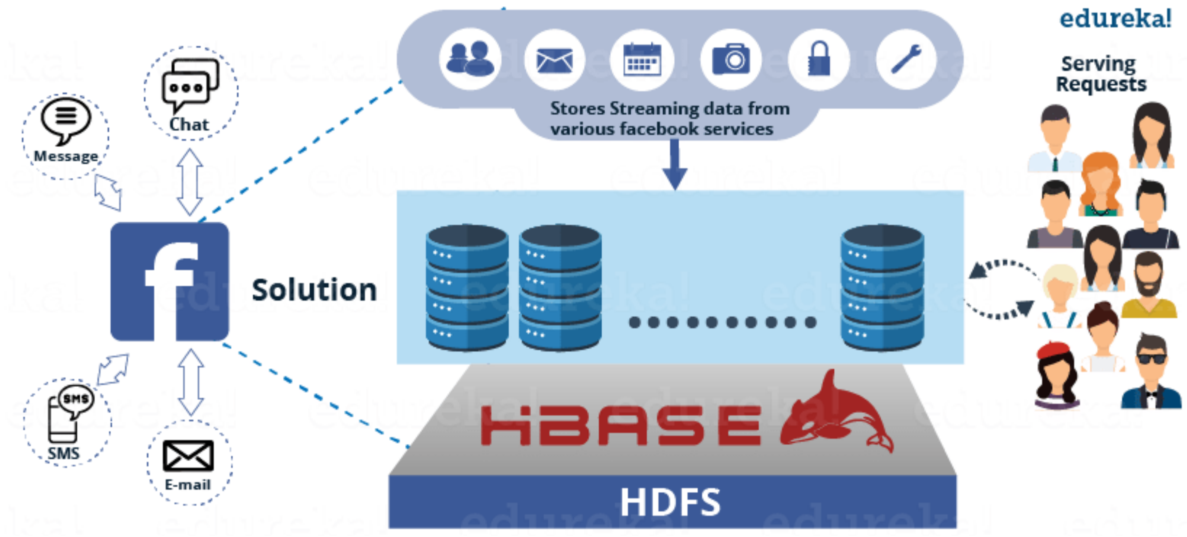

Figure: HBase as a solution to Facebook messenger

As they adopted HBase, they also focused on committing the results back to HBase itself and started working closely with the Apache community.

Since messages accept data from different sources such as SMS, chats and emails, they wrote an application server to handle all decision making for a user's message. It interfaces with large number of other services. The attachments are stored in a Haystack (which works on HBase). They also wrote a user discovery service on top of Apache ZooKeeper which talk to other infrastructure services for friend relationships, email account verification, delivery decisions and privacy decisions.

Facebook team spent a lot of time confirming that each of these services is robust, reliable and providing good performance to handle a real-time messaging system.

I hope this HBase tutorial blog is informative and you liked it. In this blog, you got to know the basics of HBase and its features. In my next blog of *Hadoop Tutorial Series*, I will be explaining the ***architecture of HBase*** and working of HBase which makes it popular for fast and random read/write.

Now that you have understood the basics of HBase, check out the Hadoop training by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka Big Data Hadoop Certification Training course helps learners become expert in HDFS, Yarn, MapReduce, Pig, Hive, HBase, Oozie, Flume and Sqoop using real-time use cases on Retail, Social Media, Aviation, Tourism, Finance domain.

Got a question for us? Please mention it in the comments section and we will get back to you.