
# Big Data Hadoop and Spark Developer



simplilearn

DATA AND
ARTIFICIAL INTELLIGENCE

# MapReduce: Advanced Concepts

# Learning Objectives

By the end of this lesson, you will be able to:

- Explain MapReduce data types
- Identify how partitioners partition the data
- Move data from mapper to reducer using shuffling and sorting
- Illustrate distributed cache, job chaining, and scheduling

DATA AND
ARTIFICIAL INTELLIGENCE

# Data Types in Hadoop

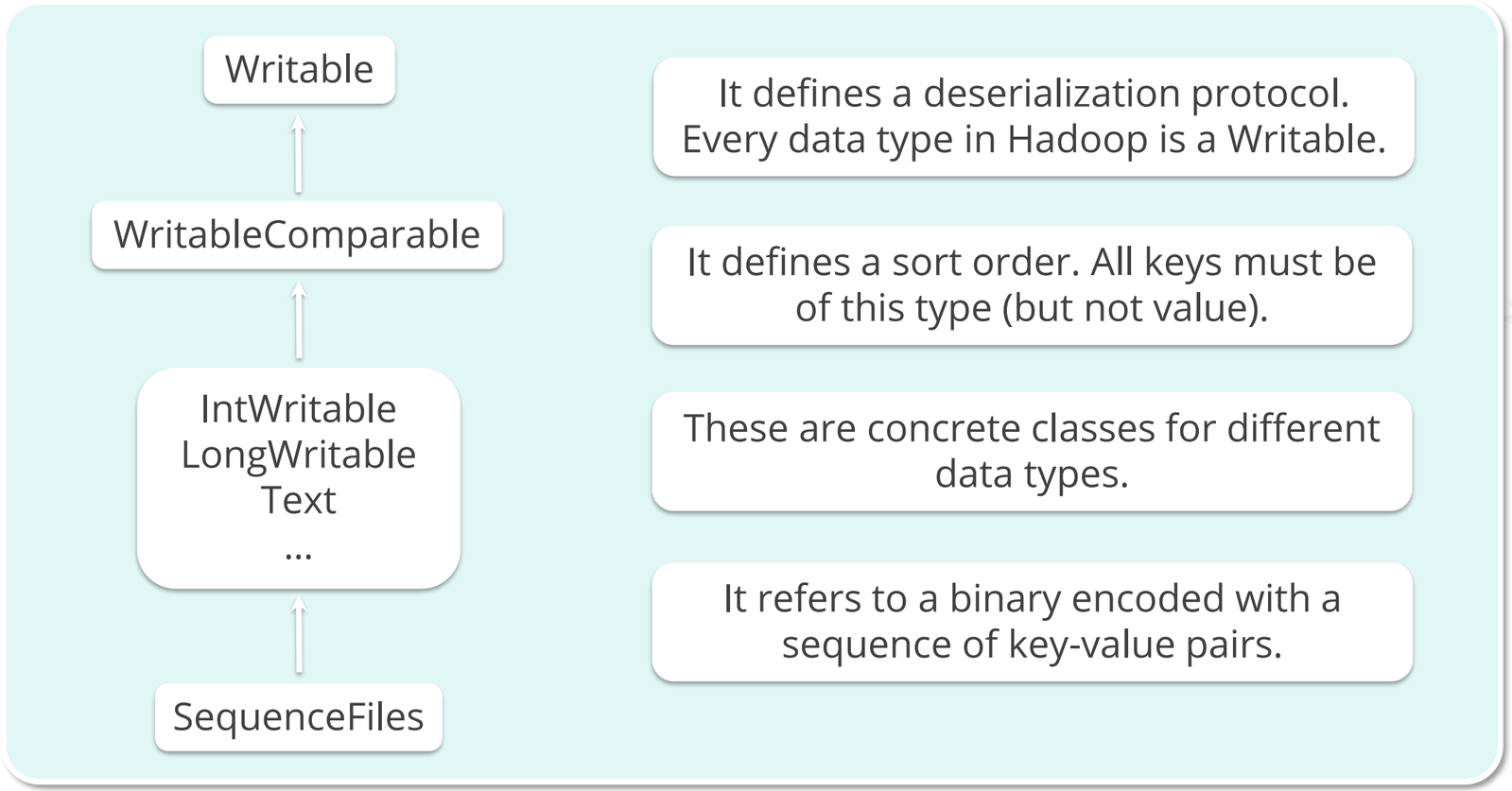
# Data Types in Hadoop

The table lists a few important data types and their functions:

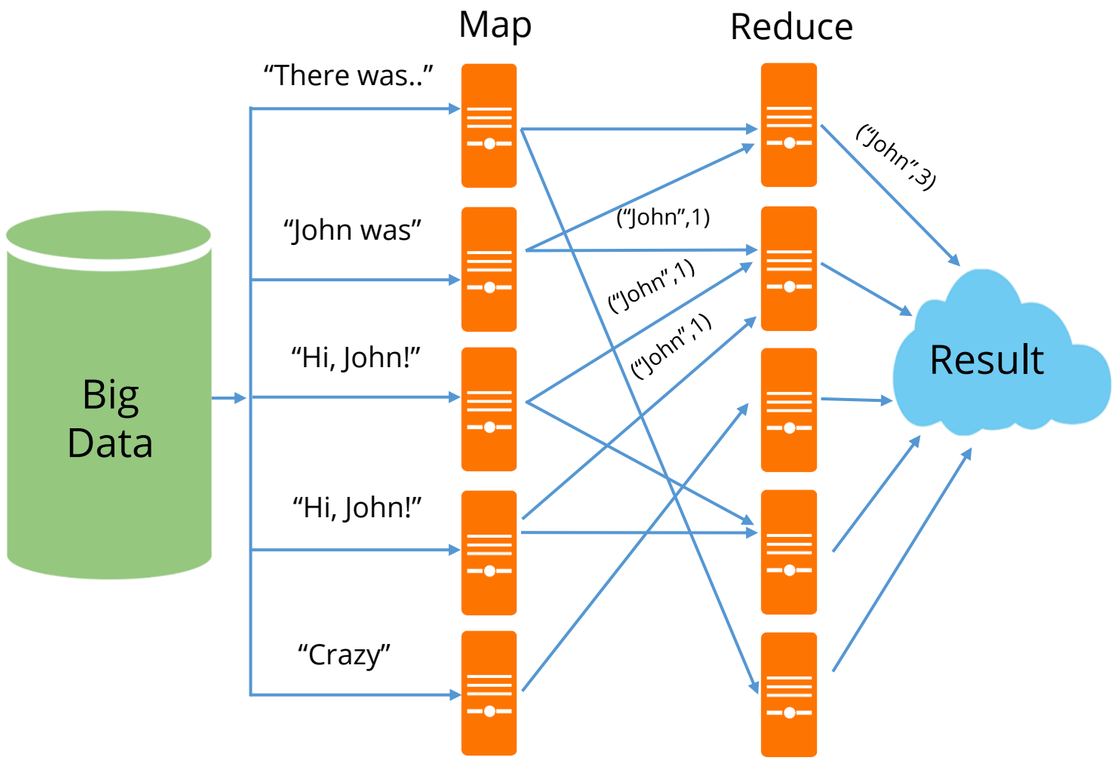
| Data types | Functions |
|---|---|
| Text | Stores string data |
| IntWritable | Stores integer data |
| LongWritable | Stores long data |
| FloatWritable | Stores float data |
| DoubleWritable | Stores double data |
| BooleanWritable | Stores boolean data |
| ByteWritable | Stores byte data |
| NullWritable | Placeholder when value is not needed |

# Data Types in Hadoop

A sample data type related to the Writable interface is displayed here:

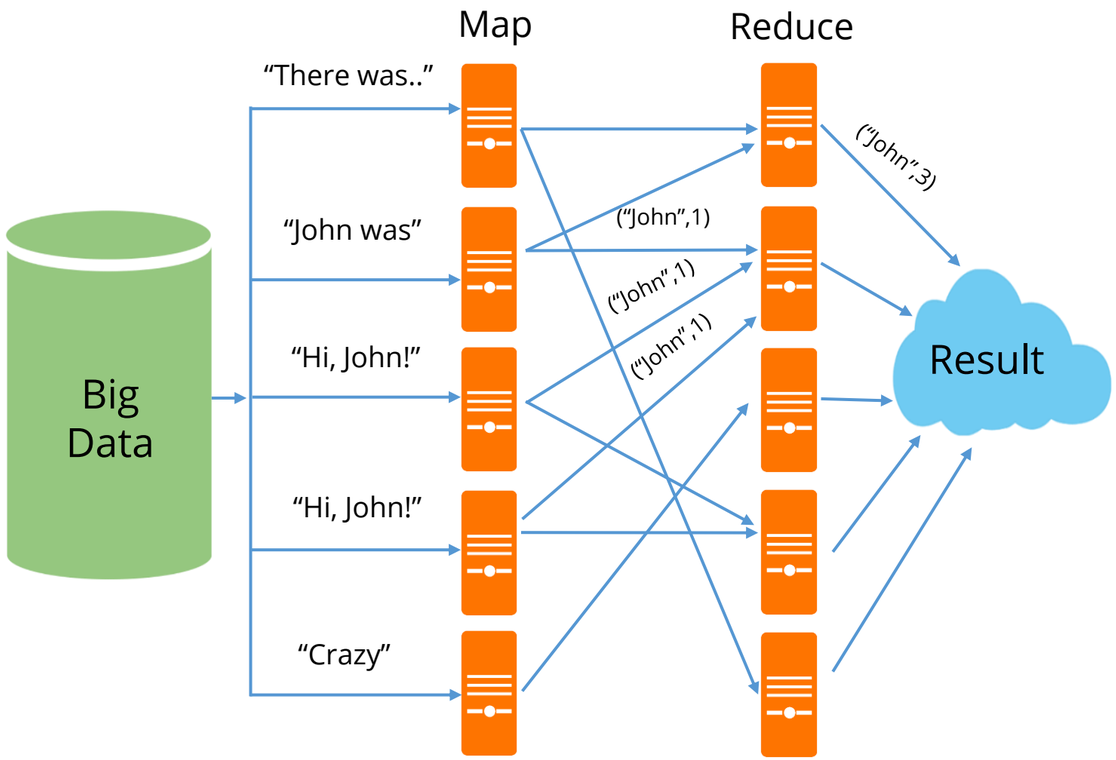


# Advanced MapReduce




- Hadoop MapReduce uses data types when it works with user-given mappers and reducers.
- The data is read from files into mappers and emitted by mappers to reducers.
- Reducers send back the processed data.

# Advanced MapReduce




- Reducers emit the data that goes into output files. At every step, data is stored in Java objects.

- Writable data types: In the Hadoop environment, objects that can be put to or received from files and across the network must obey the Writable interface.

# Interfaces

The interfaces in Hadoop are as follows:

**Writable**

**Writable Comparable**

A writable interface allows Hadoop to read and write the data in a serialized form for transmission.

```
interface Writable {
 public voidreadFields(DataInput in);
 public void write(DataOutput out);
}
```

# Interfaces

The interfaces in Hadoop are as follows:

**Writable**

**Writable Comparable**

A WritableComparable interface extends the Writable interface so that the data can be used as a key and not as a value.

```
int compareTo(Object what)
int hashCode()
```

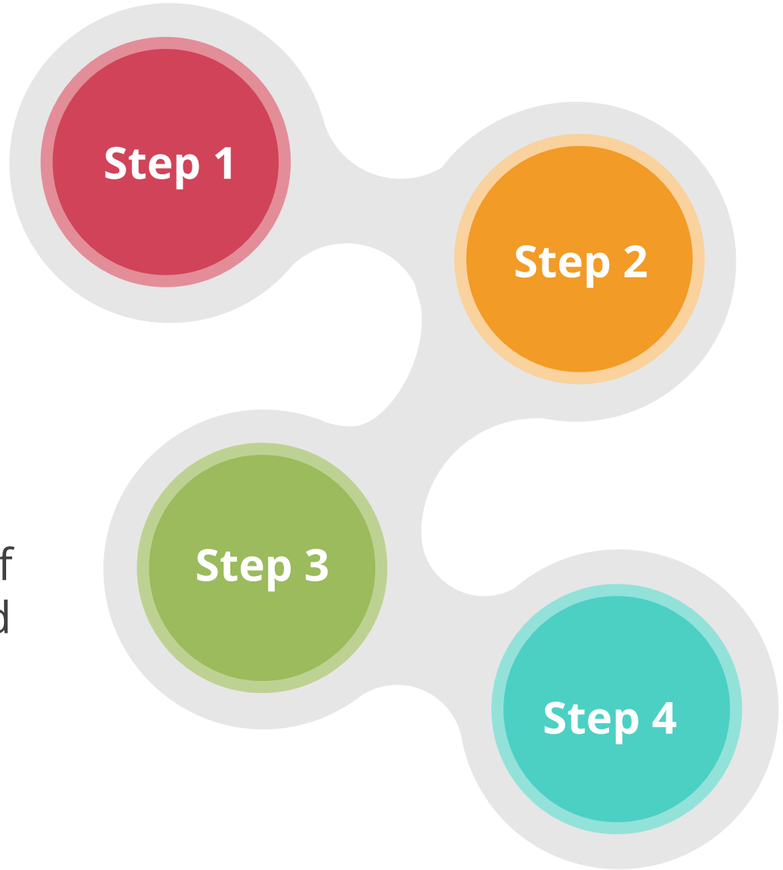
DATA AND
ARTIFICIAL INTELLIGENCE

# Custom Data Type Using WritableComparable Interface

# Custom Data Type Using WritableComparable Interface

Rules for creating custom data type:

Implement **WritableComparable** Interface




Provide the implementation of **write** method

Provide the implementation of **readField** method

Override **compareTo** method for doing custom sorting

# Sample Program to Implement WritableComparable Interface

Main Class

public static class sampleWritable implements WritableComparable<sampleWritable>

Sample code: Write method

```
public void write(DataOutput out)
throws IOException    {
    ipaddress.write(out);
    timestamp.write(out);
    reqDate.write(out);
    reqNo.write(out);
    siteURL.write(out);}
```

# Sample Program to Implement WritableComparable Interface

Main Class

public static class sampleWritable implements WritableComparable<sampleWritable>

Sample code : readFields method

```
public void readFields(DataInput in)
throws IOException
   {
    ipaddress.readFields(in);
    timestamp.readFields(in);
    reqDate.readFields(in);
    reqNo.readFields(in);
    siteURL.readFields(in);
   }
```

# Sample Program to Implement WritableComparable Interface

Main Class

public static class sampleWritable implements WritableComparable<sampleWritable>

Sample code: Override compareTo method

```
public int compareTo(sampleWritable o)
   {
     if (ipaddress.compareTo(o.ipaddress)==0)
     {
       return
(timestamp.compareTo(o.timestamp));
     }
     else return
(ipaddress.compareTo(o.ipaddress));
   }
```

# InputFormats in MapReduce

MapReduce can specify how its input is to be read by defining an InputFormat. The Hadoop framework provides the following some of the key classes of InputFormats:

| InputFormat classes | Description |
|---|---|
| KeyValueTextInputFormat | There is one key-value pair per line. |
| TextInputFormat | The key is the line number, and the value is the line. |
| NLineInputFormat | It is similar to TextInputFormat, but the difference is that there is N number of lines that make an input split. |
| MultiFileInputFormat | The input format aggregates multiple files into one split. |
| SequenceFileInputFormat | The input file is a Hadoop sequence file which contains a serialized key-value pair. |

# OutputFormats in MapReduce

The Hadoop framework provides the following some of the key classes of OutputFormats:

| OutputFormat classes | Description |
|---|---|
| TextOutputFormat | It is the default OutputFormat and writes records as lines of text. Each key-value pair is separated by a TAB character. This can be customized by using the mapred.textoutputformat.separator property. The corresponding InputFormat is KeyValueTextInputFormat. |
| SequenceFileOutputFormat | It writes sequence files to save the output. It is compact and compressed. |
| SequenceFileAsBinaryOutputFormat | It writes key and value in raw binary format into a sequential file container. |
| MapFileOutputFormat | It writes MapFiles as the output. The keys in a MapFile must be added in an order, and the reducer will emit keys in the sorted order. |

# OutputFormats in MapReduce

The table lists some of the key classes of OutputFormats provided by the Hadoop framework:

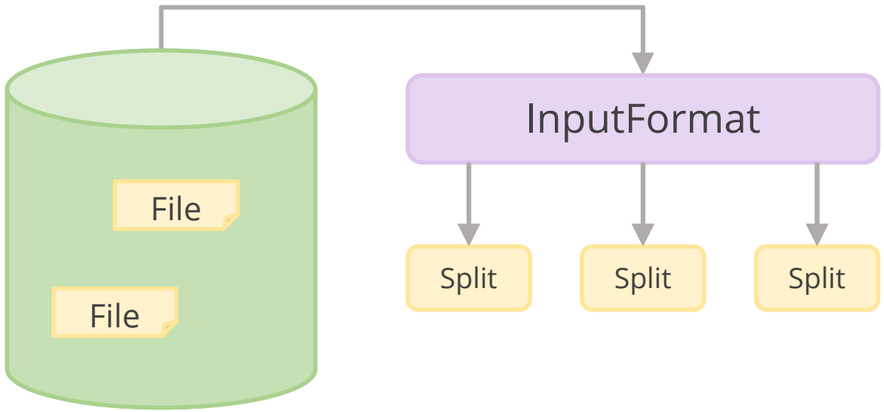
| OutputFormat classes | Description |
|---|---|
| MultipleTextOutputFormat | It writes data to multiple files whose names are derived from output keys and values. |
| MultipleSequenceFileOutputFormat | It creates output in multiple files in a compressed form. |

DATA AND
ARTIFICIAL INTELLIGENCE

# InputSplit

# What Is InputSplit?

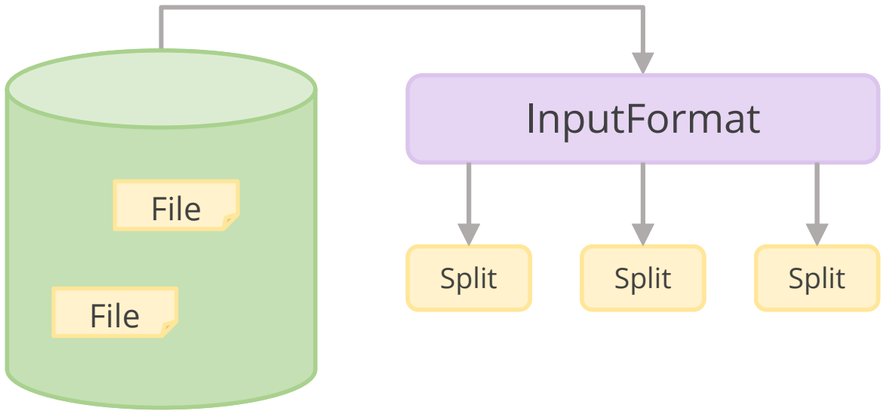
Files loaded from local HDFS stores




- It represents a logical chunk of data that a mapper processes.
- InputSplit does not contain the input data. It is a **reference** to the data.
- By default, each chunk is logically 128-megabyte size same as that of HDFS file block size.
- It can override the default size by changing the **mapred.min.split.size** param in **mapred-site.xml.**

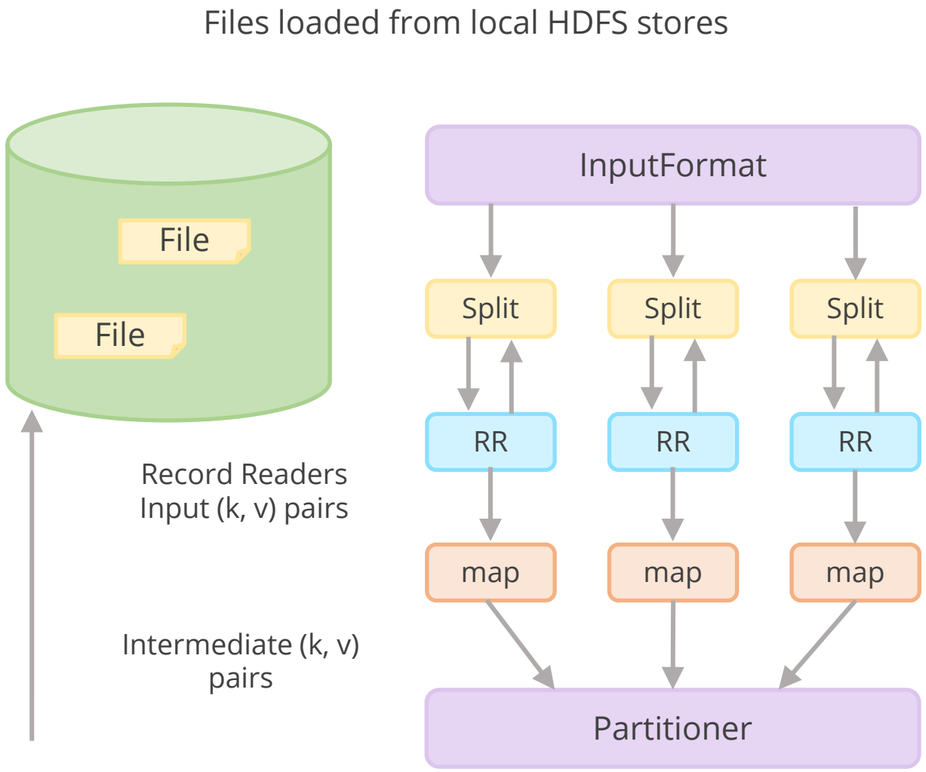
# What Is InputSplit?

Files loaded from local HDFS stores



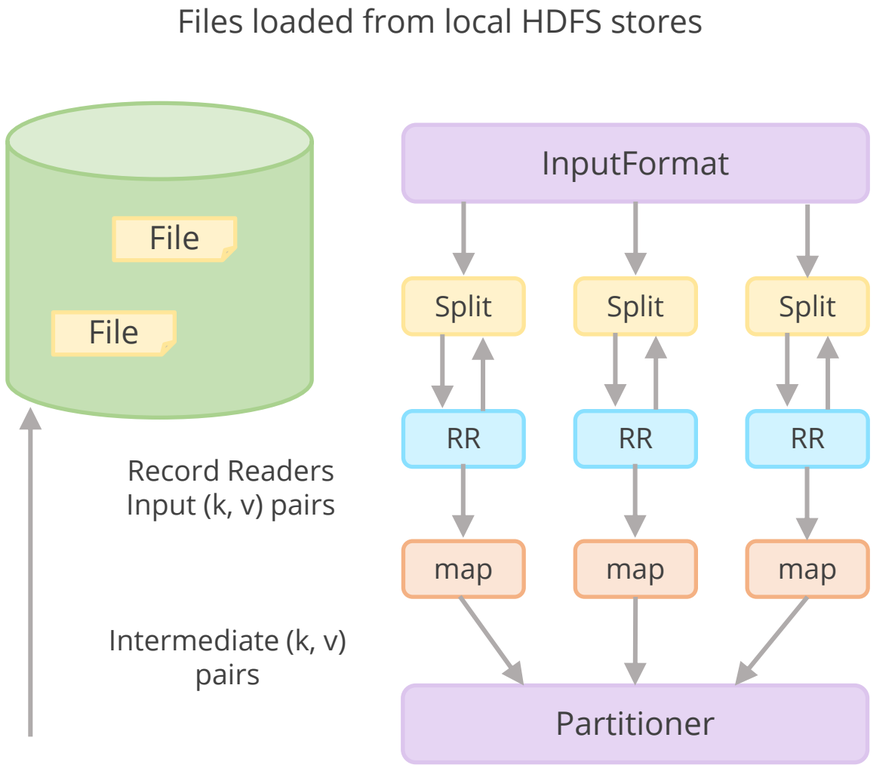

- Calling **getSplit()** method, sends splits to Application Master which uses their storage locations to schedule map tasks that will process them on the cluster.
- After this, recordReader converts logical data into key-value pairs and the same is sent to mappers.

# How Partitioners Partition Data?




- A partitioner partitions the key-value pairs of intermediate map outputs.
- The total number of partitions for the job is the same as the number of Reducer jobs.

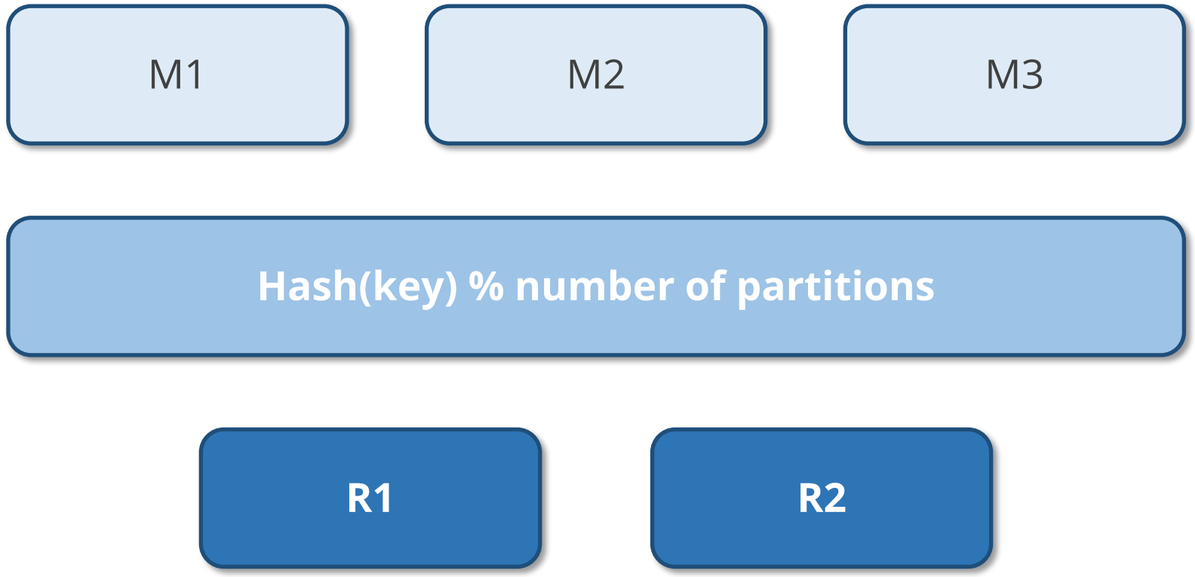
# How Partitioners Partition Data?




- Partitioner in a MapReduce job redirects the mapper output to the reducer by determining which reducer handles the particular key.
- Hence, it determines the total count of Reducers that a program can have.

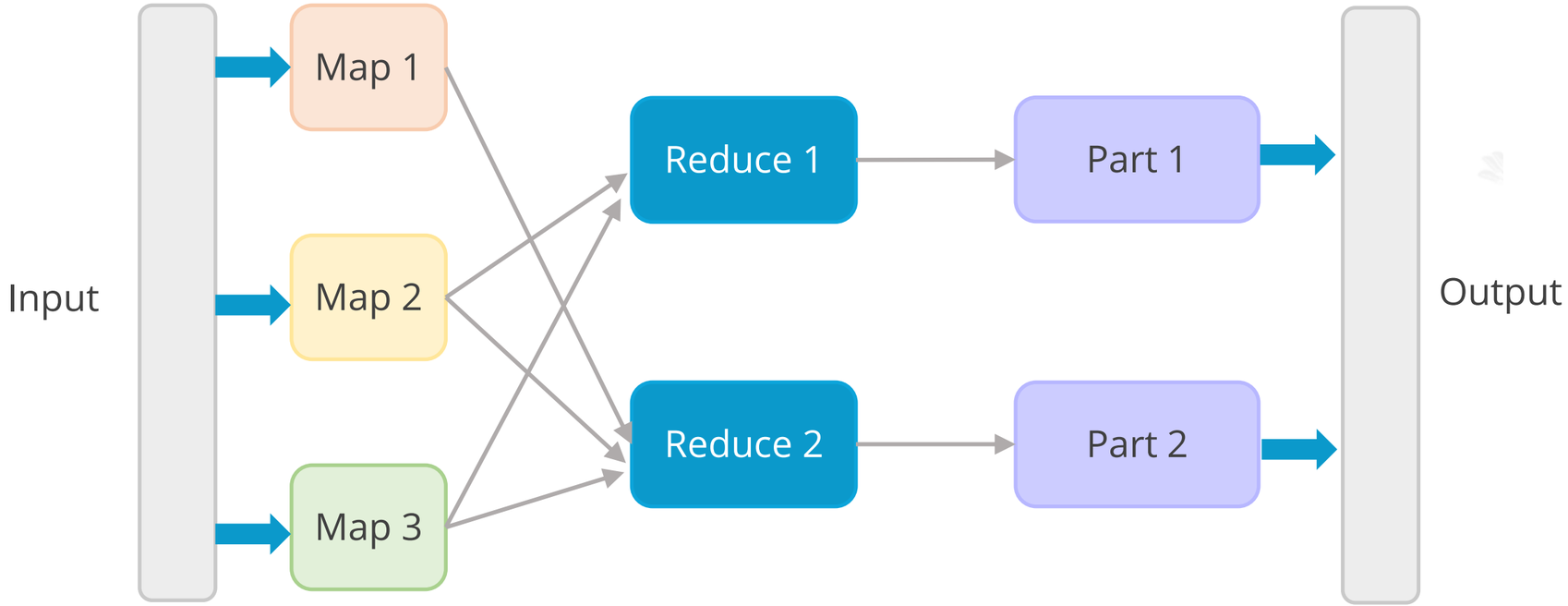
# How Partitioners Partition Data?

The MapReduce framework uses Hash Partitioner by default. Internally it creates a hashCode() method of the key objects modulo the number of partitions total to determine which partition to send a given (key, value) pair to.



# Partition Data: Example

Assume that we receive three mappers from the map phase, and the partitioner determines that only two reducers are required. As a result, only two reducers will execute the task, resulting in two-part files being generated.



DATA AND
ARTIFICIAL INTELLIGENCE

# Custom Partitioner

# Custom Partitioner


- In real-time use cases, there arises the need to write partitioner logic of our own as the default will not suit all situations and can cause some reducers and MapReduce to take more time due to the uneven distribution of the key.
- In some cases, one may want to store the output data based on certain conditions. Here, one can create a custom definition of partitioner.
- In real-time use cases, it is required to write custom partitioner logic.

# Custom Partitioner

These are the steps to implement the custom partitioner:

**Step 1**

Identify the key based on the use case

**Step 2**

Override the **getPartition** method

# Custom Partitioner: Example

iPhone is a global supplier that requires information about the most commonly used currency for transactions.

Sample Dataset:

| Country | Product | Currency |
|---|---|---|
| United States | iPhone | USD |
| United Kingdom | iPhone | GBP |
| Australia | iPhone | USD |
| New Zealand | iPhone | USD |
| India | iPhone | USD |

# Custom Partitioner: Solution

The currency will act as a key in the partition process. The Custom Partitioner class divides the dataset into two partitions, one with a currency as a key and a country as a value.

Code Snippet:

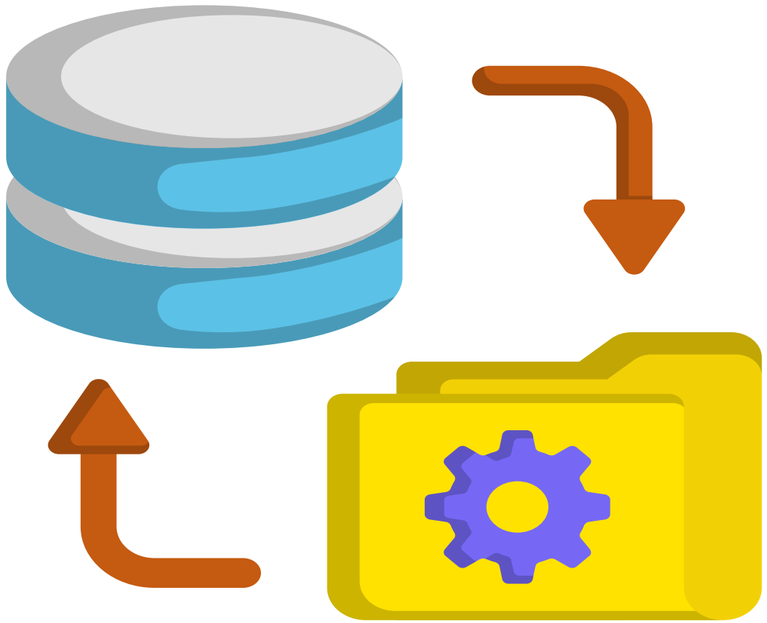
```
public static class customPartitioner extends
Partitioner<Text,Text>{
                public int getPartition(Text key, Text value,
int numReduceTasks){
                if(numReduceTasks==0)
                        return 0;
                if(key.equals(new Text("currency")) &&
!value.equals(new Text("country")))
                        return 0;
                if(key.equals(new Text("currency")) &&
value.equals(new Text("country")))
                        return 1;
                else
                        return 2;
                }
        }
Output:
currency  country
USD         4
GBP         1
```

DATA AND
ARTIFICIAL INTELLIGENCE

# Distributed Cache and Job Chaining

# Distributed Cache

The MapReduce framework provides Distributed Cache, which allows applications to cache files, such as text, archives, jars, and others.



# Distributed Cache: Features

Helps to boost efficiency when a map or a reduce task needs access to common data

Allows a cluster node to read the imported files from its local file system instead of retrieving the files from other cluster nodes

Allows both single files and archives (such as zip and tar.gz)

# Distributed Cache: Features

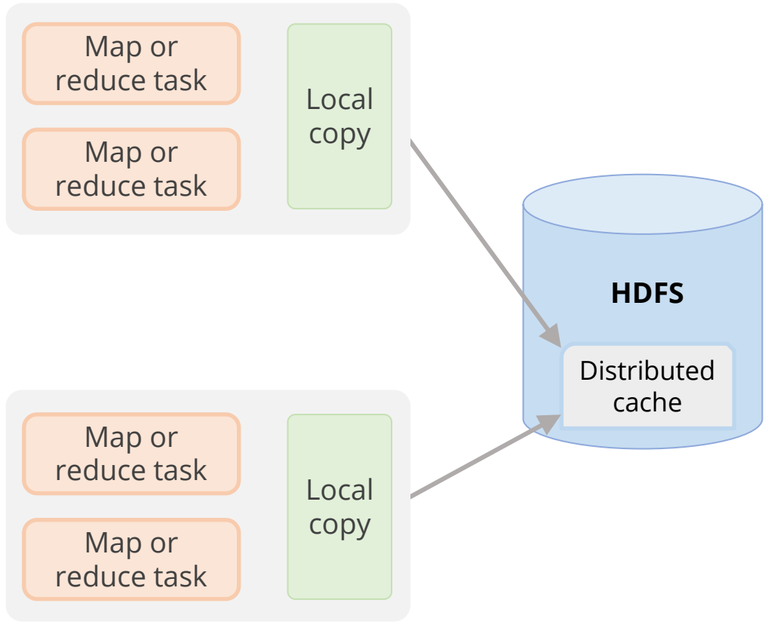
Copies file only to slave nodes; if there are no slave nodes in the cluster, distributed cache copies the files to the master node

Allows access to the cached files from mapper or reducer applications to make sure that the current working directory (./) is added to the application path

Allows one to reference the cached files as though they are present in the current working directory

# Distributed Cache MapReduce

Hadoop Distributed Cache distributes read-only file-based resources to the map and reduce tasks. These resources can be simple data files, archives, or JAR files that are needed for the computations performed by the mappers or the reducers.



# How It Works?

The following steps showcase how to add a file to the Hadoop Distributed Cache and how to retrieve it from the Map and Reduce tasks.

**Step 1**

Create a sample file

cmd:

```
ip-10-0-42-218 login: alpikaguptasimplilearn
Password:
Last login: Wed Apr 27 11:54:35 on pts/1584
[alpikaguptasimplilearn@ip-10-0-42-218 ~]$ vi sample.txt
```

# How It Works?

**Step 2**

Copy the sample.txt file to the HDFS

cmd:

```
hdfs dfs -put sample.txt /user/alpikaguptasimplilearn
```

- Users have to consider sample.txt file contains data that needs to be present to the worker nodes.
- Distributed Cache can also be used to distribute archives (tar, jar, and tgz). Hadoop extracts the archives in the worker nodes.

# How It Works?

**Step 3**

Add the resource to the Distributed Cache from your driver program

cmd:

```
Job job = new Job(getConf(), "apache-log4j.config ");
……
DistributedCache.addCacheFile(new
URI("sample.txt"),job.getConfiguration());
```

- Hadoop copies the files added to the Distributed Cache to all the worker nodes before the execution of any task of the job.
- Distributed Cache copies these files only once per job. Here, all workers will be having at least one copy of sample.txt file.

# How It Works?

**Step 4**

Retrieve the resource in the setup() method of mapper or reducer and use the data in the map or reduce function

cmd:

```
public class LogProcessorMap extends Mapper<Object,
LogWritable, Text, IntWritable> {
private IPLookup lookupTable;
public void setup(Context context) throws IOException{
File lookupDb = new File("sample.txt ");
// Load the IP lookup table to memory
lookupTable = IPLookup.LoadData(lookupDb);}
public void map(…) {
String country =
lookupTable.getCountry(value.ipAddress);
…… }}
```

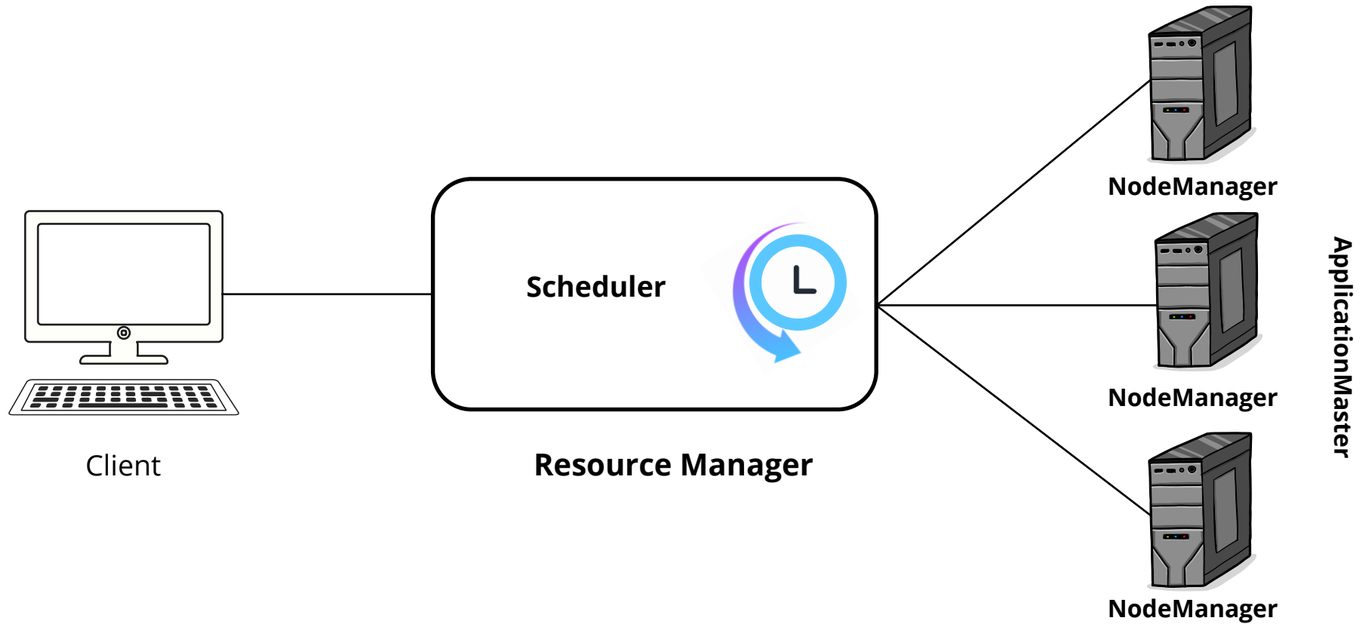
- Distributed Cache parses and loads the data in the setup() method of the mapper or the reducer.
- The data will be available for processing while creating reference lookupDB in the distributed file. MapReduce job can process this.

DATA AND
ARTIFICIAL INTELLIGENCE

# Hadoop Scheduler and Its Types

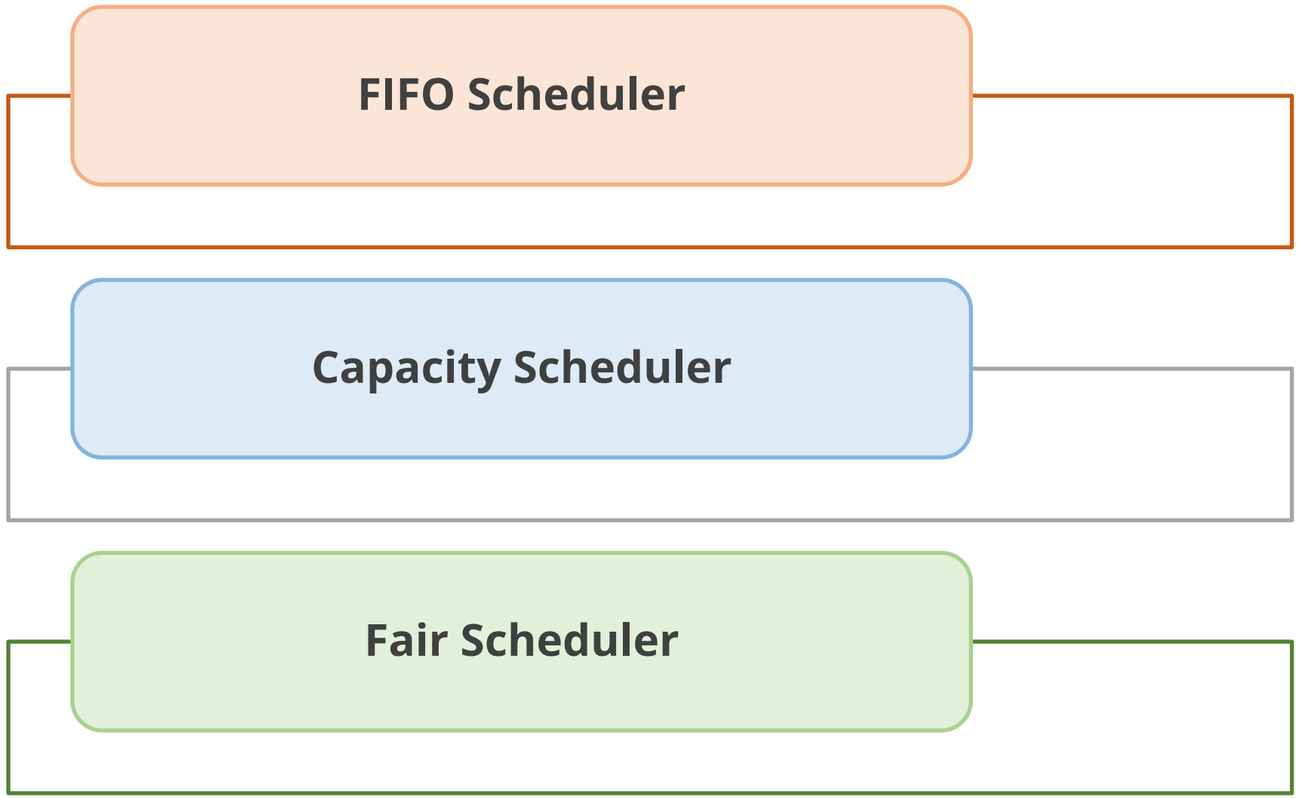
# Hadoop Scheduler

The Scheduler in YARN schedules the jobs, but it can not track the status of the application. It schedules the jobs based on the required resources.

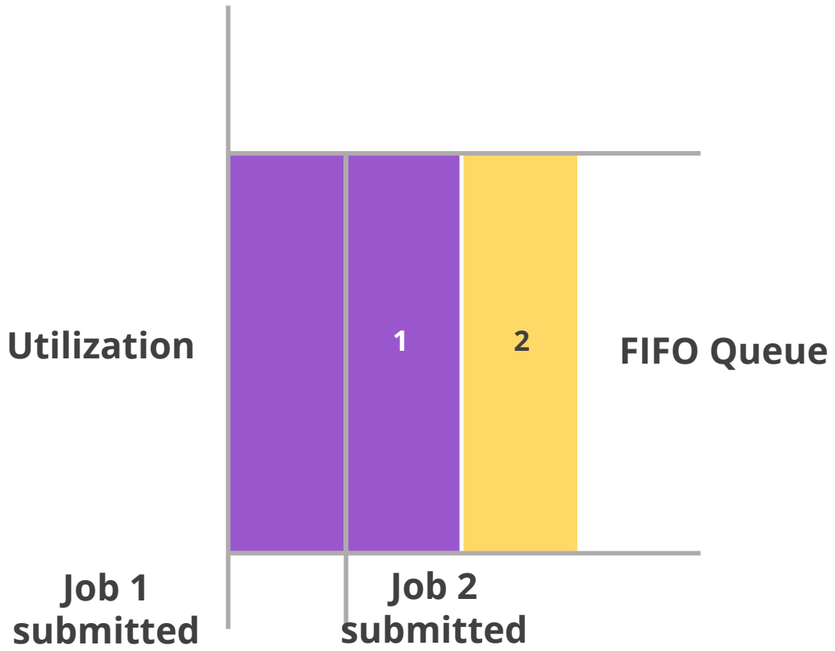


# Hadoop Scheduler: Types

There are the following three types of schedulers:



# FIFO Scheduler




- FIFO stands for First In First Out.
- It is a default scheduling technique.
- It keeps all the applications in a queue and executes them one by one as they arrive.
- There is no preference given on the size or priority of the tasks.
- It is good for testing in standalone Hadoop environments.
- In the case of clusters, FIFO is not successful.

# FIFO Scheduler

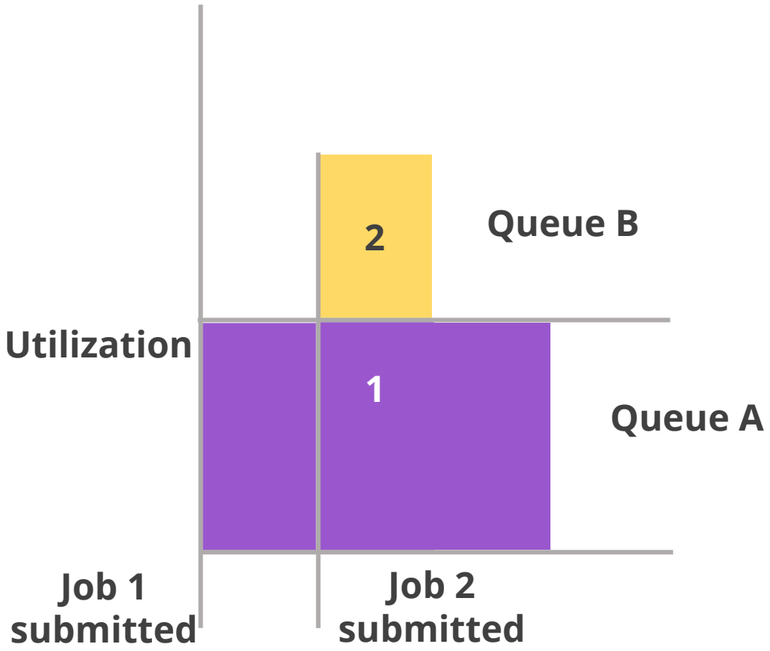
## Advantages

- It is easy to use and does not require any configuration.
- It is based on first come first serve.
- Execution of tasks is straightforward.

## Disadvantages

- It does not care about the task's priority, therefore high-priority tasks need to wait.
- It is not compatible with a shared cluster.

# Capacity Scheduler




- It supports hierarchical queues.
- It allows for multiple tenants to securely share a large cluster.
- It guarantees each organization a minimum capacity.
- It also provides elasticity for groups or organizations in a cost-effective manner.

# Capacity Scheduler

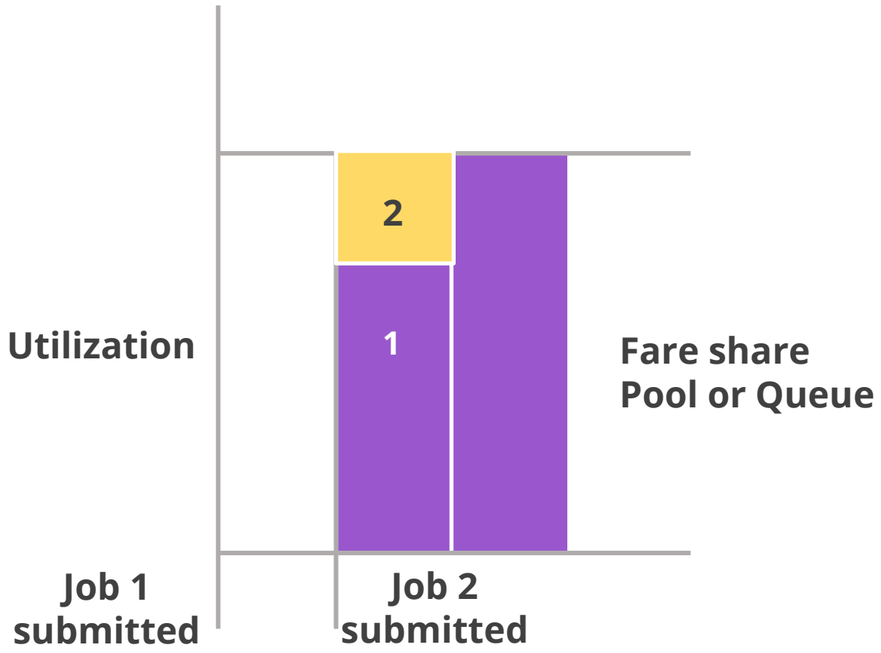
## Advantages

- It works well while working with multiple clients or if there are priority jobs available in a Hadoop cluster.
- It improves the throughput in the Hadoop cluster.

## Disadvantages

- It is more complex.
- It is not easy for everyone to configure.

# Fair Scheduler




- It assigns an equal share of resources to the jobs over time.
- There is no need to reserve resources, it will be provided based on incoming jobs requirement.
- It supports a hierarchical queue to reflect the structure of a shared cluster.
- It has inbuilt support for application priorities.
- Priorities determine the fraction of the total resources that each application should get.

# Fair Scheduler

## Advantages

- The resources allotted to each application is determined by its priority.
- It has the ability to limit the number of concurrently running tasks in a pool or queue.

## Disadvantages

- The configuration is required.
- There is no concept of job weight of each node.

# Assisted Practice: Execution of MapReduce Job Using Custom Partitioner

**Duration**: 10 mins

**Problem Scenario:** Write the commands to perform custom partitions using the JAR files for the execution of a MapReduce job

**Objective:** In this demonstration, you will explore the execution of a MapReduce job in Simplilearn's Lab by creating a custom partitioner.

**Dataset Name:** "**wordcount.txt**"

# Assisted Practice: Execution of MapReduce Job Using Custom Partitioner

**Tasks to Perform:**

**Step 1:** Download the **hadoop-custom-demo.jar** file and **wordcount.txt** file

**Step 2:** Log in to the FTP using the username and password from the lab and upload the file

**Step 3:** Log in to the **"Webconsole"** using the username and password from the lab and create a new directory **CustomPartitioner20** in HDFS using the **mkdir** command

**Step 4:** Push the **wordcount.txt** file into the directory using the put command

**Step 5:** Execute the command to move the **hadoop-custom-demo.jar** file to the HDFS directory

**Step 6:** View the files in the **final** folder with the part files

**Note: The solution to this assisted practice is provided under the course resources section.**

# Key Takeaways

- Hadoop MapReduce uses data types when it works with user-given mappers and reducers.
- A writable interface allows Hadoop to read and write the data in a serialized form for transmission.
- InputSplit represents a logical chunk of data that a mapper processes.
- It divides the data into sections based on a user-defined criterion that operates like a hash function.

DATA AND
ARTIFICIAL INTELLIGENCE


# Knowledge Check

**Knowledge Check**

**1**

**In MapReduce, which scheduling reserves at least minimum resources for any kind of job that comes in?**

A. FIFO

B. Fair scheduling

C. Capacity scheduling

D. None of the above

**Knowledge Check**

**1**

**In MapReduce, which scheduling reserves at least minimum resources for any kind of job that comes in?**

A. FIFO

B. Fair scheduling

C. Capacity scheduling

D. None of the above


The correct answer is **C**

**All jobs submitted to a queue will have access to the capacity allocated to the queue.**

**Knowledge Check 2**

**What exactly does the InputSplit operation represent in MapReduce?**

A. It is a physical representation of data.

B. An individual Mapper processes this data.

C. An individual Reducer processes this data.

D. It is the data that can be written in the temporary directory after the map phase.

**Knowledge Check**

**2**

**What exactly does the InputSplit operation represents in MapReduce?**

A. It is a physical representation of data.

B. An individual Mapper processes this data.

C. An individual Reducer processes this data.

D. It is the data that can be written in the temporary directory after the map phase.

The correct answer is **B**

**InputSplit represents the data that an individual Mapper processes. It presents a byte-oriented view of the input and is the responsibility of RecordReader of the job to process this and present a record-oriented view.**

**Knowledge Check**

**3**

**While writing custom partitioners, which method should need to be overridden?**

A. getPartitioner(Text key, Text value, int numReduceTasks)

B. getPartition(String key, Text value, int numReduceTasks)

C. getNumReducer((Text key, Text value, int totalTask)

D. getPartition(Text key, Text value, int numReduceTasks)

**Knowledge Check**

**3**

**While writing custom partitioners, which method should need to be overridden?**

A. getPartitioner(Text key, Text value, int numReduceTasks)

B. getPartition(String key, Text value, int numReduceTasks)

C. getNumReducer((Text key, Text value, int totalTask)

D. getPartition(Text key, Text value, int numReduceTasks)

The correct answer is **D**

**public int getPartition(Text key, Text value, int numReduceTasks) is used for writing custom partition.**

**Knowledge Check**

**4**

**Which of the following is a strong Hadoop interface for serializing data?**

A. ReadWritable and WritableComparable

B. Writable and readFields

C. Writable and compareTo

D. Writable and WritableComparable

**Knowledge Check**

**4**

**Which of the following is a strong Hadoop interface for serializing data?**

A. ReadWritable and WritableComparable

B. Writable and readFields

C. Writable and compareTo

D. Writable and WritableComparable

The correct answer is **D**

**Writable and Writable Comparable is a strong interface in Hadoop which while serializing the data, reduces the data size enormously so that data can be exchanged easily within the networks.**

# Lesson-End Project: Flipkart Analysis

**Problem Scenario:**

Alex is working in the Security council of an MNC whose main focus is to detect any security vulnerability in any open-source platform. Soon, he figured out, a vulnerability in Apache Log4j that can cause major issues in company inbuilt products as almost all products are using the same for logging. Alex already informed teams and managers about the same and recommended to upgrade in the new version, but product owners are slightly reluctant over this as they have very less time to upgrade. So, Alex suggested they search for the hour of the day when requests are minimum. Everyone agreed and started working on the same.

The data collected is present in the files: Apache-log.log
LoggerAnalysis.jar

# Lesson-End Project: Flipkart Analysis

**Tasks to Perform:**

1. Download the “Apache-log.log“ log file from the course resources
2. Create a directory into an HDFS and upload the file
3. Write a Map function to extract the hour of the day from a log file
4. Write Reduce function to aggregate with the Mapper and get the final output file
5. Create a JAR file from the Mapper and Reducer file
6. Execute the JAR file to see the output

DATA AND
ARTIFICIAL INTELLIGENCE

# Thank You