MapReduce

It is a software framework that enables you to write applications that process vast amounts of data, in-parallel on large clusters of commodity hardware in a reliable and fault-tolerant manner.

- A MapReduce job usually splits the input data set into independent chunks, which are processed by the map tasks in a completely parallel manner.
- The framework sorts the outputs of the maps, which are then inputted to the reduce tasks.
- Typically, both the input and the output of the job are stored in a file system.

MapReduce phases:

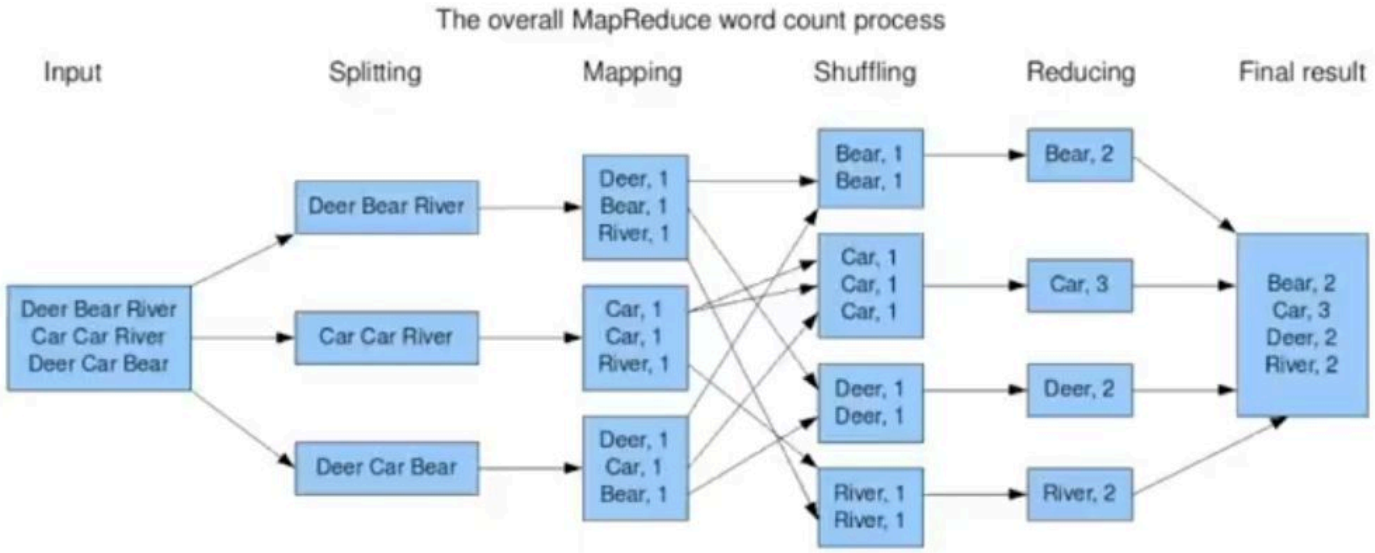
- Mapping
- Shuffle and Sort
- Reducing

MapReduce

Characteristics:

- **Distributed:** The MapReduce is a distributed framework consisting of clusters of commodity hardware which run map or reduce tasks.
- **Parallel:** The map and reduce tasks always work in parallel.
- **Fault tolerant:** If any task fails, it is rescheduled on a different node.
- **Scalable:** It can scale arbitrarily. As the problem becomes bigger, more machines can be added to solve the problem in a reasonable amount of time; the framework can scale horizontally rather than vertically.

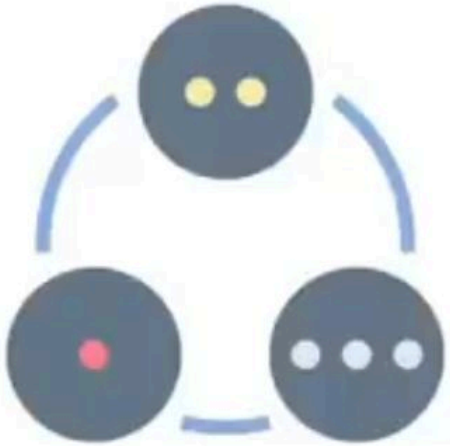
How Map Reduce works



Developing a Map Reduce application

Phases

- Configuration API
- Configuration the development environment
- GenericOptionsParser, Tool and ToolRunnerr
- Writing unit tests
- Running locally and in a cluster on test data
- The MapReduce web UI
- Hadoop logs
- Tuning a job to improve performance



Developing a Map Reduce application

Stage: 1

- Writing a program in MapReduce follows a certain pattern.
- You start by writing your map and reduce function, ideally with unit tests to make sure they do what you expect.
- Then you write a driver program to run a job, which can run from your IDE using a small subset of the data to check that it is working.
- If it fails, you can use your IDE's debugger to find the source of the problem.
- With this information, you can expand your unit tests to cover this case and improve your mapper or reducer as appropriate to handle such input correctly.

Developing a Map Reduce application

Stage: 2

- When the program runs as expected against the small dataset, you are ready to unleash it on a cluster.
- Running against the full dataset is likely to expose some more issue, which you can fix as before, by expanding your tests and mapper to handle the new cases.
- Debugging failing programs in the cluster is a challenge, so we look at some common techniques to make it easier.

Developing a Map Reduce application

Stage: 3

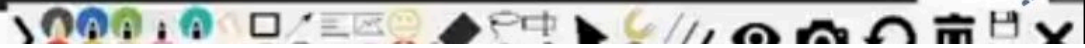
- After the program is working, you may wish to done some tuning, first by running through some standard checks for making MapReduce programs faster and then by doing task profiling.
- Profiling distributed programs is not easy, but Hadoop has hooks to aid the process.

Mapper

It maps input key/value pairs to a set of intermediate key/value pairs.

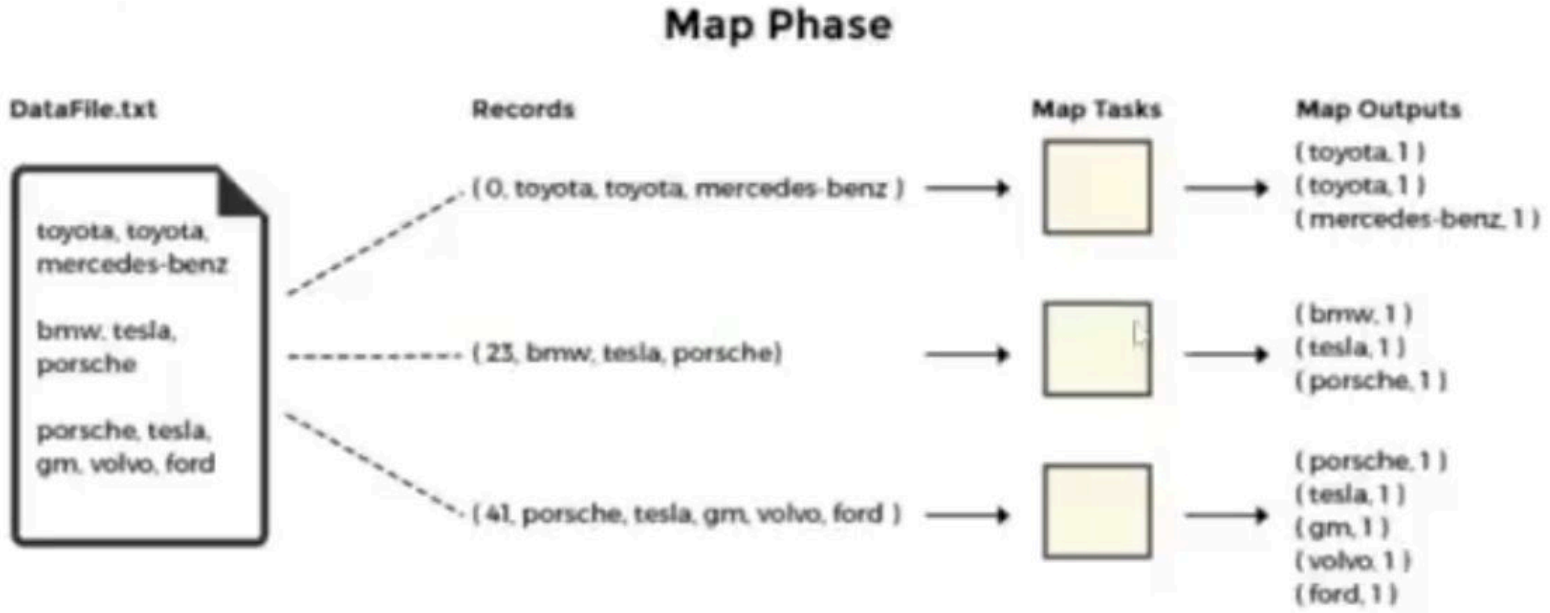
Toyota Toyota Mercedes
BMW Tesla Porsche
Porsche Tesla GM Volvo ford

```java
public class CarMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        // We can ignore the key and only work with value
        String[] words = value.toString().split(" ");

        for (String word : words) {
            context.write(new Text(word.toLowerCase()), new IntWritable(1));
        }
    }
}
```



Mapper

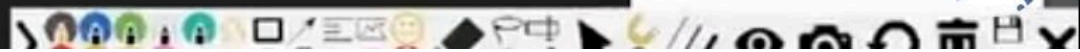
Note that sorting of data happens on the map side, and not on the reduce side.



Unit tests with MR unit

The Java library MRUnit can be used to unit test MapReduce jobs. MRUnit has been retired; frameworks like Mockito are used instead to unit test MapReduce jobs. Nevertheless, we use MRUnit to get a feel for unit testing MapReduce. The unit test to exercise our CarMapper is shown below:

```
@Test
public void testMapper() throws IOException {

    MapDriver<LongWritable, Text, Text, IntWritable> driver =
            MapDriver.<LongWritable, Text, Text, IntWritable>newMapDriver()
                    .withMapper(new CarMapper())
                    .withInput(new LongWritable(0), new Text("BMW BMW Toyota"))
                    .withInput(new LongWritable(0), new Text("Rolls-Royce Honda Honda"))
                    .withOutput(new Text("bmw"), new IntWritable(1))
                    .withOutput(new Text("bmw"), new IntWritable(1))
                    .withOutput(new Text("toyota"), new IntWritable(1))
                    .withOutput(new Text("rolls-royce"), new IntWritable(1))
                    .withOutput(new Text("honda"), new IntWritable(2));

    driver.runTest();
}
```



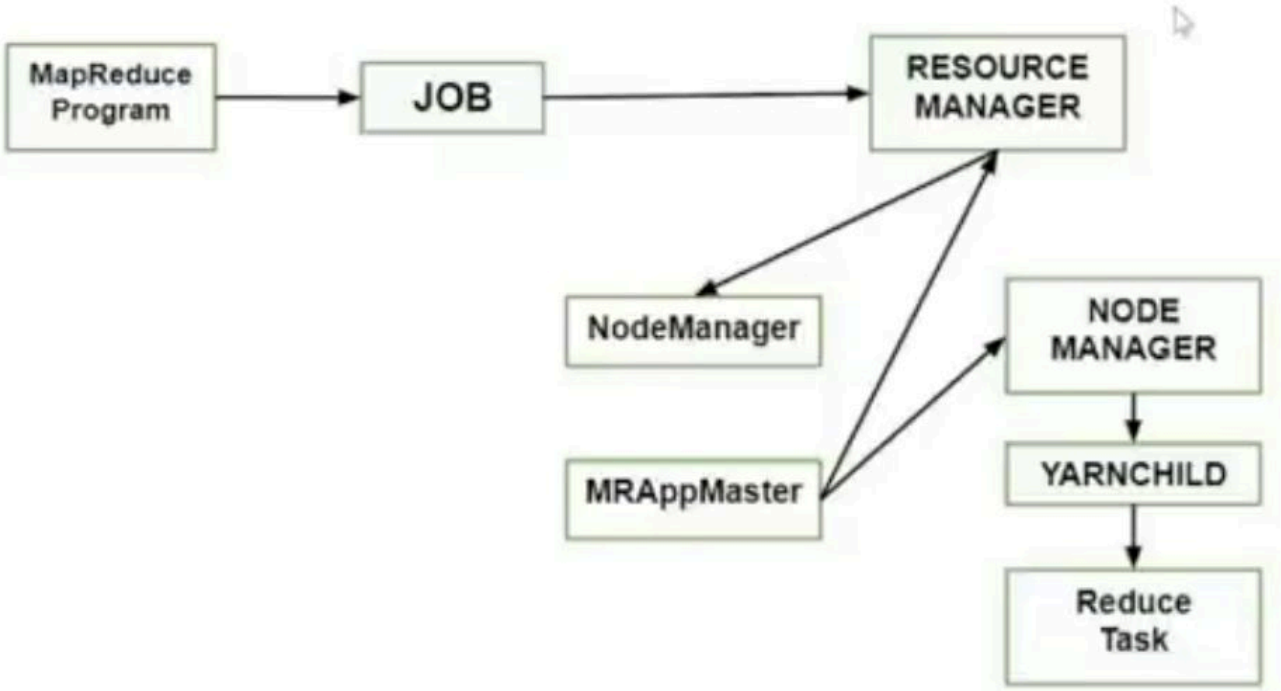
Anatomy of a Map Reduce job run

MapReduce can be used to work with a solitary method call: submit() on a Job object

Components:

- **Client :** Submitting the MapReduce job.
- **Yarn node manager :** In a cluster , it monitors and launches the compute containers on machines.
- **Yarn resource manager :** Handles the allocation of compute resources coordination on the cluster.
- **MapReduce application master :** Facilitates the tasks running the MapReduce work.
- **Distributed Filesystem :** Shares job files with other entities.

Anatomy of a Map Reduce job run



Anatomy of a Map Reduce job run

How to submit Job?

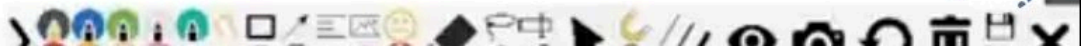
To ceate an internal JobSubmitter instance, use the submit() which further calls submitJobInternal() on it.

Processes implemented by JobSubmitter for submitting the Job :

- The resource manager askes for a new application ID that is used for MapReduce Job ID.
- Output specification of the job is checked.
- If the splits cannot be computed, it computes the input splits for the job. This can be due to the job is not submitted and an error is thrown to the MapReduce program.
- Resources needed to run the job is copied – it includes the job JAR file, the computed input splits, to the shared file system in a directory named after the job ID.
- It copies job JAR with a high replication factor, which is controlled by mapreduce.client.submit.file.replication property. AS there are the number of copies across the cluster for the node managers to access.
- By calling submitApplication(), submits the job on the resource manager.

Job scheduling

- Hadoop MapReduce is a software framework for writing applications that process huge amounts of data (terabytes to petabytes) in-parallel on the large Hadoop cluster. This framework is responsible for scheduling tasks, monitoring them, and re-executes the failed task.
- In Hadoop 2, a YARN called Yet Another Resource Negotiator was introduced. The basic idea behind the YARN introduction is to split the functionalities of resource management and job scheduling.
- The ResourceManager has two main components that are Schedulers and ApplicationsManager.
- Schedulers in YARN ResourceManager is a pure scheduler which is responsible for allocating resources to the various running applications.
- The FIFO Scheduler, CapacityScheduler, and FairScheduler are such pluggable policies that are responsible for allocating resources to the applications.



Job Scheduling

There are mainly 3 types of schedulers in Hadoop:

1. FIFO scheduler
2. Capacity scheduler
3. Fair scheduler

A Job Queue is a collection of various tasks that we have received from our various clients. The tasks are available in the queue and we need to schedule this task on the basis of our requirements.

Job Queue

Task1	Task2	Task3	Task4

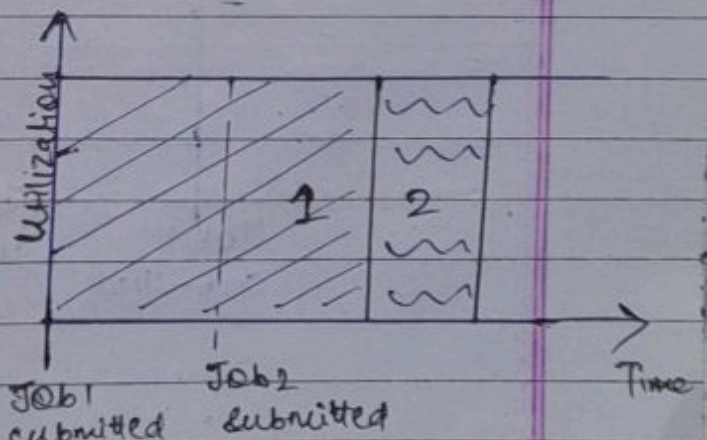
FIFO Scheduler

It stands for first In First Out so, tasks/application that comes first will be served first.

This is the default scheduler for Hadoop.

The tasks are placed in a queue and are performed in their submission order. Once the job is scheduled, no intervention is allowed.

Advantages
- No need for configuration
- Simple to execute
- First come first serve

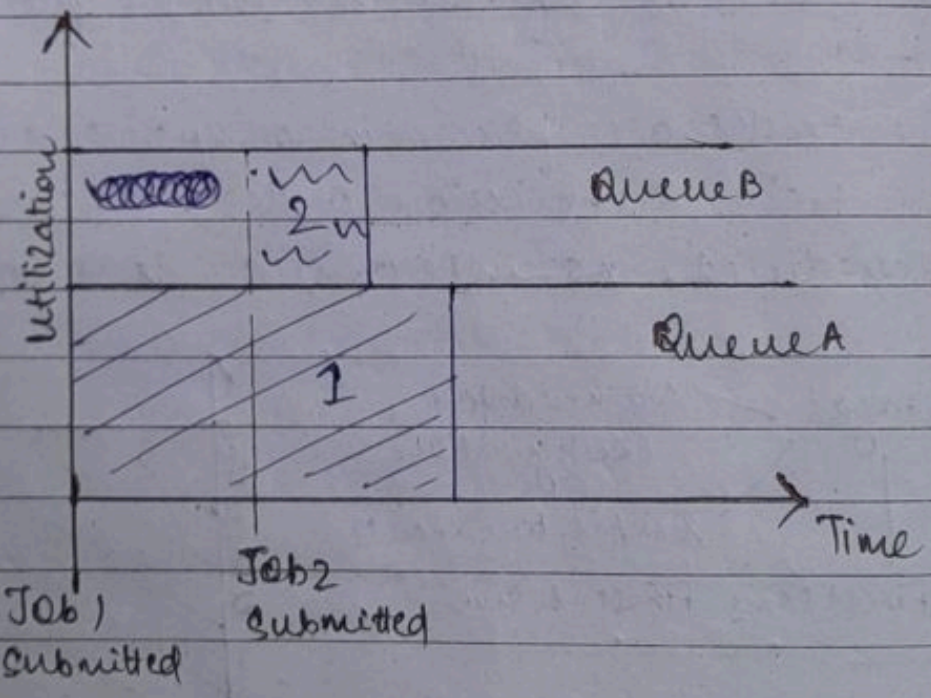


Capacity Scheduler

- Here we have multiple job queues scheduling over racks. It allows multiple occupants to share a large size Hadoop cluster.
- It contains 3 types of queue that are root, parent and leaf which are used to represent cluster, organization, application submission respectively.

Advantages:- 1. Maximizes throughput in Hadoop cluster

2. Best for working with multiple clients or priority jobs in a Hadoop cluster.

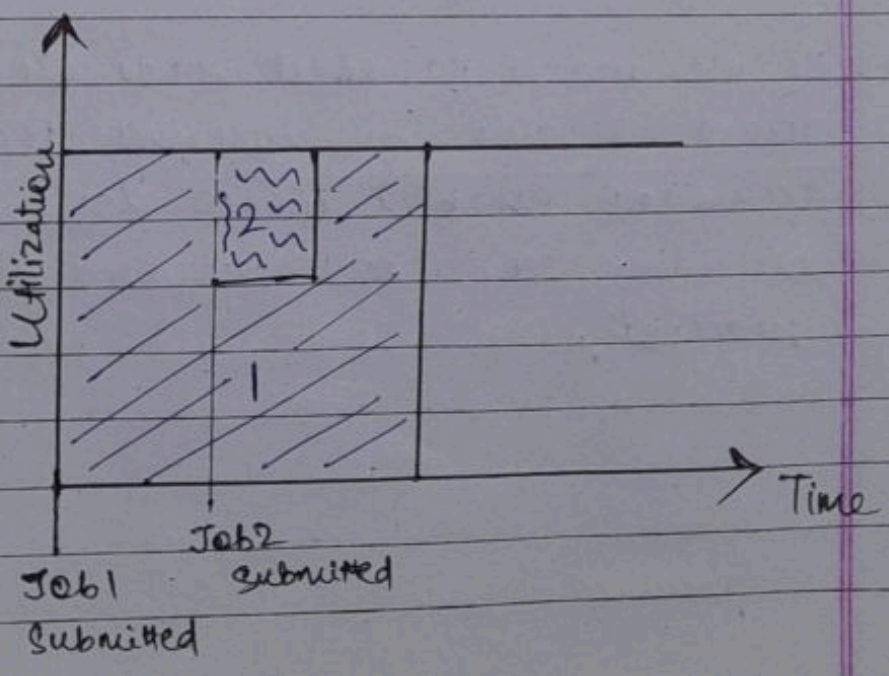


Fair Scheduler With the help of these scheduler, YARN applications can share the resources in large Hadoop cluster and these resources are maintained dynamically hence there is no need for prior capacity.

The resources are distributed in such a manner that all applications within a cluster get an equal amount of time.

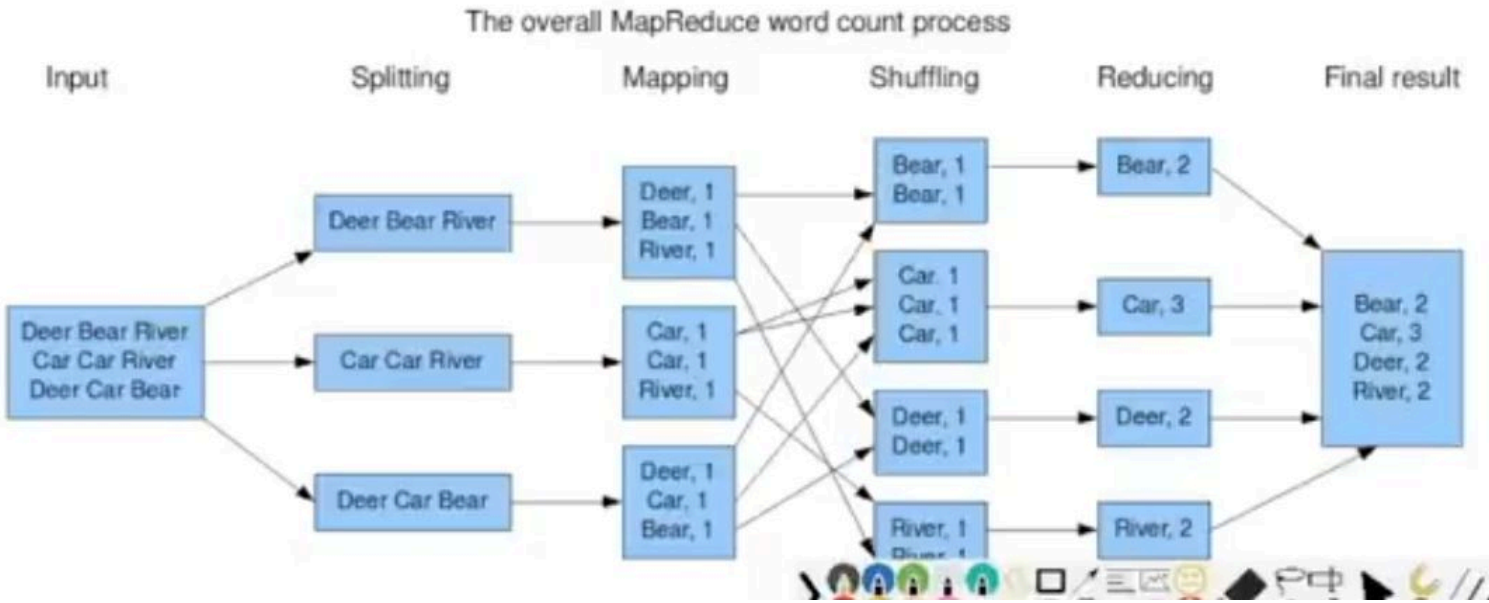
In Fair scheduler whenever any high priority job arises in the same queue, the task is processed in parallel by replacing some portion from the already dedicated slots.

Advantages :- Resources assigned to each application depends upon its priority.
It can limit the concurrent running task in a particular pool / queue.



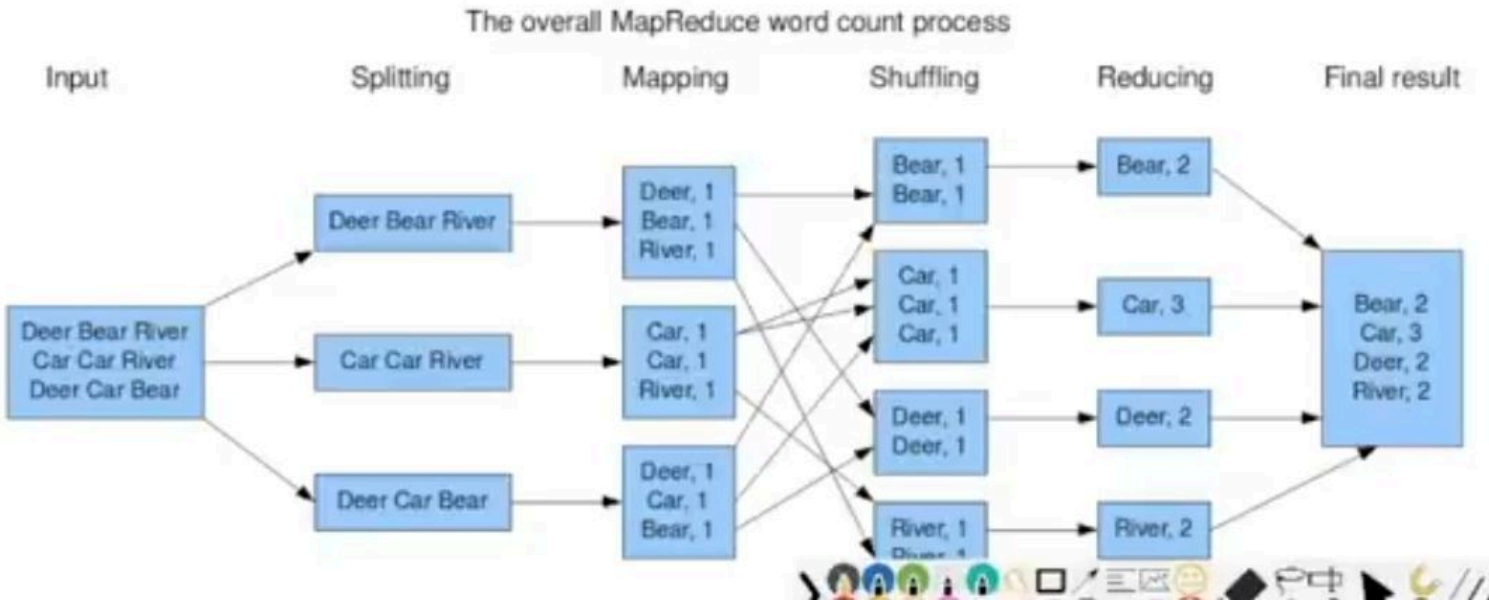
Shuffling in MapReduce

The process of transferring data from the mappers to reducers is shuffling. It is also the process by which the system performs the sort. Then it transfers the map output to the reducer as input. This is the reason shuffle phase is necessary for the reducers.



Sorting in MapReduce

MapReduce Framework automatically sort the keys generated by the mapper. Thus, before starting of reducer, all intermediate key-value pairs get sorted by key and not by value. It does not sort values passed to each reducer. They can be in any order.



Map Reduce types

Mapping is the core technique of processing a list of data elements that come in pairs of keys and values. The map function applies to individual elements defined as key-value pairs of a list and produces a new list. The general idea of map and reduce function of Hadoop can be illustrated as follows:

map: (K1, V1) -> list (K2, V2)
reduce: (K2, list(V2)) -> list (K3, V3)

The input parameters of the key and value pair, represented by K1 and V1 respectively, are different from the output pair type: K2 and V2. The reduce function accepts the same format output by the map, but the type of output again of the reduce operation is different: K3 and V3.

Input formats

Hadoop InputFormat describes the input-specification for execution of the Map-Reduce job. InputFormat describes how to split up and read input files. In MapReduce job execution, InputFormat is the first step. It is also responsible for creating the input splits and dividing them into records.
Input files store the data for MapReduce job. Input files reside in HDFS. Although these files format is arbitrary, we can also use line-based log files and binary format.

InputFormat class is one of the fundamental classes which provides below functionality:

- InputFormat selects the files or other objects for input.
- It also defines the Data splits. It defines both the size of individual Map tasks and its potential execution server.
- Hadoop InputFormat defines the RecordReader. It is also responsible for reading actual records from the input files.

Types of Input formats

- **FileInputFormat:** When we start a MapReduce job execution, FileInputFormat provides a path containing files to read. This InputFormat will read all files. Then it divides these files into one or more InputSplits.
- **TextInputFormat:** It is the default InputFormat. It performs no parsing. It is useful for unformatted data or line-based records like log files.
- **KeyValueTextInputFormat:** The difference is that TextInputFormat treats entire line as the value, but the KeyValueTextInputFormat breaks the line itself into key and value by a tab character ('/t').
- **SequenceFileInputFormat:** It is an InputFormat which reads sequence files. Sequence files are binary files. These files also store sequences of binary key-value pairs.
- **DBInputFormat**: This InputFormat reads data from a relational database, using JDBC.

Output formats

Hadoop RecordWriter:

As we know, Reducer takes as input a set of an intermediate key-value pair produced by the mapper and runs a reducer function on them to generate output that is again zero or more key-value pairs.
RecordWriter writes these output key-value pairs from the Reducer phase to output files.

Hadoop Output Format:

As we saw above, Hadoop RecordWriter takes output data from Reducer and writes this data to output files. The way these output key-value pairs are written in output files by RecordWriter is determined by the Output Format.

FileOutputFormat.setOutputPath() method is used to set the output directory. Every Reducer writes a separate file in a common output directory.

Types of Output formats

- **TextOutputFormat:** MapReduce default Hadoop reducer Output Format is TextOutputFormat, which writes (key, value) pairs on individual lines of text files and its keys and values can be of any type since TextOutputFormat turns them to string by calling toString() on them. Each key-value pair is separated by a tab character KeyValueTextOutputFormat is used for reading these output text files since it breaks lines into key-value pairs based on a configurable separator.

- **Multiple Outputs:** It allows writing data to files whose names are derived from the output keys and values, or in fact from an arbitrary string.

- **LazyOutputFormat:** Sometimes FileOutputFormat will create output files, even if they are empty. LazyOutputFormat is a wrapper OutputFormat which ensures that the output file will be created only when the record is emitted for a given partition.

- **DBOutputFormat**: This InputFormat reads data from a relational database, using JDBC.

Map Reduce features

1. **Scalability:** Apache Hadoop is a highly scalable framework. This is because of its ability to store and distribute huge data across plenty of servers. All these servers were inexpensive and can operate in parallel. We can easily scale the storage and computation power by adding servers to the cluster.
2. **Flexibility:** MapReduce programming enables companies to access new sources of data. It enables companies to operate on different types of data.
3. **Security and Authentication:** model uses HBase and HDFS security platform.
4. **Cost-effective solution:** allows the storage and processing of large data sets in a very affordable manner.
5. **Fast:** Hadoop uses a distributed storage method called as a Hadoop Distributed File System.
6. **Parallel Processing:** The parallel processing allows multiple processors to execute these divided tasks. So the entire program is run in less time.