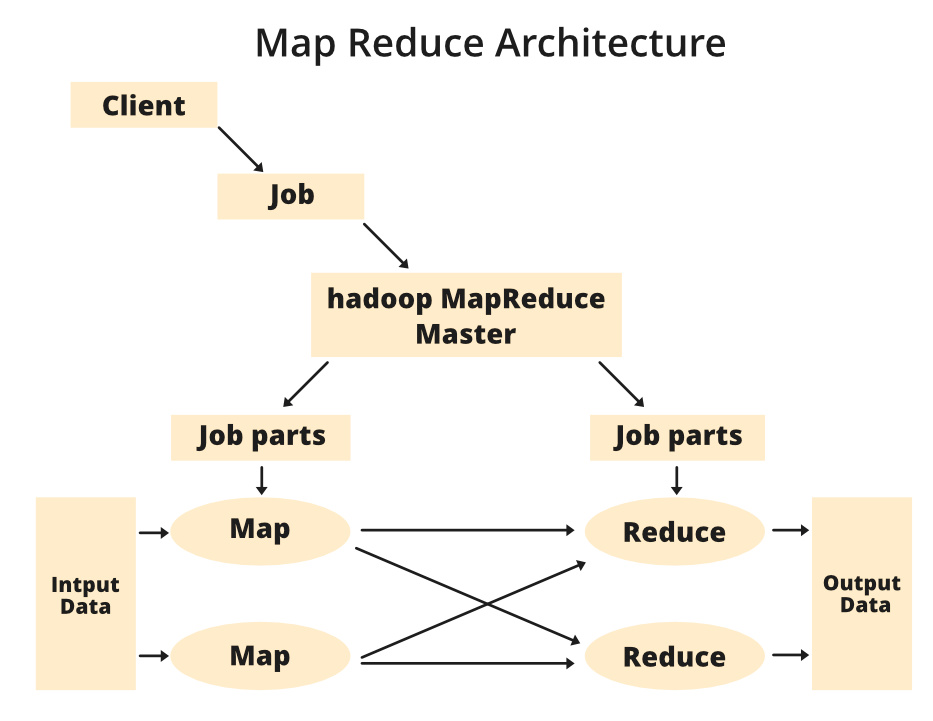
**MapReduce Architecture**

[MapReduce](https://www.geeksforgeeks.org/map-reduce-in-hadoop/) and [HDFS](https://www.geeksforgeeks.org/hadoop-hdfs-hadoop-distributed-file-system/) are the two major components of [Hadoop](https://www.geeksforgeeks.org/hadoop-introduction/) which makes it so powerful and efficient to use. MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations. The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The MapReduce task is mainly divided into two phases [Map Phase](https://www.geeksforgeeks.org/hadoop-mapper-in-mapreduce/) and [Reduce Phase](https://www.geeksforgeeks.org/hadoop-reducer-in-map-reduce/).

**MapReduce Architecture:**



**Components of MapReduce Architecture:**

1. **Client:** The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.
2. **Job:** The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.
3. **Hadoop MapReduce Master:** It divides the particular job into subsequent job-parts.
4. **Job-Parts:**  The task or sub-jobs that are obtained after dividing the main job. The result of all the job-parts combined to produce the final output.
5. **Input Data:** The data set that is fed to the MapReduce for processing.
6. **Output Data:** The final result is obtained after the processing.

In **MapReduce**, we have a client. The client will submit the job of a particular size to the Hadoop MapReduce Master. Now, the MapReduce master will divide this job into further equivalent job-parts. These job-parts are then made available for the Map and Reduce Task. This Map and Reduce task will contain the program as per the requirement of the use-case that the particular company is solving. The developer writes their logic to fulfill the requirement that the industry requires. The input data which we are using is then fed to the Map Task and the Map will generate intermediate key-value pair as its output. The output of Map i.e. these key-value pairs are then fed to the Reducer and the final output is stored on the HDFS. There can be n number of Map and Reduce tasks made available for processing the data as per the requirement. The algorithm for Map and Reduce is made with a very optimized way such that the time complexity or space complexity is minimum.

Let’s discuss the MapReduce phases to get a better understanding of its architecture:

The MapReduce task is mainly divided into **2 phases** i.e. Map phase and Reduce phase.

1. **Map:** As the name suggests its main use is to map the input data in key-value pairs. The input to the map may be a key-value pair where the key can be the id of some kind of address and value is the actual value that it keeps. The *Map()* function will be executed in its memory repository on each of these input key-value pairs and generates the intermediate key-value pair which works as input for the Reducer or *Reduce()* function.
2. **Reduce:** The intermediate key-value pairs that work as input for Reducer are shuffled and sort and send to the *Reduce()* function. Reducer aggregate or group the data based on its key-value pair as per the reducer algorithm written by the developer.

How Job tracker and the task tracker deal with MapReduce:

1. **Job Tracker:** The work of Job tracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the Task Tracker running on the same data node since there can be hundreds of data nodes available in the cluster.
2. **Task Tracker:** The Task Tracker can be considered as the actual slaves that are working on the instruction given by the Job Tracker. This Task Tracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by Job Tracker.

There is also one important component of MapReduce Architecture known as **Job History Server**. The Job History Server is a daemon process that saves and stores historical information about the task or application, like the logs which are generated during or after the job execution are stored on Job History Server.

## **Inputs and Outputs**

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a **MapReduce job** − (Input) <k1, v1> → map → <k2, v2> → reduce → <k3, v3>(Output).

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| **Map** | <k1, v1> | list (<k2, v2>) |
| **Reduce** | <k2, list(v2)> | list (<k3, v3>) |

## MapReduce Architecture in Big Data explained with Example

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

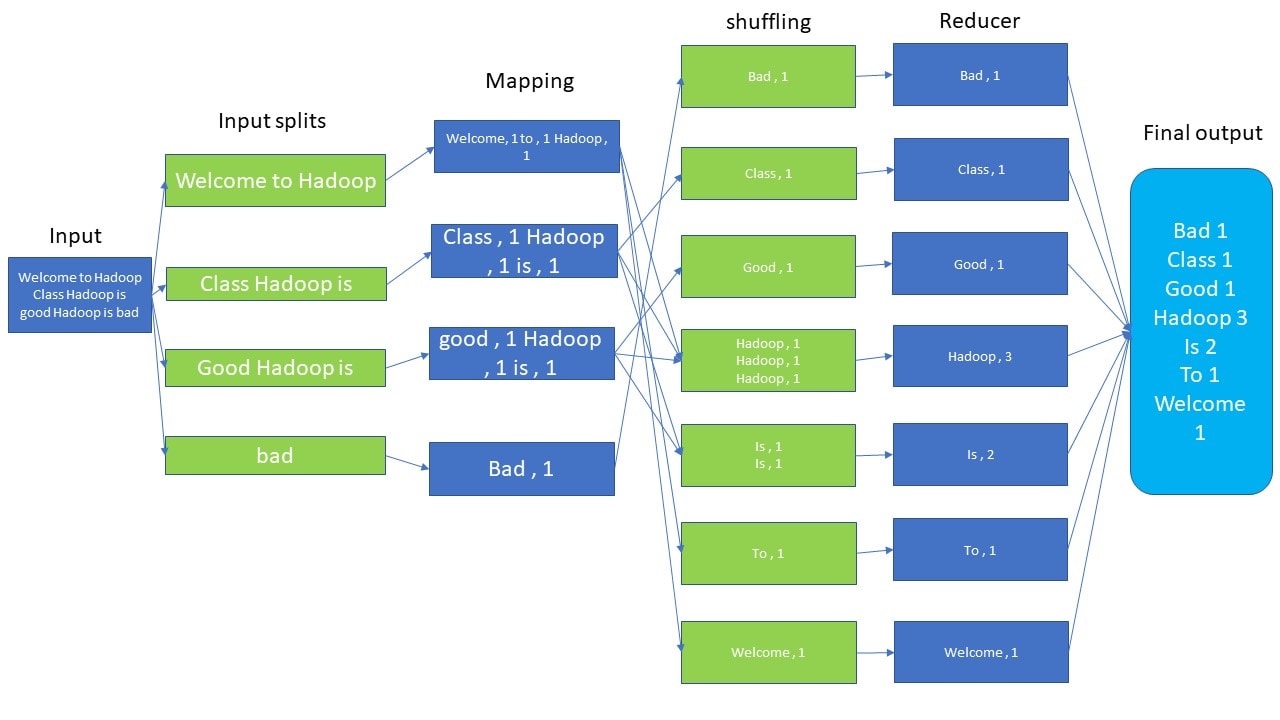
Now in this MapReduce tutorial, let’s understand with a MapReduce example–

Consider you have following input data for your MapReduce in Big data Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad



MapReduce Architecture

**The final output of the MapReduce task is**

|  |  |
| --- | --- |
| bad | 1 |
| Class | 1 |
| good | 1 |
| Hadoop | 3 |
| is | 2 |
| to | 1 |
| Welcome | 1 |

The data goes through the following phases of MapReduce in Big Data

**Input Splits:**

An input to a MapReduce in Big Data job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map

**Mapping**

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

**Shuffling**

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

**Reducing**

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

## MapReduce Architecture explained in detail

* One map task is created for each split which then executes map function for each record in the split.
* It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.
* However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
* For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
* Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
* Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
* Map output is intermediate output which is processed by reduce tasks to produce the final output.
* Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
* In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
* Reduce task doesn’t work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
* On this machine, the output is merged and then passed to the user-defined reduce function.
* Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

## How MapReduce Organizes Work?

Now in this MapReduce tutorial, we will learn how MapReduce works

[Hadoop](https://www.guru99.com/learn-hadoop-in-10-minutes.html) divides the job into tasks. There are two types of tasks:

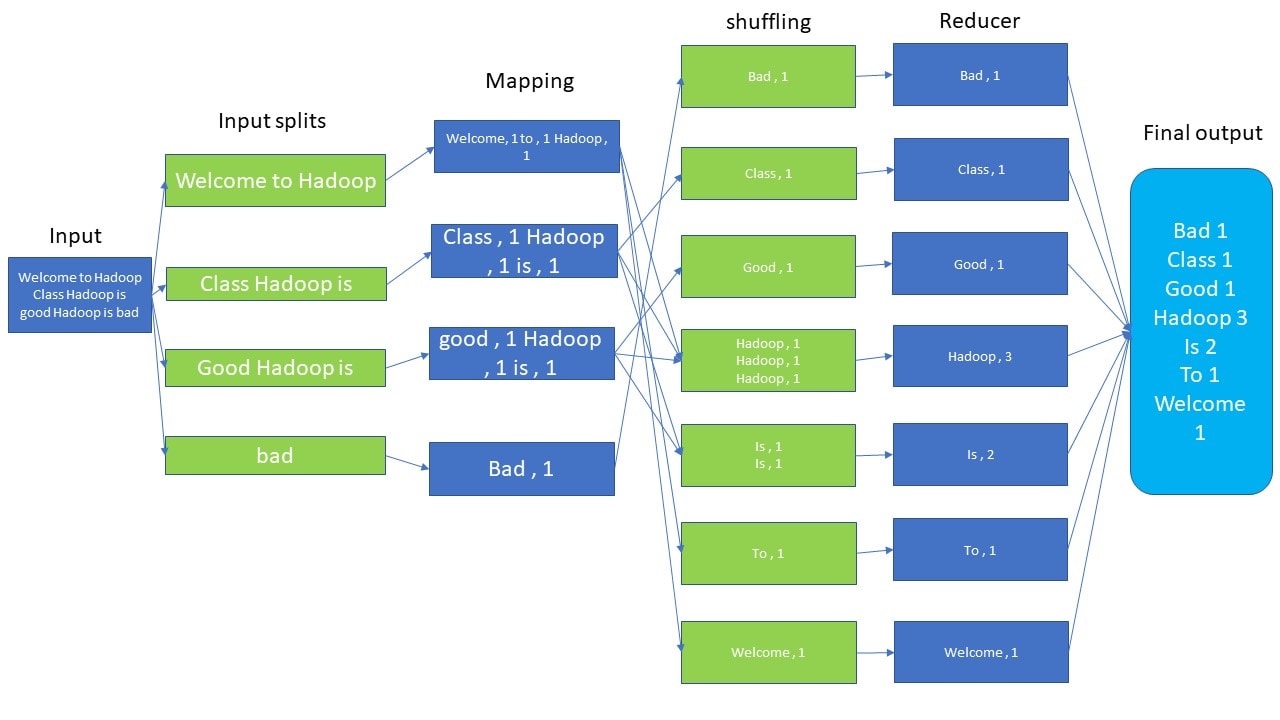
1. **Map tasks** (Splits & Mapping)
2. **Reduce tasks** (Shuffling, Reducing)

as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. **Jobtracker**: Acts like a **master** (responsible for complete execution of submitted job)
2. **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job

For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.

How Hadoop MapReduce Works

* A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
* It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
* Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
* Task tracker’s responsibility is to send the progress report to the job tracker.
* In addition, task tracker periodically sends **‘heartbeat’** signal to the Jobtracker so as to notify him of the current state of the system.
* Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.