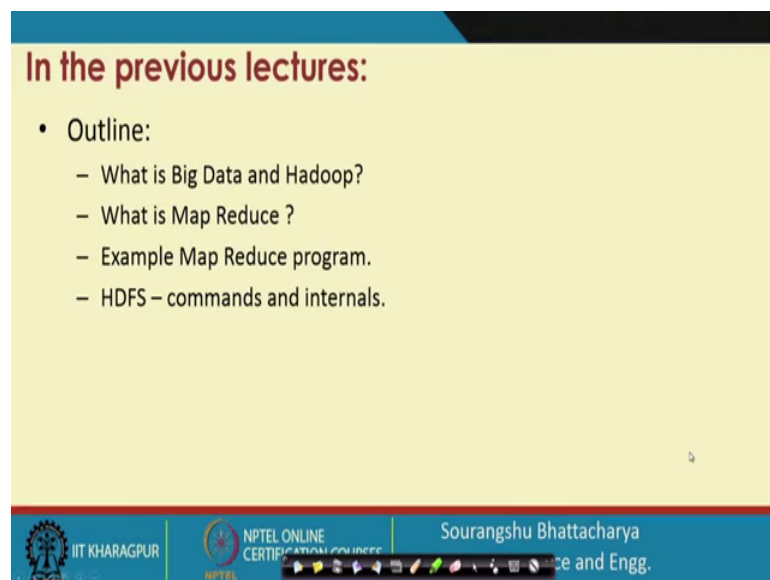


Scalable Data Science
Prof. Sourangshu Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 18b
Hadoop System (Contd.)

Hello everyone, welcome to the second part of 18th lecture of the NPTEL course on Scalable Data Science. I am Professor Sourangshu Bhattacharya from IIT from computer science department IIT, Kharagpur. So, today's lecture is going to be on Hadoop System.

(Refer Slide Time: 00:40)



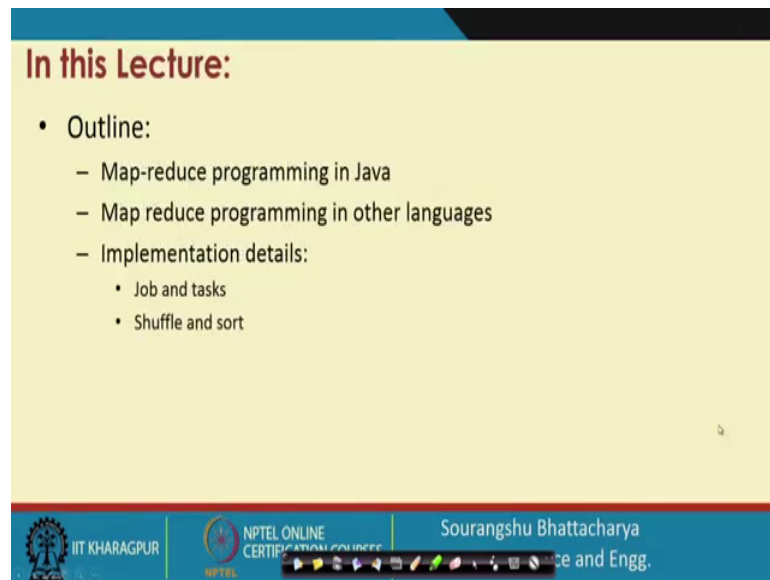
In the previous lectures:

- Outline:
 - What is Big Data and Hadoop?
 - What is Map Reduce ?
 - Example Map Reduce program.
 - HDFS – commands and internals.

The slide footer contains the IIT Kharagpur logo, NPTEL Online Certification Course logo, and the name Sourangshu Bhattacharya, Department of Computer Science and Engineering.

So, in the last lecture we have seen what is Big Data and Hadoop. We have seen what is Map Reduce? We have seen an Example Map Reduce Program and we have seen the HDFS, commands and also the internals of HDFS.

(Refer Slide Time: 01:01)



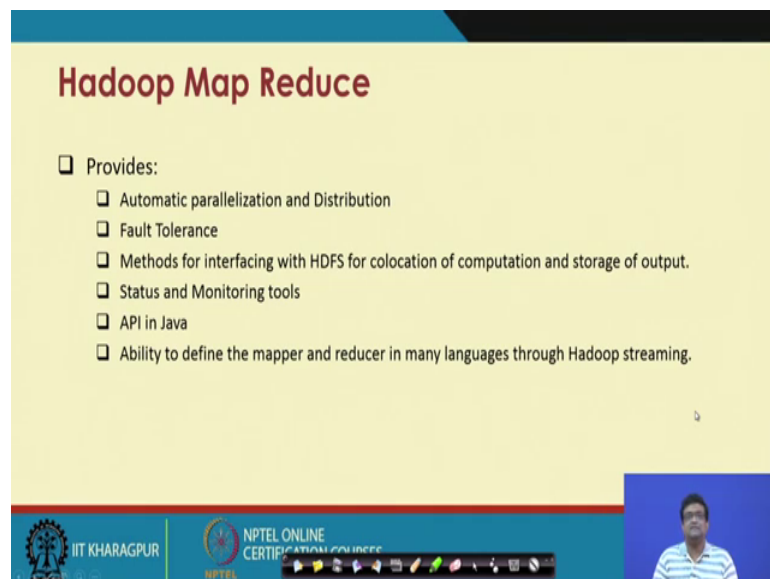
In this Lecture:

- Outline:
 - Map-reduce programming in Java
 - Map reduce programming in other languages
 - Implementation details:
 - Job and tasks
 - Shuffle and sort

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSE | Sourangshu Bhattacharya | Computer Science and Engg.

In this lecture, we are going to see an actual map reduce program in Java and also we are going to see how you can write map reduce program in other languages, and then we will start looking at implementation details of a map reduce task. So, we will look at the concepts of jobs and tasks and we will see the implementation of shuffle and sort.

(Refer Slide Time: 01:33)



Hadoop Map Reduce

- Provides:
 - Automatic parallelization and Distribution
 - Fault Tolerance
 - Methods for interfacing with HDFS for colocation of computation and storage of output.
 - Status and Monitoring tools
 - API in Java
 - Ability to define the mapper and reducer in many languages through Hadoop streaming.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSE | Sourangshu Bhattacharya | Computer Science and Engg.

So, as discussed earlier the Hadoop Map Reduce framework provides for a distributed or for an environment for big data computation which includes facilities related to automatic parallelization and distribution.

So, whatever program you whatever programs you write they are automatically parallelized given the current cluster and the available systems and distributed to various nodes in the current cluster.

It provides fault tolerance, so if a particular node fails the framework automatically provides fault tolerance that is it executes the portions of the job which was getting executed on the failing node on to other nodes and this is at no additional cost. So, the user does not have to do anything extra.

Then it provides methods for interfacing with the Hadoop distributed file system and in particular it provides a facility for collocation of computation with storage as we have discussed in the HDFS lecture. Then it provides in certain cases status monitoring tools.

So, basically if there is a heavily loaded cluster where multiple users are submitting multiple jobs and various jobs may be queued or some certain tasks of certain jobs may be taking more time. So, one can see the status of the jobs that the person has submitted using these tools.

So, Hadoop Map Reduce also provides an API in java and it also provides a way to write map reduce programs especially the mapper function and the reducer function. In other languages through a protocol which is called the Hadoop streaming protocol. So, we will see this in today's lecture.

(Refer Slide Time: 04:01)

```
Wordcount program

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATE COURSE

So, this is the map reduce so first I am going to describe the map reduce program for the word count problem which we have described earlier. So, the word count problem is basically the problem where you are given a large file of unstructured text and you have to count the number of occurrences of each word in this file. So, the as you can see this screen or this part of the program shows the in the libraries which are required to write a Hadoop map reduce program. So, the first two are simple java library. So, one is for exception IO exception and the second one is for Tokenizing String. So, string handling it is a utility file.

But the second portion of the libraries are the Hadoop specific libraries which are required to write the map reduce program. So, the first library is the or the first class that is taken is the configuration class which basically reads the cluster configuration for a given Hadoop cluster and it provides this configuration.

So, it creates an object of type configuration and provides this to the job object. Then we see the path the path class which basically creates HDFS path from where the input or the output will be read Hadoop

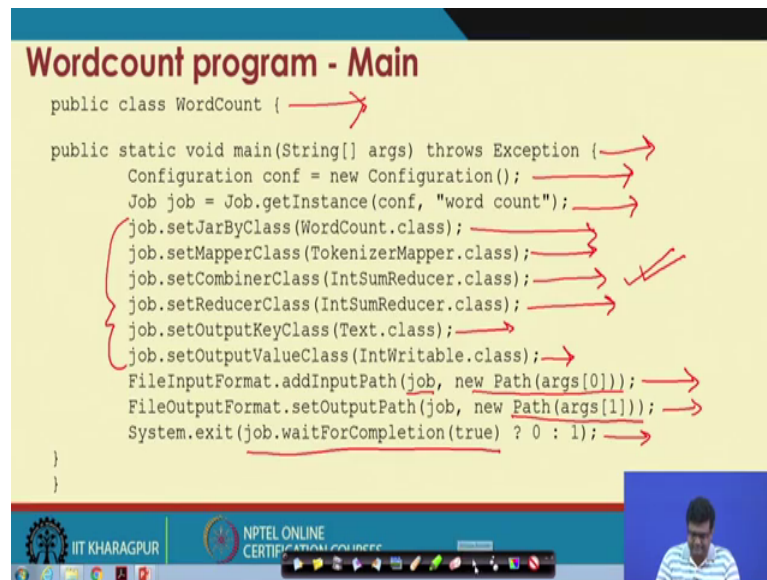
The third is the Hadoop specific types for example, in this case it is the IntWritable type that we will be using which is basically a serialized version of the integer type and the second one is the Text type which is again a serialized version of the string type which is in java.

Then we have the job class which is the main class that one needs because this defines so every map reduce program needs to have an object at least one object of type job and this is basically the main object which controls a map reduce job ok. So, any map reduce job is controlled through this job object.

Then there is the mapper class which basically is the base class for any mapper function that the user may want to define and similarly the reducer class is the base class for the reducer function that the user may want to define.

And finally, there are the input and output file format classes these are basically the utility classes which allow Hadoop to read files from HDFS and convert them into the mapper input record and take the mapper the reducer output wave card and convert it back into the HDFS output file format.

(Refer Slide Time: 08:01)



```
public class WordCount {  
  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        Job job = Job.getInstance(conf, "word count");  
        job.setJarByClass(WordCount.class);  
        job.setMapperClass(TokenizerMapper.class);  
        job.setCombinerClass(IntSumReducer.class);  
        job.setReducerClass(IntSumReducer.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
        System.exit(job.waitForCompletion(true) ? 0 : 1);  
    }  
}
```

Now, we describe the next portion of the program which is the main program. So, as you can see the far so this is. So, for those of you who are familiar with java, you know that this is the main class called word count. So, this is the class inside which your word count program will reside ok. So, this is a java class inside which your whole map reduce word count program will recite and the main function is the starting point of execution for this class.

So, first statement creates the configuration object which basically reads the configuration for the present cluster and then the next statements creates the job object where you pass the configuration and the title stream for the job.

Then the next command sets the main class of this job basically all these commands manipulate the job object ok. So, you said the main class for the job, then you send the mapper class you set the mapper class for the job, you set the combiner class for the job. So, we will discuss what is the combiner class at a later time, for the time being you can think that it is a optimized version of the reducer class then you set the reducer class for the job, and then you set the output key class and output value class for the mapper for the for the mapper.

So, so in this case as you can see the output key is of type text because the mapper output key is a string and output value is of type IntWritable. And then finally, you can set you

can so. So, you add to the input path of a particular job the path which is which is going to be provided as an argument when the main program is run.

So, this is the input path and the second argument for the main program to be run is going to be the output path. So, you set these two to the appropriate attribute of the job object and then finally, you so this job dot wait for completion actually starts execution of the job and you wait for this and it returns when complete is true with a 0 or a 1 flag and which at which point in time you exit the system.

(Refer Slide Time: 11:25)

```
public static class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

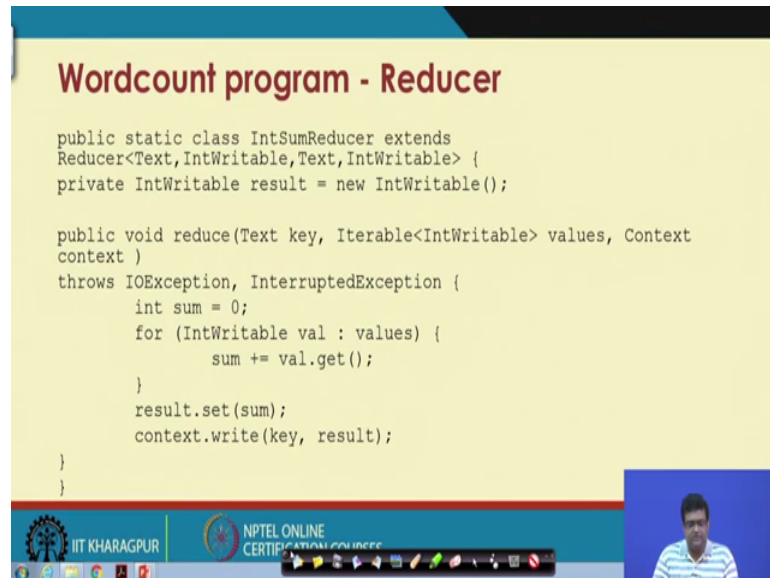
    public void map(Object key, Text value, Context context )
    throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken()); context.write(word, one);
        }
    }
}
```

Now, this is the mapper class so you so, we have defined we know what all libraries we need and we know what the main function is now we define the mapper and the reducer class and then we are done. So, this program defines the mapper class. So, the mapper class has.

So, so in this case the name of the mapper class is Tokenizer Mapper as you can see and it has to extend of course, the Mapper class which we have which we have imported earlier and then it defines a local variable which is the word the main logic here is that the so you have in the map object of the of the mapper of the mapper record the input key the input value and the context in which you write the output key and the output value ok.

So, so now, you define a string Tokenizer and then you pass the value of the input key which is the Mapper input record and it tokenises the input key into words and then you iterate through the tokens which are the words in the input value of the mapper record and you write to the context the word and 1 is nothing, but the, So, the Int Writable constant 1, ok. So, basically you are writing something like word comma 1.

(Refer Slide Time: 13:52)



```
public static class IntSumReducer extends
Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context
context )
throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
```

Finally you define the reducer class which is a very similar to the mapper class. So, as previously it has to extend the reducer class. So, in this case the name of the reducer class is Int some reducer so it is adding or it is calculating the sum of the integers.

Now, in this case again one has to define the reduce function the reduce function takes as input the key and the value. So, these are the input keys and input values and of course, the context object in which you have to write the output and then.

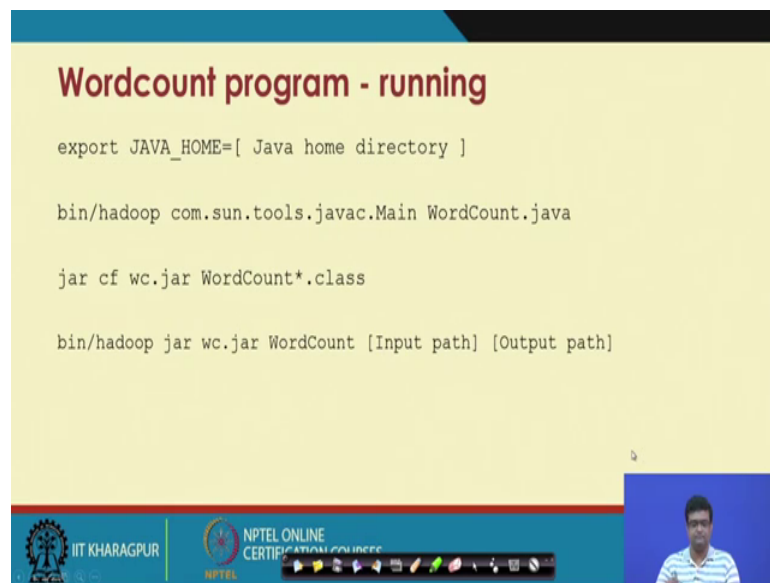
So, so actually in this case it takes as input. So, so observe that actually it takes as input and Iterable object of type IntWritable values ok. So, actually the reducer input value is not a single value, but it is an Iterator over value. So, it is a list of values ok.

So, now in the reducer function you first define sum to 0 you first initialize sum to 0 and then you Iterate over all the values and you add all the values to the sum and then write the result in the in the result variable ok. So, you define the result variable earlier so yeah. So, you define the result variable here and then you write to the context the key

which is a single value, but the result is the sum over all the input values that you have received in this values Iterator.

So, that concludes the description of the word count program in java in map reduce ok. So, as you can see the program is very lengthy and quite complicated to write; however, one can actually understand the internal details of functioning of a map reduce system in this manner.

(Refer Slide Time: 16:42)



```
Wordcount program - running

export JAVA_HOME=[ Java home directory ]

bin/hadoop com.sun.tools.javac.Main WordCount.java

jar cf wc.jar WordCount*.class

bin/hadoop jar wc.jar WordCount [Input path] [Output path]
```

The slide features a yellow background with a blue header and footer. The footer contains the IIT KHARAGPUR logo on the left and the NPTEL ONLINE CERTIFICATION COURSES logo on the right. A small video inset of a presenter is visible in the bottom right corner.

Now so this is taken from the Apache Hadoop tutorial. So, you can then once you have once you have the Hadoop distribution downloaded from apache Hadoop site you can just follow these steps. So, you have to set the java home variable to the java home directory.

And then you have to compile this word count or java program that I have just described and then you have to create a jar of all the classes. So, it will have the word count class, the mapper class, and the reducer class, in this case and then you can just run using this command Hadoop jar and then you provide the jar file and then you provide the name of the main class or the class where the main function is there and where it should start executing and then you provide all the user defined arguments.

In this case remember you have to provide the input path and the output path. So, this is the complete description of how you can run a map reduce program.

(Refer Slide Time: 18:05)

Wordcount in python

Mapper.py

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

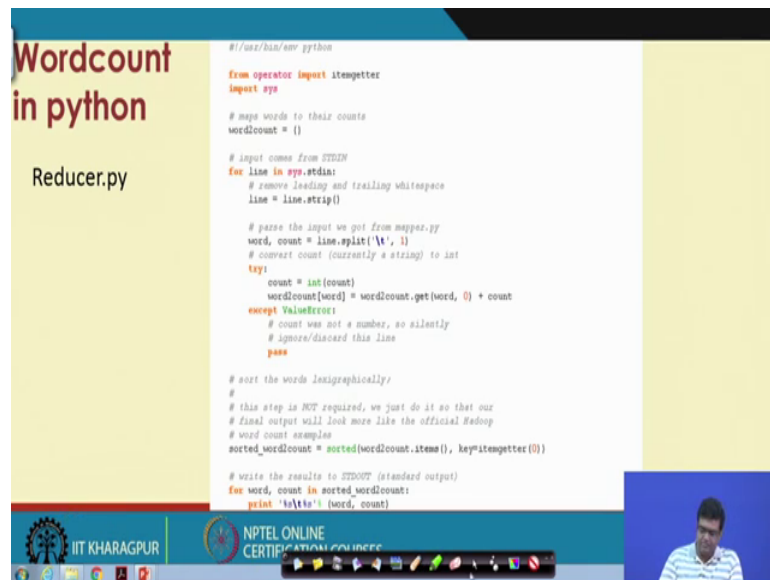
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, I will describe a much simpler way to write the same program. So, here I have taken python as the language, but this particular program can be written in any language and one just has to follow certain conventions while writing this program. So,. So, let us see what.

So, here what I am displaying is the same mapper the word count mapper, but now in python. So, let us see what the mapper does. So, the mapper takes all the lines in the stdin ok. So, the mapper is a process or it is a program it is a python program and also a process running inside the map reduce system and now this is take reading from its own stdin and then these are standard processing.

So, it is removing the whitespace character and then it is splitting the line into words ok. So, this word is a list of words that you get when you split the line and then for each word in words it is writing this word comma 1 and this is the formatting string so it is writing the word followed by a tab which is very important and then followed by the string 1 ok. So, it is writing this string records of this form ok. So, now let us see what the reducer looks like.

(Refer Slide Time: 20:14)



```
#!/usr/bin/env python
from operator import itemgetter
import sys

# map words to their counts
wordcount = {}

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
        wordcount[word] = wordcount.get(word, 0) + count
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        pass

# sort the words lexicographically
#
# this step is NOT required, we just do it so that our
# final output will look more like the official Redoop
# word count example
sorted_wordcount = sorted(wordcount.items(), key=itemgetter(0))

# write the results to STDOUT (standard output)
for word, count in sorted_wordcount:
    print '%s\t%s' % (word, count)
```

So, the reducer looks something like this that. So, so you have you import the Itemgetter and then again in the reducer you read the line from every line and you strip it off trailing whitespace.

Now, you see you just split the line using the delimited tab ok. So, you split using the delimited tab and then you assign the first portion of this and you create you only split it once and you assign the first portion to the word and the last portion to the count.

Now remember your input was of form word comma one because this is what you wrote sorry the comma is not there. So, your input was of the form word and then there was a tab here. So, there was a tab here. So, there was word and then there was a tab and then there was 1 ok. So,

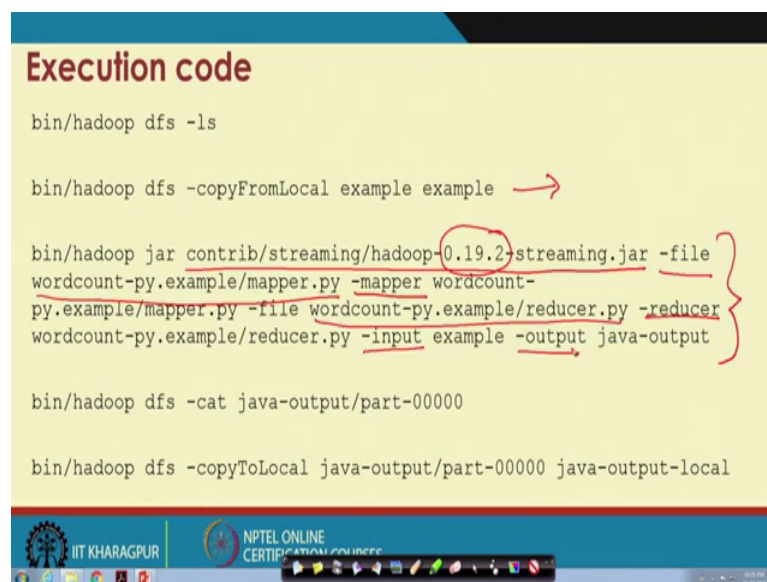
So, in this case you are getting the word and the count using this command ok. Now you try to convert the count into the number count and then what you do you maintain a list of called what to count and then you add the count of the word to count to this particular.

So, you basically increment the count of that particular word by the count that you have received and once you have got this you sort the word to come and then you print the sorted word to count ok. Sorting here is not absolutely essential, but if you sort then your final output will come sorted ok. At least each portion for each reducer the output will come sorted ok.

Now, what is happening here so the main key thing is this tab. So, what happens is that when in the previous if you see the previous mapper ok. When you write when you write it in the tab delimited format the streaming API of the Hadoop understands that the portion or the string before the mapper or before the tab is the mapper output key and hence it is also the reducer input key and hence it sends all the records having the same mapper output key to the same reducer. So, this is what is ensured by the Hadoop streaming framework.

The other important thing is that it ensures that each record output by the word count is treated as 1 mapper output record while also

(Refer Slide Time: 24:29)



```
Execution code

bin/hadoop dfs -ls

bin/hadoop dfs -copyFromLocal example example →

bin/hadoop jar contrib/streaming/hadoop-0.19.2-streaming.jar -file
wordcount-py.example/mapper.py -mapper wordcount-
py.example/mapper.py -file wordcount-py.example/reducer.py -reducer
wordcount-py.example/reducer.py -input example -output java-output

bin/hadoop dfs -cat java-output/part-00000

bin/hadoop dfs -copyToLocal java-output/part-00000 java-output-local
```

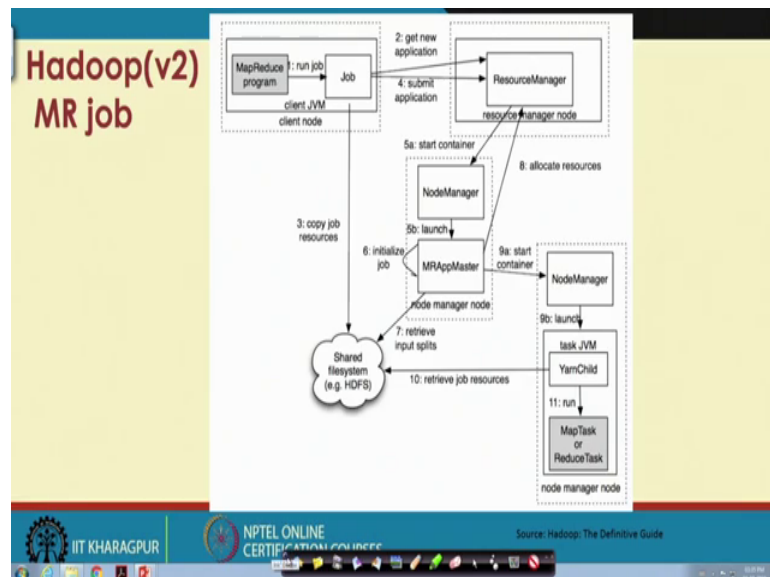
And each record inputted to the reducer is counted as 1 mapper input record ok. So, in this case the mapper input records may arrive in jumbled manner. Hence you have to maintain an array where you maintain all the counts of all the words and you increment as and when you get a particular word.

So, finally, this is the command to execute the streaming job that you have defined. So, basically the you just execute the Hadoop jar command. So, this command is actually copying the data from local file system to the HDFS. Now here you are just executing the jar Hadoop streaming dot jar this is the version of the streaming and then you give in minus file the mapper file and also in minus file the reducer file.

So, these files must be shipped along with the jar to all the nodes in which this streaming program is being executed hence you provide it in the file command and then you with the minus mapper command you specify which is the program to be executed as the mapper and in the minus reducer command you provide the program to be executed as the reducer.

Then you provide the input and the output ok. So, in this, So, if you provide all these things then the streaming framework is able to execute the map reduce job ok. Now, since we have discussed.

(Refer Slide Time: 27:09)



So, this is a detailed view of what a Hadoop map reduce job looks like or what are the components and how the control flows in a Hadoop map reduce job ok. So, we start with a map reduce program that we have already shown you and then the map reduce program defines a job one map reduce program can define also multiple jobs which will get executed as multiple Hadoop jobs.

Now, each job first creates an application or it contacts the resource manager which in this case is either yard or it is if it is just a local job then there is no resource manager needed, but in the general case it contacts the resource manager and creates the application and also it provides the input files which it fetches from the HDFS and then it provides them to the to the resource manager and at this point in time. It submits the application to be executed by the resource manager ok.

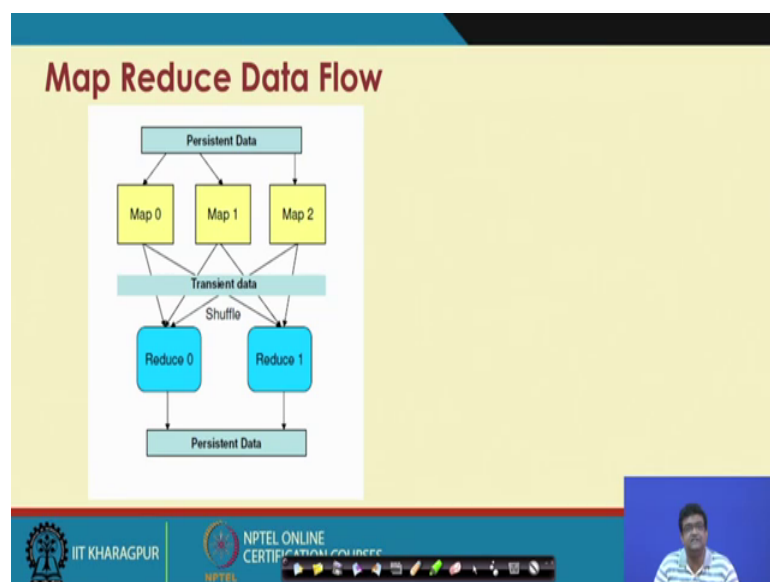
Now, the resource manager contacts the node manager node manager is a program which or is a yeah it is a software that runs on each of the nodes of the Hadoop cluster and resource manager contacts the node manager and starts what is called a container.

So, container is basically a unit of execution in which either a mapper task or a reducer task will execute and the first task that the node manager does is it creates what is called the app master ok. So, the map reduce app master also called the master, which is basically responsible for coordinating between the all the map tasks and the reduce tasks ok.

The app master then contacts the node manager it could be the same node manager or other node managers and it already has the input splits or the location of the files where the input data is there. So, it can now decide the and contact node managers which are local to the data and the node manager finally, runs child task.

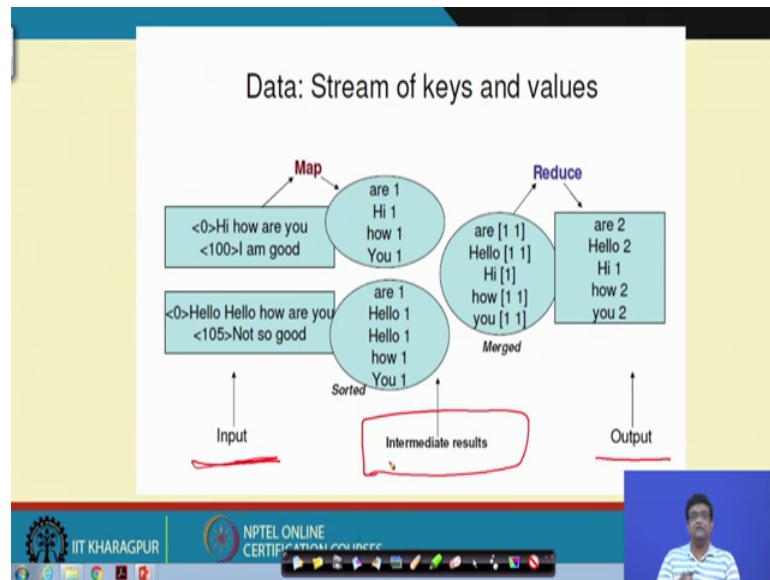
So, this is a separate process run in the same node by the node manager and as the cases the it is either. So, initially the map tasks will be run and then once the map tasks are complete the reduce tasks will be run and this will whole thing will be coordinated by the app master which we will discuss next or later.

(Refer Slide Time: 30:53)



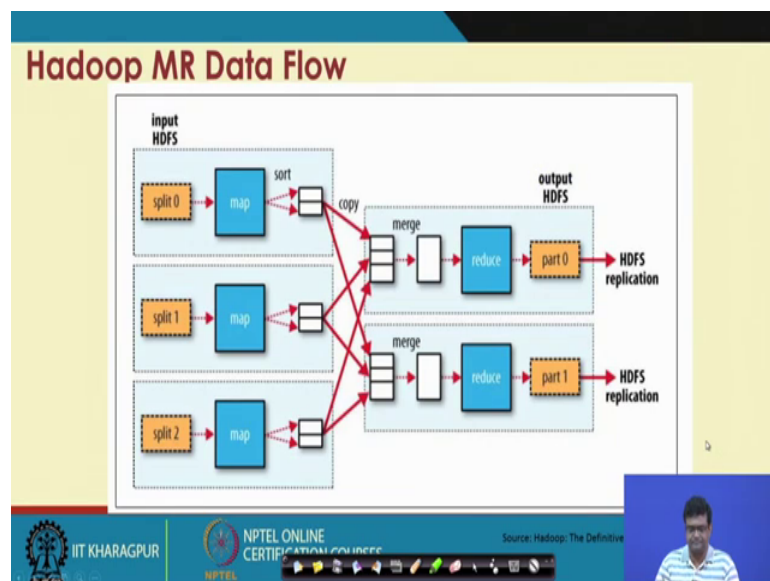
So, as we have seen this is the broad map reduce data flow. So, there is the persistent data which is on a HDFS then we process using mappers and create some transient data which is further processed by reducers to create some persistent data.

(Refer Slide Time: 31:17)



And this is a run through of the program which is the word count program and this is how it works.

(Refer Slide Time: 31:31)



Now, we discuss so as you can see that as you can see that there is so there is. So, so the main problem with this map reduce system is that there is a lot of intermediate results. So, so as you can understand here that this is the input which is written on to a HDFS ok.

This is the output which is also written to an HDFS and these are the intermediate results which go from a mapper to which go from mapper to the reducer.

The problem with big data processing is that there is a huge amount of intermediate results which go from the mapper to the reducer. So, how do we process this huge number of intermediate results which go from the mapper to the reducer ok.

So, in the next lecture we will start discussing the details of Implementations of these Data Processing records which are responsible for making the System Scalable.

Thank you.