Chapter 5: Big Data and Health care

5.1 Introduction

5.2 Big Data Definition

5.3 Big Data Application and Health care

5.4 Big Data Analytics

5.5 Apache Hadoop and MapReduce framework

5.6 Apache Spark, Hive, Hbase and Pig.

**5.2 Big Data Definition**

Today’s **Big data** is a hot topic that directs ways to investigate analytically to extract information from data sets that are too large in size and it becomes complex to be processed by traditional data-processing application software within the tolerable acceptable time.Big data can be unstructured, semi-structured and structured data. However, the main focus is on unstructured data.There are several numbers of characteristics of Big Data. The most important features of big data are given below.

1. **Volume**

The amount of produced and stored data is too large in size. For example, Facebook and generate above 500TB and 20 PB data in day respectively and about 1TB of new trade data is generated per day in The New York Stock Exchange.

1. **Variety**

This describes the different types and natures of the data. Big data can be generated from text, images, audio, video.

1. **Velocity**

The velocity means the speed at which the data is generated and processed.

1. **Veracity**

It refers to the uncertainty of data quality and the data value.

5.3 Big Data Application and Health care

Big data has augmented the mandate of information supervision experts so much so that Software AG, Oracle Corporation, IBM, Microsoft, SAP, EMC, HP and Dell have paid more than $150 billion on data management and analytics. The following are the application fields of big data.

1. Government
2. International development
3. Manufacturing
4. Healthcare
5. Education
6. Media
7. Insurance
8. Internet of Things (IoT)
9. Information technology
10. Sports

Big data analytics has been facilitating healthcare improvement by providing personalized medicine, medical risk interference and predictive analytics, automatic reporting of patient data. Some fields of development are more ambitious than actually implemented. The volume of data produced within healthcare organizations is not small. With the additional implementation of m-Health, s-Health and e-Health the volume of data continuously increased. The electronic health record data, imaging data, patient created data, sensor data, and other systems generated data are the sources of data and difficult to process. The practice of big data in healthcare has elevated important moral challenges ranging from risks for distinct rights, confidentiality and independence, to transparency and faith.

Big data in health research is predominantly auspicious in terms of tentative biomedical research, as data-driven investigation can change more rapidly than hypothesis-driven research.

**5.4 Big Data Analytics**

Big data analytics is a complex process of investigating huge and diverse data sets and to reveal information like hidden patterns, unidentified relationships, marketplace trends and client preferences that can help making well-versed decisions.

Big data analytics allow big data analysts, data scientists, analytical modelers and other professional to scrutinise increasing bulks of structured transaction data, semi-structured and unstructured data. Internet search data, server logs, social media, survey and sensors data that are the examples of the mix of semi-structured and unstructured data should be dealt with big data analytics.

**5.5 Apache Hadoop and MapReduce framework**

The most powerful tool for analysing big data is known as Apache Hadoop. Apache Hadoop is defined as an open source framework for storage and processing data. Hadoop consists of four main parts:

* The Hadoop Distributed File System (HDFS) is a distributed file system designed to store data on commodity machines and it provides very high bandwidth across the cluster. Dananode and Namenode are the two components of HDFS.
* YARN (Yet Another Resource Negotiator) is a platform for the management of Hadoop's resources and job scheduling programs that will run on the Hadoop distributed processing framework. It has two basic components, those are the ResourceManager and NodeManager.
* Hadoop Common is a set libraries and utilities.
* Hadoop MapReduce is an implementation of the programming model. MapReduce performs two basic functions. The functions are mapping and reducing.

**Hadoop Architecture**

Hadoop has a master and slave topology. One master node and several nodes for slaves are the main components. The function of master node is to allocate a job to multiple slave nodes and to manage resources. the actual execution is performed by the slave nods.

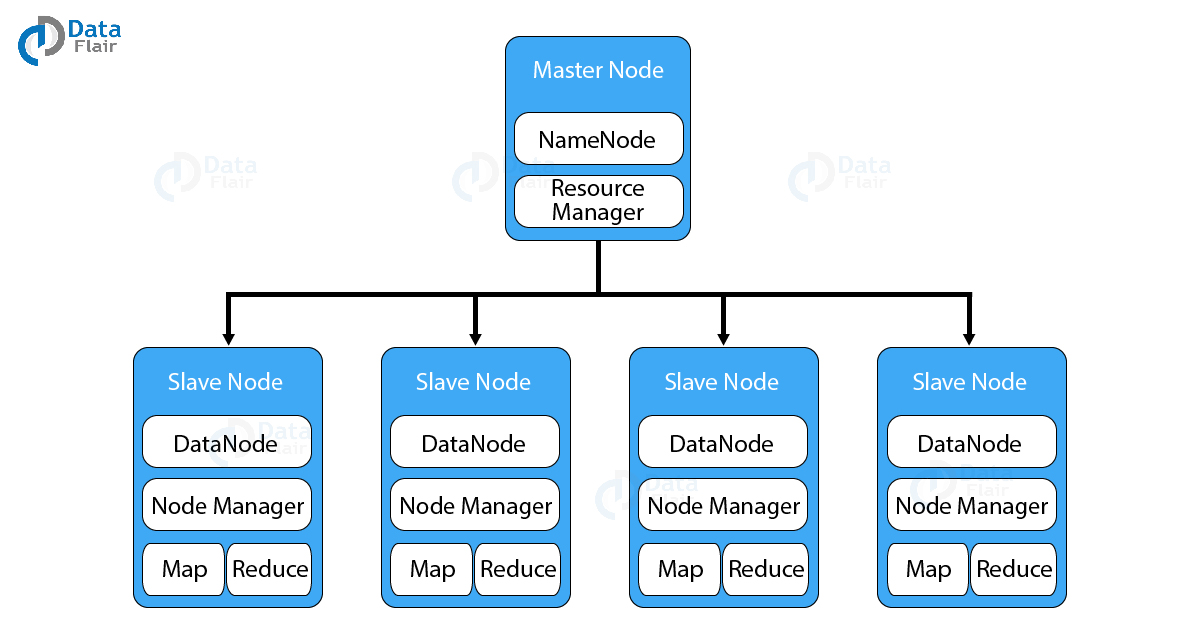
Hadoop Architecture encompasses three major layers. They are

* HDFS
* Yarn
* MapReduce

HDFS runs for data storage of Hadoop. HDFS divide the data into smaller blocks units and stores blocks in a distributed manner. The default block size is 128MB or 256 MB. Two daemons running in HDFS are one for master node called NameNode and other for slave nodes called DataNode.

NameNode daemon controls the space management and file access by the client, opening, closing and renaming files, keeps track of mapping of modifications to file system and blocks to DataNodes or directories.

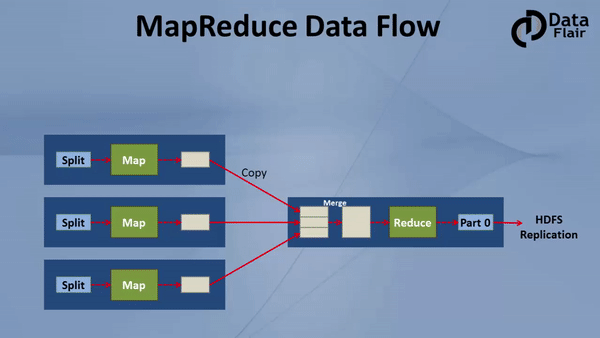
DataNode daemons are used for storing actual data. The smaller data unit are stored on a group of slave machines. This DataNodes perform the read and write request from the file system’s client. DataNode is responsible for the creation, deletion and replication of blocks on demand from NameNode.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/02/Hadoop-Architecture2-01.jpg)

2. MapReduce

**MapReduce** is a software framework that allows to write applications for processing big data. These applications can be run in parallel on a cluster. It is so reliable and fault-tolerant.

MapReduce job encompasses mapping tasks and reducing tasks.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2017/01/hadoop-mapreduce-data-flow-execution-1.gif)

Map Task

***The Map task run in the following phases:-***

#### a. RecordReader

The [**recordreader**](https://data-flair.training/blogs/hadoop-recordreader/)transforms the input split into records. It parses the data into records but does not parse records itself. It provides the data to the mapper function in key-value pairs. Usually, the key is the positional information and value is the data that comprises the record.

#### b. Map

In this phase, the [**mapper**](https://data-flair.training/blogs/hadoop-mapper-in-mapreduce/) which is the user-defined function processes the key-value pair from the recordreader. It produces zero or multiple intermediate key-value pairs.

The decision of what will be the key-value pair lies on the mapper function. The key is usually the data on which the reducer function does the grouping operation. And value is the data which gets aggregated to get the final result in the reducer function.

#### c. Combiner

The [**combiner is actually a localized reducer**](https://data-flair.training/blogs/hadoop-combiner-tutoria) which groups the data in the map phase. It is optional. Combiner takes the intermediate data from the mapper and aggregates them. It does so within the small scope of one mapper. In many situations, this decreases the amount of data needed to move over the network. For example, moving (Hello World, 1) three times consumes more network bandwidth than moving (Hello World, 3). Combiner provides extreme performance gain with no drawbacks. The combiner is not guaranteed to execute. Hence it is not of overall algorithm.

#### d. Partitioner

[**Partitioner pulls the intermediate key-value pairs**](https://data-flair.training/blogs/hadoop-partitioner-tutorial/) from the mapper. It splits them into shards, one shard per reducer. By default, partitioner fetches the hashcode of the key. The partitioner performs modulus operation by a number of reducers: key.hashcode()%(number of reducers). This distributes the keyspace evenly over the reducers. It also ensures that key with the same value but from different mappers end up into the same reducer. The partitioned data gets written on the local file system from each map task. It waits there so that reducer can pull it.

### b. Reduce Task

***The various phases in reduce task are as follows:***

#### i. Shuffle and Sort

The reducer starts with shuffle and sort step. This step downloads the data written by partitioner to the machine where reducer is running. This step sorts the individual data pieces into a large data list. The purpose of this sort is to collect the equivalent keys together. The framework does this so that we could iterate over it easily in the reduce task. This phase is not customizable. The framework handles everything automatically. However, the developer has control over how the keys get sorted and grouped through a comparator object.

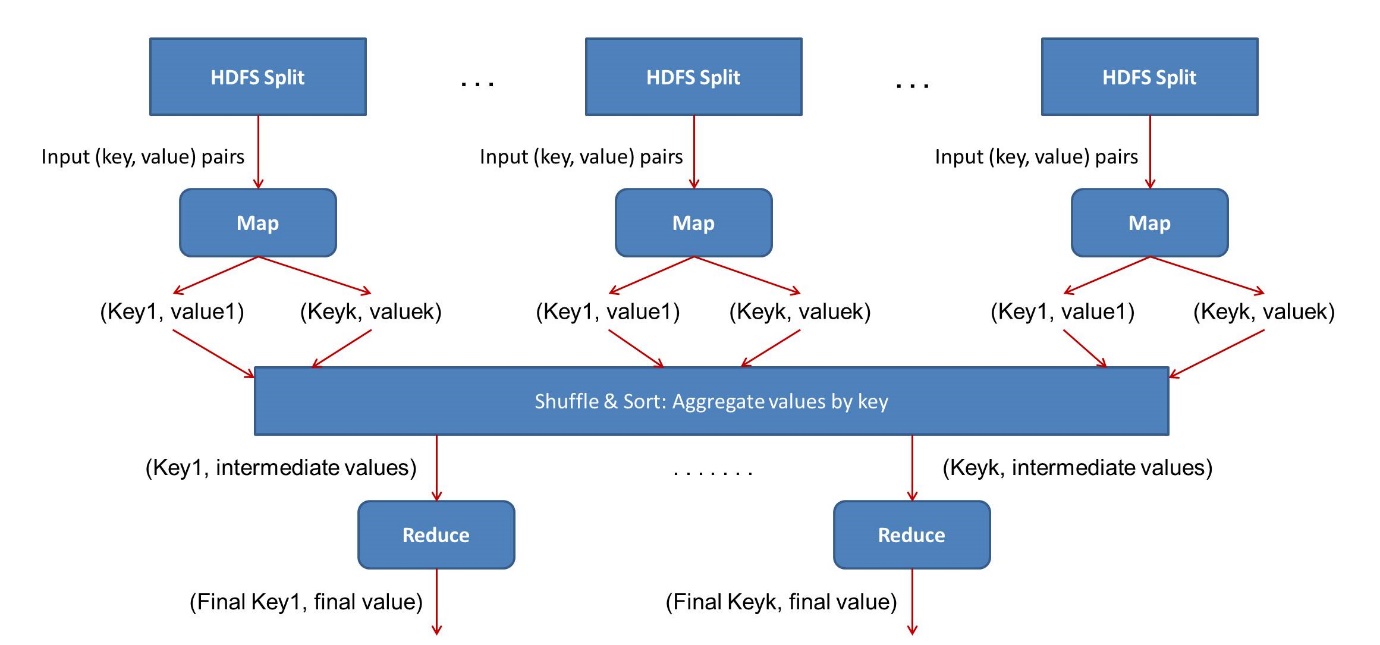
#### ii. Reduce

The [**reducer performs the reduce function**](https://data-flair.training/blogs/hadoop-reducer/) once per key grouping. The framework passes the function key and an iterator object containing all the values pertaining to the key.

We can write reducer to filter, aggregate and combine data in a number of different ways. Once the reduce function gets finished it gives zero or more key-value pairs to the outputformat. Like map function, reduce function changes from job to job. As it is the core logic of the solution.

#### iii. OutputFormat

This is the final step. It takes the key-value pair from the reducer and writes it to the file by recordwriter. By default, it separates the key and value by a tab and each record by a newline character. We can customize it to provide richer output format. But none the less final data gets written to HDFS.

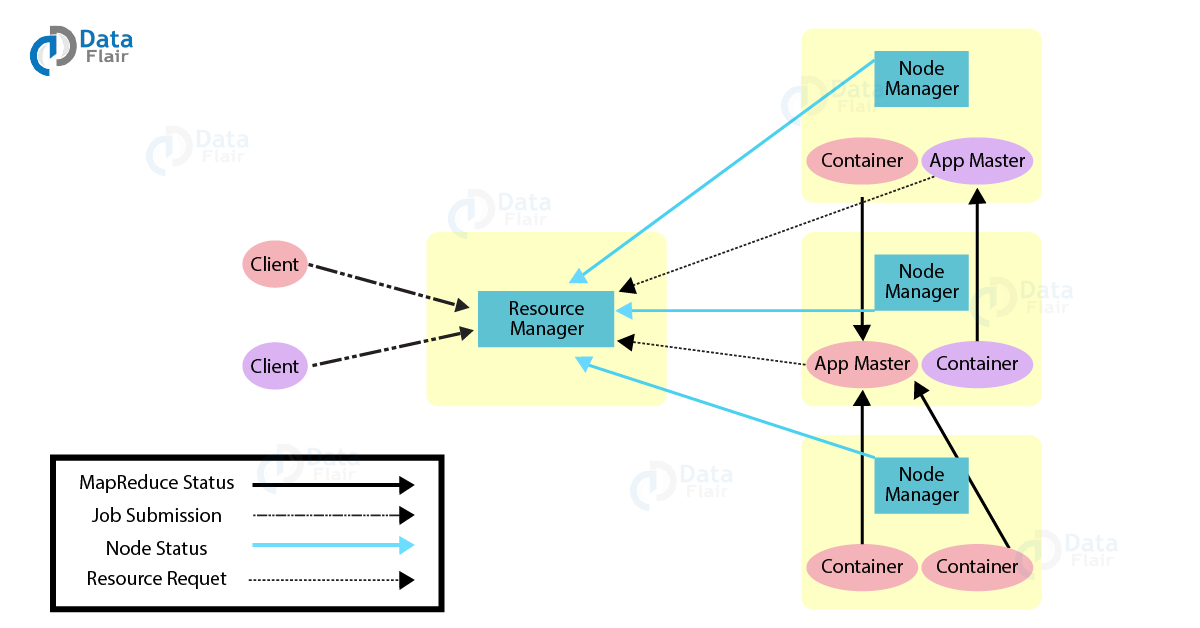
[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2016/11/how-map-reduce-work-together-tutorial.jpg)

## 3. YARN

YARN or Yet Another Resource Negotiator is the resource management layer of Hadoop. The [**basic principle behind YARN**](https://data-flair.training/blogs/hadoop-yarn-tutorial/) is to separate resource management and job scheduling/monitoring function into separate daemons. In YARN there is one global ResourceManager and per-application ApplicationMaster. An Application can be a single job or a DAG of jobs.

Inside the YARN framework, we have two daemons [**ResourceManager**](https://data-flair.training/blogs/hadoop-yarn-resource-manager/) and NodeManager. The ResourceManager arbitrates resources among all the competing applications in the system. The job of NodeManger is to monitor the resource usage by the container and report the same to ResourceManger. The resources are like CPU, memory, disk, network and so on.

The ApplcationMaster negotiates resources with ResourceManager and [**works with NodeManger**](https://data-flair.training/blogs/hadoop-yarn-node-manager-tutorial/) to execute and monitor the job.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/02/Yarn-Architecture.png)

The ResourceManger has two important components – Scheduler and ApplicationManager

### i. Scheduler

Scheduler is responsible for allocating resources to various applications. This is a pure scheduler as it does not perform tracking of status for the application. It also does not reschedule the tasks which fail due to software or hardware errors. The scheduler allocates the resources based on the requirements of the applications.

### ii. Application Manager

[](https://data-flair.training/blogs/hadoop-tutorials-home/#quizzes)

Following are the functions of ApplicationManager

* Accepts job submission.
* Negotiates the first container for executing ApplicationMaster. A container incorporates elements such as CPU, memory, disk, and network.
* Restarts the ApplicationMaster container on failure.

Functions of ApplicationMaster:-

* Negotiates resource container from Scheduler.
* Tracks the resource container status.
* Monitors progress of the application.

We can scale the YARN beyond a few thousand nodes through YARN Federation feature. This feature enables us to tie multiple **YARN clusters** into a single massive cluster. This allows for using independent clusters, clubbed together for a very large job.