

Edureka

Fundamentals of MapReduce with MapReduce Example



Shubham Sinha

Follow

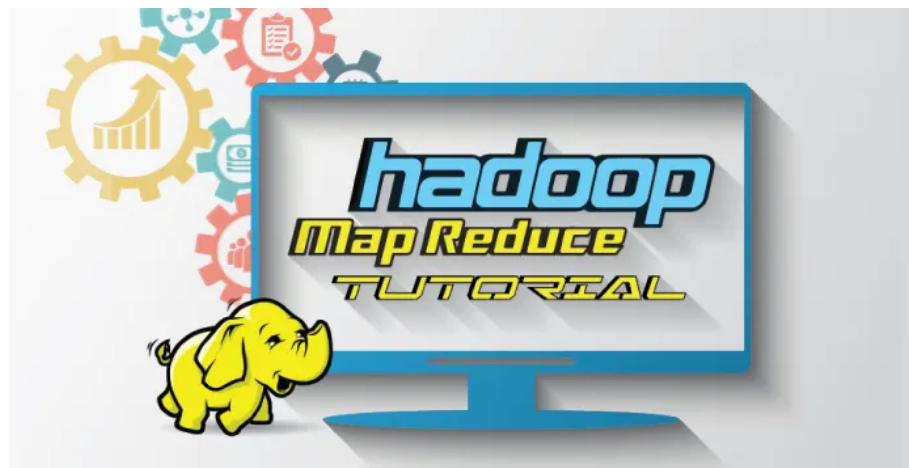
10 min read · Nov 15, 2016

387

3



...



MapReduce Tutorial - Edureka

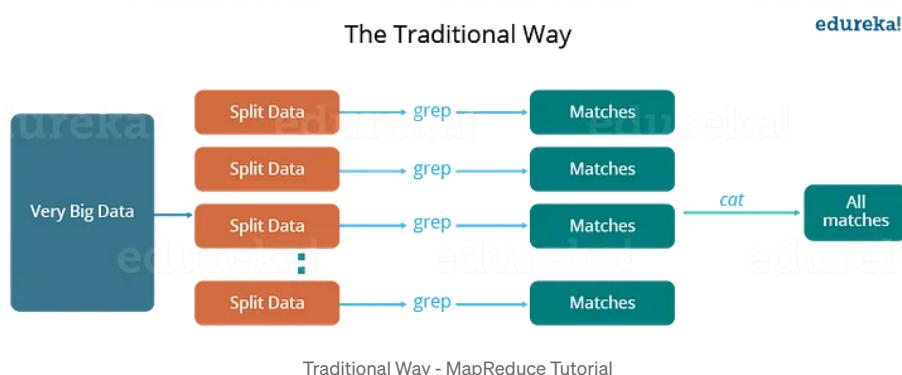
In this MapReduce Tutorial blog, I am going to introduce you to MapReduce, which is one of the core building blocks of processing in the *Hadoop framework*. Before moving ahead, I would suggest you get familiar with HDFS concepts which I have covered in my previous *HDFS tutorial* blog. This will help you to understand the MapReduce concepts quickly and easily.

Google released a paper on MapReduce technology in December 2004. This became the genesis of the Hadoop Processing Model. So, MapReduce is a programming model that allows us to perform parallel and distributed processing on huge datasets. The topics that I have covered in this MapReduce tutorial blog are as follows:

- Traditional Way for parallel and distributed processing
- What is MapReduce?
- MapReduce Example
- MapReduce Advantages
- MapReduce Program

- MapReduce Program Explained

Traditional Way



Let us understand, when the MapReduce framework was not there, how parallel and distributed processing used to happen in a traditional way. So, let us take an example where I have a weather log containing the daily average temperature of the years from 2000 to 2015. Here, I want to calculate the day having the highest temperature each year.

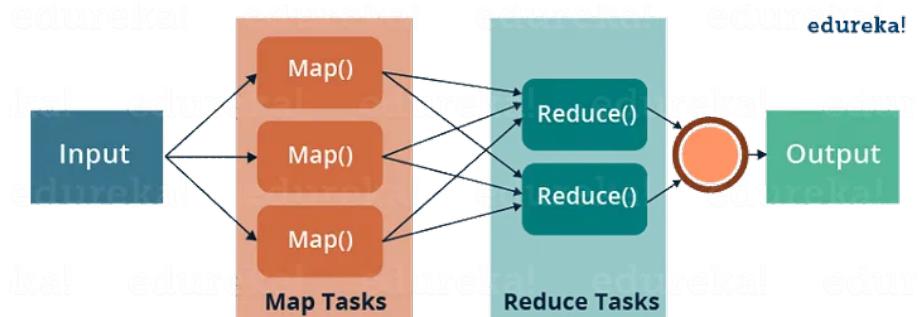
So, just like in the traditional way, I will split the data into smaller parts or blocks and store them in different machines. Then, I will find the highest temperature in each part stored in the corresponding machine. At last, I will combine the results received from each of the machines to have the final output. Let us look at the challenges associated with this traditional approach:

1. **Critical path problem:** It is the amount of time taken to finish the job without delaying the next milestone or actual completion date. So, if, any of the machines delay the job, the whole work gets delayed.
2. **Reliability problem:** What if, any of the machines which are working with a part of data fails? The management of this failover becomes a challenge.
3. **Equal split issue:** How will I divide the data into smaller chunks so that each machine gets even part of data to work with. In other words, how to equally divide the data such that no individual machine is overloaded or underutilized.
4. **Single split may fail:** If any of the machines fail to provide the output, I will not be able to calculate the result. So, there should be a mechanism to ensure this fault tolerance capability of the system.
5. **Aggregation of the result:** There should be a mechanism to aggregate the result generated by each of the machines to produce the final output.

These are the issues which I will have to take care individually while performing parallel processing of huge datasets when using traditional approaches.

To overcome these issues, we have the MapReduce framework which allows us to perform such parallel computations without bothering about the issues like reliability, fault tolerance etc. Therefore, MapReduce gives you the flexibility to write code logic without caring about the design issues of the system.

What is MapReduce?



What is MapReduce - MapReduce Tutorial

MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.

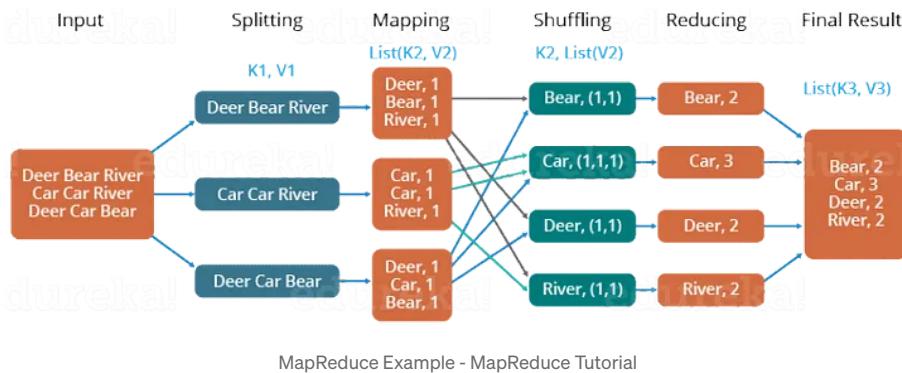
- MapReduce consists of two distinct tasks — Map and Reduce.
- As the name MapReduce suggests, reducer phase takes place after the mapper phase has been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.
- Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

A Word Count Example of MapReduce

Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding unique words and the number of occurrences of those unique words.



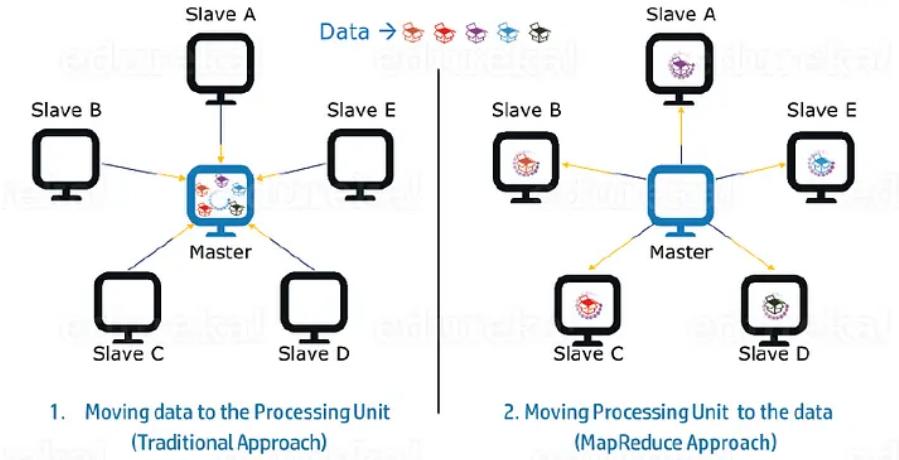
- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Advantages of MapReduce

The two biggest advantages of MapReduce are:

1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount as shown in the figure below (2).



Traditional Way Vs. MapReduce Way - MapReduce Tutorial

Top highlight

2. Data Locality:

Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- Master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

- It is very cost effective to move the processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

MapReduce Example Program

Before jumping into the details, let us have a glance at a MapReduce example program to have a basic idea about how things work in a MapReduce environment practically. I have taken the same word count example where I have to find out the number of occurrences of each word. And Don't worry guys, if you don't understand the code when you look at it for the first time, just bear with me while I walk you through each part of the MapReduce

code.

Source code:

```
package co.edureka.mapreduce;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;

public class WordCount
{
    public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {
        public void map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                value.set(tokenizer.nextToken());
                context.write(value, new IntWritable(1));
            }
        }
    }

    public static class Reduce extends
Reducer<Text,IntWritable,Text,IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values,Context
context) throws IOException,InterruptedException {
            int sum=0;
            for(IntWritable x: values)
            {
                sum+=x.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf= new Configuration();
        Job job = new Job(conf,"My Word Count Program");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        Path outputPath = new Path(args[1]);
        //Configuring the input/output path from the filesystem into the job
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //deleting the output path automatically from hdfs so that we don't
        have to delete it explicitly
        outputPath.getFileSystem(conf).delete(outputPath);
        //exiting the job only if the flag value becomes false
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Explanation of MapReduce Program

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code:

```
public static class Map extends
Mapper<LongWritable,Text,Text,IntWritable> {

public void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException {

String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
}
```

- We have created a class `Map` that extends the class `Mapper` which is already defined in the MapReduce Framework.
- We define the data types of input and output key/value pair after the class declaration using angle brackets.



- Both the input and output of the Mapper is a key/value pair.
- Input:
 1. The *key* is nothing but the offset of each line in the text file: *LongWritable*
 2. The *value* is each individual line (as shown in the figure at the right): *Text*
- Output:
 1. The *key* is the tokenized words: *Text*
 2. We have the hardcoded *value* in our case which is 1: *IntWritable*
 3. Example — Dear 1, Bear 1, etc.

- We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

Reducer Code:

```
public static class Reduce extends
Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context
    context)
    throws IOException, InterruptedException {
        int sum=0;
        for(IntWritable x: values)
        {
            sum+=x.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

- We have created a class Reduce which extends class Reducer like that of Mapper.
- We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a key-value pair.
- Input:
 1. The *key* nothing but those unique words which have been generated after the sorting and shuffling phase: *Text*
 2. The *value* is a list of integers corresponding to each key: *IntWritable*
 3. Example — Bear, [1, 1], etc.
- Output:
 1. The *key* is all the unique words present in the input text file: *Text*
 2. The *value* is the number of occurrences of each of the unique words: *IntWritable*
 3. Example — Bear, 2; Car, 3, etc.
- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:

```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
```

```

job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);

//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job, the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method `setInputFormatClass()` is used for specifying that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen `TextInputFormat` so that single line is read by the mapper at a time from the input text file.
- The `main()` method is the entry point for the driver. In this method, we instantiate a new `Configuration` object for the job.

Run the MapReduce code:

The command for running a MapReduce code is:

```

hadoop jar hadoop-mapreduce-example.jar WordCount /sample/input
/samples/output

```

Now, you guys have a basic understanding of the MapReduce framework. You would have realized how the MapReduce framework facilitates us to write code to process huge data present in the HDFS.

If you wish to check out more articles on the market's most trending technologies like Artificial Intelligence, Python, Ethical Hacking, then you can refer to [Edureka's official site](#).

Do look out for other articles in this series which will explain the various other aspects of Big data.

1. [Hadoop Tutorial](#)

2. [Hive Tutorial](#)

3. [Pig Tutorial](#)

4. [Big Data Tutorial](#)

5. [HBase Tutorial](#)

6. [HDFS Tutorial](#)

7. [Hadoop 3](#)

8. [Sqoop tutorial](#)

9. [Flume tutorial](#)

10. [Oozie tutorial](#)

11. [Hadoop ecosystem](#)

12. [Top Hive Commands with Examples in HQL](#)

13. [Hadoop Cluster With Amazon EMR?](#)

14. [Big Data Engineer Resume](#)

15. [Hadoop Developer- Job Trends and Salary](#)

16. [Hadoop Interview Questions](#)

• • •

Originally published at www.edureka.co on November 15, 2016.

Big Data

Mapreduce

Hadoop

Hdfs

Hadoop Training



Published in Edureka

5.3K followers · Last published Jul 30, 2022

Follow

There are many e-learning platforms on the internet & then there's us. We are not the biggest, but we are the fastest growing. We have the highest course completion rate in the industry. We provide live, instructor-led online programs in trending tech with 24x7 lifetime support.



Written by Shubham Sinha

184 followers · 1 following

Follow

Big Data enthusiast, loves solving real-world Big Data problems.

Responses (3)



 Ishu

What are your thoughts?



Nithin Warrier
May 6, 2021

...

I watched many videos but didn't quite get the concept, you explained pretty well with examples, superb, keep posting!!

 1 [Reply](#)



fadhil alnassar
Jun 30, 2020

...

. Suppose we have a text file named initial.txt with the following content:

Book, River, Dear,
Bear, Car, River,
Deer, Car, Bear, Car

a. Similar to what you see in the course slides, show the MapReduce steps to count the number of times each word... [more](#)

 1  1 reply [Reply](#)



A Andrew Hancock
Oct 7, 2023

...

What Nithin said. There's lots online about MapReduce that descends into a morass of implementation terminology that isn't very meaningful. This is the one page I've found that describes the concept. Thank you, Shubham.

 [Reply](#)

More from Shubham Sinha and Edureka



In Edureka by Shubham Sinha



In Edureka by Vishal Padghan

Medium

Search

Write



This blog on HDFS Tutorial will cover what is HDFS, its advantages and features.

Oct 21, 2016 12



...

This blog on AWS Fargate will cover the need and working of AWS Fargate with a hands-on.

Nov 27, 2018 55



...



In Edureka by Swatee Chand

Dependency Injection Using Spring Boot

This article on Dependency Injection is a comprehensive guide to the technique with ...

Jul 2, 2019 414 6



...

In Edureka by Shubham Sinha

Earthquake Detection System Using Apache Spark

This article on Apache Spark Tutorial will give you an introduction to Apache Spark with a...

May 4, 2017 5



...

[See all from Shubham Sinha](#)

[See all from Edureka](#)

Recommended from Medium

 In Data Engineer Things by Vu Trinh

What Makes OLTP Databases So Quick at Finding Your Data?

Things you need to know about the most used data structure for optimizing query...

 4d ago  93  2

 Puja Pandey

How SQL Views and Window Functions Helped Me in Interviews

In one of my interviews, I was asked a SQL question:

 Sep 14

 Ben Meehan

Understanding Change Data Capture (CDC): The Backbone of...

Modern software systems generate and consume massive volumes of data at every...

 6d ago  7

 Prem Vishnoi(cloudvala)

When Integrity Meets Hierarchy: The Ethical Dilemma Every Data...

Not thinking respect for authority is the greatest enemy of truth— Albert Einstein

 3d ago

 In Javarevisited by Vikas Singh

High Level System Design—TinyURL or bitly etc

TinyURL provides services such as shortening, analytics, management of URL.

 Jun 9  2

 Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.

 Sep 23  7.1K  251

See more recommendations