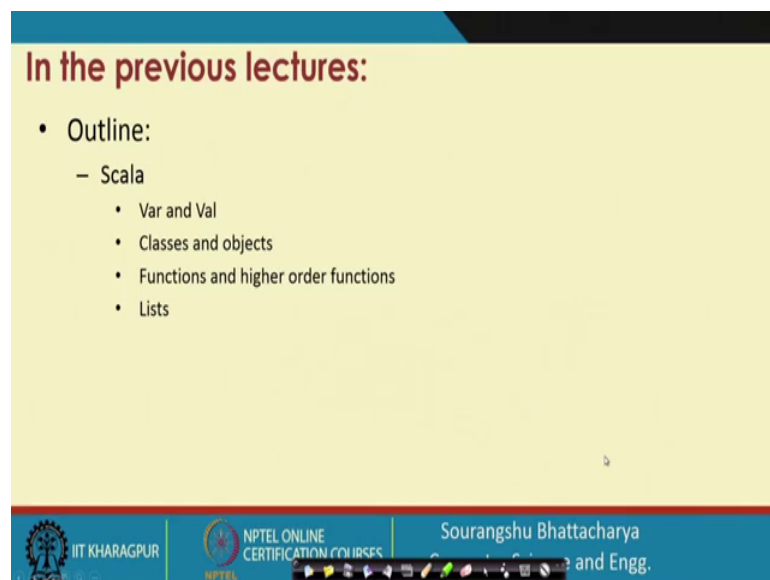


Scalable Data Science
Prof. Sourangshu Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 19b
Spark (Contd.)

Hello everyone. Welcome to the 19th lecture of NPTEL course on Scalable Data Science. I am Prof. Sourangshu Bhattacharya from Computer Science Department in IIT, Kharagpur. This is the 2nd lecture on spark.

(Refer Slide Time: 00:30)



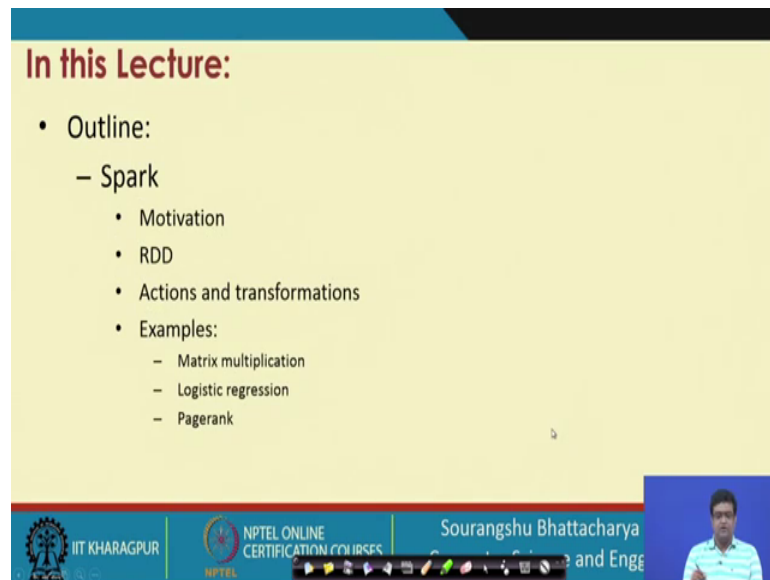
In the previous lectures:

- Outline:
 - Scala
 - Var and Val
 - Classes and objects
 - Functions and higher order functions
 - Lists

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

So, in the last lecture, we have seen such as Scala and we have seen its immutable verses mutable variables, which are Vars and Val. And then we have seen functions which are objects, and also higher order functions and we have seen lists.

(Refer Slide Time: 00:53)



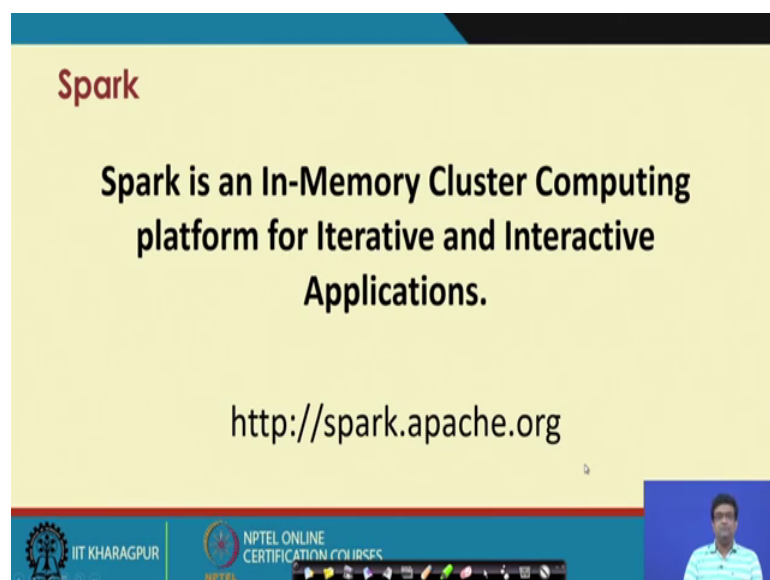
In this Lecture:

- Outline:
 - Spark
 - Motivation
 - RDD
 - Actions and transformations
 - Examples:
 - Matrix multiplication
 - Logistic regression
 - Pagerank

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Sourangshu Bhattacharya | NPTEL

In this lecture, we are going to see spark. So, we will see the motivation, which we have already seen. We will briefly go through the motivation once again for the spark. Then we will see what are RDDs or Resilient Distributed Datasets. And then we shall we shall see some see what are actions and transformations. And then finally, we will see some programming examples using spark.

(Refer Slide Time: 01:24)



Spark

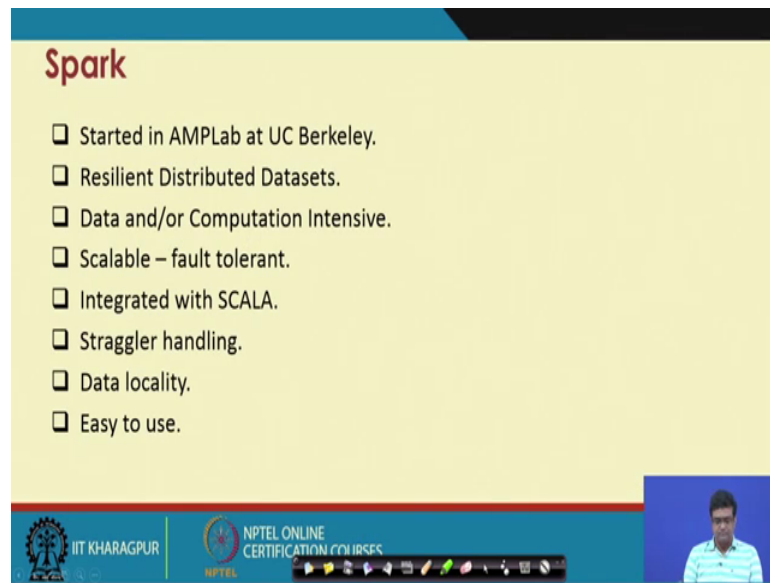
Spark is an In-Memory Cluster Computing platform for Iterative and Interactive Applications.

<http://spark.apache.org>

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, as already mentioned spark is a distributed in-memory cluster computing platform for iterative and interactive applications.

(Refer Slide Time: 01:35)



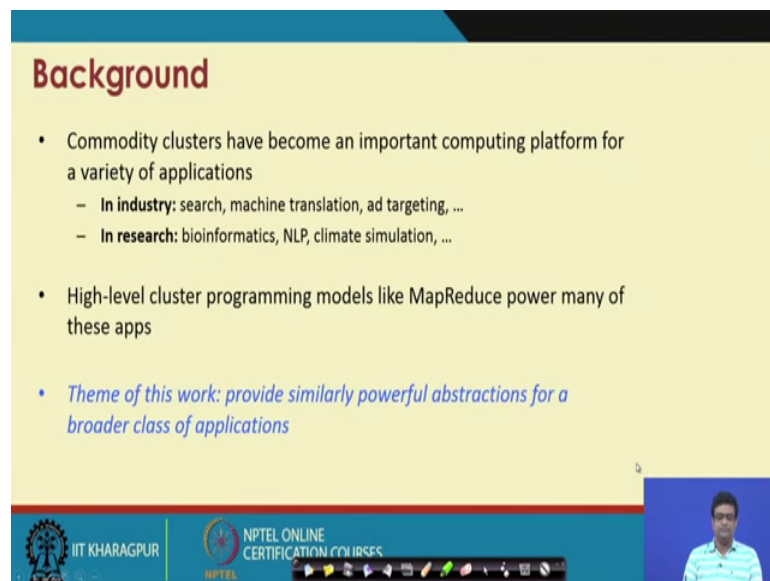
Spark

- ❑ Started in AMPLab at UC Berkeley.
- ❑ Resilient Distributed Datasets.
- ❑ Data and/or Computation Intensive.
- ❑ Scalable – fault tolerant.
- ❑ Integrated with SCALA.
- ❑ Straggler handling.
- ❑ Data locality.
- ❑ Easy to use.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And we have discussed about the origins and properties of spark.

(Refer Slide Time: 01:42)



Background

- Commodity clusters have become an important computing platform for a variety of applications
 - In **industry**: search, machine translation, ad targeting, ...
 - In **research**: bioinformatics, NLP, climate simulation, ...
- High-level cluster programming models like MapReduce power many of these apps
- *Theme of this work: provide similarly powerful abstractions for a broader class of applications*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

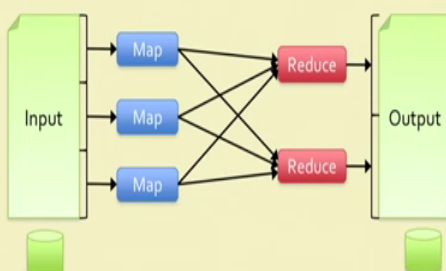
And we have seen the background also.

(Refer Slide Time: 01:46)

Motivation

Current popular programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:



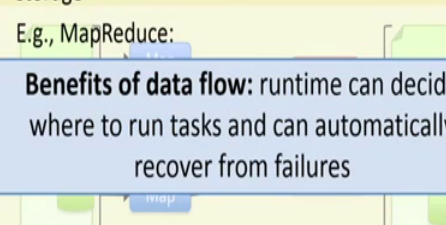
The diagram illustrates the MapReduce process. On the left, a green folder icon labeled 'Input' is connected to three blue boxes labeled 'Map'. Each 'Map' box is connected to two red boxes labeled 'Reduce'. The 'Reduce' boxes are connected to a green folder icon labeled 'Output' on the right. Below the input and output folders are small green cylinder icons representing storage. The slide footer includes the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

(Refer Slide Time: 01:59)

Motivation

- Current popular programming models for clusters transform data flowing from stable storage to stable storage
- E.g., MapReduce:

Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures



The diagram shows a simplified MapReduce flow with 'Input' and 'Output' folders and 'Map' and 'Reduce' boxes. A blue callout box is overlaid on the diagram, containing the text: 'Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures'. The slide footer includes the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

And we have also seen that what, so what is the benefit of this map reduced programming model which is to say that the runtime can decide where to run the tasks and also it can automatically recover from the failures ok.

(Refer Slide Time: 02:18)

Motivation

- Acyclic data flow is a powerful abstraction, but is not efficient for applications that repeatedly reuse a *working set* of data:
 - **Iterative** algorithms (many in machine learning)
 - **Interactive** data mining tools (R, Excel, Python)
- Spark makes working sets a first-class concept to efficiently support these apps

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And we have also seen that for an iterative computations and interactive computations. This kind of this kind of the programming model is inefficient, and also sometimes more difficult to run programs in ok. So, so spark, so one of the main properties of spark is that it makes this working sets a first class concept to efficiently support this kind of computations ok.

(Refer Slide Time: 02:54)

Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- Retain the attractive properties of MapReduce:
 - Fault tolerance (for crashes & stragglers) →
 - Data locality →
 - Scalability →

Solution: augment data flow model with "resilient distributed datasets" (RDDs)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the goal here is to provide distributed memory abstractions for clusters to support apps with working sets ok. So, working sets are basically the distributed datasets on

which you do various computations ok. At the same time what you want to do is you want to retain these attractive properties of fault tolerance, data locality, and scalability ok. So, you and also additionally you want to also retain the property of ease of use ok. So you do not want to write program for you know distributing, you should be able to automatically schedule the tasks as they come. And so, the solution is to augment the data flow model with resilient distributed datasets. So, so what are resilient distributed datasets.

(Refer Slide Time: 04:13)

Resilient Distributed Datasets

- ❑ Immutable distributed SCALA collections.
 - ❑ Array, List, Map, Set, etc.
- ❑ Transformations on RDDs create new RDDs. }
 - ❑ Map, ReduceByKey, Filter, Join, etc.
- ❑ Actions on RDD return values. } →
 - ❑ Reduce, collect, count, take, etc.
- ❑ Seamlessly integrated into a SCALA program. →
- ❑ RDDs are materialized when needed. →
- ❑ RDDs are cached to disk – graceful degradation. →
- ❑ Spark framework re-computes lost splits of RDDs. →

The slide also features the IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES logos at the bottom, and a small video inset of a presenter in the bottom right corner.

So, resilient distributed datasets are immutable distributed SCALA collections. So, what are SCALA collections, SCALA collections are something like arrays, lists, maps, sets, etcetera. So, these are basically collection of objects either sequential collection like in arrays or lists or sometimes non-sequential collections like sets or maps, where there is no particular order, but the elements are together or a collection of elements are together help in this data structures. And importantly these data structures are immutable that is once you initialize them, you cannot change their values ok. So, from the programmer's point of view, this is what a resilient distributed datasets is ok.

Now, moreover the programmer can do two types of operations on resilient distributed datasets ok. So, these resilient distributed datasets are created and managed through these kind of bulk operations such as map operation, which we have already seen which is to perform a particular function on a particular a particular on all the elements of a resilient

distributed datasets. An operation can be something like reduce by key, which is an associative operation on set of on a set of elements in an RDD ok. So, we will see how to use reduce by key in sometime time.

Then one can have operations like filter, which basically again operates on every element of an RDD or every record of an RDD. And we can have operations like join, which join the records in two different RDDs based on the key the keys of those records ok. So, the thing about transformations is that all transformations when applied on one or more RDDs create new RDDs ok. So, this is the property of a transformation.

On the other hand, there is another class of actions like reduce, collect, etcetera. So, reduce action for example, takes in a function which operates on elements of an RDD, and then finally, produces a single value or a single object. Similarly, collect, produces, local object or a local list from an RDD. So, if you call collect on an RDD, it will take all the elements of that RDD, and produce a local SCALA array or a SCALA object. Similarly, count we will just count the number of records in an RDD and so on and so forth. The common thing about these actions is that they operate on an RDD, and they report values or they return values, so they do not create any new RDDs. So, this is a major difference ok.

So, as you can as you will see these are seamlessly integrated into the SCALA program. Now, another important thing is that RDDs are conceptual objects. So, these collections in spark are for the understanding of the programmer, but in reality they may not exist. So, they are exist or they exist or they are materialized, only when needed ok. So, just because you define an RDD does not mean that the RDD is computed. And then, so RDDs can be fairly large ok, so you can have very very large distributed RDDs. So, RDDs have the capability, so it may not fit into the main memory of the clusters. So, in that case, there is the facility of RDDs being cached on to disk. So, portions of the RDD may be return on to disk or even sometimes they can be cached on to memory ok.

Another thing is that since RDDs are these distributed collections. And the basically so the RDD data is distributed into different splits on different machines. So, fault tolerance basically means that if a certain portion of an RDD also called a split of an RDD is lost, let us say because you are doing big data computation and thousands of server, and one

of the server is lost ok. Then the spark framework has the capability to re- compute only those lost splits without you know computing the other splits ok.

(Refer Slide Time: 10:24)

The slide is titled "RDDs in More Detail" in red text. It contains four bullet points, each with a red checkmark icon. The first bullet point is "An RDD is an immutable, partitioned, logical collection of records", with red arrows pointing to "immutable", "partitioned", and "logical". Below it is a sub-bullet: "Need not be materialized, but rather contains information to rebuild a dataset from stable storage". The second bullet point is "Partitioning can be based on a key in each record (using hash or range partitioning)". The third is "Built using bulk transformations on other RDDs". The fourth is "Can be cached for future reuse". The slide footer includes the IIT Kharagpur logo and "NPTEL ONLINE CERTIFICATION COURSES". A small video inset shows a person in the bottom right corner.

- ❑ An RDD is an immutable, partitioned, logical collection of records
 - ❑ Need not be materialized, but rather contains information to rebuild a dataset from stable storage
- ❑ Partitioning can be based on a key in each record (using hash or range partitioning)
- ❑ Built using bulk transformations on other RDDs
- ❑ Can be cached for future reuse

So, again what is an RDD? So, an RDD is an immutable, partitioned and logical collection of records; we have already discussed what it means to say immutable, which means that an RDD from the point of view of logic, cannot be change. So, once you define an RDD, the RDD remains the same. If you do some operations like transformations on this RDD, new RDD is created ok.

These are partitioned in the sense that these are split into different portions or also sometimes called partitions or splits. And each split can stay on different machines of a cluster depending on the requirement. And these are logical collections as we have said, because they need not always exist. So, even if we actually define an RDD, the actual data of that RDD may not be computed which may be really large. Instead, what is present is the information to build this RDD from the dataset, which is there in the stable storage.

So, partitioning can be based on in many ways either on key or using some hash value. And as you have discussed, typically RDDs are created using bulk transformation, which are also called transformations on other RDDs; and they can be cached for future reuse ok.

(Refer Slide Time: 12:22)

RDD Operations

Transformations (define a new RDD)	Actions (return a result to driver)
map filter sample union groupByKey reduceByKey join cache	reduce collect count save lookupKey ...

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, here is slightly more complete list of transformations, which define new RDDs and actions, which compute value ok.

(Refer Slide Time: 12:41)

RDD Fault Tolerance

- RDDs maintain *lineage* information that can be used to reconstruct lost partitions
- Ex: `cachedMsgs = textFile(...).filter(_.contains("error")).map(_.split('\t')(2)).cache()`

Handwritten notes:
- `textFile` → DFS
- `filter` → transformations parallelize → local array
- `map` → transformations parallelize → local array
- `cache()` → local array

Diagram: A flow diagram showing RDD lineage: `HdfsRDD` (path: hdfs://...) → `FilteredRDD` (func: contains(...)) → `MappedRDD` (func: split(...)) → `CachedRDD`. Handwritten notes include "Error to" and ".contains".

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, now RDDs maintain what is called lineage information to construct lost partitions. So, even before this, let us see so let us see an RDD program ok. So, suppose you have you have file which has; so let us say here you would provide a file name ok. So, suppose you have a file which has a lot of error messages or something like let us say web server log or a operating system log ok, so you will have some information

messages and you will have some error messages. Now, these files are typically very large ok.

And suppose, you want to do some processing where you want to extract the messages of the errors ok; so, the errors which have occurred, you want the messages of that ok. So, the first command is called the text file command ok. So, the text file command creates its one of the ways of creating an RDD. So, there are broadly two ways of creating an RDD; one is the text file using the text file command, which creates an RDD from a DFS file, so it creates an RDD from a distributed file system file, and the other command is called the parallelized command ok. So, this creates an RDD from a local array. So, you can pass it a local array and it will create an RDD ok.

So, in this case, we are creating an RDD from a text file command ok, so the output of this is an RDD. On that RDD, we are calling the filter operation which so this underscore here as we have seen means that it is a function literal, which takes in the particular argument, which is there in underscore, and then call the contains function on this with the string error. So, if the string passed to filter contains the string error, then it is evaluated to positive otherwise negative. And then this filter is a transformation ok. So, the filter is a transformation, which creates a new RDD with only those lines which contain the error file the string error ok.

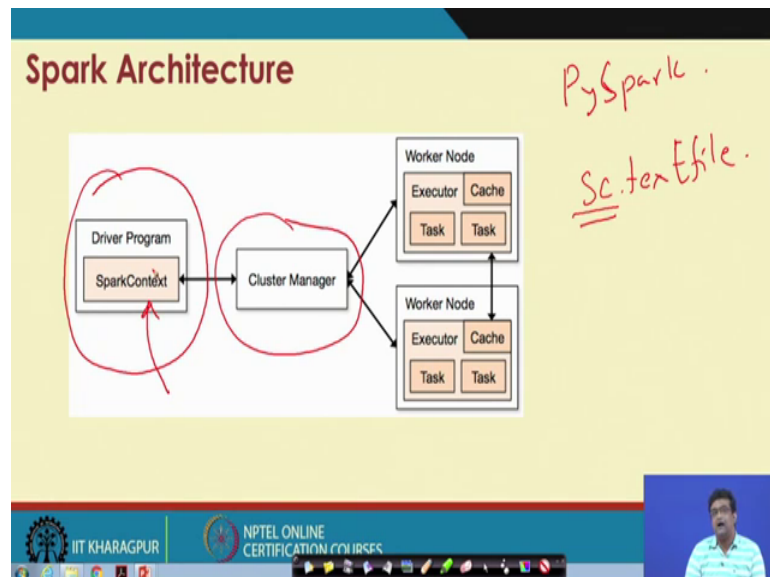
And then on this filtered RDD, we apply the map function with the split command. So, the split command so typically your file will have this format that you will have error, and then tab, and then the actual error message may be some sort of an error message ok. What you want is you want to extract this error message. So, you split the line in to 2, and you only take the 2nd field of the error and create a new RDD, which has only the 2nd field. And finally, the cached command says that the resultant RDD can be very large. So, you either store it on memory if it is possible, otherwise you store it on to over disk. So, this creates a new RDD called cached messages ok. So, this the resultant of all this operation is a new RDD called cached messages ok.

And so, now when you give this command, this RDD cached messages is created, but the actual cached message data is not created. Instead, what is created is a lineage graph something like this. So, this is a lineage graph that is created ok. So, what is the lineage graph, lineage graph is that you first have an hdfs RDD, which is the output of this text

file. Then you have a filtered RDD, which is output of this filter command. Then you have a mapped RDD, which is the output of this mapped command. And then finally, you have a cached RDD.

Moreover, you have these dependencies. So, you have that in order to compute filtered RDD, you have to compute this hdfs RDD and so on and so forth. So, you have all the RDDs, and you have all the dependencies among the RDD. So, these faults are graph in general. In fact, it forms a directed acyclic graph as we shall see ok. So, this directed acyclic graph is also called a lineage graph.

(Refer Slide Time: 18:49)



Now, how does a spark program work. So, we have already seen how the Hadoop program works. So, spark program works in a similar context. You have a driver program, which is written which is a user program written mostly in SCALA or it can be also written in it can be also written in python, when you are using something like a pyspark, but for this class we will use mostly SCALA ok.

And in this, there is a object called spark context ok. So, spark context object is something like the job object in Hadoop, where basically this is the object, which collects to a spark cluster which collects rather to a spark cluster, and then it does all the distributed operations ok. So, whenever you create an RDD you create an RDD using a spark context object, so you will have to call something like sc dot text file ok. So, sc is a spark context object ok. So, all the RDDs are created using this spark context object.

Now, in practice what happens is the spark context object has the code to contact the cluster manager, which creates some executors nodes, when spark job needs to be executed that is when a RDD needs to be materialized ok. So, whenever so when you define an RDD, nothing happens, so the cluster need not be contacted, only the lineage graph needs to be built, so that is done within the driver program.

But, whenever you RDD needs to be materialized, which is when actions are called an RDD, we will come to that. The basically the spark context will contact the cluster manager, and it will create a lot of worker nodes for the for the spark job to be executed ok. And each of these worker nodes will (Refer Time: 21:30) a certain number of tasks, which will then execute all the all the tasks which are needed for materializing all the partitions of the RDD ok. So, in general, this is the structure of a spark program ok.

(Refer Slide Time: 21:50)

Example: MapReduce

- MapReduce data flow can be expressed using RDD transformations

```
res = data.flatMap(rec => myMapFunc(rec))
           .groupByKey()
           .map((key, vals) => myReduceFunc(key, vals))
```

Or with combiners:

```
res = data.flatMap(rec => myMapFunc(rec))
           .reduceByKey(myCombiner)
           .map((key, val) => myReduceFunc(key, val))
```

The slide includes a footer with the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. A small video inset in the bottom right corner shows a man in a light blue shirt.

Now, let us see how the spark program spark is so, for example, any map reduced program can be you know created or rather can be can be mimicked using the spark platform. So, first of all in map reduce, you specify two functions; so the first is the mapper, and the second is the reducer. So, in this case, the mapper function is called my map function, and the reducer function is called my reduce func ok.

So, so in this case basically as you can see, suppose data is an RDD ok, which is storing the input dataset of a map reduce function. So, in case of remember in case of map reduce, your input comes from a hdfs file. But, in case of RDD in case of spark, your

input can come from an RDD. So, let data be that input RDD, and finally you will get the output RDD, which is the result ok. So, then you can call the flat map function which basically takes in every record, which is there in rec of the RDD data ok, and it passes through myMapFunc and creates which creates the output records, output records are of func key value pair ok.

And then what you can do is you can just so basically myMapFunc, remember can produce multiple output records, so flat map basically creates an RDD with multi which is with multiple output records per input records. So, each input record may result in multiple output records for the RDD ok.

Then you use a group by key function, which is just a shuffle function. So, it basically takes all records with the same key and groups them together. And then finally, you just apply the reduce function with the key and the value ok. So, my reduce function takes the key and the value, and outputs the reducer output record w which is the output of each of the records ok. And basically this is applied on the mapped data ok. So, this is one way of reproducing using the map reduce functionality in spark ok. Another way can be using combiner functions, where instead of group by key, you use the reduceByKey command, which first groups by key and then also acts the combiner or puts the combiner in the in the first in the in each reducer record ok.

(Refer Slide Time: 25:40)

Word Count in Spark

```
val lines = spark.textFile("hdfs://...")
val counts = lines.flatMap(_.split(" "))
                  .reduceByKey(_+_ )
counts.save("hdfs://...")
```

Handwritten annotations:

- $a+b+c$ and $(a+b) + (c+c)$ pointing to the textFile argument.
- (a, b, c, d, \dots) and $a+b \rightarrow c$ pointing to the split function.
- $(n: Int, y: Int)$ and $z: Int$ pointing to the reduceByKey function.
- $z = n + y$ pointing to the reduceByKey function signature.

Footer: IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, how will one execute word count in spark ok? So, as you can see the first line is creating RDD called lines using this so, this is the spark context object. So, it is calling the text file command on the spark context object, and creating the RDD lines. Next, it is calling flat map on lines, so that it splits the line into words using void space. And then for each word, it is outputting just the word; after this, it is just doing reduce by key operation, and it is just adding. So, reduce by key takes in two functions ok.

So, reduce by key or rather reduce by key actually takes in a function with two arguments. So, this function is actually a function of this type int comma y int and then it produces the output, let us say Z , which is of type int ok. And in this particular case, your Z is nothing but x plus y ok. So, this is what it is saying that it is taking in two arguments, and basically computing the sum of those two arguments.

Now, when reduce by key function is called to the records of an RDD, what it does is basically it takes all the records a, b, c, d a, b, c, d like this, and then it takes pairs of records in random order. So, let us say so it has to be associative the function has to be associative, so it may first compute a plus b , and then apply the result and c , so it will apply a plus b and c . And then again it will take a plus b plus c and then apply it to d and so on and so forth. So, this it may do in arbitrary order, so it can form a reduction graph. So, it can either compute a plus b plus c and then plus d or it can compute a plus b and c plus d separately, and then add the two ok. So, this is what any reduce or reduce by key operation does. And then finally, it is saving it to a hdfs file ok.

(Refer Slide Time: 28:48)

Matrix Multiplication

- ◆ Representation of Matrix:
 - ◆ List <Row index, Col index, Value>
 - ◆ Size of matrices: First matrix (A): $m \times k$, Second matrix (B): $k \times n$
- ◆ Scheme:
 - ◆ For each input record: If input record
 - ◆ Mapper key: <row_index_matrix_1, Column_index_matrix_2>
 - ◆ Mapper value: <column_index_1 / row_index_2, value>
 - ◆ GroupByKey: List(Mapper Values)
 - ◆ Collect all (two) records with the same first field multiply them and add to the sum.

Handwritten notes: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (circled), $\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix}$ (with row indices 1, 2, 2 and column indices 1, 2, 0), $A \times B$ with arrows pointing to 'row_index_2, col_1'.

Now, let us take the problem of matrix multiplication ok. So, so the input here is that you are given a matrix, which is of this form, list of row index, column index and value. So, for example, if you have a two cross two matrix of 1 0 0 1, then it will be represented as 1 comma 1 comma 1, 1 comma 2 comma 0, 2 comma 1 comma 0 and 2 comma 2 comma 1. So, this will be your RD ok. So, this is how you represent this matrix ok.

Now, let us say there are m cross k and k cross n matrices. So, then there will be two matrices A and B ok. So, then you take so you know that for each row of A so, if you have to compute A times B for each row of A , you have to compute for each row you have to compute one entry for each row of A , and each column of B ok.

So, hence your mapper key for the 1st, so your mapper key for the if you have the 1st matrix, should be the row index of the 1st matrix ok. And if it is the 2nd matrix, your mapper key should be the column index of the 2nd matrix. So, it should either be the row index of the 1st matrix or the column index of the 2nd matrix ok. And your sorry your mapper key should be the row index of the 2nd matrix, and the column index of the of the 1st matrix ok, let me let me start again.

(Refer Slide Time: 31:37)

Matrix Multiplication (row_index - 1, col_index - 1)

- ◆ Representation of Matrix:
 - ◆ List <Row index, Col index, Value>
 - ◆ Size of matrices: First matrix (A): m*k, Second matrix (B): k*n
- ◆ Scheme:
 - ◆ For each input record: If input record
- ◆ Mapper key: <row_index_matrix_1, Column_index_matrix_2>
- ◆ Mapper value: <column_index_1 / row_index_2, value>
- ◆ GroupByKey: List(Mapper Values)
- ◆ Collect all (two) records with the same first field multiply them and add to the sum.

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

So, your mapper key should be so, you should have an output for all. So, all row index of the 1st matrix, and all column index of 2nd matrix should come to one place ok. So, this so these should be your mapper keys ok. And your mapper value should have the either the column index of the 1st matrix or the row index of the 2nd matrix, and then it should also contain the value ok.

So, now when in the reducer, you get you get all the row indices and the all the column indices corresponding to corresponding to the first so corresponding to all the particular row and all the columns and a particular column, so this for the 1st matrix, and a particular column for the 2nd matrix ok. You can just match the corresponding row or the column indices, so and then you take the product of the corresponding values, and you sum them. So, in other words, if you have A_{ik} and you have B_{kj} , you just sum over all values of k , and this becomes your resultant C_{ij} entry ok. So, so you can compute this in a distributed manner using this particular algorithm ok.

(Refer Slide Time: 33:40)

Logistic Regression

- Binary Classification. $y \in \{+1, -1\}$
- Probability of classes given by linear model:
$$p(y | x, w) = \frac{1}{1 + e^{(-y w^T x)}}$$

Handwritten note: $\sigma(y w^T x)$
- Regularized ML estimate of w given dataset (x_i, y_i) is obtained by minimizing:
$$l(w) = \sum_i \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2} w^T w$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The next very important application is the logistic regression application, which is used for binary classification. So, we have seen this application. So, you have the binary classification problem, where you have the labels as plus 1 and minus 1, and you have the sigmoid of w transpose sigmoid of y minus y times w transpose x as the probability of y , and then you have to minimize this particular loss function in a distributed manner ok.

(Refer Slide Time: 34:27)

Logistic Regression

- Gradient of the objective is given by:
$$\nabla l(w) = \sum_i (1 - \sigma(y_i w^T x_i)) y_i x_i - \lambda w$$
- Gradient Descent updates are:
$$w^{t+1} = w^t - s \nabla l(w^t)$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And you have the gradient, which is given by this function; you can check that the gradient of the loss function is given by this function. And your algorithm for computing w is something like this ok. Now, the question is how will you implement this algorithm? So, this is the gradient descent update that you want to implement.

(Refer Slide Time: 35:01)

```
Spark Implementation

val x = loadData(file) //creates RDD
var w = 0
do {
  //creates RDD
  val g = x.map(a => grad(w,a)).reduce(+)
  s = linesearch(x,w,g)
  w = w - s * g
}while(norm(g) > e)
```

Handwritten annotations:

- (x, y) circled in red.
- Red arrow pointing to `var w = 0`.
- Red matrix notation: $a = \begin{pmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$

So, this is a simple spark program that tries to implement that algorithm ok. So, first you have an RDD x , the records of which are data points for the data sets. So, this RDD x has both x and y in the logistic regression parlance. And then you have a local variable w in the driver program, which stores the weights of each of these matrices or weights for the logistic regression ok. So, the first you have to have a gradient function ok, which computes the gradient of a dataset or a subset of data a , let us say a is equal to $x_1, y_1, \dots, x_k, y_k$ ok. So, this gradient function should compute the gradient given this dataset a ok. And then you can just the reduce function just adds those gradients ok.


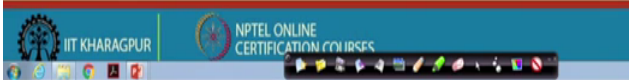
(Refer Slide Time: 36:40)

Logistic Regression

- Gradient of the objective is given by:

$$\nabla l(w) = \sum (1 - \sigma(y_i w^T x_i)) y_i x_i + \lambda w$$

- Gradient Descent updates are:

$$w^{t+1} = w^t - s \nabla l(w^t)$$



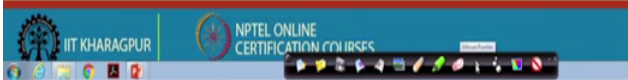
So, why is this working, this is working because if you see here, your gradient is actually sum over the gradients of all data points ok. So, if i is the index of the gradient of the data points, and if you sum over the gradient of all data points, you get the total gradient of the objective function, you have to subtract at a later time the regularizer, but for the timing, you can also assume lambda to be 0 ok.

(Refer Slide Time: 37:24)

Spark Implementation

```
val x = loadData(file) //creates RDD
var w = 0
do {
  //creates RDD
  val g = x.map(a => grad(w, a)).reduce(+)
  s = linesearch(x, w, g)
  w = w - s * g
}while(norm(g) > e)
```

Handwritten notes: Closure, $X - 1B$, $x 1A$, $20M$, 201 , $1B \rightarrow 1M$, $2(1M)$



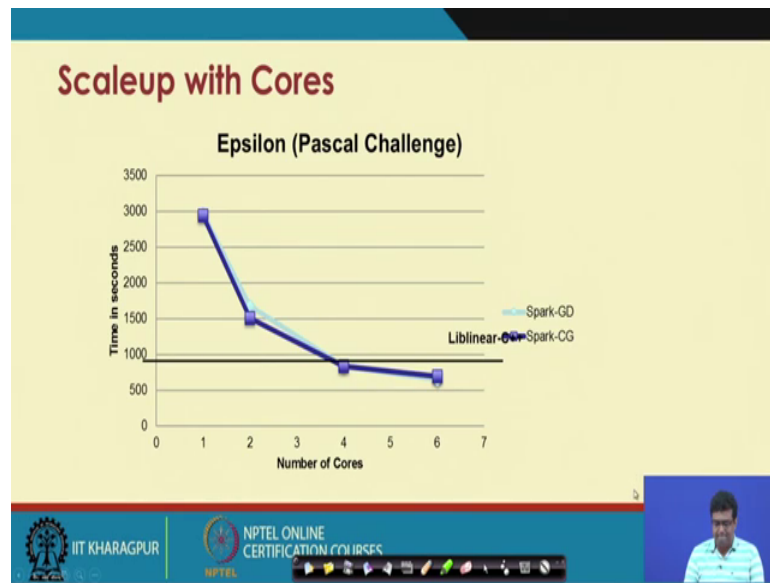
So, if this is the case, you can compute the total gradient in a distributed manner using this line of code that is you compute for each subset of data the gradient with

respect to that subset of data. Of course, for computing the gradient, you need the current parameter value, which is a local variable. So, it has to be passed to this as a closure. So, this is a closure that is passed to each and every server in order to compute the gradient of this particular value, and then you sum the gradient ok. And then s is another function, which given the gradient direction and the parameter current parameter w , computes the step length ok. And then in the local machine, you just update the parameter w ; so, just to give you a feeling of how this program would work.

So, if you have many machines, and you have a very large dataset, so let us say you have 1 billion data points each having 1 million features ok, so then you would have a central machine, where you would have the parameter w , which is of size 1 million. And now you have a data matrix which is X , which is stored as an RDD which is of size 1 billion cross 1 million, because it has one billion data points each of which is of size 1 million. So, then each of these each of these machines, so this is distributed in each of these machines. So, this may have 20 million of the data, this may have another 20 million of the data and so on and so forth, and it may be distributed on to five machines.

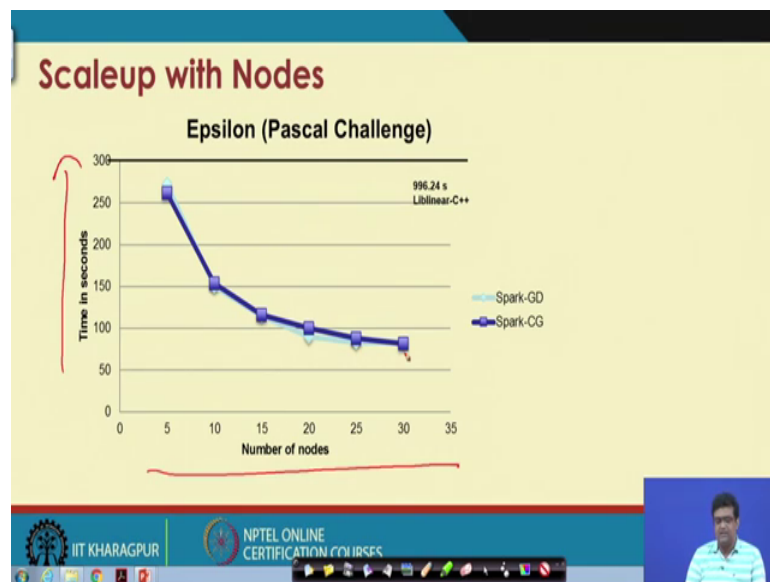
Now, the gradient computation happens parallelly in each of these machines, and then the sum is computed in the local machine, and then the update is also computed in the local machine, so that computation is ordered 1 million, whereas the computation on this distributed machine is of order billion. So, hence it will result in a speed up of the total computation.

(Refer Slide Time: 40:16)



So, these are some results of implementing this implementing this program using the map reduce paradigm.

(Refer Slide Time: 40:30)

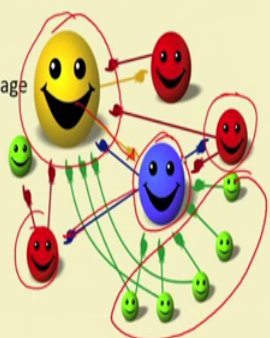


So, here in the x-axis, you can see the number of nodes ok. And in the y-axis, you can see the time taken in seconds. So, you can see that initially there is a larger drop as you increase the number of nodes from 5 to 10, but as you keep on increasing the number of nodes after a while, it saturates ok.

(Refer Slide Time: 41:05)

Basic Idea

- Give pages ranks (scores) based on links to them
 - Links from many pages → high rank
 - Link from a high-rank page → high rank



The diagram shows a network of nodes represented by colorful smiley faces (yellow, red, blue, green) connected by arrows. A large yellow smiley face is the central node, receiving multiple arrows from other nodes. A blue smiley face also receives arrows from several nodes. The diagram demonstrates how a page's rank is determined by the number of links it receives and the rank of the pages linking to it.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We take the last example here, which is the example of the page rank algorithm, which is actually a very complex algorithm to implement ok. So, what is the page rank algorithm? The page rank algorithm means used to compute the score of web pages using the links structure of those web pages. So, we all know that web pages point to other web pages, or they have links to other web pages. So, the idea behind page rank is very simple, the idea is that if a page receives links from many other pages or many other high ranking pages, then it has a very high rank. Whereas, if a high ranking page points to another page, then the page that it points to will also receive a very high rank. So, this is the idea ok.

So, many so for example, in this case, these are many low ranking pages, which are pointing to this high ranking linkage. So, this receives a somewhat higher rank ok. And these are other somewhat high ranking pages, which are pointing to these, this page. So, this is a very high ranking page. And now, this page is pointing to this page, so this page has intermediate rank, but this page points to again these pages ok. So, mathematically the way this works is the following ok.

(Refer Slide Time: 42:57)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\frac{\text{rank}_p}{|\text{neighbors}_p|}$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, you start with the rank or the score of each page as being 1. And then on each iteration; you have a page p contribute the rank of p by the number of neighbors of p to its neighbors. And then and then you just take the sum of all the contributions from all the incoming links, and multiply it with 0.85 and then add 0.15 ok. So, this is the page rank algorithm.

(Refer Slide Time: 43:47)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\frac{\text{rank}_p}{|\text{neighbors}_p|}$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, for example, you start with 1 for all of these, then because so this web page contributes 0.5 to both of its neighbors. So, this web page contributes 0.5, because it has

two outgoing links; this contributes 1, this contributes 0.5, because it has again two outgoing links, and this contributes 1 ok.

(Refer Slide Time: 44:15)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

The diagram shows four pages represented as document icons. The top page has a rank of 1.85. The left page has a rank of 0.58. The right page has a rank of 1.0. The bottom page has a rank of 0.58. Arrows indicate links: the top page links to the left and right pages; the left page links to the top and bottom pages; the right page links to the top and bottom pages; and the bottom page links to the left and right pages. Red circles are drawn around the top and right pages.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, at the end of the first iteration, the page rank of this will remain 1.

(Refer Slide Time: 44:23)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

The diagram shows the same four pages as the previous slide. The top page has a rank of 1.31. The left page has a rank of 0.39. The right page has a rank of 1.72. The bottom page has a rank of 0.58. Arrows indicate links: the top page links to the left and right pages; the left page links to the top and bottom pages; the right page links to the top and bottom pages; and the bottom page links to the left and right pages. Red circles are drawn around the top and right pages.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The page rank of this sorry page rank of this will become 0.58 sorry yeah. So, and the page rank of this will become 0.15 ok. Now, you can go forward further.

(Refer Slide Time: 44:39)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

The diagram shows four pages arranged in a diamond shape. The top page has a rank of 1.85, the bottom page 0.58, the left page 0.58, and the right page 1.0. Green arrows indicate contributions: 0.58 from the left page to the top page, 1.85 from the top page to the right page, 1.0 from the right page to the bottom page, 0.58 from the bottom page to the left page, and 0.29 from the bottom page to the top page.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And then again, these are the contributions in green. And you can see that now the page rank of this one has become 1.72, because a high ranking page is pointing here, and this has reduced a little bit and so on and so forth.

(Refer Slide Time: 44:57)

Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$

Final state:

The diagram shows the same four pages as before, but with updated ranks: top page 1.44, bottom page 0.73, left page 0.46, and right page 1.37. The arrows and contributions are no longer shown.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And you keep doing like this. And it will converge to this one being 1.37, this one being 0.46 and so on and so forth. So, this is the algorithm.

(Refer Slide Time: 45:08)

Spark Implementation

```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    (url, (nhb, rank)) =>
      nhb(dest => (dest, rank/nhb.size))
  }
  ranks = contribs.reduceByKey(+)
  .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```

The slide includes handwritten annotations in red: arrows pointing from 'contribs' to the flatMap function, from 'URL' to the 'url' parameter, and from 'URL' to the 'dest' parameter in the lambda function. Another arrow points from 'URL' to the 'rank/nhb.size' expression. The 'reduceByKey(+)' and 'mapValues' lines are also annotated with arrows.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the question is how will we implement this algorithm in spark? So, for this, we need two RDDs ok. So, the first RDD is the links RDD, which stores the url. So, so for each url, it stores it the neighbors of that url or the url's that point to the this particular url or rather the url's that this current url points too ok. And the second RDD stores a url and its rank ok.

Now, what you do is you take the links RDD and you join it with the ranks RDD. So, now you have for each url, its neighbors, so the url that it points to and you have the rank of the current url ok. So, what you do is you take out the neighbors for from this url. And for all the neighbors, you emit one record which has the following. So, it has the url of that particular neighbor, and it has the rank of the current url divided by the size of the neighbors ok. So, this is basically doing all the contributions to all the neighbors ok. So, we call this RDD the contribs RDD ok.

Now this contribs RDD takes a so now, then you call the reduce by key on this contribs RDD, because the key here is the url which is receiving this contribution, because which is the neighbor of the current url ok. And those contributions get added up. And then finally, for each of these you each so, this creates an RDD with all the contributions, and then you multiply that with 0.15 and or multiply that with 0.85 and add 0.1, and then you get the new rank RDD of the for each url. So, then you keep doing this for one to a

certain number of iterations, at the time when this algorithm will converge. So, this is a spark implementation of the page rank algorithm.

(Refer Slide Time: 48:30)

Conclusion:

- We have seen:
 - Spark
 - Motivation
 - RDD
 - Actions and transformations
 - Examples:
 - Matrix multiplication
 - Logistic regression
 - Pagerank

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Sourangshu Bhattacharya

So, in conclusion, we have seen the motivation, RDD, and we have seen what are what are actions and transformations. And then we have seen these three programming examples using spark.

Thank you.