MapReduce

parallel & distributed programming model

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

MapReduce is a programming paradigm that runs in the background of Hadoop to provide scalability and easy data-processing solutions. This tutorial explains the features of MapReduce and how it works to analyze Big Data.

Audience

This tutorial has been prepared for professionals aspiring to learn the basics of Big Data Analytics using the Hadoop Framework and become a Hadoop Developer. Software Professionals, Analytics Professionals, and ETL developers are the key beneficiaries of this course.

Prerequisites

It is expected that the readers of this tutorial have a good understanding of the basics of Core Java and that they have prior exposure to any of the Linux operating system flavors.

Copyright & Disclaimer

Table of Contents

1. MAPREDUCE – INTRODUCTION

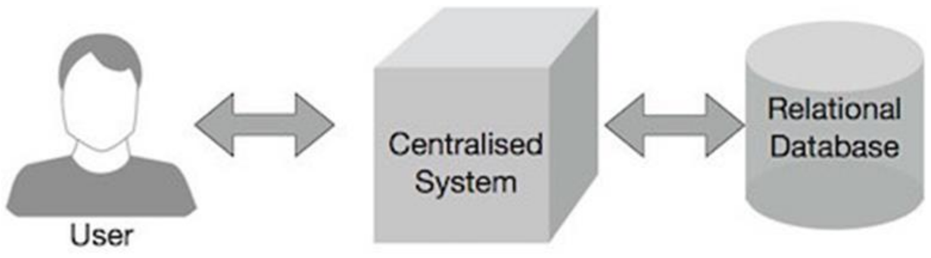
MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

What is Big Data?

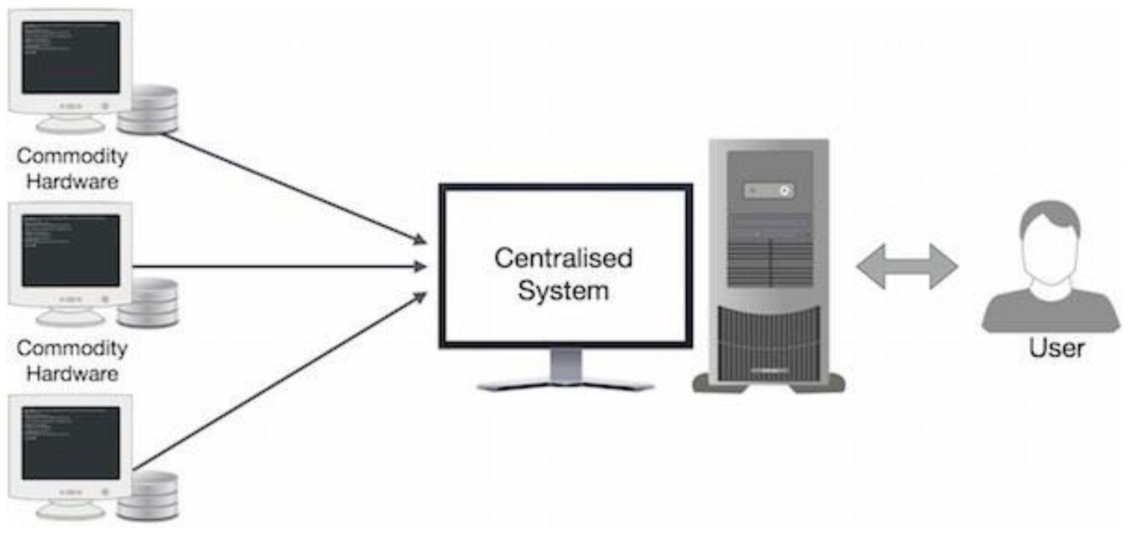
Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. For example, the volume of data Facebook or YouTube need require it to collect and manage on a daily basis, can fall under the category of Big Data. However, Big Data is not only about scale and 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?

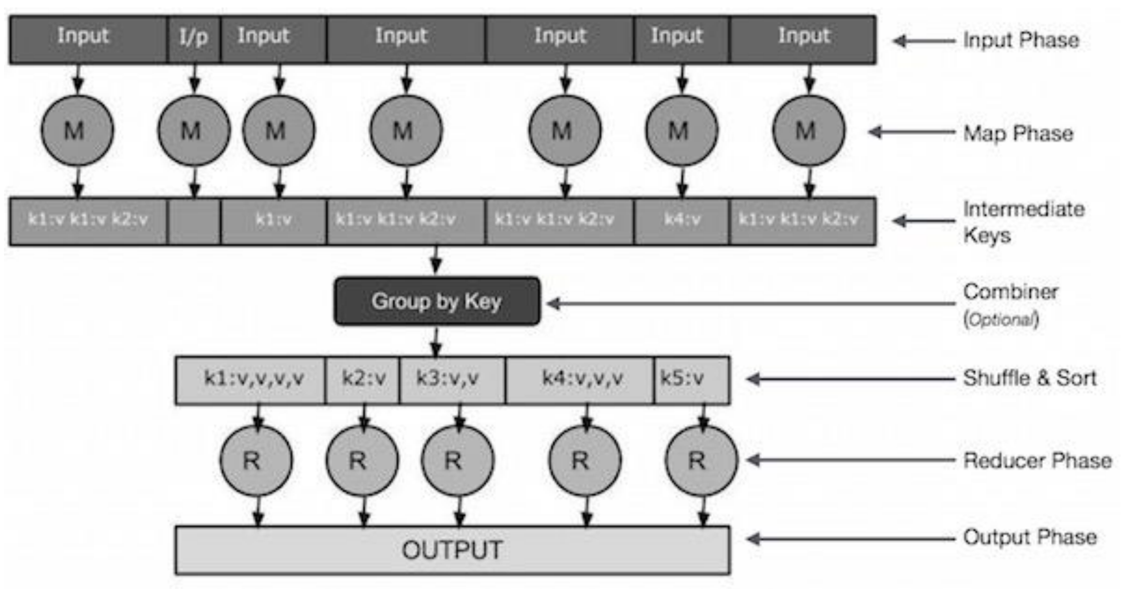
The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- 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-value pairs).

- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) 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** – The key-value pairs generated by the mapper are known as intermediate keys.

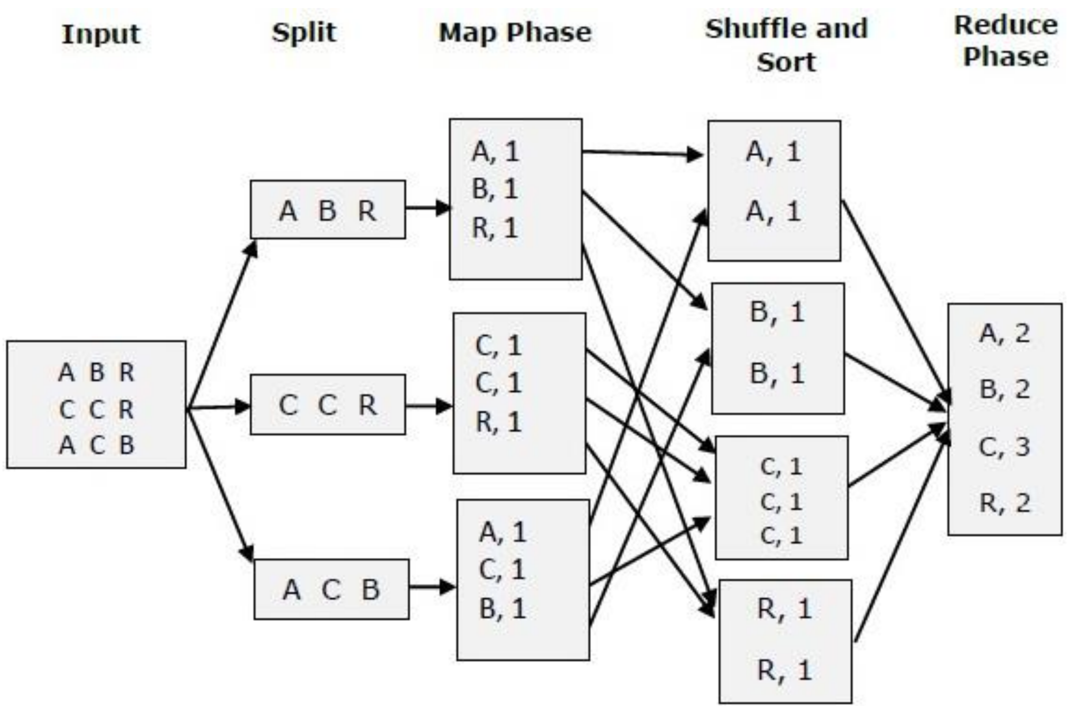
- **Combiner** – A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce 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.

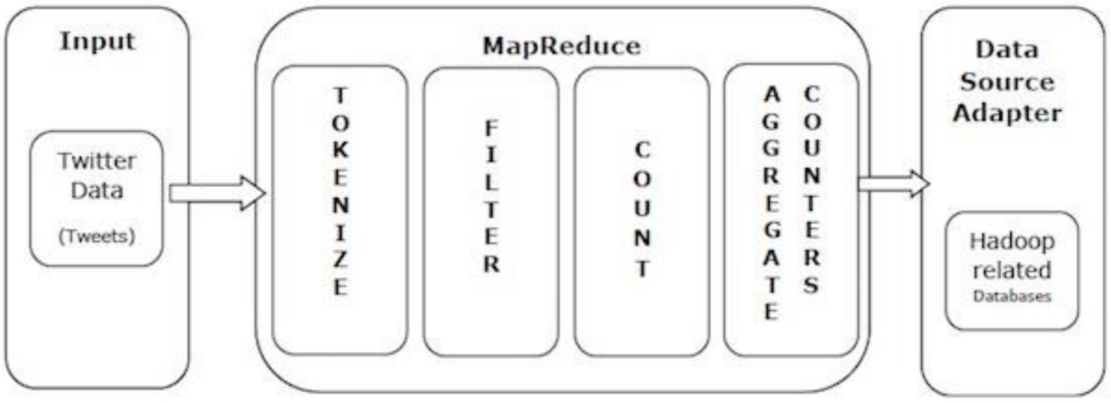
- **Output Phase** – In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

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 –

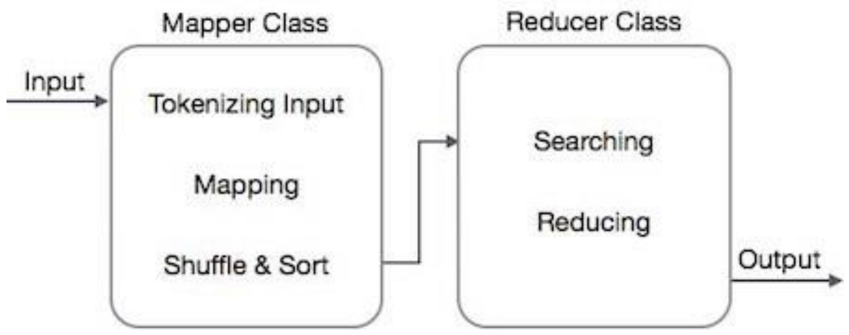
- **Tokenize** – Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter** – Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** – Generates a token counter per word.
- **Aggregate Counters** – Prepares an aggregate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

2. MAPREDUCE – ALGORITHM

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

- The map task is done by means of Mapper Class
- The reduce task is done by means of Reducer Class.

Mapper class takes the input, tokenizes it, maps, and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.



MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. In technical terms, MapReduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.

These mathematical algorithms may include the following –

- Sorting
- Searching
- Indexing
- TF-IDF

Sorting

Sorting is one of the basic MapReduce algorithms to process and analyze data. MapReduce implements sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys.

- Sorting methods are implemented in the mapper class itself.

- In the Shuffle and Sort phase, after tokenizing the values in the mapper class, the**Context** class (user-defined class) collects the matching valued keys as a collection.

- To collect similar key-value pairs (intermediate keys), the Mapper class takes the help of **RawComparator** class to sort the key-value pairs.

- The set of intermediate key-value pairs for a given Reducer is automatically sorted by Hadoop to form key-values (K2, {V2, V2…}) before they are presented to the Reducer.

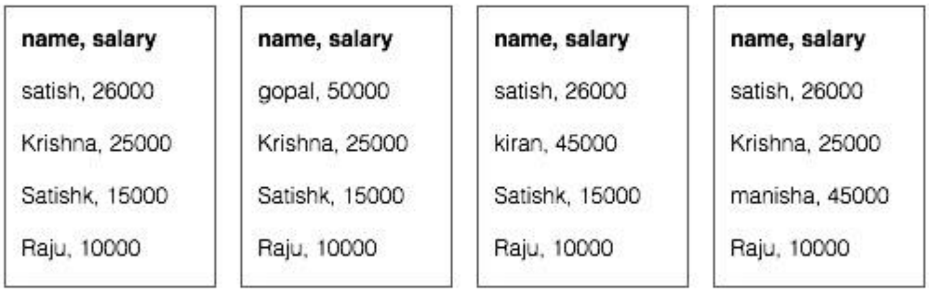
Searching

Searching plays an important role in MapReduce algorithm. It helps in the combiner phase (optional) and in the Reducer phase. Let us try to understand how Searching works with the help of an example.

Example

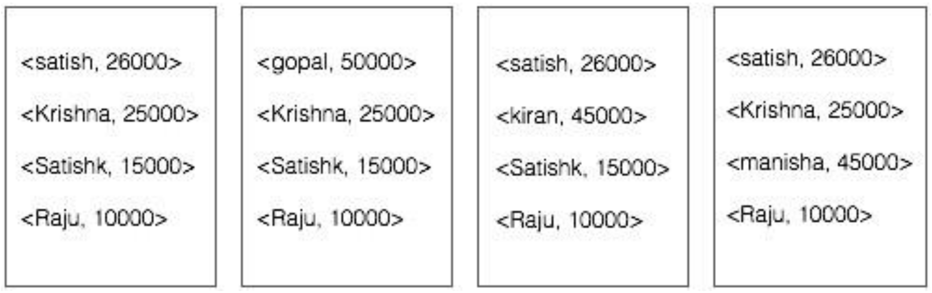
The following example shows how MapReduce employs Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset.

- Let us assume we have employee data in four different files – A, B, C, and D. Let us also assume there are duplicate employee records in all four files because of importing the employee data from all database tables repeatedly. See the following illustration.



name, salary	**name, salary**	**name, salary**	**name, salary**
satish, 26000	gopal, 50000	satish, 26000	satish, 26000
Krishna, 25000	Krishna, 25000	kiran, 45000	Krishna, 25000
Satishk, 15000	Satishk, 15000	Satishk, 15000	manisha, 45000
Raju, 10000	Raju, 10000	Raju, 10000	Raju, 10000

- **The Map phase** processes each input file and provides the employee data in key-value pairs (<k, v> : <emp name, salary>). See the following illustration.



<satish, 26000>	<gopal, 50000>	<satish, 26000>	<satish, 26000>
<Krishna, 25000>	<Krishna, 25000>	<kiran, 45000>	<Krishna, 25000>
<Satishk, 15000>	<Satishk, 15000>	<Satishk, 15000>	<manisha, 45000>
<Raju, 10000>	<Raju, 10000>	<Raju, 10000>	<Raju, 10000>

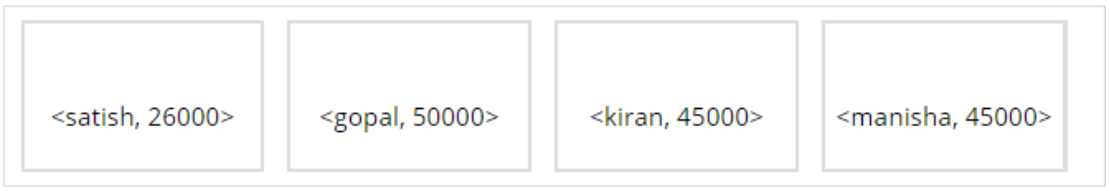
- **The combiner phase** (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary. Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file. See the following snippet.

```
<k: employee name, v: salary>
Max= the salary of an first employee. Treated as max salary

if(v(second employee).salary > Max){
   Max = v(salary);
}

else{
   Continue checking;
}
```

The expected result is as follows –



<satish, 26000>	<gopal, 50000>	<kiran, 45000>	<manisha, 45000>

- **Reducer phase** – Form each file, you will find the highest salaried employee. To avoid redundancy, check all the <k, v> pairs and eliminate duplicate entries, if any. The same algorithm is used in between the four <k, v> pairs, which are coming from four input files. The final output should be as follows –

```
<gopal, 50000>
```

Indexing

Normally indexing is used to point to a particular data and its address. It performs batch indexing on the input files for a particular Mapper.

The indexing technique that is normally used in MapReduce is known as **inverted index.**Search engines like Google and Bing use inverted indexing technique. Let us try to understand how Indexing works with the help of a simple example.

Example

The following text is the input for inverted indexing. Here T[0], T[1], and t[2] are the file names and their content are in double quotes.

```
T[0] = "it is what it is"

T[1] = "what is it"

T[2] = "it is a banana"
```

After applying the Indexing algorithm, we get the following output –

```
"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}
```

Here "a": {2} implies the term "a" appears in the T[2] file. Similarly, "is": {0, 1, 2} implies the term "is" appears in the files T[0], T[1], and T[2].

TF-IDF

TF-IDF is a text processing algorithm which is short for Term Frequency – Inverse Document Frequency. It is one of the common web analysis algorithms. Here, the term 'frequency' refers to the number of times a term appears in a document.

Term Frequency (TF)

It measures how frequently a particular term occurs in a document. It is calculated by the number of times a word appears in a document divided by the total number of words in that document.

```
TF(the) = (Number of times term the ‘the’ appears in a document) / (Total
number of terms in the document)
```

Inverse Document Frequency (IDF)

It measures the importance of a term. It is calculated by the number of documents in the text database divided by the number of documents where a specific term appears.

While computing TF, all the terms are considered equally important. That means, TF counts the term frequency for normal words like “is”, “a”, “what”, etc. Thus we need to know the frequent terms while scaling up the rare ones, by computing the following –

```
IDF(the) = log_e(Total number of documents / Number of documents with term
‘the’ in it).
```

The algorithm is explained below with the help of a small example.

Example

Consider a document containing 1000 words, wherein the word **hive** appears 50 times. The TF for **hive** is then (50 / 1000) = 0.05.

Now, assume we have 10 million documents and the word **hive** appears in 1000 of these. Then, the IDF is calculated as log (10,000,000 / 1,000) = 4.

The TF-IDF weight is the product of these quantities – 0.05 × 4 = 0.20.