Map Reduce Framework

How is Flipkart able to dodge the famous "Wanna Cry" ransomware attack?

A famous world wide attack due to a vulnerability exposed by WinOS in May 2017 was encrypting data and making it unusable to be used.

Flipkart was using almost 4000 servers on windows and was under threat so they decided to have a call with microsoft in order to get a proper patch. Microsoft was quick to give this to flipkart , Now the issue is when to apply the patch in order to shield its servers from the same as this patch requires restart of all servers. They have below mentioned major problems in hand:

a. What will be the optimum time to bring servers down?
b. Do we need to do this as a single server one by one or in batches?
c. How much time will it take to reboot a single server ?
d. How many business hours can be impacted through this?

Solution:

After so much brainstorming, flipkart thought of bringing batches of 50 servers down but for this they need exact time on when they should go ahead and bring them down.

To determine exact time, they need to analyze almost 200-300 GB of logs and then figure out on which particular hour did requests were minimum.

Ingesting and processing huge amount of logs can't be done with :

a. Writing python/Java scripts
b. Through MYSQL, Oracle as logs don't have any structure
c. Writing logic and processing using distributed systems like

i. DASK
ii. NoSQL DB (MongoDB)
iii. Map Reduce

DASK : **Dask** is a flexible parallel computing library for analytics and can only be used for python also for bigger workloads that start giving performance issues.

NoSQL DB : NoSQL DB like mongoDB can provide processing capabilities but it is full licensed which makes it quite expensive

Map Reduce: An open source framework designed to process TB of data on 1000 of machines parallelly fits perfect here. Also one can write code in Java, Python, even c++ too.

What is Map Reduce then?

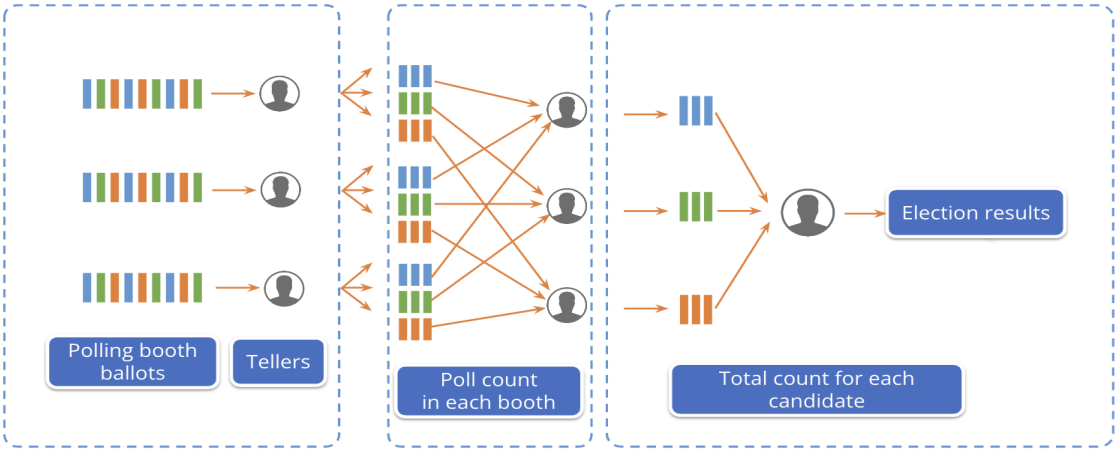
MapReduce is a **programming** model that simultaneously processes and analyzes huge data sets logically into separate clusters.

Map is a function which "transforms" items in some kind of list to another kind of item and put them back in the same kind of list.

Reduce is a function which "collects" the items in lists and perform some computation on all of them, thus reducing them to a single value.

While **Map** sorts the data, **Reduce** segregates it into logical clusters, thus removing the bad data and retaining the necessary information.

Simple way to understand MR via Analogy



Suppose there is an election going on in the country and for the same, people have casted their votes to respected parties. Now it's the result time so counting of votes are underway.

Each Teller(person who is responsible for counting votes) has been handed over the ballot boxes. So what will the process followed by them:

a. Segregate each party vote based on color/symbol. We call this the map phase.

b. Now after segregation , it's time to split the votes among tellers. For ex teller A will take responsibility for counting party A votes , same with B and C.

c. After this, when each teller gets all votes of a single party , counting will begin and results from each teller will be shared. (Reducer Phase)

d. Party with the most votes is declared as the WINNER!!!!

Note : From a to d , everything is happening in parallel.

On the same analogy, Map Reduce works. Divide and Conquer.

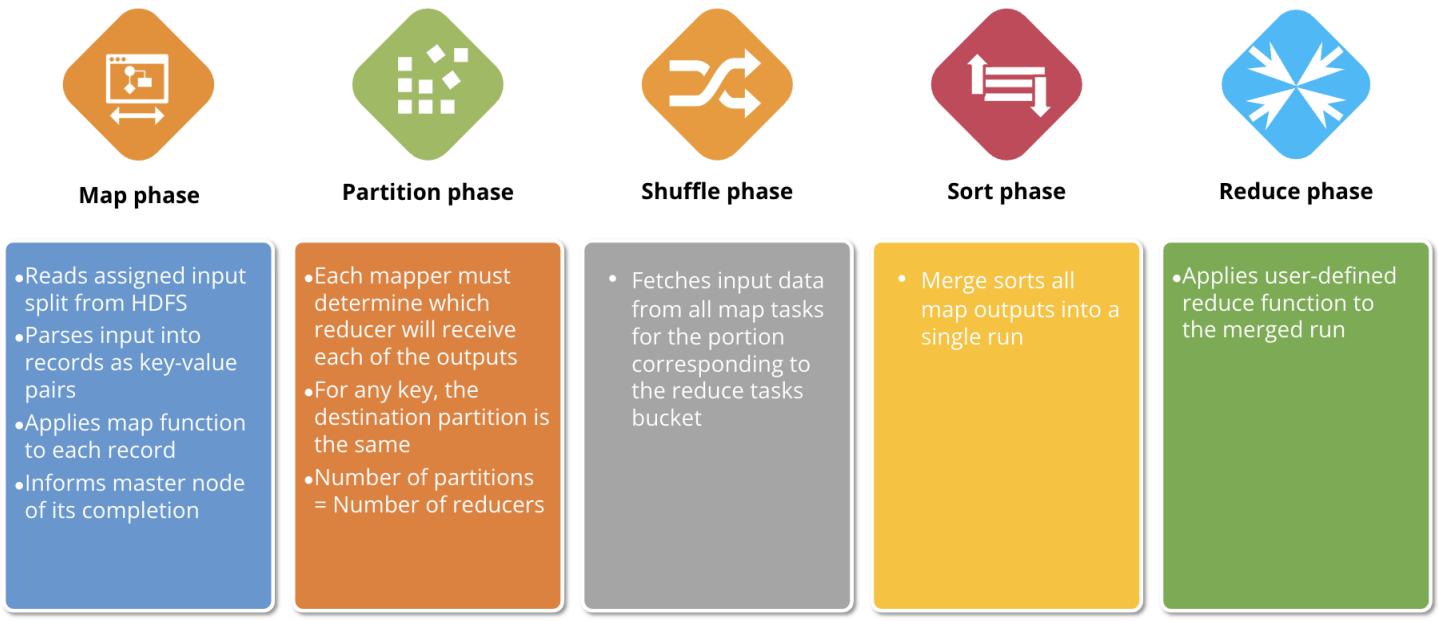
Individual Work

Parallel Work

How Does Map reduce work?

To understand MR, we need to know about its phases. A typical MR job has 5 major phases:

a. Map Phase
b. Partition Phase
c. Shuffle Phase
d. Sort Phase
e. Reduce Phase



How did Flipkart actually use MapReduce to overcome the WannaCry crisis?

Assumptions :

a. To understand this we will assume a **2 node** cluster
b. OS used will be **ubuntu** for both nodes
c. Open source **Apache Log4j** is used for logging

Implementation :

To implement this we need good hands on python and a little bit of the idea of a log file.

a. Mapper File in python (mapper.py)
b. Reducer File in python (reducer.py)
c. Job File in python (jobFile.py)
d. Log file as input data file (Apache-log.log)

What will be the logic behind implementation?

Our logic will be :

a. Gather the input data file (Apache-log.log) from single server and will analyze the same as this can be expanded to multiple servers

b. After this we will focus on each row to extract a timestamp when a request arrives by writing code in a mapper file.

c. In the mapper file , we will write logic to extract the exact hour from timestamp as we need to get at which hour requests are minimum so that patch can be applied to servers

d. Now it's time for the Reducer file to jump in. In this file, we will apply an aggregation function on the data given by Mapper.

e. And finally in the Job file we will glue mapper , reducer file for framework to start work on it.

f. And eventually will explore the final outcome.

How Did execution take place here ?

Once we create Mapper and Reducer file code, then we pass them to Map Reduce Framework and execution starts in below order :

Note: All below steps are internal to MapReduce Framework and they are actual a java file.

Step 1 : **InputFormat** →

- FileInputFormat indicates the path of input files and where the output files should be written.

- InputFormat creates the Inputsplit and divides it into records, FileInputFormat, by default, breaks a file into 128MB chunks which is default for HDFS.

Step 2 : **InputSplit** →

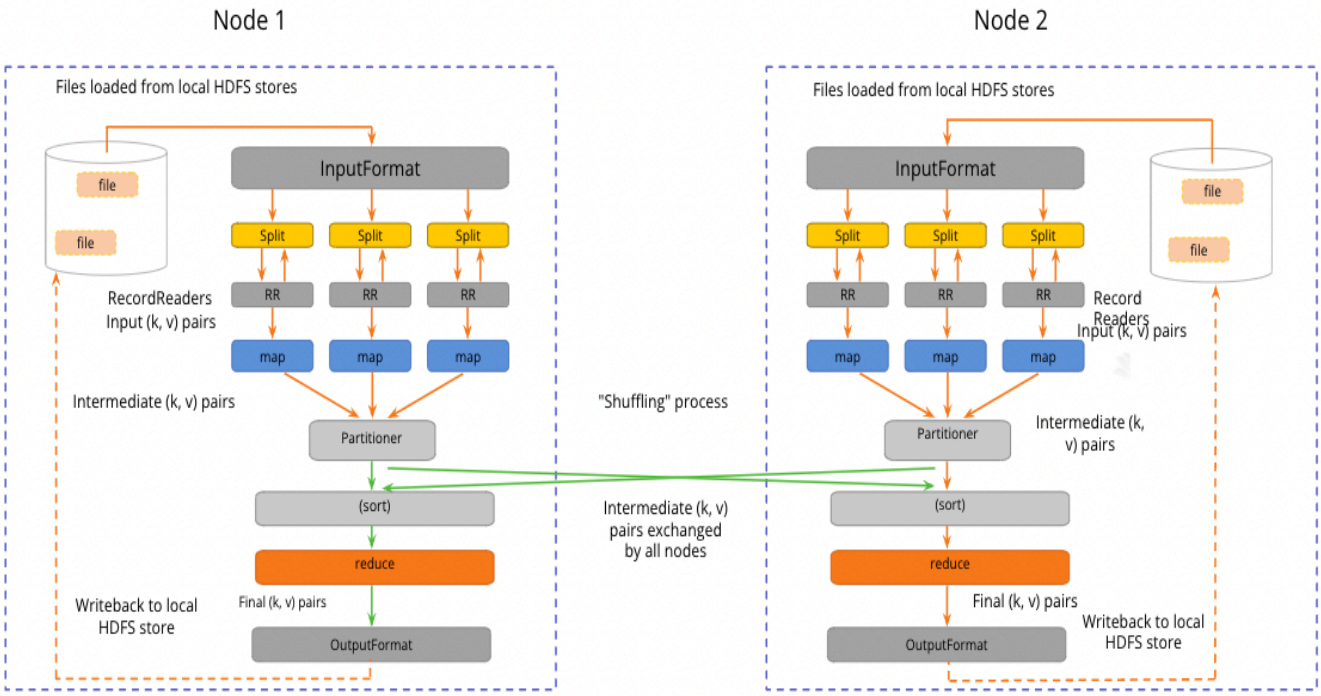
- It represents the data to be processed by an individual Mapper.

- Typically InputSplit presents a byte-oriented view of the input.
- The important thing to notice is that Inputsplit does not contain the input data; it is just a reference to the data.
- As a user, we don't need to deal with InputSplit directly, because they are created by an InputFormat.

Step 3: **RecordReader** →

- Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented view to the Mapper implementations for processing.
- RecordReader thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values.
- RR is responsible for creating key value pairs through byte oriented split, by default key will become **rowNumber** and value will be complete **content of Row.**
- As in flipkart example, input file will read by RR as 1 being row number as key and rest content as value that will be passed to mapper:

(Key,value)(1, 64.242.88.10 - - [01/Jun/2017:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12846)



Step 4: Mapper →

- **RR** output will serve as input to Map Reduce.
- **Mapper** is a function or task which is used to process all input records from a file and generate the output which works as input for Reducer.

- It produces the **intermediate key-value** as output by returning new key-value pairs.
- Mapper is a simple user-defined program that performs some operations on input-splits as per it is designed.
- Finally, the **intermediate key-value pair** output will be stored on the local disk as intermediate output. There is no need to store the data on HDFS as it is an intermediate output.

Mapper Code :

```
#!/usr/bin/env python
# package: com.hadoop.logger
import java.io.IOException

import java.text.ParseException

import java.text.SimpleDateFormat

import java.util.Calendar

import java.util.Date

import java.util.regex.Matcher

import java.util.regex.Pattern

import org.apache.hadoop.io.IntWritable

import org.apache.hadoop.io.LongWritable
```

```
import org.apache.hadoop.io.Text

import org.apache.hadoop.mapreduce.Mapper

class LogMapper(Mapper, LongWritable, Text, IntWritable,
IntWritable):
    """ generated source for class LogMapper """
    hour = IntWritable()
    one = IntWritable(1)
    logPattern = Pattern.compile("([^ ]*) ([^ ]*) ([^ ]*)
\\[([^]]*)\\]" + " \"([^\"]*)\"" + " ([^ ]*) ([^ ]*).*")
    sdf = SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss")

    def map(self, key, value, context):
        """ generated source for method map """
        date = None
        line = (Text(value)).__str__()
        matcher = self.logPattern.matcher(line)
        if matcher.matches():
            try:
                date = Date(sdf.parse(timestamp))
            except ParseException as ex:
                ex.printStackTrace()
            cal.setTime(date)
            self.hour.set(cal.get(Calendar.HOUR_OF_DAY))
            context.write(self.hour, self.one)
```

How Many Maps?

The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

The right level of parallelism for maps seems to be around **10-100 maps** per-node, although it has been set up to **300 maps** for very cpu-light map tasks.

Step 5: Partitioner →

- Partitioner controls the partitioning of the keys of the intermediate map-outputs.
- For example if map output is <1,10> , <1,15> , <1,20> , <2, 13> , <2,6> , <4,8> , <4,20> etc.
    - we can see that there are 3 different keys which are 1, 2 and 4.
    - In map-reduce, the number of reduced tasks are fixed and each reduced task should handle all the data related to one key.
    - That means map output like <1,10>, <1,15>, <1,20> should be handled by the same reduced tasks.
    - It is not possible that <1,10> is handled by one reduce task and <1,15> is handled by another reduce task as the key which is 1 is the same.
- Partitioning is to club the data which should go to the same reducer based on keys.
- HashPartitioner is the default Partitioner.
    - hash(key) % number_of_reducers.

For Instructor : Can explain the same thing with examples from Flipkart too.

Step 6: Shuffling & Sorting →

- In the final output of the map task there can be multiple partitions and these partitions should go to different reduced tasks.
- **Shuffling** is basically transferring map output partitions to the corresponding reduce tasks

Step 7: Reducer →

- Hadoop Reducer takes a set of an intermediate key-value pair produced by the mapper as the input and runs a Reducer function on each of them.
- One can aggregate, filter, and combine this data (key, value) in a number of ways for a wide range of processing.
- Reducer first processes the intermediate values for a particular key generated by the map function and then generates the output (zero or more key-value pair).
- One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another.
- The user can decide the number of reducers. By default the number of reducers is 1.

Reducer Code:

https://github.com/scaleracademy/dsml-de/tree/main/hadoop/Map-Reduce-Flipkart-Issue

Job File Code:

https://github.com/scaleracademy/dsml-de/tree/main/hadoop/Map-Reduce-Flipkart-Issue

How Many Reduces?

The right number of reduces seems to be **0.95 or 1.75** multiplied by (<no. of nodes> * <no. of maximum containers per node>).

What if we have Reducer NONE ?

It is legal to set the number of reduce-tasks to zero if no reduction is desired. Reducer NONE means only mapper will do the job and no aggregation is required.

Step 8 : OutputFormat →

Hadoop RecordWriter takes output data from Reducer and writes this data to output files.

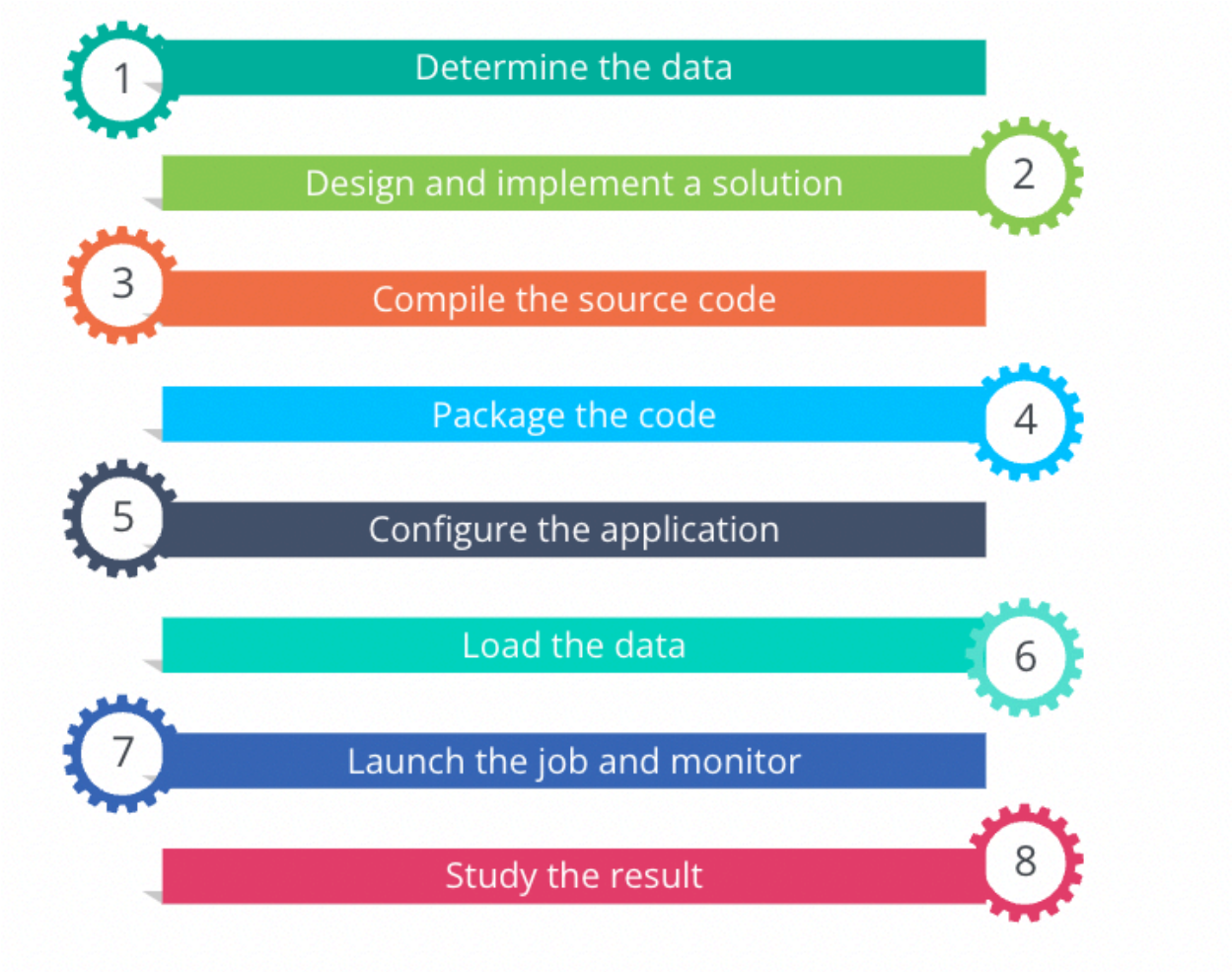
The way these output key-value pairs are written in output files by RecordWriter is determined by the **Output Format.**

The Output Format and InputFormat functions are alike. OutputFormat instances provided by Hadoop are used to write to files on the HDFS or local disk.

OutputFormat describes the output-specification for a Map-Reduce job. On the basis of output specification;

- MapReduce job checks that the output directory does not already exist.
- **OutputFormat** provides the **RecordWriter** implementation to be used to write the output files of the job. Output files are stored in a FileSystem.

What are the Steps for building a map Reduce Program?



a. **Determine the data:**
   i. Identify the data and its format that needs to be processed. In our case we have a log file(**apache-log.log**).
   ii. Determine if the data can be made parallel and solved using Map Reduce. For ex : you need to analyze whether the data is WORM(Write Once Read Many) in nature.

b. **Design and Implementation a solution :**

i. Once we have identified the relevant data , now build your own logic by writing 3 files:
    1. **Mapper File** → Strictly write only and only segregation logic
    2. **Reducer File** → Only write custom Aggregation logic like count,sum,
    3. **Job Config File** → To let map reduce framework know which is our custom mapper and custom reducer code , with few more imp config needs to be written here.

c. Compile and Package the code:

i. Once all files are created, compile the source code with hadoop core and package the code

d. Configure the Application:

i. Configure the job as to the number of mappers and reducer tasks and to the number of input and output streams.

ii. We can set it through dynamically while executing our mapper or reducer files or can keep it in the config file.

e. Load the data:

i. Finally load the data in HDFS via command and copy the path handy.

f. Launch the job and Monitor:

i. Start the job as:

```
>>>> $HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/mapred/contrib/streaming/hadoop-streaming*.j
ar \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/hduser/apache-log.log -output /user/hduser/output
```

Where

Mapper.py,reducer.py → these are our logic

/user/hduser/input.txt → from previous step, location of file which we need for execution.

/user/hduser/output → **HDFS** file location where we want to see our output file

What are the roles and responsibilities of a developer in Map Reduce?

Developer	Framework Provides
Setting up the job	Distributing the job
Specifying the input/output location	Running the map operation
Write Mapper , Reducer Logic	Performing shuffling and sorting
Ensuring correct format and location of input	Reducing phases
	Placing the output
	Informing the user of job completion status

What are different data types provided by hadoop?

As in our flipkart code we have used various data types. Below are the details for them :

Data Types	Usage

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

Interestingly, if you have observed that mostly data types are similar to Java/Python but with them there is Writable attached to them.

What is the reason for this?

Answer lies in **Serialization**!!!!

So what is **Serialization** and why do we need it ?

To transfer the data between the nodes across the network data needs to be compressed which means instead of sending **100 mb of data** over the network.

Can't we compress it to **30/40mb** and then send it across ?

This is why we need data to be serialized .

"The process of translating data structures or objects state into binary or textual form, that process is what we call **Serialization**"

"The process of translating binary or textual form back to object state, is known as **Deserialization**"

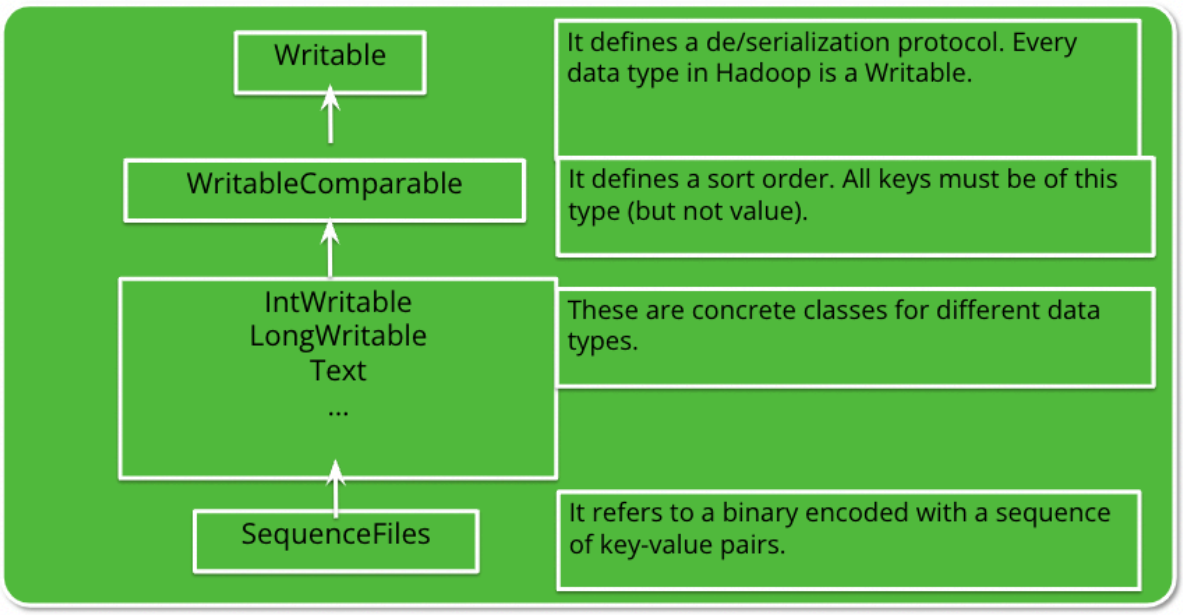
For Instructor : Serialize to translate data structures into a stream of data. Transmit this stream of data over the network or store it in DB regardless of the system architecture.

To implement **serde**(**serialization and deserialization**)Hadoop uses its own **Writable Interface** which provides methods for serialization and deserialization with methods as:

```
public void readFields(DataInput in);
public void write(DataOutput out);
```

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()
```

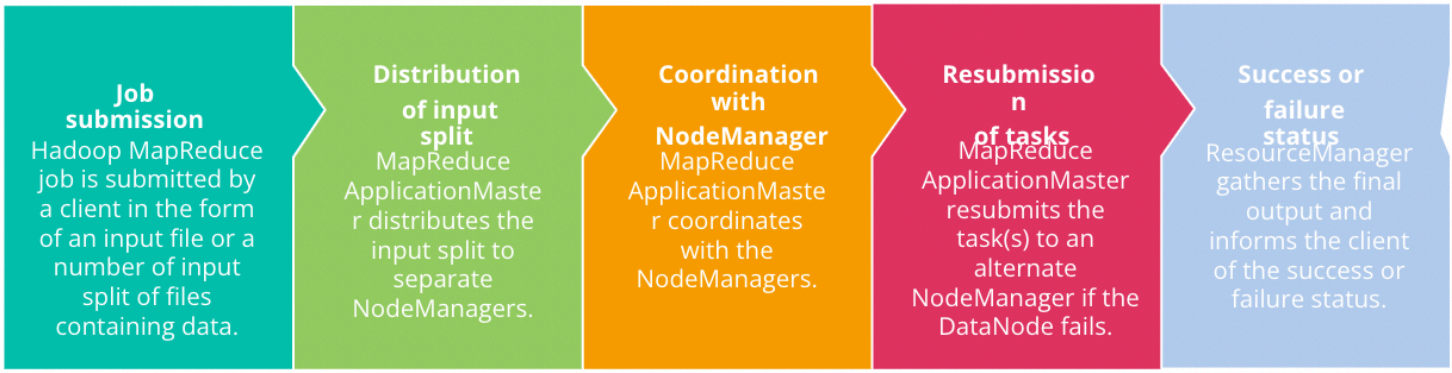


How Does Map Reduce get resources from YARN?

Map Reduce requires resources to execute itself, so to understand this let's see what different tasks performed.

a. **Map Process** : An initial ingestion and transformation step where initial input records are processed in parallel.
b. **Reduce Process** : An aggregation or summarization step in which all associated records are processed together
c. **Node Manager :** Keeps track of individual map tasks and can run in parallel
d. **Application Manager :** Keeps track of a MapReduce job

And workflow of these tasks works in below mentioned flow :



How to optimize the Map Reduce job ?

Plain vanilla MR code is good for the initial level of job but when one gets into industry , expectations are quite high and focus is on how to write more conducive MR jobs.

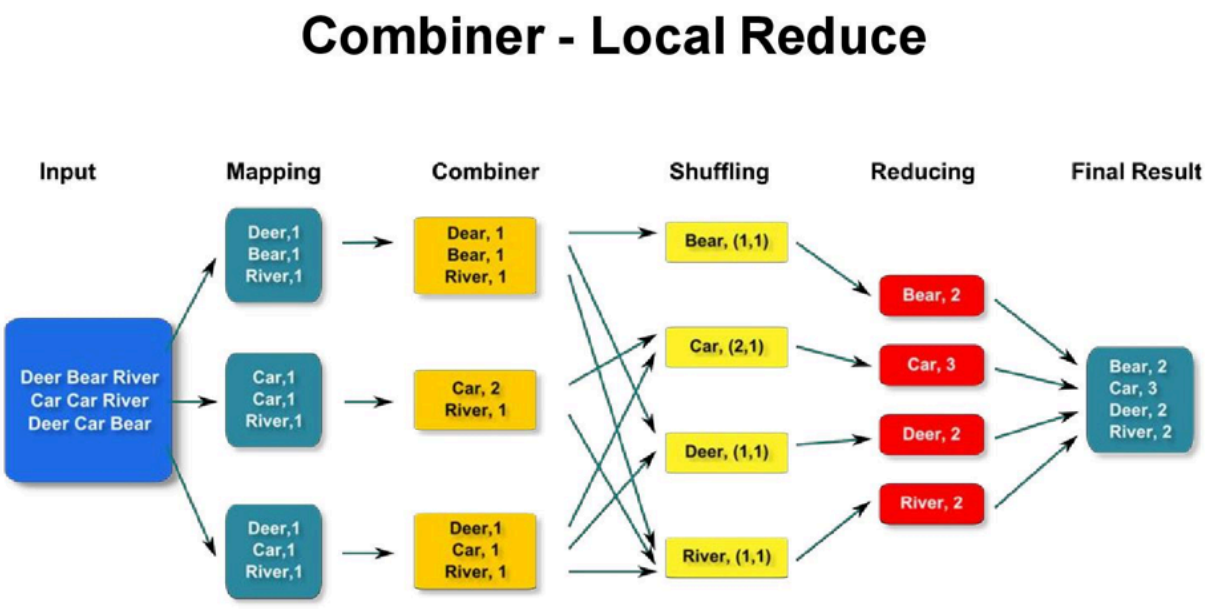
Below are recommended optimization strategies:

a. **Usage of Distributed Cache :**

i. Hadoop's MapReduce framework provides the facility to cache *small to moderate* **read-only files** such as text files, zip files, jar files etc. and broadcast them to all the Datanodes(worker-nodes) where the MapReduce job is running.
ii. Each Datanode gets a copy of the file*(local-copy)* which is sent through **Distributed Cache**. When the job is finished these files are deleted from the DataNodes.

b. **Take help from Combiners** :
   i. An optional class that is used at map reduce side to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.
   ii. Same code for reducer can be used to specify combiner.
   iii. Sample code will be :

Job.setCombinerClass(ReducerClass.py)



c. **Do some Memory tuning :**
   - In Hadoop MapReduce performance tuning the most basic and common rule for memory tuning is: using as much memory as possible without triggering swapping.
   - The role memory parameter is mapred.child.java.opts which can be inserted into your config file.

d. **Tuning of Mappers/Reducers :**

i. Hadoop breaks the file into smaller chunks when dealing with large datasets, so that the mapper can run it in parallel.
ii. Nevertheless, initializing a new mapper job usually takes a few seconds to minimize which is also an overhead.
iii. The suggestions for the same are as follows: Reuse jvm task Target to map tasks that run 1–3 minutes each.
iv. For this, if the average running time of the mapper is less than one minute, increase the **mapred.min.split.size**, assign less mappers in the slot, thus reducing the initializing overhead mapper.