MapReduce is a programming model suitable for processing of huge data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

**MapReduce programs work in two phases:**

1. Map phase
2. Reduce phase.

Input to each phase are **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

The whole process goes through three phase of execution namely,

**How MapReduce works**

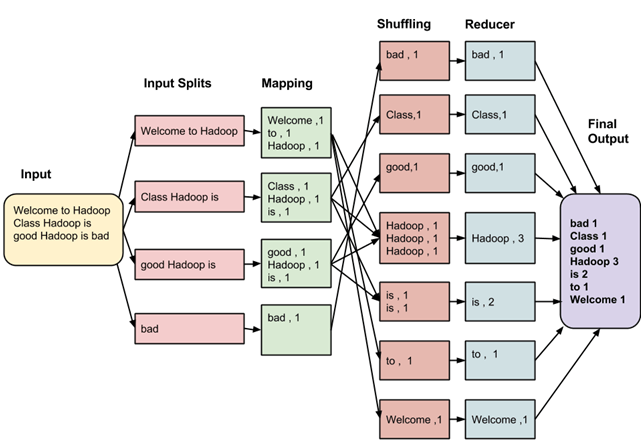
Lets understand this with an example –

Consider you have following input data for your MapReduce Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad



The data goes through following phases

**Input Splits:**

Input to a MapReduce 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 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, job of mapping phase is to count 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 output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, same words are clubed together along with their respective frequency.

**Reducing**

In this phase, output values from 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.

**The overall process 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 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 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 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 don't work on the concept of data locality. 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 to 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?**

Hadoop divides the job into tasks. There are two types of tasks:

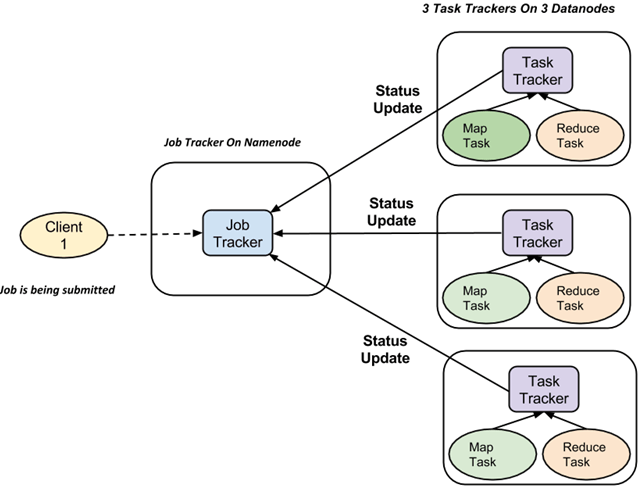
1. **Map tasks** (Spilts & Mapping)
2. **Reduce tasks** (Shuffling, Reducing)

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**.



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