Chapter - I

**Introduction to Hadoop**

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage. Hadoop can manage SQL as well as No SQL databases efficiently. However, it is not preferred for smaller databases.

**Key Aspects of Hadoop:**

1. **Open source software**
2. **Highly Scalable Cluster:**

Hadoop includes everything that we need to develop and execute.

**3.** **Distributed:**

Hadoop divides and stores data across multiple computers. Computation/ Processing is done in parallel across multiple connected nodes/computers.

**4.** **Massive Storage:**

Hadoop stores vast amount of data across nodes of low-cost commodity hardware.

**5. Fault Tolerance is Available:**

In Hadoop data is replicated on various DataNodes in a Hadoop cluster which ensures the availability of data if somehow any of your systems got crashed. You can read all of the data from a single machine if this machine faces a technical issue data can also be read from other nodes in a Hadoop cluster because the data is copied or replicated by default.

**History of Hadoop:**

Apache Software Foundation is the developer of Hadoop, and its co-founders are Doug Cutting and Mike Cafarella. It’s co-founder Doug Cutting named it on his son’s toy elephant. In October 2003 the first paper release was Google File System. In January 2006, MapReduce development started on the Apache Nutch which consisted of around 6000 lines coding for it and around 5000 lines coding for HDFS. In April 2006 Hadoop 0.1.0 was released.

**Core Components of Hadoop Eco-system:**

Over the years different generations of hadoop were released and additional components were included in each release. Here, we will take a brief loop at different components in hadoop eco-system

**1. HDFS:**

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project

**2. YARN:**

It was introduced in Hadoop 2.0. It allows processing and running batch processing on data, stream processing, interactive processing and graph processing which are stored in HDFS. In this way, it helps to run different types of distributed applications other than MapReduce.

**3. MapReduce:**

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

Other Components are,

* **Spark:** It allows In-Memory data processing
* **PIG, HIVE:** It used for query-based processing of data services
* **HBase:** It is used for NoSQL Database
* **Mahout, Spark MLLib:** These contain Machine Learning algorithm libraries which can be used for data analysis
* **Solar, Lucene:** These are used for Searching and Indexing
* **Zookeeper:** Managing cluster
* **Oozie:** Job Scheduling

Following figure summarizes all the essential components of Hadoop 2.0,

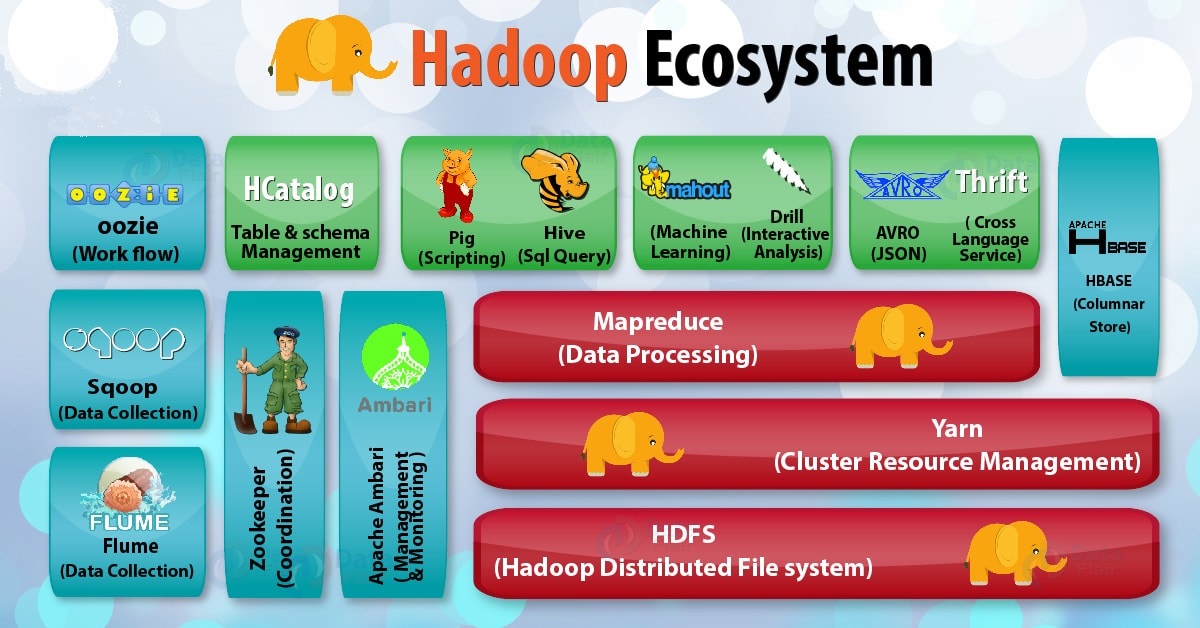


Figure 1: Hadoop Ecosystem

Chapter – II

**Introduction to MapReduce**

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Map task takes care of loading, parsing, transforming, and filtering. The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. Each map task is broken into the following phases:

1. Record Reader
2. Mapper
3. Combiner
4. Partitioner

The output produced by map task is known as intermediate keys and values. These intermediate keys and values are sent to reducer. The reduce tasks are broken into the following phases:

* + - 1. Shuffle
      2. Sort
      3. Compression
      4. Reducer
      5. Output Format

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. Data locality means that data is not moved over network; only computational code is moved to process data which saves network bandwidth.

**MapReduce Workflow:**

Following diagram gives a brief idea about MapReduce working schema.

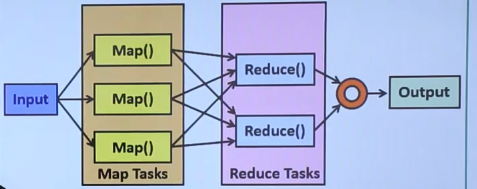


Fig 5: MapReduce Workflow

**MapReduce Architecture:**

The following diagram shows a MapReduce architecture.

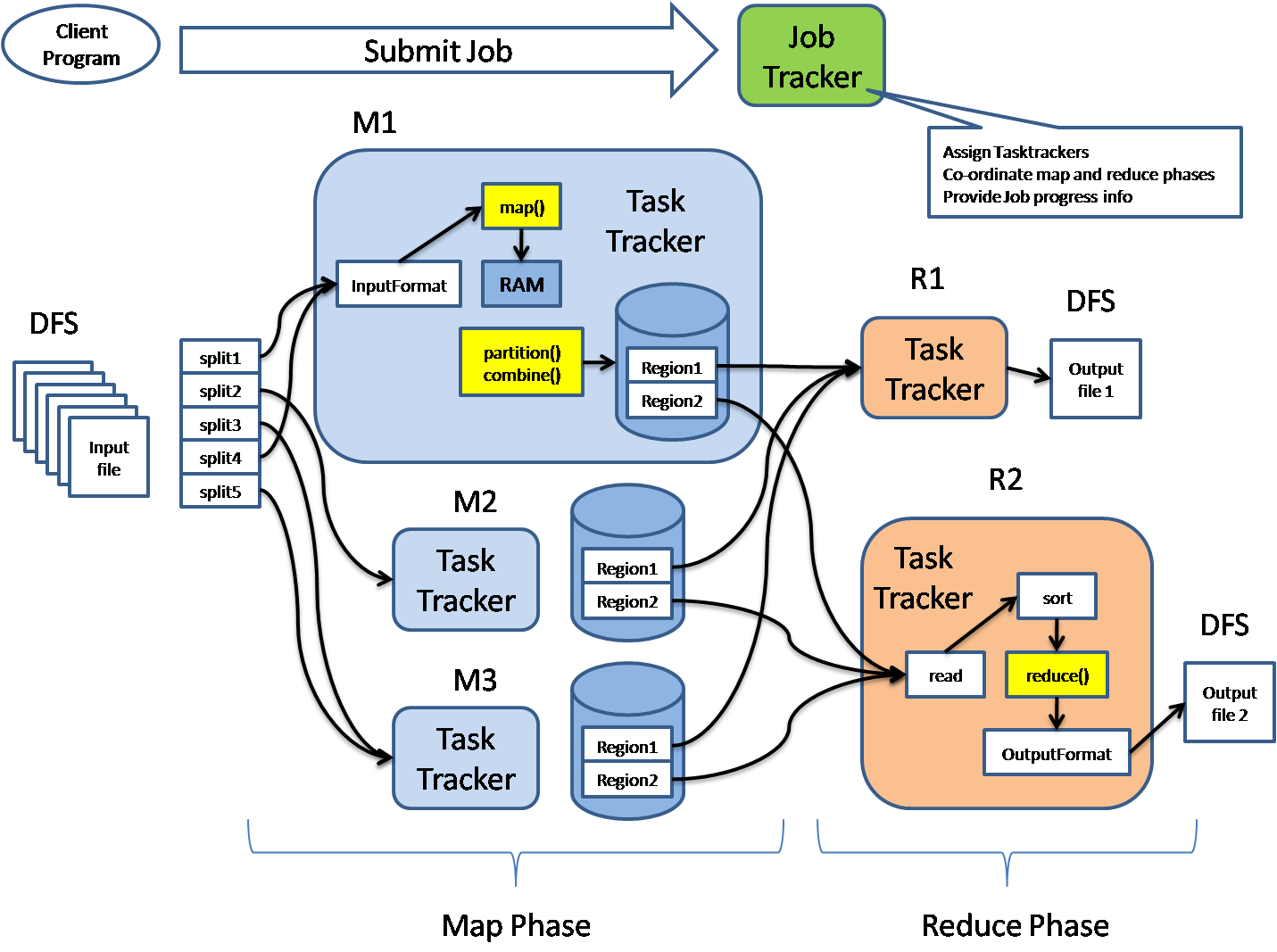


Fig 6: MapReduce Architecture

MapReduce architecture consists of various components. A brief description of these components can improve our understanding on how MapReduce works.

* **Job:** This is the actual work that needs to be executed or processed
* **Task:** This is a piece of the actual work that needs to be executed or processed. A MapReduce job comprises many small tasks that need to be executed.
* **Job Tracker:** This tracker plays the role of scheduling jobs and tracking all jobs assigned to the task tracker.
* **Task Tracker:** This tracker plays the role of tracking tasks and reporting the status of tasks to the job tracker.
* **Input data:** This is the data used to process in the mapping phase.
* **Output data:** This is the result of mapping and reducing.
* **Client:** This is a program or Application Programming Interface (API) that submits jobs to the MapReduce. MapReduce can accept jobs from many clients.
* **Hadoop MapReduce Master:** This plays the role of dividing jobs into job-parts.
* **Job-parts:** These are sub-jobs that result from the division of the main job.

In the MapReduce architecture, clients submit jobs to the MapReduce Master. This master will then sub-divide the job into equal sub-parts. The job-parts will be used for the two main tasks in MapReduce: mapping and reducing. The developer will write logic that satisfies the requirements of the organization or company. The input data will be split and mapped. The intermediate data will then be sorted and merged. The reducer that will generate a final output stored in the HDFS will process the resulting output.

The following diagram shows a simplified flow diagram for the MapReduce program.



Fig 7: MapReduce Flowchart

**Job Trackers and Task Trackers:**

Every job consists of two key components: mapping task and reducing task. The map task plays the role of splitting jobs into job-parts and mapping intermediate data. The reduce task plays the role of shuffling and reducing intermediate data into smaller units.

ssThe job tracker acts as a master. It ensures that we execute all jobs. The job tracker schedules jobs that have been submitted by clients. It will assign jobs to task trackers. Each task tracker consists of a map task and reduces the task. Task trackers report the status of each assigned job to the job tracker. The following diagram summarizes how job trackers and task trackers work.

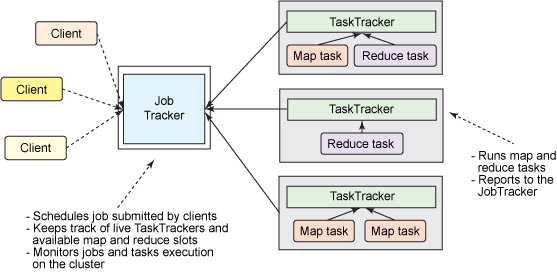


Fig 8: Job Tracker and Task Trackers workflow

Visit following link to have a look at MapReduce program for word count ([link](https://www.javatpoint.com/mapreduce-word-count-example)). Also, all the execution steps are provided there.

Chapter – III

**Mapper & Reducer**

A mapper maps the input key−value pairs into a set of intermediate key–value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key–value pairs. We’ll first see stages of Mapper in detail and then go to reducer part.

1. **Stages/Phases in Mapper**
2. **RecordReader:**

RecordReader converts a byte-oriented view of the input (as generated by the InputSplit) into a record-oriented view and presents it to the Mapper tasks. It presents the tasks with keys and values. Generally, the key is the positional information and value is a chunk of data that constitutes the record.

1. **Map:**

Map function works on the key–value pair produced by RecordReader and generates zero or more intermediate key–value pairs. The MapReduce decides the key–value pair based on the context.

1. **Combiner**:

It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key–value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.

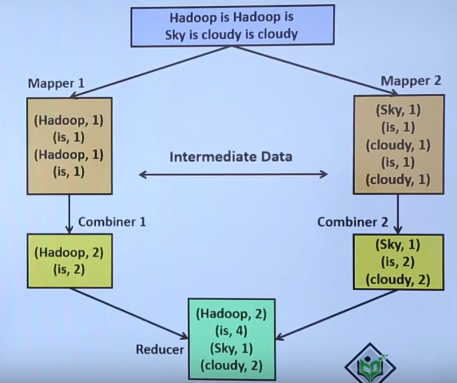


Fig 9: Combiner in action

1. **Partitioner:**

The partitioner takes the intermediate key–value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

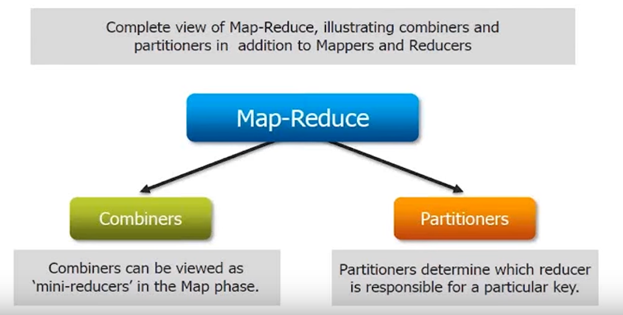


Fig 10: Partitioner

1. **Stages/Phases in Reducer**

The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.

* + - * 1. **Shuffle and Sort:**

This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.

1. **Reduce**:

The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key–value pairs) of reducer is sent to the output format.

1. **Compression:**

It is a very important part which is needed to save storage and processing cost of hadoop ecosystem. As hadoop has to work over large sized problem sets so it must be compressed and saved to HDFS otherwise the resources require will be of enormous amount for storing, processing and sending over networks.

Key features to select right format of compression

1. Faster read and write can be done after compression.
2. Compressed file should be distributive in nature (means compressed file should split if needed).
3. Compressed file should be optimal in storage.
4. Must follow the dynamic schema even after compression.

These are the commonly used format used for compression in hadoop

1. Gzip (It is slower but can perform high compression ratio.)
2. Snappy (It is faster but compression ratio is less than Gzip.)
3. LZO (It is split able when indexed but slower and has less compression ratio.)
4. bZip2 (It is split able and can compress more effectively than Gzip but the process is slow.)
5. AVRO
6. Parquet
7. **Output Format:**

The output format separates key–value pair with tab (default) and writes it out to a file using record writer.

Chapter – IV

**Workflow of MapReduce**

The whole process goes through various MapReduce phases of execution, namely, splitting, mapping, sorting and shuffling, and reducing. Let us explore each phase in detail.

* 1. **InputFiles**

The data that is to be processed by the MapReduce task is stored in input files. These input files are stored in the Hadoop Distributed File System. The file format is arbitrary, while the line-based log files and the binary format can also be used.

* 1. **InputFormat**

It specifies the input-specification for the job. InputFormat validates the MapReduce job input-specification and splits-up the input files into logical InputSplit instances. Each InputSplit is then assigned to the individual Mapper. TextInputFormat is the default InputFormat.

1. **InputSplit**

It represents the data for processing by the individual Mapper. InputSplit typically presents the byte-oriented view of the input. It is the RecordReader responsibility to process and present the record-oriented view. The default InputSplit is the FileSplit.

1. **RecordReader**

RecordReader reads the <key, value> pairs from the InputSplit. It converts a byte-oriented view of the input and presents a record-oriented view to the Mapper implementations for processing. It is responsible for processing record boundaries and presenting the Map tasks with keys and values. The record reader breaks the data into the <key, value> pairs for input to the Mapper.

1. **Mapper**

Mapper maps the input <key, value> pairs to a set of intermediate <key, value> pairs. It processes the input records from the RecordReader and generates the new <key, value> pairs. The <key, value> pairs generated by Mapper are different from the input <key, value> pairs. The generated <key, value> pairs is the output of Mapper known as intermediate output. These intermediate outputs of the Mappers are written to the local disk. The Mappers output is not stored on the Hadoop Distributed File System because this is the temporary data, and writing this data on HDFS will create unnecessary copies. The output of the Mappers is then passed to the Combiner for further processing.

1. **Combiner**

It is also known as the ‘Mini-reducer’. Combiner performs local aggregation on the output of the Mappers. This helps in minimizing data transfer between the Mapper and the Reducer. After the execution of the Combiner function, the output is passed to the Partitioner for further processing.

1. **Partitioner**

When we are working on the MapReduce program with more than one Reducer then only the Partitioner comes into the picture. For only one reducer, we do not use Partitioner. It partitions the keyspace. It controls the partitioning of keys of the Mapper intermediate outputs. Partitioner takes the output from the Combiner and performs partitioning. Key is for deriving the partition typically through the hash function. The number of partitions is similar to the number of reduce tasks. HashPartitioner is the default Partitioner.

1. **Shuffling and Sorting**

The input to the Reducer is always the sorted intermediate output of the mappers. After combining and partitioning, the framework via HTTP fetches all the relevant partitions of the output of all the mappers. Once the output of all the mappers is shuffled, the framework groups the Reducer inputs on the basis of the keys. This is then provided as an input to the Reducer.

1. **Reducer**

Reducer then reduces the set of intermediate values who shares a key to the smaller set of values. The output of reducer is the final output. This output is stored in the Hadoop Distributed File System.

1. **RecordWriter**

RecordWriter writes the output (key, value pairs) of Reducer to an output file. It writes the MapReduce job outputs to the FileSystem.

1. **OutputFormat**

The OutputFormat specifies the way in which these output key-value pairs are written to the output files. It validates the output specification for a MapReduce job. OutputFormat basically provides the RecordWriter implementation used for writing the output files of the MapReduce job. The output files are stored in a FileSystem.

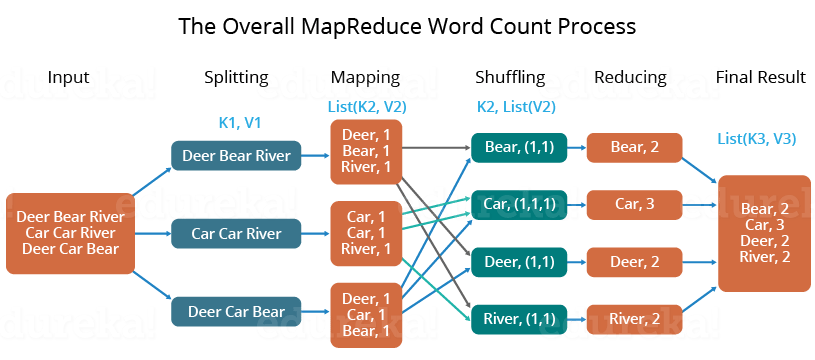


Fig 11. Word Count process Demonstration

In this manner, MapReduce works on the Hadoop cluster in different phases.

Chapter – VI

**Advantages and Disadvantages of Hadoop**

After studying almost everything about MapReduce, Now we’ll discuss the advantages of Hadoop. They are as follows

1. **Scalability**

Hadoop is a platform that is highly scalable. This is largely because of its ability to store as well as distribute large data sets across plenty of servers. These servers can be inexpensive and can operate in parallel. And with each addition of servers one adds more processing power.

Contrary to the traditional relational database management systems (RDMS) that cannot scale in order to process huge amounts of data, Hadoop MapReduce programming enables business organizations to run applications from a huge number of nodes that could involve the usage of thousands of terabytes of data.

1. **Cost-effective solution**

Hadoop’s highly scalable structure also implies that it comes across as a very cost-effective solution for businesses that need to store ever growing data dictated by today’s requirements.

In the case of traditional relational database management systems, it becomes massively cost prohibitive to scale to the degrees possible with Hadoop, just to process data. As such, many of the businesses would have to downsize data and further implement classifications based on assumptions of how certain data could be more valuable that the other. In the process, raw data would have to be deleted. This basically serves short term priorities, and if a business happens to change its plans somewhere down the line, the complete set of raw data would be unavailable for later usage.

Hadoop’s scale-out architecture with MapReduce programming, allows the storage and processing of data in a very affordable manner. It can also be used in later times. In fact, the cost savings are massive and costs can reduce from thousands and figures to hundred figures for every terabyte of data.

1. **Flexibility**

Business organizations can make use of Hadoop MapReduce programming to have access to various new sources of data and also operate on different types of data, whether they are structured or unstructured. This allows them to generate value from all of the data that can be accessed by them.

Along such lines, Hadoop offers support for numerous languages that can be used for data processing and storage. Whether the data source is social media, email, or clickstream, MapReduce can work on all of them. Also, Hadoop MapReduce programming allows for many applications, such as recommendation systems, processing of logs, marketing analysis, warehousing of data and fraud detection.

1. **Speed**

Hadoop uses a storage method known as distributed file system, which basically implements a mapping system to locate data in a cluster. The tools used for data processing, such as MapReduce programming, are also generally located in the very same servers, which allows for faster processing of data.

Even if you happen to be dealing with large volumes of data that is unstructured, Hadoop MapReduce takes minutes to process terabytes of data, and hours for petabytes of data.

1. **Security and Authentication**

Security is a vital aspect of any application. If any unlawful person or organization had access to multiple petabytes of your organization’s data, it can do you massive harm in terms of business dealings and operations.

In this regard, MapReduce works with HDFS and HBase security that allows only approved users to operate on data stored in the system.

1. **Parallel processing**

One of the primary aspects of the working of MapReduce programming is that it divides tasks in a manner that allows their execution in parallel.

Parallel processing allows multiple processors to take on these divided tasks, such that they run entire programs in less time.

1. **Availability and resilient nature**

When data is sent to an individual node in the entire network, the very same set of data is also forwarded to the other numerous nodes that make up the network. Thus, if there is any failure that affects a particular node, there are always other copies that can still be accessed whenever the need may arise. This always assures the availability of data.

One of the biggest advantages offered by Hadoop is that of its fault tolerance. Hadoop MapReduce has the ability to quickly recognize faults that occur and then apply a quick and automatic recovery solution. This makes it a game changer when it comes to big data processing.

1. **Simple model of programming**

Among the various advantages that Hadoop MapReduce offers, one of the most important ones is that it is based on a simple programming model. This basically allows programmers to develop MapReduce programs that can handle tasks with more ease and efficiency.

The programs for MapReduce can be written using Java, which is a language that isn’t very hard to pick up and is also used widespread. Thus, it is easy for people to learn and write programs that meets their data processing needs sufficiently.

Now we’ll discuss disadvantages of Hadoop.

1. **Not beginner friendly**

From installing hadoop to using MapReduce programming in it is a big task for beginners. Hadoop being open-source, there are many resources on internet and one might get confused while using those. Although from release of first version of hadoop lot has changed now, the apache hadoop documentation is well organised and neatly prepared. So we emphasis more on use it at beginning.

1. **Not Fit for Small Data**

While big data is not exclusively made for big businesses, not all big data platforms are suited for small data needs. Unfortunately, Hadoop happens to be one of them. Due to its high capacity design, the Hadoop Distributed File System, lacks the ability to efficiently support the random reading of small files. As a result, it is not recommended for organizations with small quantities of data.

1. **Potential Stability Issues**

Like all open source software, Hadoop has had its fair share of stability issues. To avoid these issues, organizations are strongly recommended to make sure they are running the latest stable version, or run it under a third-party vendor equipped to handle such problems.