# Chapter 11 – Map-Reduce, Hadoop, HDFS, Hbase, MongoDB, Apache HIVE, and Related

Xiangzhe Li

**Summary**

Nowadays, there are more and more data everyday about everything. For instance, here are some of the astonishing data from the book Hadoop the Definitive Guide: "The New York Stock Exchange generates about one terabyte of new trade data per day. Facebook hosts approximately 10 billion photos, taking up one petabyte of storage. Ancestry.com, the genealogy site, stores around 2.5 petabytes of data. The Internet Archive stores around 2 petabytes of data, and is growing at a rate of 20 terabytes per month." (Whites) The Large Hadron Collider near Geneva, Switzerland, will produce about 15 petabytes of data per year. In the business world, having a precise way of determining the accurate information from the big set of data is very critical and can help the company reduce the cost of information retrieval. In this chapter, we will talk about the different components of the software architecture and frameworks that process massive amount of unstructured data. Some of the topics include Map-Reduce, Hadoop, HDFS, Hbase, MongoDB, and Apache HIVE.

## Introduction

### High Level Concepts

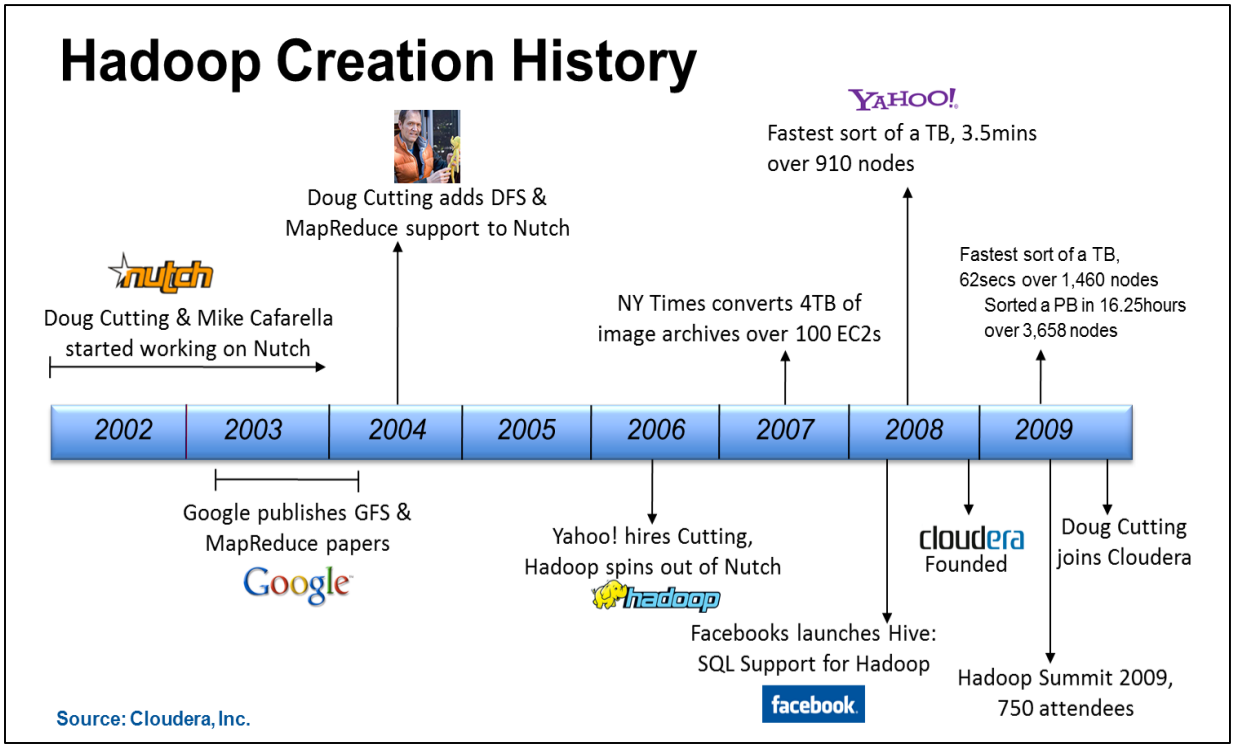
In reality, even though the technology improvement allows the storage capacities of hard drive to increase, the speed at which the data is accessed has not made significant progress. For instance, research found that "One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s, you could read all the data from a full drive in around five minutes. Over 20 years later, one terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data off the disk." (Whites) Wow, even though the transfer speed has increased over 20 times, the storage has increased so significantly that the time for accessing the data become 30 times longer under the old processing methods. For this reason, the concept of parallel computing has brought the initial invention of big data processing with tools such as the Hadoop family.

### History

The initial version of Hadoop was created in early 2005 by Doug Cutting and Michael Cafarella, while Cutting was working at Yahoo! at the time. The name of the project came after his son's toy elephant. The original purpose of the project was to support an open-source web search platform called Nutch. Nutch was initiated in 2002 and it was based on open source information retrieval framework called Apache Lucene and using the Java language as its backbone for the structure. After the initiate architecture of Nutch was created, Cutting and Cafarella realized that it was not able to support the billions of pages on the web. In 2003, Google published a paper about a fully functioning product called the Googles Distributed File System. Then later in 2004, following the Googles Distributed File System, Cutting and

Cafarella started the implementation of the Nutch Distributed File System. In the same year, Map-reduce were introduced by Google in a research paper. During 2005, the Nutch developers have completely integrated a production version of Map-reduce and Nutch Distributed File System into Nutch. By 2006, developers found out that Nutch Distributed File System and Map-reduce can be used in many other fields other than for search engine so the project was separated from Nutch and formed its own subproject of Lucene called Hadoop.

**Figure 1. The history of Hadoop**

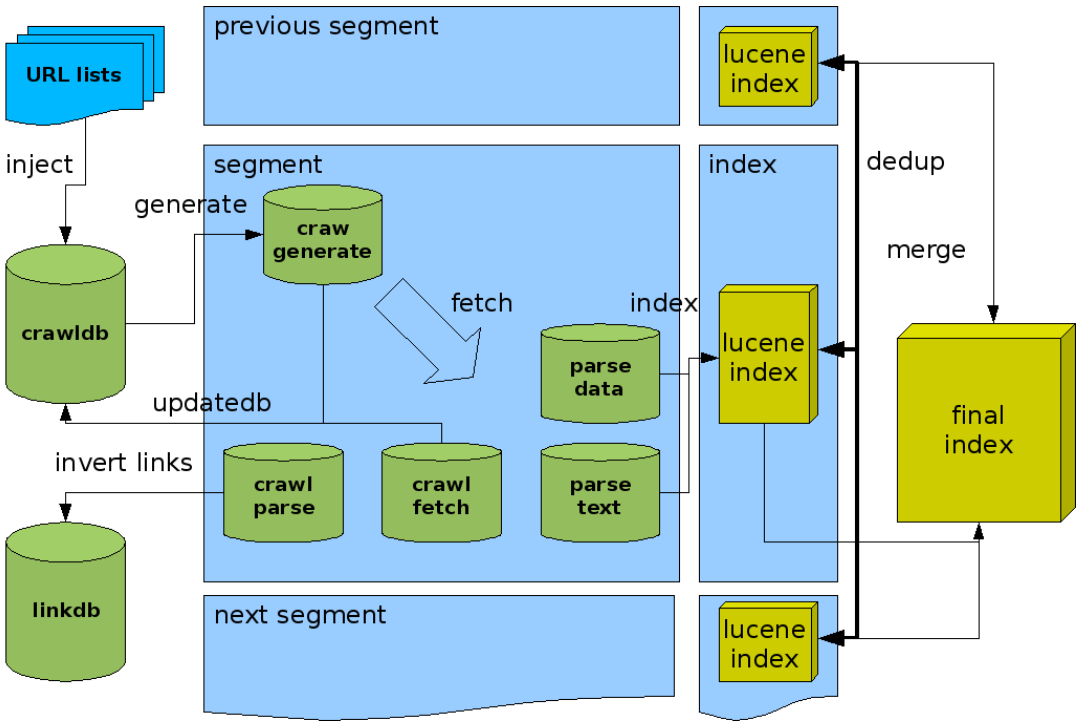



## Topics

### Nutch

The goal of the Nutch project was to be able to realize a web-scale, crawler-based search engine. As for the current version of the project, it has two main versions and has significant difference in its purpose. The first version of Nutch 1.x is a well finished and in production. It mainly relies on the Apache Hadoop data structure and the main strengths of its functionality are for batch job processing. The second version of Nutch 2.x is build off the first version, but the main difference is that it no longer has a well-defined data structure thanks to Apache Gora. This way, objects are handled much more flexibly and one data model can be implemented to be compatible with storing all kind of data.

**Figure 2. The architecture of Nutch**

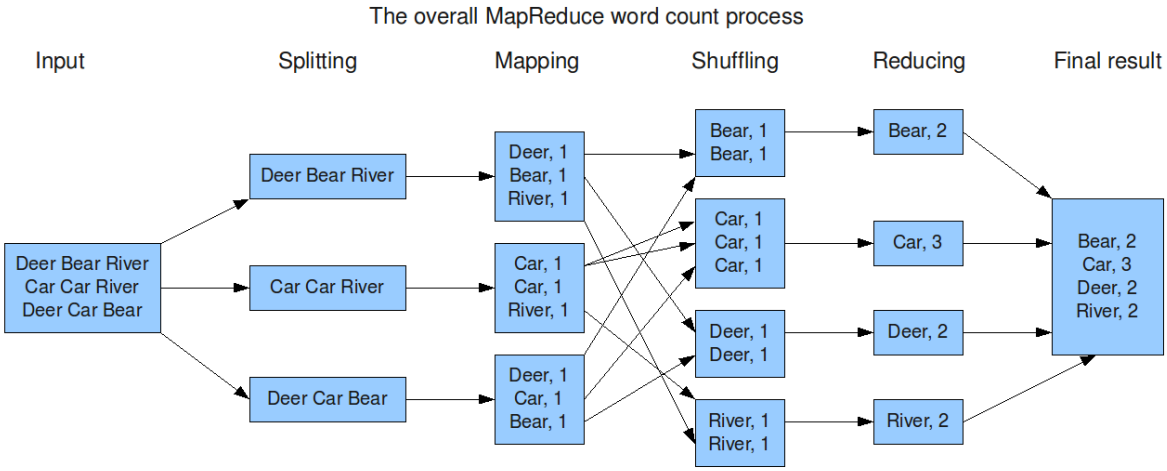



Nutch is divided into two main parts: The crawler and the searcher. The crawler gets the URL lists as inputs and turns them into invert links. The searcher then uses the invert links and turns them into index to responds to user's request.

## Map-Reduce

Map-Reduce is widely used in many big technology companies, for instance in Google, it has been reported that "…more than ten thousand distinct Map-Reduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand Map-Reduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day." (Dean) One of the reasons why Map-Reduce is so popular is that programmer find it very easy to use. The program Map-Reduce is very self-explanatory in its naming. There are two parts to this programming model. First, the map part takes a set of data and converts it into another set of data using methods specific to the domain such as filtering or sorting algorithms. The data will be also broken down into a key-value pair and get passed on to the Reduce part of the programming model. The reduce part takes the outputs from the map function and use it as input for the combine of the data. The data are combined to form even smaller set of data, usually having a count value for the reduced set of data. The figure below is a simple example of applying Map-Reduce to a set of inputted words.

**Figure 3. The Map-Reduce example**




As you can see, in this Map-Reduce function, a set of inputted data is passed into the mapping part and split based on their names. Then it is passed into the reduce function and rearranged into the final set of individual key/value pair. Having explained how Map-Reduce work in higher structure, now it is the time to express the key detail of its usage in technical terms. The code for creating a Map-Reduce program requires 3 different components. It consists of a Map function, a Reduce function, and the code that runs the job. A brief touch on one of the 3 components, the map function is incorporated in a generic mapper, where Hadoop uses its own set of data type that works much more efficiently for the inputted data. The inputted text value is converted into a Java String and uses the substring function to retrieve the data we are looking for.

As for the progression of the updates for Map-Reduce API, there are several new changes with the newer version. For instance, the new API 0.20.0 preferred abstract classes over interfaces since it ease the integration part of implementing a new functions without breaking the old structure of the class. In addition, the configuration in the newer version has been combined into a centralized location. In the older version, the job configuration is set up in a JobConf object, which involves its own declaration of XML documents. But in the newer version, this specific declaration is removed and it is included with every other configuration.

## Hadoop

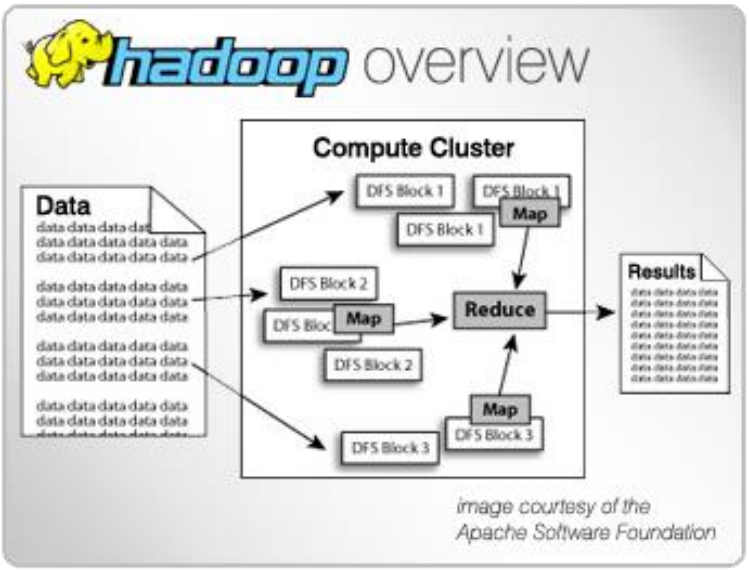
Hadoop software library is an open source framework that allows the distributed computing of large amount of data using the Map-Reduce programming model. The software itself is able to detect and handle the failure during the computation. All of the components are designed so they can detect the occurrence of failure and will let the framework handle it. Some of the components of the Hadoop project are:

- Hadoop Common – The module that contains common utilities and libraries that support the other Hadoop modules.

- HDFS – Hadoop Distributed File System, a distributed file system that stores data on common hardware and provides access to large amount of application data.
- Hadoop Yarn - A resource management platform that manages cluster resource and job scheduling.
- Hadoop MapReduce - A programming model based on Yarn for large scale data processing.
- Other related projects discussed in this chapter include:
- HBase - A scalable, distributed database built on top of HDFS that supports structured data storage for large tables.
- Apache HIVE - A data warehouse infrastructure that provides data summarization and analysis of large data set in HDFS.

In short, Hadoop project is the entire architecture of Hadoop family and consist of all the different components that provide the capability of processing big data. Hadoop applies to many fields. For example, in finance, accurate portfolio evaluation and risk analysis require very complicated model and it will be difficult to be stored in a traditional database. This is where Hadoop comes in. It will store all the data and perform deep and computationally extensive analysis.

**Figure 4. The architecture of Hadoop**




Here is the official logo of Hadoop and the overall software architecture. As you can see, data are passed into the compute cluster and divided using HDFS and Map-Reduce. The resulting data is then well formatted and outputted. The concept of Map-reduce and HDFS are presented in their individual section.

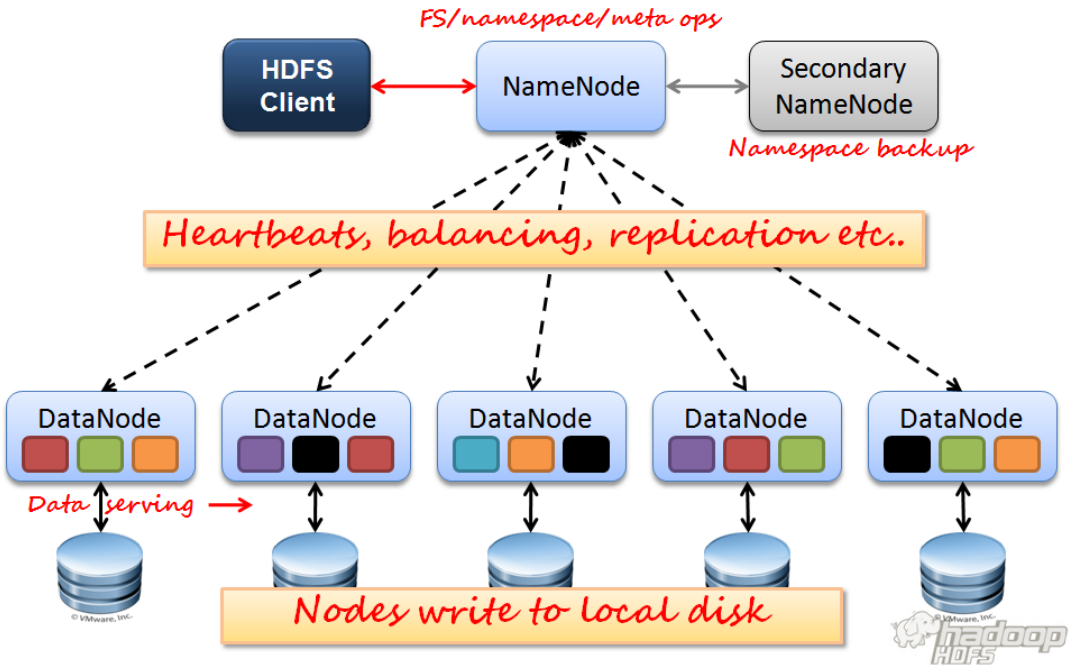
**HDFS**

HDFS stands for Hadoop Distributed File System. When a set of data exceed the storage capacity of the system that is processing the data, the HDFS comes in to distribute the data across multiple system. When this distribution occurs, one of the biggest issues that need to be handled is having a suitable failure tolerable mechanism and recovery method within the system to ensure no data loss.

HDFS has several great capabilities. HDFS is designed for storing very large files; it can go up to megabytes to even terabytes in size. HDFS is also designed so it can run on commonly available hardware that is not very expensive or highly reliable. With the system designed to handle node failure the tasks can be carrying on without visible disruption to the user. On the other hand, HDFS has a few currently known issues that make it not as compatibles in some scenarios. For instance, because HDFS is optimized for processing a big amount of data, it does not work so well with applications that requires low-latency access to data. In this scenario, an additional layer of HBase on top of HDFS is a more suitable choice and it will be discussed under HBase section. When the number of files became too big, the distributed system will not be able to store them. Each of the name nodes requires memory in the system, on average, a file and directory takes about 150 bytes of memory, so even though the data within the file can be stored without problem, the number of files will go beyond the storage capacity.

The structure of HDFS is a master/slave model. The HDFS cluster will have one single name node, the master server that organize the namespace and control the files that are accessed by clients. Then under the name node, there are several data nodes that manage storage attached to the nodes. They store and retrieve blocks as the name node or the clients requested and send back the set of blocks that carry those information. The blocks are stored internally in the name node and they are much larger than a normal block in a disk. The default for the block is 64MB and files are broken into block-sized chunks to be stored. There are several benefits of having a block structure for the distributed system. First, since a file can be larger than the disk in the network, the file can be divided into several blocks and to be stored on different disks. This way, the file can actually be processed in parallel. In addition, for fault tolerance and recovery, block structure is easily replicated from another disk and bring the process back to normal.

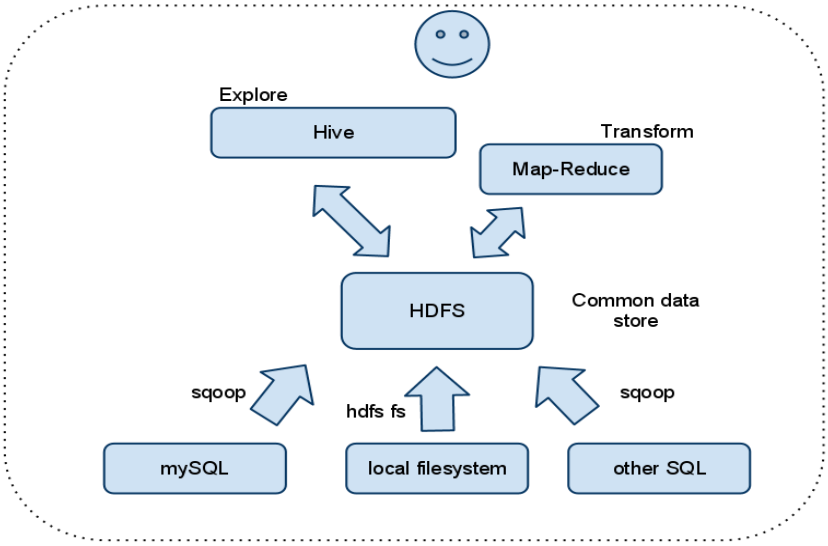
**Figure 5. The architecture of HDFS**




Since HDFS is built using the Java language, any machine that supports Java can run the name node or the data node software. There exists a variety of other interfaces that are compatible using HDFS by different methods, this include Thrift, C, FUSE, WebDAV, HTTP

and FTP. Usually, the other file system interfaces need additional integration in order to access HDFS. For example, for some non-Java applications that have Thrift bindings, they use the Thrift API in their implementation by accessing the Thrift service and ease the interaction to Hadoop.

**Figure 6. Interaction of HDFS with other components**

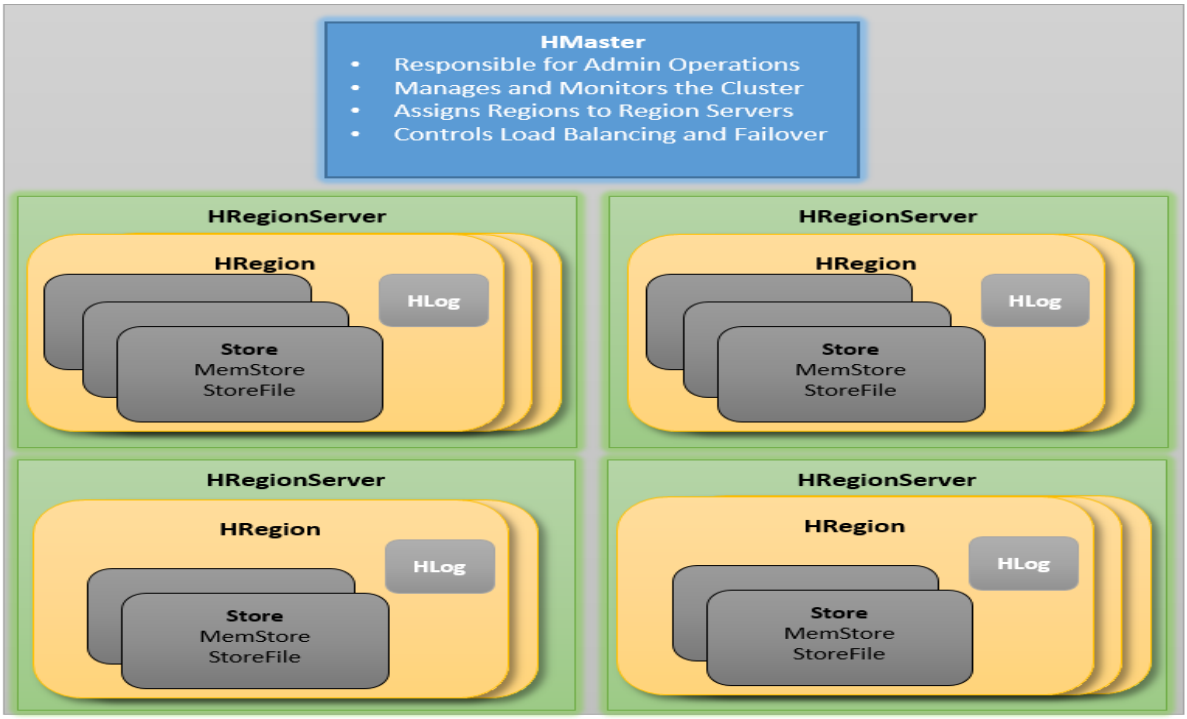



As you can see in this architecture diagram, HDFS interacts with other components of Apache Hadoop to distribute files and data as requested.

## HBase

HBase is a scalable, distributed database built on top of HDFS that supports structured data storage for large tables. It is used when the application requires real time read and write random access to large data set. HBase is designed to solve the scaling problem from a different perspective than most other solutions. It is built from scratch just by adding nodes. In comparison with the relational database systems, HBase applications are actually written in Java. For this reason, HBase is a NoSQL type database and it is neither relational nor supporting SQL. But it is capable of solving the problem a relational database management system cannot: it can store large data table on clusters made from commodity hardware. It lacks several features that are in RDBMS, for example, common functionality such as secondary indexes, triggers, typed columns, and advance query language are not part of HBase. But it also features several benefits in sacrificing those properties. Here are a few key features of HBase: since it is written in Java, it facilitates clients' access through Java API. It has been designed so the base classes provide great recovery for MapReduce jobs by storing information in the HBase table. HBase table has the capability of automatically redistribute data to different regions as it grows. In addition, the architecture of HBase is constructed so reads and writes to the table are very consistent throughout the access.

**Figure 7. The architure of HBase**



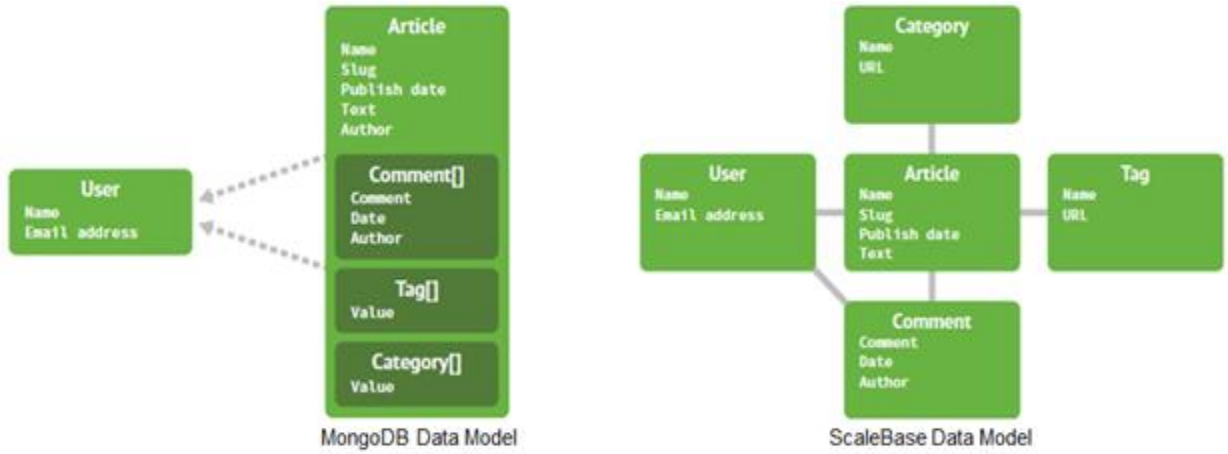

Similar to the structure of HDFS, the architecture of HBase is also in the form of Master/Slave relationship. HBase application typically will have a master node and multiple region servers as work horses. Each region server contains several region and data are stored in tables and these tables are then stored in each region. From a top down perspective, the architecture of HBase starts with the master node with responsibilities such as managing and monitoring the cluster and assigning regions to the region servers. Then under the master node there are the region servers that manage the regions. The region servers communicate directly with clients and handle the read and write requests accordingly. When the regions' data exceed a limit that is set, the region server automatically gives order to the region and let it split into two region of the same size. Under the region servers are the regions. In this component, a set of table's row are stored within it. As the data grows larger, the region is split into two new regions of similar size. Now under regions there are tables that consist of rows and columns. Similar to RDBMS, each row has a primary key; the main differences in HBase are that the intersection of row and column coordinates is versioned and the rows are sorted.

## MongoDB

MongoDB is one of the best examples of NoSQL database and it is widely used by many Fortune 500 companies to make their businesses more agile and scalable. MongoDB is a cross-platform document oriented database. MongoDB was originally created by 10gen in October 2007 and went open source in 2009. Since then, MongoDB has been widely used by several top websites and services as their back end component, this include "Craigslist, eBay, Foursquare, SourceForge, and the New York Times."(MongoDB) It is an agile database that can change its schemas as the application evolves, while keeping the basic functionalities from the traditional

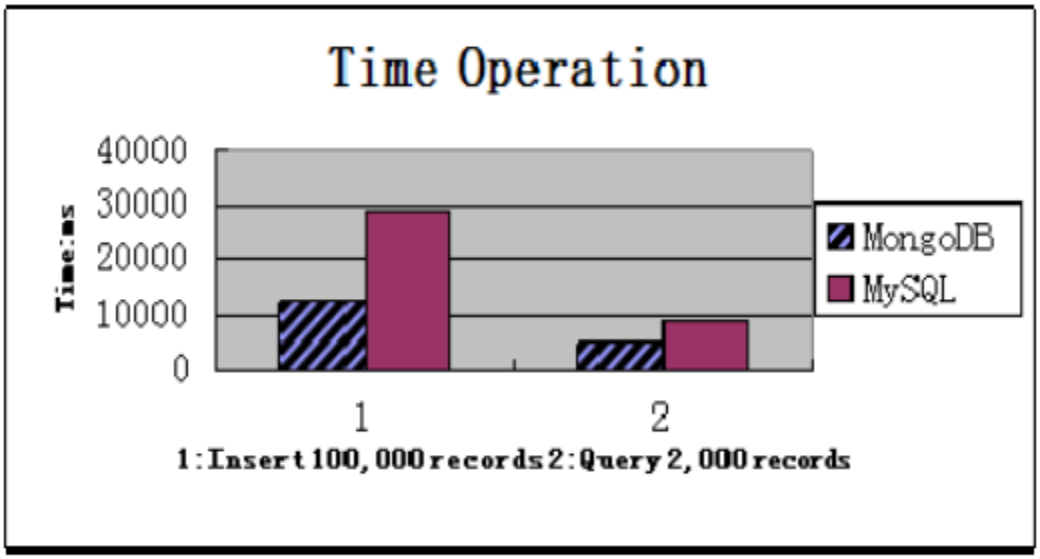
databases such as secondary indexing and have an advanced query language. MongoDB is designed so data has a dynamic schema. Rather than having the data stored in a typical 2 dimensional database, MongoDB stores data in a binary representation called BSON, which stands for Binary JavaScript Object Annotation. In the database, there are several collections of documents, and these documents do not have a specific format. For this reason, the data model can be adjusted based on the application requirements and optimize its performance. To make it easy to visualize, you can think of the collections as the tables and the documents as the rows in the relational database.

**Figure 8. MongoDB versus MySQL**




To compare the performance of MongoDB versus the performance of MySQL in certain application, consider the following blogger application. In our scenario, let's consider having information on users, articles, comments, and categories. As image 6 indicates, in a traditional relational database, all data would be stored in several tables with each table having one type of information. Each table will be connected through a foreign key. In order to find an article with all necessary information, the application would have to query at least several keys to obtain the complete information on one specific article. For the data model created with MongoDB, data will only need to be stored in two distinct collections, namely users and articles. Within each collection, category, comments, and other relevant information about the same article will be stored. This way, an article can be easily retrieved by accessing a single collection versus querying several tables. In summary, MongoDB stores all information in a single item within the same collection while the traditional database stores information scarcely across several tables in the system.

**Figure 9. Performance Comparison for textbook insertion and query**




The above comparison shows the time it takes to insert 100000 textbook records in the first operation and the time it takes to query 2000 textbook records. As you can see the runtime for inserting records in MySQL exceeds MongoDB by almost three times and the querying runtime almost doubled.

**Apache HIVE**

Apache HIVE is a data warehouse solution on top of Hadoop Map-Reduce framework that provides similar functionalities to RDBMS. It was initially developed by Facebook, but later on it was also implemented and developed by Netflix and Amazon. Apache HIVE allows users to access the data stored in it the same way as how user would access them in a regular relational database. Hive provides the capability of generating tables and also has a query language called HiveQL. HiveQL is based on SQL thus it is very easy for common database users to learn and use it in practice. HiveQL currently has several capabilities similar to SQL. For instance, it has the functionality CREATE and DROP to manipulate tables and partitions. Most importantly, it features the function SELECT capable of joining tables on a mutual key, and filter data using the row selection techniques. Here is an example of HiveQL query.

**Figure 10. HiveQL query**

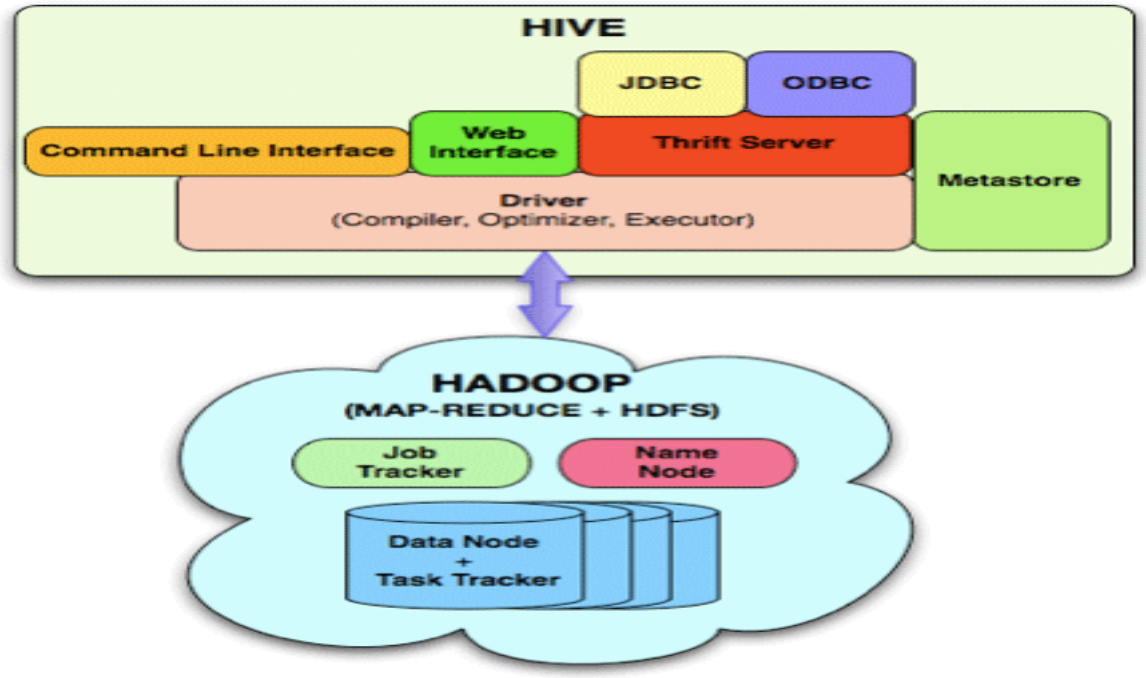
```
SELECT o_orderkey, o_custkey, c_custkey
FROM customer c JOIN
        orders o ON c.c_custkey = o.o_custkey JOIN
        lineitem l ON o.o_orderkey = l.l_orderkey;
```

When a query is performed against Hive, the query is analyzed by a semantic analyzer and translated into a query execution. This process is then send to Map-Reduce framework and uses data stored in Hadoop Distributed File System as input.

Even though Apache HIVE provides similar capabilities to SQL, it cannot be compare with traditional system in certain perspectives. Hadoop jobs tend to have very long runtime in job submission and scheduling. For this reason, HiveQL query also tends to take long time before it can be completed. As a comparison, system such as Oracle will use much less data for analysis and can be completed in a fast pace. Hive is definitely not optimal in compare with traditionally established system but rather used for interactive data browsing, querying smaller data set, and for testing non production data.

As the figure shows below, the architecture of Apache HIVE contains a few important components. Command line interface interacts with users and allows them to enter HiveQL queries. Driver is the ultimate processing tool that receives the queries and processes them with its components. The Metastore serves as storing the metadata on different tables and partitions. Compiler takes the query and metadata from Metastore to generate an execution plan. During this process, the optimizer takes the execution plan and translates it into an executable plan with multiple Map-Reduce steps. The executor then executes the plan generated by the compiler using Map-Reduce engine.

**Figure 11. Apache HIVE architecture**




**Relationship between the topics**

To summarize, the project Nutch, an open source web search engine, was created as part of the Lucene project, the text search library. Then due to non-efficient time consumption for processing large amount of data, Hadoop project was initialized as a result of it. Hadoop Distributed File System, Map-Reduce, HBase, and MongoDB were all part of the Hadoop developing projects. HBase is built on top of Hadoop Distributed File System and it is created to satisfy applications that requires low-latency access to data. MongoDB serves as a NoSQL database in Hadoop and it makes it much more efficient for application with large data because of its object oriented structure versus traditional database. Hive is a data warehousing

architecture on top of Hadoop Map-Reduce framework for users to be capable of handling data the same way users would access a traditional relational database management system. Hadoop Distributed File System is the centralized file processing architecture and it is used by Hive to store the data. Hive uses Map-Reduce engine to execute the logical plan of the HiveQL query and retrieve data from HDFS.

**References**

[1] Tom Whites, Hadoop The Definitive Guide, O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2011

[2] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, New York, NY, USA, January 2008, Pages 107-113.

[3] An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics, http://www.biomedcentral.com/content/pdf/1471-2105-11-S12-S1.pdf, accessed: 04/21/2014

[4] What Is Apache Hadoop?, http://hadoop.apache.org/, accessed: 04/26/2014

[5] Zhu Wei-ping, "Using MongoDB to implement textbook management system instead of MySQL", Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference. Page 303 – 305.

[6] MongoDB, http://en.wikipedia.org/wiki/MongoDB, accessed: 04/27/2014

Picture sources - (Other pictures are from the references documents))

http://yoyoclouds.wordpress.com/tag/hdfs/

http://xiaochongzhang.me/blog/?p=334

http://mmcg.z52.ru/drupal/node/3

http://www-01.ibm.com/software/ebusiness/jstart/hadoop/

http://xiaochongzhang.me/blog/?p=338

http://www.scalebase.com/extreme-scalability-with-mongodb-and-mysql-part-2-data-distribution-reads-writes-and-data-redistribution/

http://practicalanalytics.wordpress.com/2011/11/06/explaining-hadoop-to-management-whats-the-big-data-deal/

http://home.in.tum.de/~gruenhei/Agruenheid_ideas11.pdf

http://www.cubrid.org/blog/dev-platform/platforms-for-big-data/