

MAPREDUCE - INTRODUCTION

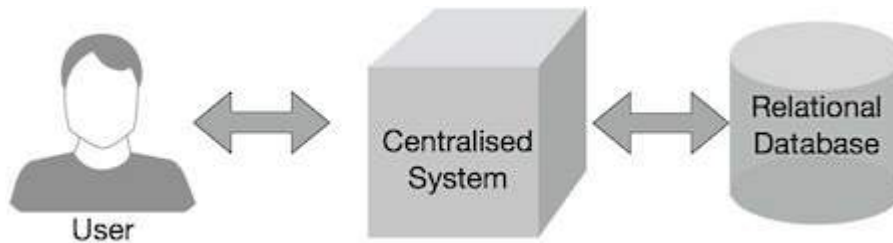
MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

What is Big Data?

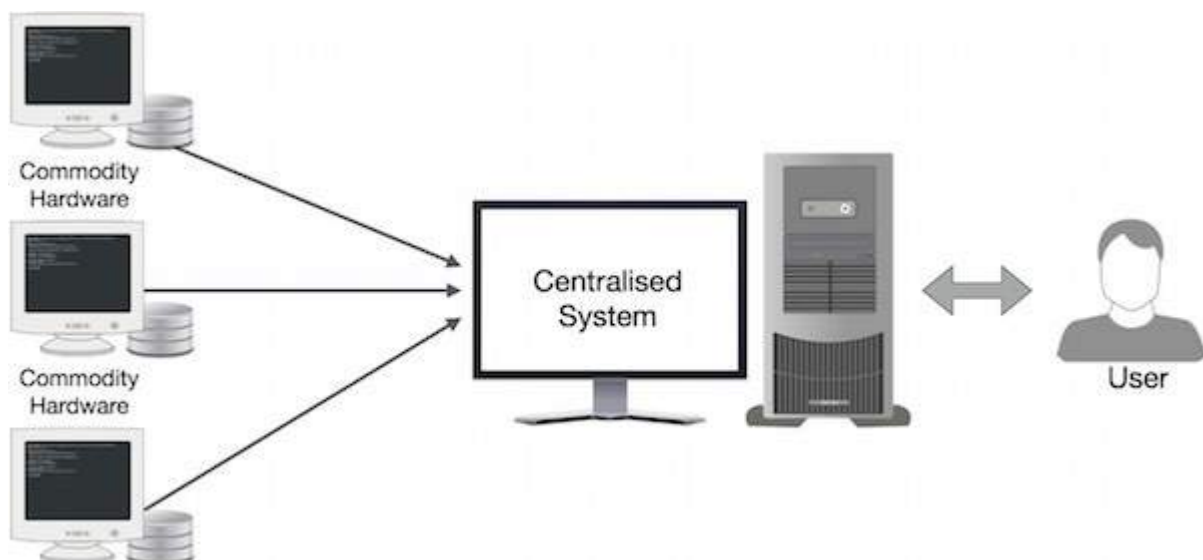
Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. For example, the volume of data Facebook or Youtube need require it to collect and manage on a daily basis, can fall under the category of Big Data. However, Big Data is not only about scale and volume, it also involves one or more of the following aspects – Velocity, Variety, Volume, and Complexity.

Why MapReduce?

Traditional Enterprise Systems normally have a centralized server to store and process data. The following illustration depicts a schematic view of a traditional enterprise system. Traditional model is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.



Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.



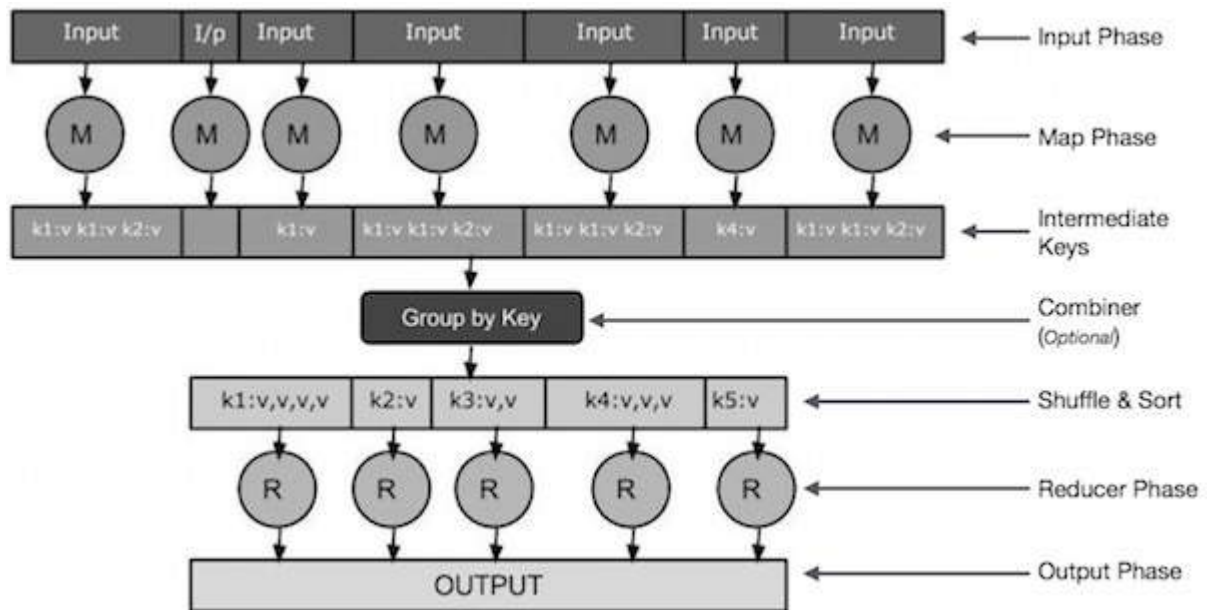
How MapReduce Works?

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples *key – valuepairs*.
- The Reduce task takes the output from the Map as an input and combines those data tuples *key – valuepairs* into a smaller set of tuples.

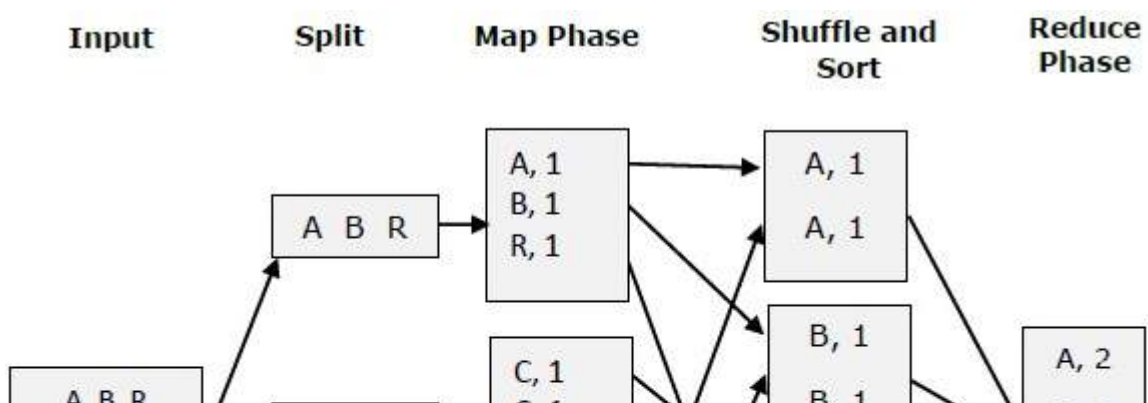
The reduce task is always performed after the map job.

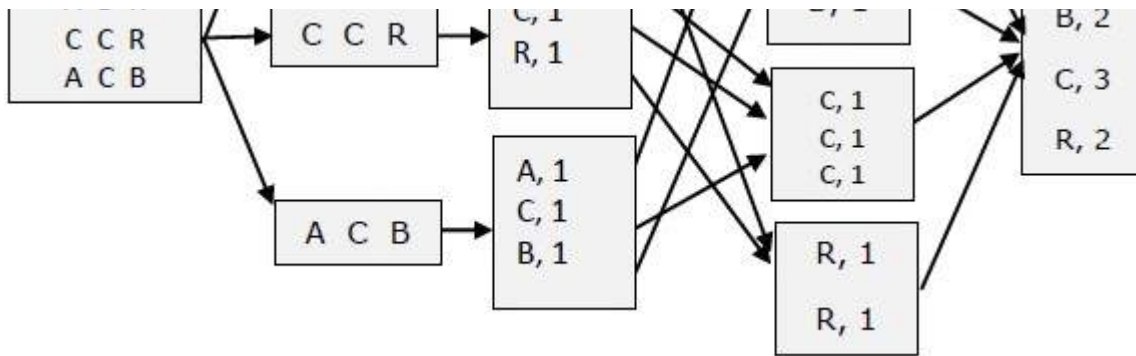
Let us now take a close look at each of the phases and try to understand their significance.



- **Input Phase** – Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map** – Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** – They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** – A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** – The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** – The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- **Output Phase** – In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

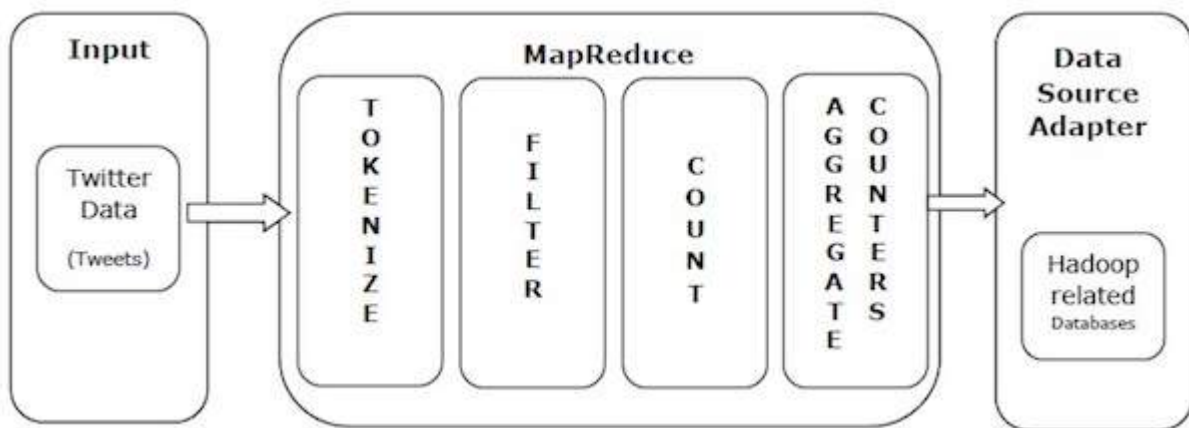
Let us try to understand the two tasks Map & Reduce with the help of a small diagram –





MapReduce-Example

Let us take a real-world example to comprehend the power of MapReduce. Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.



As shown in the illustration, the MapReduce algorithm performs the following actions –

- **Tokenize** – Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter** – Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** – Generates a token counter per word.
- **Aggregate Counters** – Prepares an aggregate of similar counter values into small manageable units.