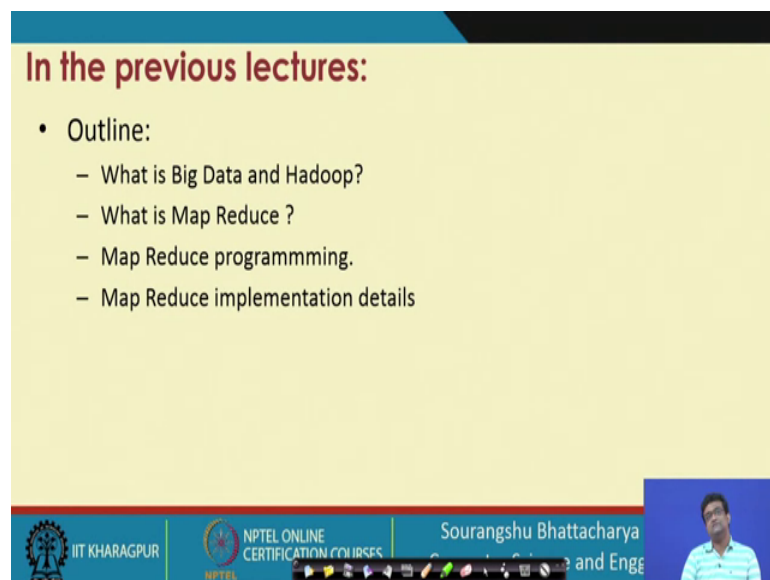


Scalable Data Science
Prof. Sourangshu Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 19a
Spark

Hello, everyone. Welcome to the 19th lecture of the NPTEL course on Scalable Data Science. I am Professor Sourangshu Bhattacharya from Computer Science Department in IIT, Kharagpur and today we are going to discuss about Spark.

(Refer Slide Time: 00:35)



In the previous lectures:

- Outline:
 - What is Big Data and Hadoop?
 - What is Map Reduce ?
 - Map Reduce programming.
 - Map Reduce implementation details

The slide features a yellow background with a blue header and footer. The footer contains the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the name Sourangshu Bhattacharya. A small video inset of the professor is visible in the bottom right corner.

So, already we have discussed what is big data we have discussed, what is hadoop which is a open source system for big data, we have discussed what is map reduce and the map reduce programming paradigm we have seen some map reduce programs and we have also seen the implementation details of the map reduce paradigm, how map reduce can be implemented such that it can really work in a distributed setting on a very large data.

(Refer Slide Time: 01:12)

In this Lecture:

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

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

So, today we are going to discuss or rather in this series of lectures we are going to discuss about spark. So, to begin with we are going to discuss a bit about the programming language scala which is functional programming language. So, so, there are two main reasons for discussing this programming language scala; the first reason is that a spark was itself developed in scala and even now the scala is the most widely used platform for writing programs for spark.

Or and secondly in general the lot of these distributed computation paradigms including spark and map reduced are actually inspired by functional programming paradigm which is an existing programming paradigm. So, spark is very strongly integrated with the functional programming language scala and hence understanding the constructs of scala we will also help us understanding the constructs of spark better.

So, we will cover about the var and the val. So, we will not cover all aspects of scala we will cover only very limited aspects of scala namely var and val then we will cover classes and objects. Most importantly we will cover the notion of functions and higher order functions which are very very important and are directly borrowed and then generalized and distributed in the spark programming paradigm and also we will discuss about lists which are also borrowed in the spark programming paradigm for a something like a distributed list ok.

(Refer Slide Time: 03:31)

Scala

- Scala is both functional and object-oriented
 - every value is an object
 - every function is a value--including methods
- Scala is interoperable with java.
- Scala is statically typed
 - includes a local type inference system:

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, as discussed earlier scala is both functional and object oriented programming language. So, basically every value in scala is an object. So, I am assuming that you are familiar with what is object oriented programming and so, every value in scala is an object and moreover every function is value and hence an object, including the methods which are part of objects, ok.

So, scala was designed on top of java. So, when you write the when you see the programming construct you will see a lot of derivatives from the java and another important piece of information is that scala is actually a statically typed language just like java. So, every variable has a type that it can do a local type inference. So, we will see. So, you do not always have to specify the type.

(Refer Slide Time: 04:35)

Var and Val

- ❑ Use `var` to declare variables:
 - ❑ `var x = 3;`
 - ❑ `x += 4;`
- ❑ Use `val` to declare values (final vars)
 - ❑ `val y = 3;`
 - ❑ `y += 4; // error`
- ❑ Notice no types, but it is statically typed
 - ❑ `var x = 3;` → *n: Int*
 - ❑ `x = "hello world"; // error`
- ❑ Type annotations:
 - ❑ `var x : Int = 3;`

So, so, first we discuss about this var and val. So, var you can use var keyword to declare variables. So, scala has the concept of variables and constants. So, for example, in this case as you can see the var x defines a variable x. So, at a later point in time in the program you can say x plus equal to 4 or x is equal to 5 over something and it will work this is because x is a variable. However, you can also use this keyword val which is used to declare values or constants.

So, in java these are called const variable constant variables which are. So, for example, if you say y is equal to 3 and then if you try to do y plus equal to 4 scala will throw an error because you cannot change a concept. So, this brings the notion of immutability. So, there are certain variables which are immutable. So, you notice that that we have not specified any type. So, that type will be inferred.

So, for example, in this case you the type of x will be taken as int because you have given x is equal to 3. So, later if you try to do x is equal to let us say hello world it will give an error, ok of course, you can also explicitly mention the type like this ok.

(Refer Slide Time: 06:29)

```
Class definition

class Point(val xc: Int, val yc: Int) {
  var x: Int = xc
  var y: Int = yc

  def move(dx: Int, dy: Int) {
    x = x + dx
    y = y + dy
    println ("Point x location : " + x);
    println ("Point y location : " + y);
  }
}
```

So, this is an example of a class definition in scala. So, for example, we are trying to define the class point where which has two values one is xc or rather two components and both are of type int and one is xc and one is yc and may be these are storing the coordinates and then this is the method which is the method move which takes dx and dy as the input. So, these are may be the amount by which you want the current point to move and then what it just computes the new x as a x plus dx and also the new y as y plus dy.

So, and then it can printing. So, this is just to get you familiar with the programming language or programming style of scala.

(Refer Slide Time: 07:34)

The slide is titled "Scala" and contains the following content:

- ❑ Class instances
 - ❑ `val c = new IntCounter[String];`
- ❑ Accessing members
 - ❑ `println(c.size); // same as c.size()`
- ❑ Defining functions:
 - ❑ `def foo(x : Int) { println(x == 42); }` (with a red arrow pointing to the right)
 - ❑ `def bar(y : Int): Int = y + 42; // no braces // needed!` (with red circles around `Int` and `y + 42`, and a red arrow pointing to the right)
 - ❑ `def return42 = 42; // No parameters either!` (with a red arrow pointing to the right)

Handwritten notes in red ink on the slide include: `bar(10) → 52` with an arrow pointing from the `foo` function definition to the right, and `bar(10) → 52` with an arrow pointing from the `bar` function definition to the right.

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a presenter is visible in the bottom right corner.

Now, once you define a class like we have defined you can define objects of this class. So, you can say `val c` which is like a value of from the class `new counter` and it can also be a parameterized class something like `string` which is the class itself is defined as a parameterized class and the `new` key word you would create a new object of this class.

And, then you can just access members as you would doing java. So, many of this constructs had just like java and you can say `print ln c dot size`. In fact, you can use. So, since a scala programs run in java virtual machines you can use much of the java library also ok. Now, in functions is where the new things starts. So, you can the first definition is a standard definition of functions. So, you have the function name and then you have the argument. So, this is the argument, `x` is the argument and then you have the function body within the basis, ok.

The second definition is a definition which is without braces ok. So, you can so, here this is `bar` is the function name, this `y` is the argument. So, whatever is within the bracket is the argument list, this is the return type of this function. So, you are saying that the function `bar` takes in an integer and returns another integer and then you are just. So, you are just saying the return value is equal to `y plus 42`.

So, as if it is just values but it is actually not a value because, now whenever you say `bar` and then you pass it a certain argument it will return the. So, let us say you add 10 then it will return 52. So, it will add 42 to it and return 52 and the lastly of course, you can

define functions which have no argument, no parameters also and no arguments also in which case of course, these are same as defining constants.

(Refer Slide Time: 10:22)

Functions are first-class objects

- Functions are **values** (like integers, etc.) and can be assigned to variables, passed to and returned from functions, and so on
- Wherever you see the **=>** symbol, it's a literal function
- Example (assigning a literal function to the variable **foo**):
- `scala> val foo = (x: Int) => if (x % 2 == 0) x / 2 else 3 * x + 1`
`foo: (Int) => Int = <function1>`

`scala> foo(7)`
`res28: Int = 22`

- The basic syntax of a function literal is **parameter_list => function_body**
- In this example, **foreach** is a function that takes a function as a parameter:
- `myList.foreach(j => println(2 * j))`

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

So, we have already discussed that functions are like values because it is a functional programming language. So, one way of defining a function literal is to use equal to greater than symbol, ok. So, there so, so for example, if you say val foo equal to and then you write within bracket x. So, x is now the parameter of this function, then you write the symbol of function which is the equal to greater than symbol and then you write the body of the function ok. So, this defines the new function with the name foo. So, whenever you have to invoke this function you can just call foo with the parameter and then it will execute.

So, for example, in this case it checks if x modular 2 is 0, then it returns x by 2 otherwise it returns 3 x plus 1. So, when you pass 7 it returns 3 into 7 plus 1 which is 22 ok. So, in general this is the a function literal, ok. So, function literal is of this form that you have the parameter list followed by the equal to greater than sign or is equal to greater than symbol followed by the function body and then so, we will come to this for each function in a minute ok.

(Refer Slide Time: 12:19)

Functions as parameters

- To have a function parameter, you must know how to write its type:
 - $(type1, type2, \dots, typeN) \Rightarrow return_type$
 - $type \Rightarrow return_type$ // if only one parameter
 - $() \Rightarrow return_type$ // if no parameters
- Example:
 - `scala> def doTwice(f: Int => Int, n: Int) = f(f(n))`
`doTwice: (f: (Int) => Int, n: Int) Int`
 - `scala> def collatz(n: Int) = if (n % 2 == 0) n / 2 else 3 * n + 1`
`collatz: (n: Int) Int`
 - `scala> doTwice(collatz, 7)`
`res2: Int = 11`
 - `scala> doTwice(a => 101 * a, 3)`
`res4: Int = 30603`

Handwritten notes: 22 , $a \Rightarrow 101 * a$, $101 * 303$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, so, now, the advantage of this is that now you can for example, you can use pass around functions like as if they are values. So, you can for example, pass functions as parameters to other functions which you could earlier not do not only that you can define functions which take in as one of the parameters other functions ok. So, for example, take this example. So, we have already said that you can define the function using this equal to greater than sign. For example, this is one function where you have taken N arguments and you have given a certain return or you can take just one argument and take a return and so on and so forth.

So, let us take this example. So, in this example the first statement is defining a function do twice, ok. So, this function takes in as the first argument of function. So, this first argument for this function is another function, ok. So, what we have giving here is just the prototype of this functions. So, basically this function f takes in one integer and returns one integer and that happens to be the first argument of this do twice function and the second argument is just an integer, ok.

And, then the return values for this do twice function is that you apply this f function twice to this argument n which is also an argument to this function and return the value. So, so, for example, now you define another function let us say collatz which takes in one integer as an argument and returns another integer as an argument. So, if an integer is even number then it returns n by 2 else it returns 3 into n plus 1.

So, now if you pass this collatz function to doTwice. So, you call the doTwice function with collatz and the arguments 7. So, according to this definition it basically applies this function two times, ok. So, the function so, collatz function two time. So, that the first time as we all know the output is 22. Now, second time when it applies it knows that it is a even numbers. So, it just returns n by 2. So, hence the final output is 11,. Similarly, you can pass another function literal. So, instead of defining a function and then passing it to doTwice you can just define a function literal here which is like. So, this is your function literal which basically takes in value a and then returns 101 times a, ok. So, this is your function.

So, this is the first argument which is the function literal and the second argument is of course, 3. So, it will apply this 101 times a two times. So, first it will apply 101 times 2 a. So, it will get 303 and then again you will multiply this by 101 and then it gets this result, ok.

(Refer Slide Time: 16:22)

Scala

- ❑ Defining lambdas – nameless functions (types sometimes needed)
 - ❑ `val f = x: Int => x + 42;`
- ❑ Closures (context sensitive functions)
 - ❑ `var y = 3;`
 - ❑ `val g = x: Int => y += 1; x+y; }`
- ❑ Maps (and a cool way to do some functions)
 - ❑ `List(1,2,3).map(_+10).foreach(println)`
- ❑ Filtering (and ranges!)
 - ❑ `1 to 100 filter (_ % 7 == 3) foreach (println)`

Now, we want to define two – three types of functions. So, so, as we have seen now that you can understand that this is something going towards map reduce where you basically have to pass functions. So, before going in to that you can have two types of functions is first type is called the lambdas, which are basically nameless functions which take in a value. So, lambdas are nameless functions which have no side effects, ok. So, these are functions which have no side effects. So, these are nameless functions which take in

some arguments it can take either one argument or more than one argument and it returns a value, but does not need anything else, ok. So, this is the first type of function.

Second type of functions is called what are called closures. So, closures are functions which are sensitive on the context. So, for example, you can have a variable `y` which is `y` is equal to 3 and then you can have a function `g`, which is basically taking in an argument `x` your function `g` takes in an argument `x` and which is of type `int` of course, and then it does `y` plus equal to 1 and then returns `x` plus `y`, ok. So, this function not only takes in the arguments that it is passed, but also utilizes this variable `y`, ok. So, many functions do this for example, methods in of a class can have dereference the local variables of the that class while taking in also the arguments.

So, we will see how these are tackled why these are important, but note that these functions can also be context sensitive, ok.

Now, so, we will now go into some functions.

(Refer Slide Time: 18:55)

Lists

- Scala's **Lists** are more useful, and used more often, than Arrays
 - `val list1 = List(3, 1, 4, 1, 6)` →
 - `val list2 = List[Int]()` // An empty list must have an explicit type →
- By default, **Lists**, like **Strings**, are **immutable**
 - Operations on an immutable List return a new List
- Basic operations:
 - `list.head` (or `list head`) returns the first element in the list
 - `list.tail` (or `list tail`) returns a list with the first element removed
 - `list(i)` returns the *i*th element (starting from 0) of the list →
 - `list(i) = value` is **illegal** (immutable, remember?) →
- There are over 150 built-in operations on Lists—use the API!

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, before going into functions we want to define one more thing which is called the lists ok. So, lists are like arrays ok. So, for so, this is an example of a list, ok. So, and you can also have a list which is not populated, in which case you have to give the type of course. So, lists like strings so, these are arrays, but these are immutable arrays, and so, so lists

are like a. So, what we think in the real world that these are list of values which you need to operate on ok.

So, for example, you can. So, there are many many functions which are defined on list for example, you can use list dot head to get the first entry of the list or you can use list dot tail to get the last element of the list. In general you can get the i-th element of list. However, you cannot do something like this that list dot i is equal to value ok, because it is immutable.

(Refer Slide Time: 20:27)

Higher-order methods on Lists

- **map** applies a one-parameter function to every element of a List, returning a new List
 - scala> def double(n: Int) = 2 * n →
double: (n: Int)Int
 - scala> val ll = List(2, 3, 5, 7, 11)
ll: List[Int] = List(2, 3, 5, 7, 11)
 - scala> ll.map(double)
res5: List[Int] = List(4, 6, 10, 14, 22)
 - scala> ll.map(n => 3 * n)
res6: List[Int] = List(6, 9, 15, 21, 33)

Handwritten notes: ll.map(double) → Int, String
- **filter** applies a one-parameter test to every element of a List, returning a List of those elements that pass the test
 - scala> ll.filter(n => n < 3)
res10: List[Int] = List(2, 3)
 - scala> ll.filter(_ < 5) // abbreviated function where parameter is used once
res11: List[Int] = List(2, 3)

Handwritten note: bool →

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, with lists this colour defines some of this operation. So, the first operation is called the map operations, ok. Now, these are also called higher order methods or higher order functions from scala point of view it is not much different than an existing function. However, from programmers point of view these are higher order functions because they take in another function as an argument. So, what does the map function do for example. So, the map function is a member function of the list class, ok.

And, when you call, so, let us say you define a list which is ll, and then you call the map function on the list and parameter to this map function is the double function. So, another way of writing the same thing is you call ll dot map and then as a parameter you just say double. Now, note that double is defined as a function here, ok. So, what this will do is it will return another list where every element of the original list has been doubled, ok. So, what map function is doing is it is being called on a list. So, it is the member function of

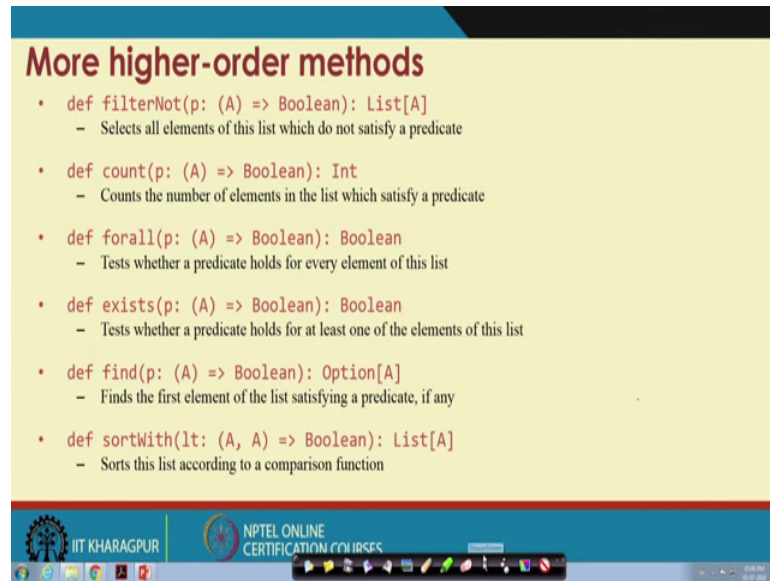
a list it takes in another function and then applies that function to every individual element in this list and returns the resulting list as a new list, ok. So, this is what map function does.

Similarly, you can have a filter function ok. So, the filter function also works on elements individual elements of a list, ok. It takes in a function which basically it is a Boolean valued function on the type of element. So, note one thing that the value the type of the list. So, list is a parameterized type. So, your list you can have list of integer, you can have list of strings etcetera. Now, this double function must be of appropriate type such that if it is being called on a list of integers, it should take in as argument an integer which was the case in this case and if it is being called on a list of strings it should take in as argument a string, ok. So, otherwise there will be error thrown, hm.

Now, similarly here so the filter function takes in as an argument a function a Boolean valued function. So, this function takes in. So, the function that is to be supplied to the filter function has to be a function which returns either true or false as an as a return value and now for the filter function simply returns a list for which the resulting elements evaluate to true according to the supplied function, ok.

So, for example, in this case a if you call filter with underscore less than or equal to 5 or which is same as this. So, if you call so, n and then equal to greater than n less than 5, which basically means that it evaluates true when the argument is less than 5 otherwise false, then it will only select those which are those elements which are less than 5.

(Refer Slide Time: 24:31)



More higher-order methods

- `def filterNot(p: (A) => Boolean): List[A]`
 - Selects all elements of this list which do not satisfy a predicate
- `def count(p: (A) => Boolean): Int`
 - Counts the number of elements in the list which satisfy a predicate
- `def forall(p: (A) => Boolean): Boolean`
 - Tests whether a predicate holds for every element of this list
- `def exists(p: (A) => Boolean): Boolean`
 - Tests whether a predicate holds for at least one of the elements of this list
- `def find(p: (A) => Boolean): Option[A]`
 - Finds the first element of the list satisfying a predicate, if any
- `def sortWith(lt: (A, A) => Boolean): List[A]`
 - Sorts this list according to a comparison function

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are many other higher order methods which one can check basically these are filter or not count for all exists find an sort with, ok. So, you can you can check the scala reference for all the higher order methods that one can use.

So, now, just to sum up. So, scala is the functional programming language which allows you to treat functions as though there are their values. So, functions can be passed to other functions, functions can be part of a function and so on and so forth, and then you have higher order function which can operate on these lower order, which can take these simple functions as input and then do adaptive kind of computation based on the input function that is given.

(Refer Slide Time: 25:58)

Spark

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

<http://spark.apache.org>

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now we start discussing sparks. So, what is spark? So, spark is an in memory cluster computing platform for iterative and interactive applications, ok. So, it is a open source aperture project which is available at this URL spark dot aperture dot arc and it is free for download and so, anybody can use it.

(Refer Slide Time: 26:38)

Spark

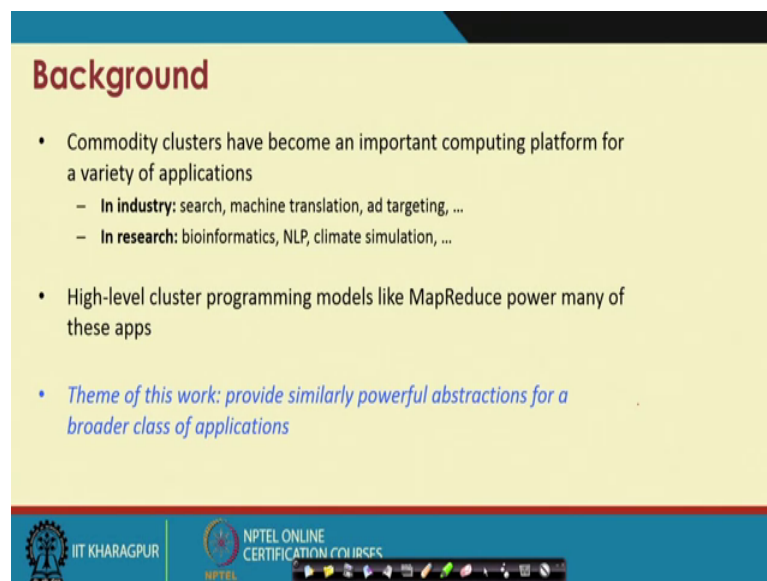
- Started in AMPLab at UC Berkeley.
- Resilient Distributed Datasets.
- Data and/or Computation Intensive.
- Scalable – fault tolerant.
- Integrated with SCALA.
- Straggler handling.
- Data locality.
- Easy to use.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so a little bit of background on spark. So, spark was started in AMPLab at UC Berkeley, this was around 2012 it and then they came up with this concept of what is called resilient distributed datasets,. So, these are the main concept behind spark, we will

come to what they are and why they are so interesting,. So, one of the advantages of spark is that unlike hadoop which is mostly for data intensive computation spark is a bit more flexible. So, it can be used for both data and computational intensive computations, ok. Then just like hadoop spark is fall tolerant it is as I have already mentioned integrated with scala, it has all the goodies that hadoop provides or all the good properties that hadoop provides like strangle handling and data locality and it is also very easy to use, ok.

(Refer Slide Time: 27:56)



Background

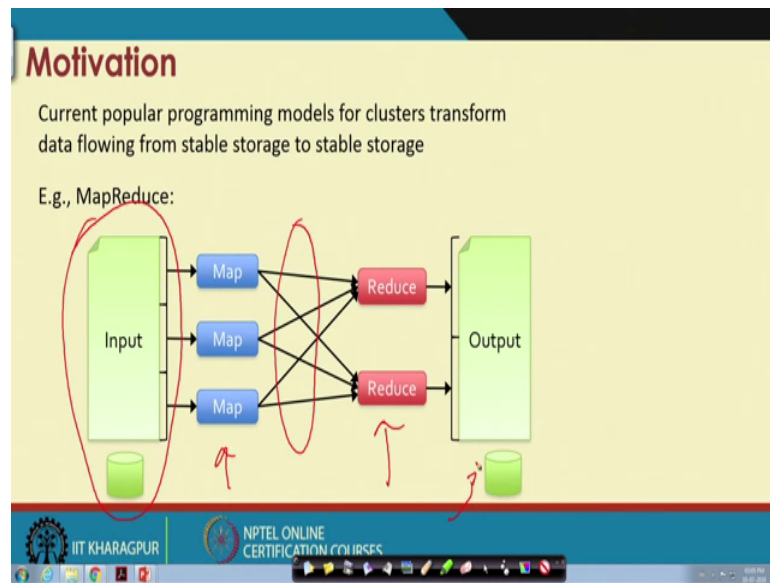
- Commodity clusters have become an important computing platform for a variety of applications
 - In **industry**: search, machine translation, ad targeting, ...
 - In **research**: bioinformatics, NLP, climate simulation, ...
- High-level cluster programming models like MapReduce power many of these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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, so, I think we already have the background. So, the basic idea is that we want to perform big data computation on a set of community clusters. So, so there are of course, many many applications like an industry you can have web search, you can have machine translation, you can have ad targeting in research you can have bioinformatics, NLP, climate simulation etcetera, ok.

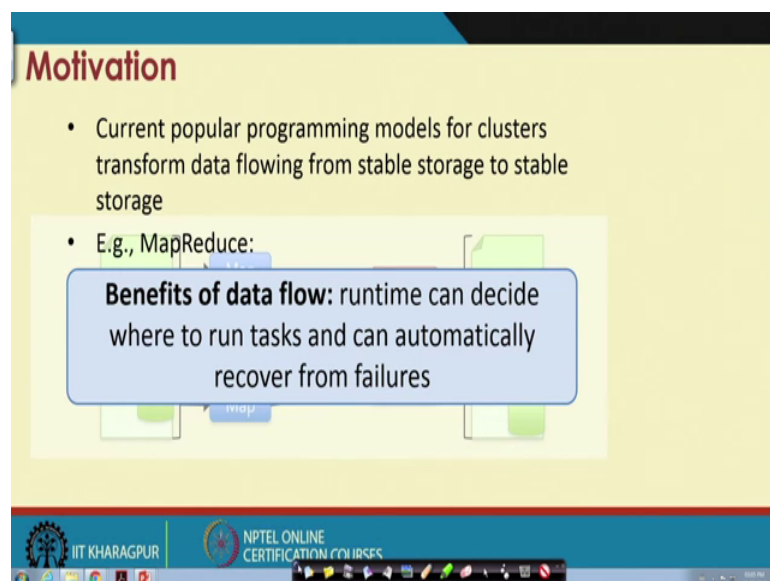
So, now typically for such data intensive computation one tries to use the map reduce frame work. So, basically the understanding here is that can we use framework or the computation that map reduce provides, but make it more powerful for the programmer or in other words the programmer should be able to express a wider array of computations than what he or she could using map reduce.

(Refer Slide Time: 29:13)



So, what is the map reduce computation model? The map reduce computation model is something like this that you have this input which is basically the a stable storage. So, typically for the big data framework we will people use distributed file system as the storage and then there are some mapper jobs which run and then there is a shuffling that goes on here and then there is a there is some reduce jobs which run and then finally, you write to some stable storage which is again typically a distributed storage, ok.

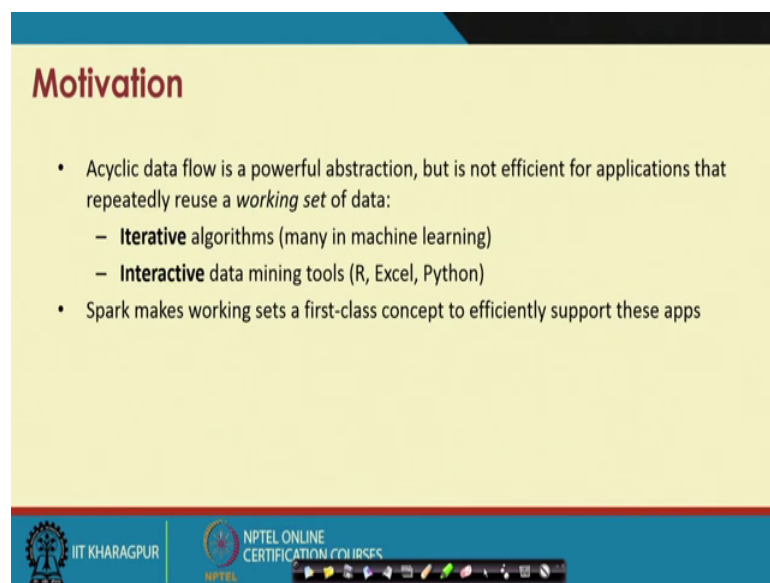
(Refer Slide Time: 30:00)



Now, the question is so why do we do this, ok. So, one thing we have. So, we have seen the advantage or the benefits of this kind of map reduce frameworks or the benefits of this. So, the first is that you have scalability and fault tolerance, then the run time can decide on everything it can decide on task on the task scheduling it can automatically recover from failures and basically it can scale very well. But, the question is this the only computation model for which we can have all this advantage, ok.

So, now let us see, what are the problems with this computation model, ok.

(Refer Slide Time: 30:54)



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

So, the problems with this computation model is that these two, ok. So, first iterative algorithms are not very easily expressed in hadoop map reduce. So, this is. So, first of all you have to write multiple hadoop jobs in a loop in order to express an iterative computation which is as we have seen already very (Refer Time: 31:25). Secondly, this will be very very slow because every time you write you your hadoop job or your map reduce job actually takes as input, data sum from stable storage does the processing distributed processing and then rights data towards stable storage, ok.

So, it is very inefficient if you want to use some iterative algorithm and many of the algorithms are in machine learning for example, are iterative algorithms. The other problem is when you have a lot of interactive tools ok. So, hadoop was designed for batch processing. So, the processing was something like you have a certain amount of data and you want to extract some information out of that data in a distributed and

scalable manner. So, you take the data as the input and run the map reduce job. However, what if you want to first extract a certain amount of information from the data and then you want to refine that.

So, for example, first you may want to see the minimum and the maximum temperatures for a particular country and then once you have found that out you are the answer you are looking for was not there may be you want to find the minimum and maximum temperature for a certain the cities of one particular country. So, the data that your processing is the same ok, but you now want a different set of output in that case you have to run a different map reduce job. However, as we shall see in spark you can do it much much faster. So, this is the interactive computation, ok. So, we will see how we are able to achieve all this in spark in the next lecture.

Thank you.