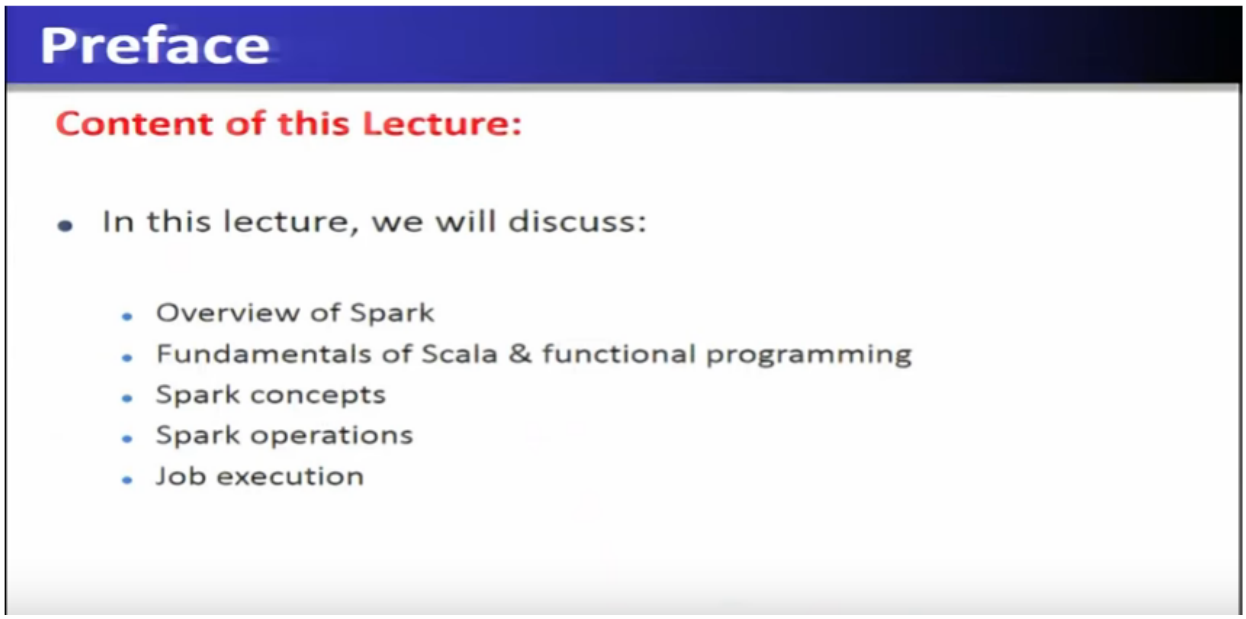
Lecture 09
Parallel programming with Spark

Parallel programming with the Spark.

Refer slide time :(0:17)

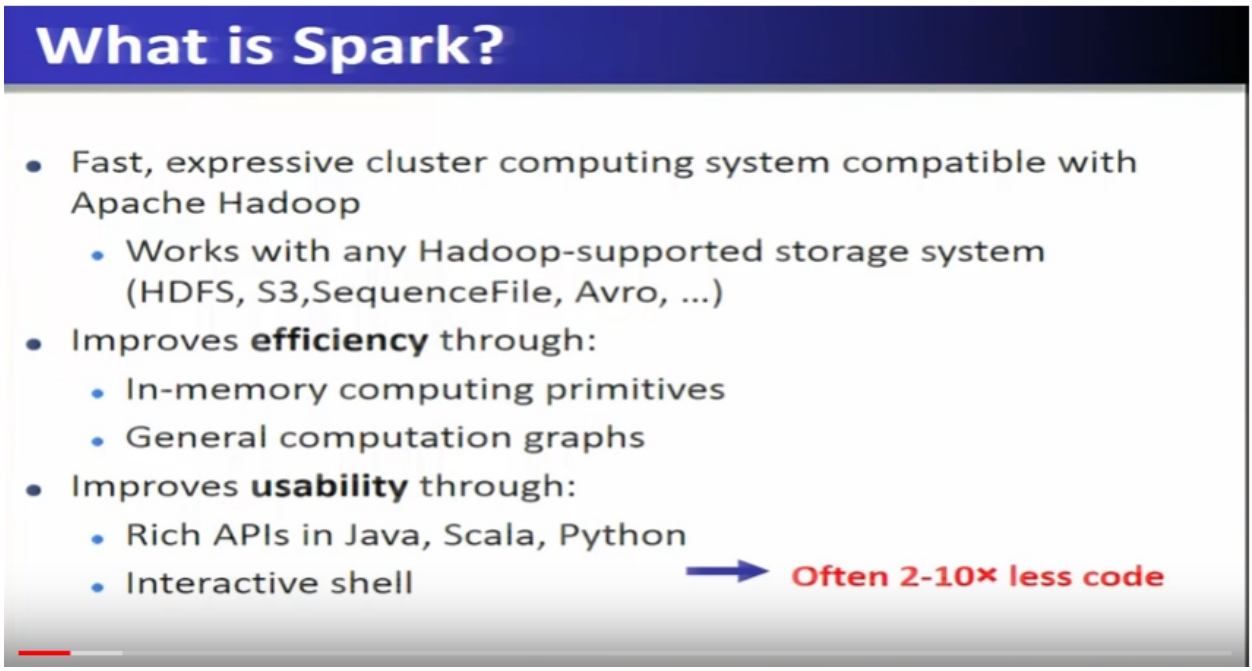


Preface: Content of this lecture. In this lecture we will discuss overview of Spark, fundamental of scalar and functional programming, Spark concepts, Spark operation and the job execution.

Refer slide time :(0:30)

Introduction to Spark.

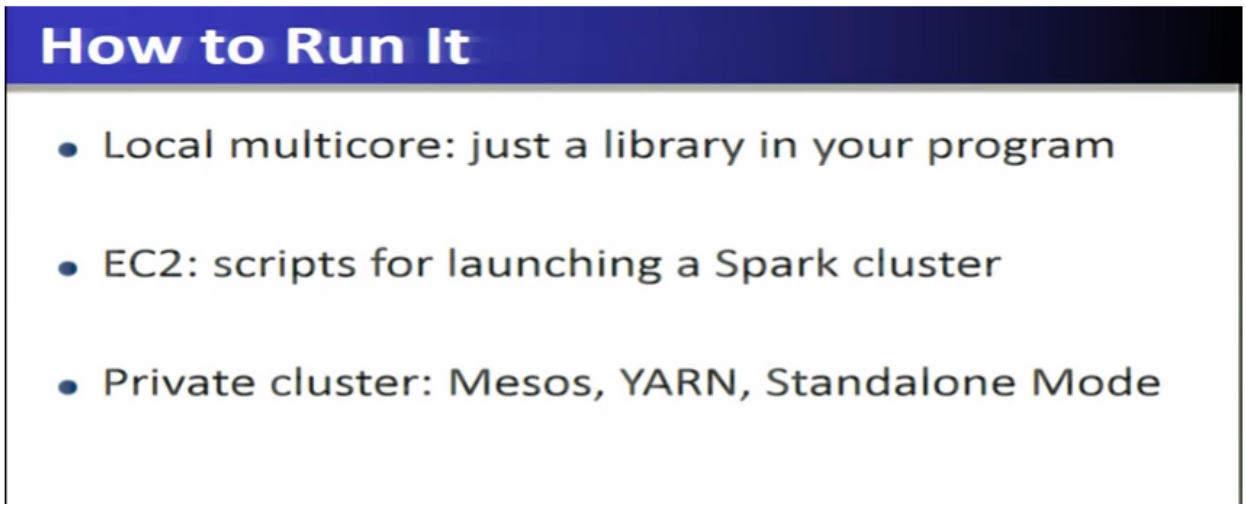
Refer slide time :(0:32)



What is this park? It is fast expressive crystal computing systems, which is compatible to Apache Hadoop, now this particular Spark system works, with any Hadoop supported storage system such as HDFS s3, sequential file. And so, on which improves, the efficiency through in-memory computational,

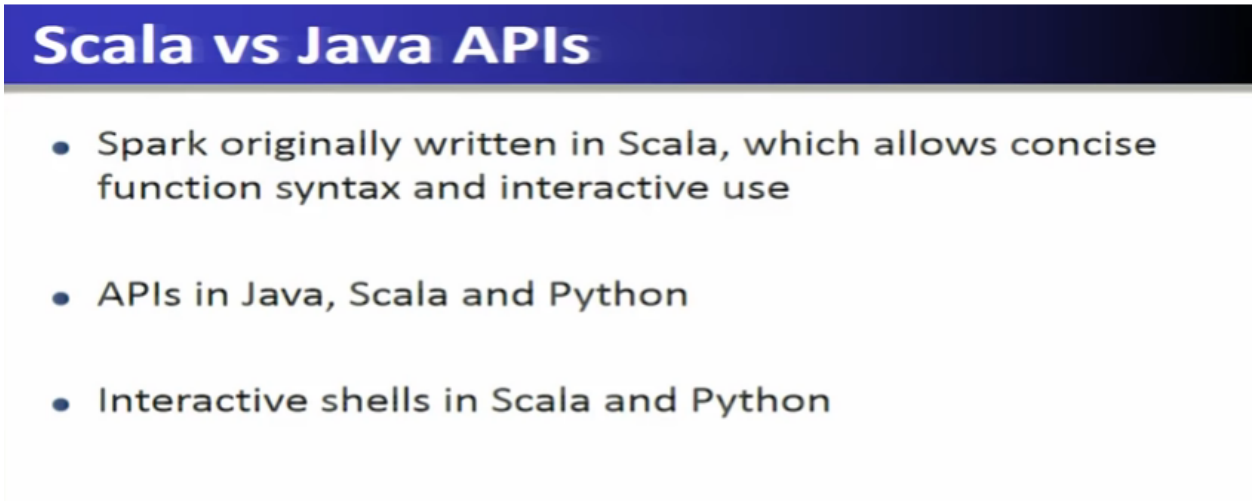
primitives and general computational graph. So, it also improves the usability through rich collection of API, is in the form of scale a Java Python, it has the interactive cell. So, this all comprises of the Spark, scenario. So, using this in memory computation, it is 100 times faster, compared to the previous generation MapReduce systems and also with the interactive, shell it has reduced up n 2 to 10 times less code here in this particular system.

Refer slide time :(01:44)



So, how to run it basically using local multi-core system or using with the, with a private cluster using Mesos YARN on and standalone mode

Refer slide time :(01:57)

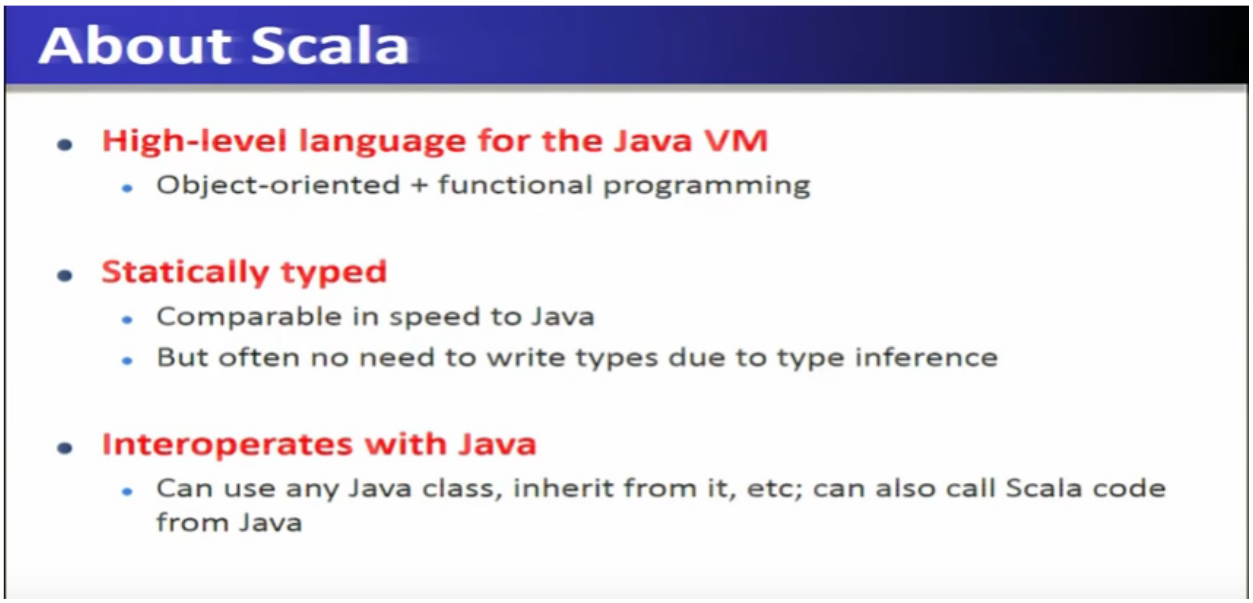


So, Spark originally was written in scalar, which allows concise function that is syntax and interactive use, there are APIs, switch are available for, Java and Scalar and Python now. So, interactive shells are available in a scalar and python.

Refer slide time :(02:18)

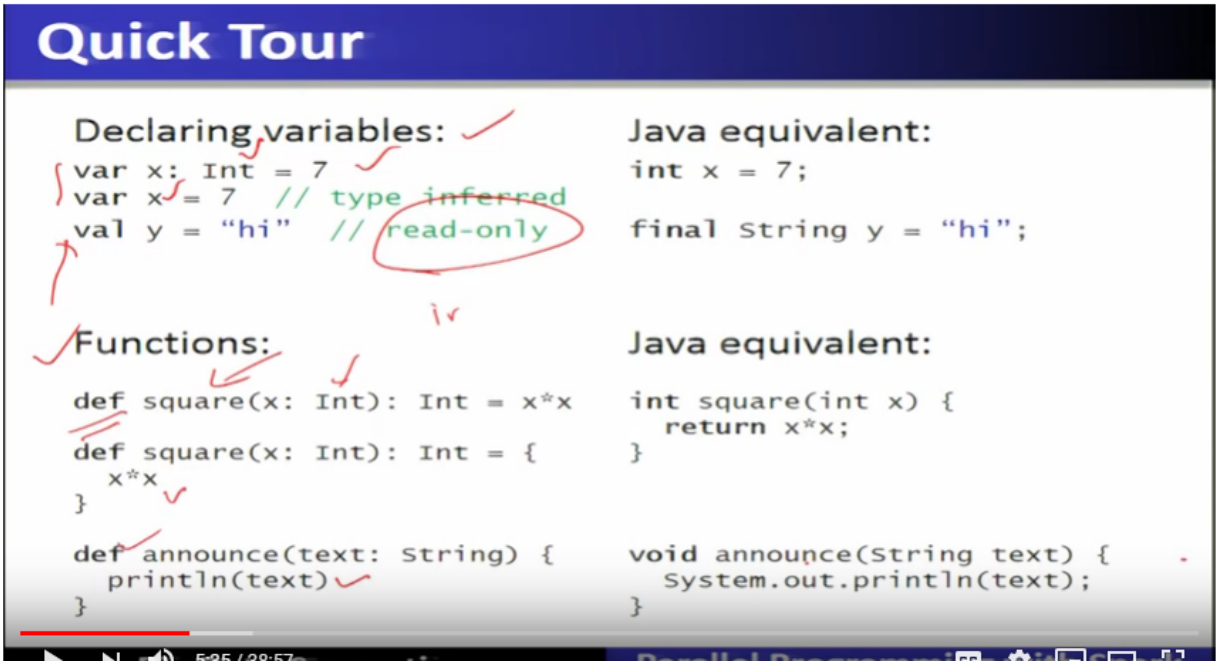
Now let us introduce, to the scalar functional programming language.

Refer slide time :(02:27)



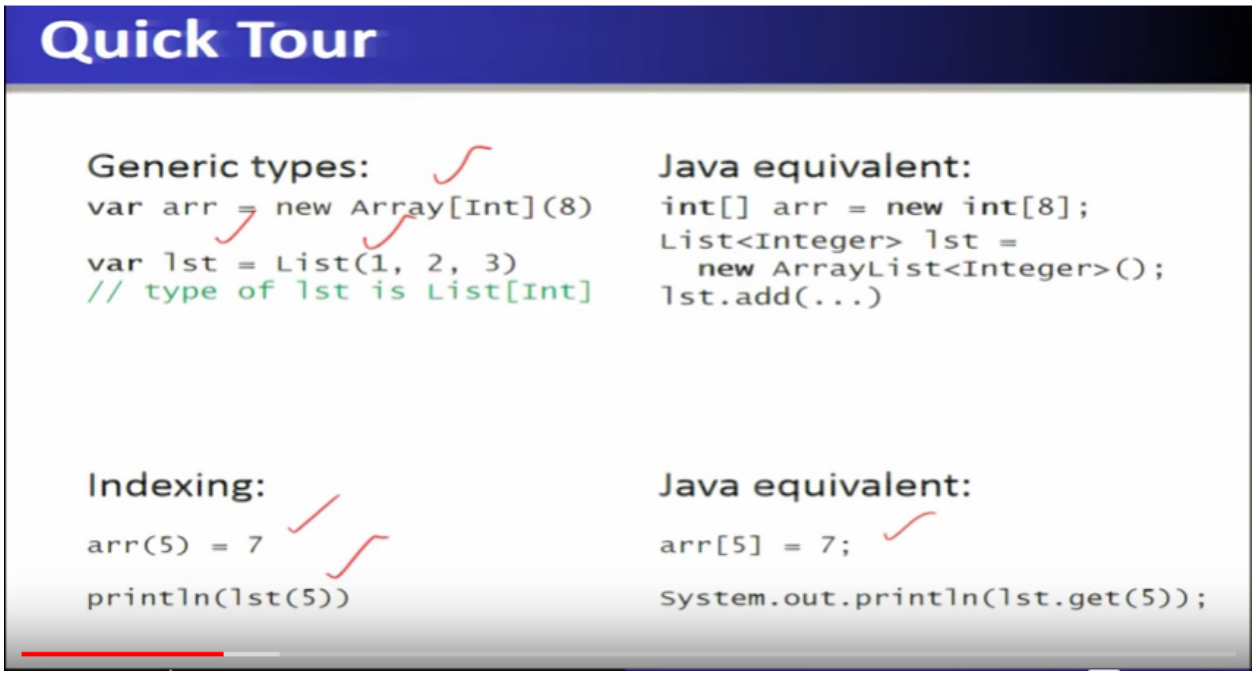
So, it is a high-level language, for the java virtual machine, that is it can compile through the java virtual machine, byte code and it is statically, typed that is and then it is interoperates with the Java.

Refer slide time :(02:46)



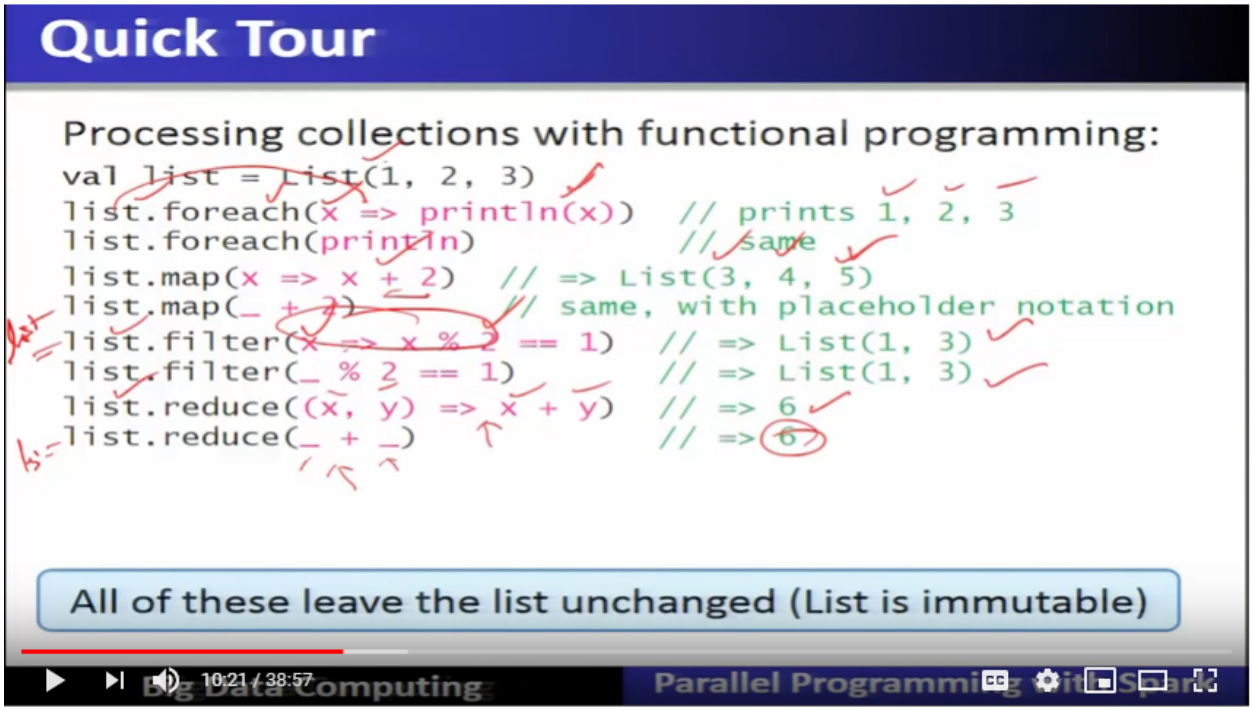
And so, here we are going to see the quick tour, of functional programming language, that is the scalar, in scalar the variables are defined using, where function, where there are two ways, you can define the variable one is by specifying the type of the variable, that is X of type int and our without is specifying the type also you can define, the variable such as X is equal, to X is equal to seven or you can also say where X assign int ,we have to specify the int. so, if we are not specifying the type automatically it will discover the type and if we say that X is equal to eight it will accept, it but if you say X is equal to let us say, is then it will not accept, because there will be a type mismatch. So, automatically, identifies the type here in this particular case. Now another way you can specify the variable is, using Val function, Val function is read only, that means it is immutable, you cannot change, the values here in this case now another thing is after declaring the variables, the next thing is called,' Functions'. So, functions in scalar can be defined, using def function and the name, of the function and the arguments in it and then we can supply the return values and for example, return is of type, in to X cross X. So, it will return the values so, there are different flavors. For example, if we want to only specify the return value, so we have to say the curly braces empty and X into X will be returned, in this particular case now if you want to return the void value, don't displace apply the type, then in that case after def only within the curly braces, whatever we write down. So, here that will not specify or it will be a void type, of a return value, that is equivalent to the, the Java functional programming here. We have to specify the void here automatically if you don't specify the type, then automatically it will take the white and rest of the commands will be written enclosing, with within the curly braces. So, we have seen the how the variables are declared how the functions are defined using def function. And the next thing is,

Refer slide time :(05:41)



is the generic types. So, we can specify the type array, as this array of int 8 type. Similarly we can also define, the list very simply, as a list of items, list of 1 2 3 4 and this is this type is of type, of first is the list of int values, here in this case, similarly we can also specify the induction. So, this is equivalent to the, Java equivalent, where array under this square bracket 5 it is written here we have to specify the, indexes that is array, at the fifth element we are storing, the value 7 in this case and if we print, the list of its fifth element, then it will print the value, which is stored in the list.

Refer slide time :(06:33)

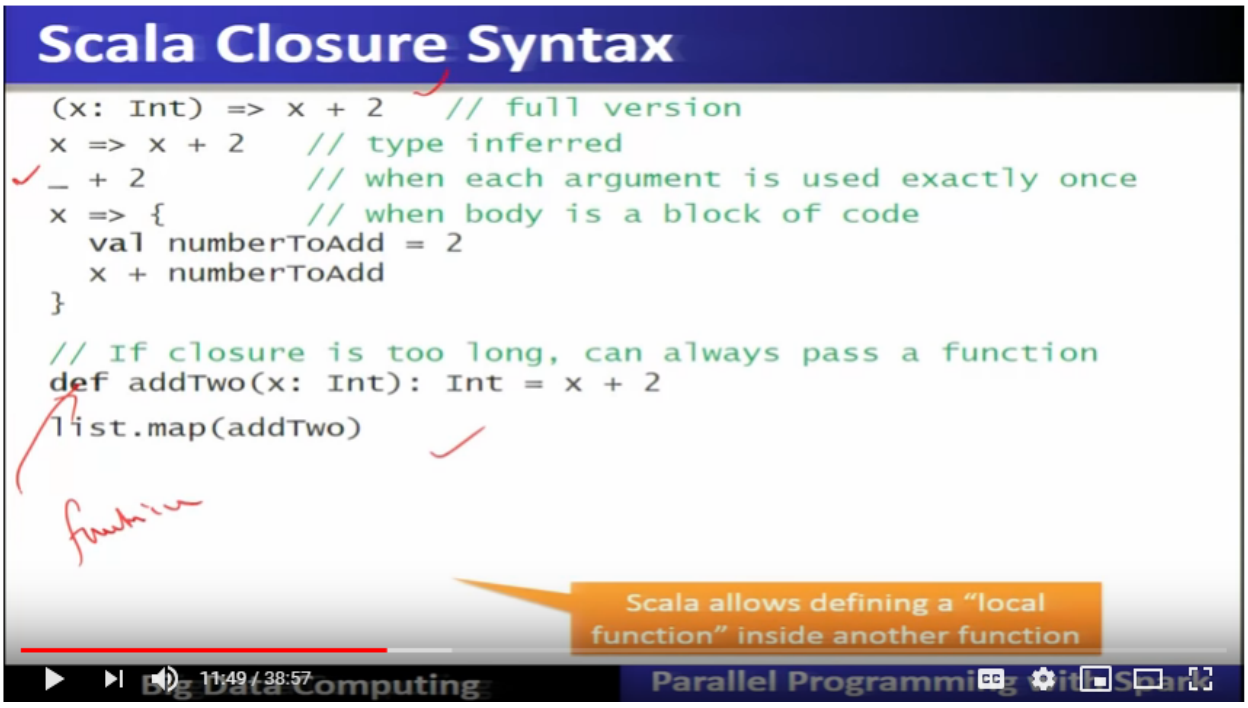


Now processing collection with the functional programming, here we can see that, it is very easy to define the list of values 1 2 3 4 as a list and then using, for each command, that is on Lister we can scan through, this particular function for each element of that particular list and for, here we can specify function, on every element. So, this says that for, every element of this list that is X we'll apply this particular function, print ln. So, hence it will print one two three that means all these values of, the list will be printed. So, the short form you can write down like, this for each list, dot for each we can specify the print ln and it will do the same things. Similarly we can also apply, we can perform the map operation on every element of the list and this map function says that on, every element of the list, you go to add plus two on it so, one two three so, if you add plus two on one it become, 3 2 plus 2 that is 4 3 plus 2 that is 5.

And the same thing we can specify using, this underscore notation, that it says that take the first element, of this take the element, from the list and add to it, it will be denoting the same thing so, therefore using this particular map function applying, on each and every element of this list and we have to specify, what

we have to do and this way this is supported. Now we can apply the filter on it. So, that means every element, of list we can apply the filter function and filter function says that, the for every element of X and if it is odd number then, then it will print, the list of items. So, list will return again a list similarly here again, it will return a list after filtering it and instead of saying this full statement, we can specify in a in a shortcut, in a shorthand that is underscore percentage, to that means it will it will check, whether it is even whether if it is odd, that means by saying that mod of 2 if it is 1 that is it is odd number, then it will print in the list as it is. So, again there is another function which is called a,' Reduce', in scalar. So, this reduced function what it will do, it will reduce on these elements x, and y is the pair of elements and it will give x plus y. so, this particular reduced function, is when it is applied on the list of elements. So, you can see that, it will add all of them and the value of 6 will be given, this reduce same thing instead of writing this complete expression we can also specify in a shorthand. So, that means it will take all the arguments and do the summation and it will return a list. So, it will be a list which contains only one element, it will perform the reduction operation. So, all of these will leave the list unchanged, hence the list is immutable.

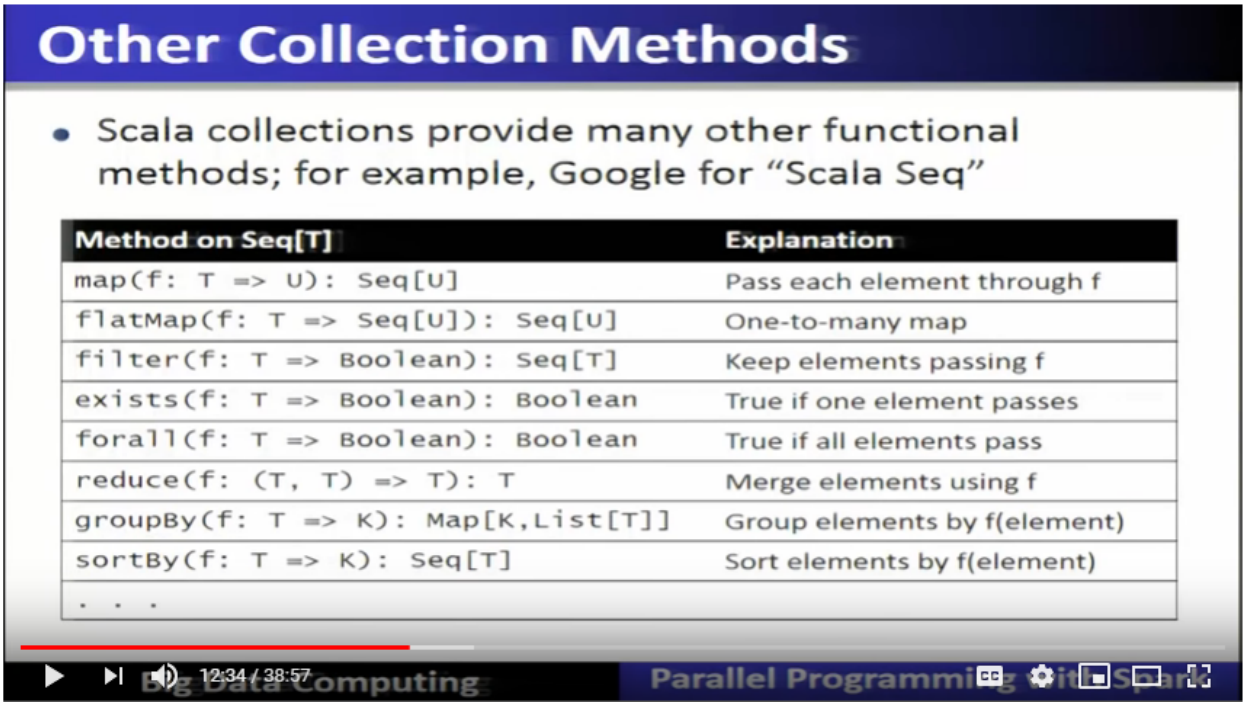
Refer slide time :(10:25)



Here in this case now let us see the scalar closure syntax more of it. So, this we have seen the full version, that is x : into will return x plus 2 and this is the type input, that is x assigned to x plus 2 that is type in front and here, the first argument plus 2 each argument plus 2 has plus 2 is also doing the same thing and x will be equivalent to, s assign X will be applied on, well number to add this is equal to 2 X plus number to add this also will be applied, on these functions. Now if the closure is too long, can always pass a function, and here in this case, we can see that using depth, we can apply that means we can define his a it is a function and add 2x colon is integer and int is equal to X plus 2 here in this case. And it

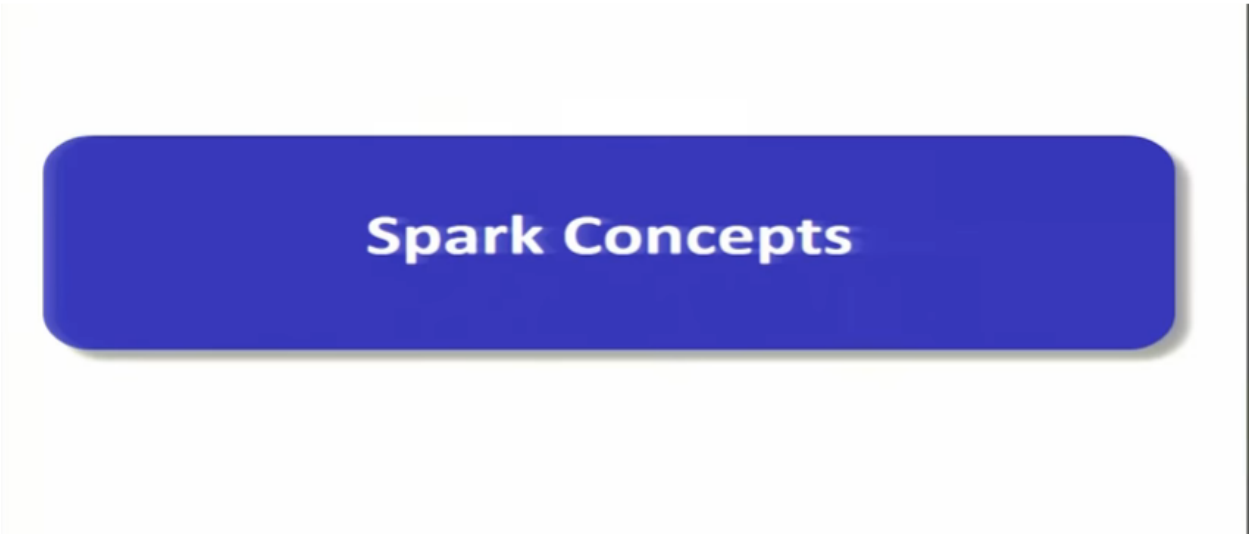
will now perform on this list the map add to, this particular same thing, we can now perform using. So, a scalar allows defining a local function.

Refer slide time :(11:54)



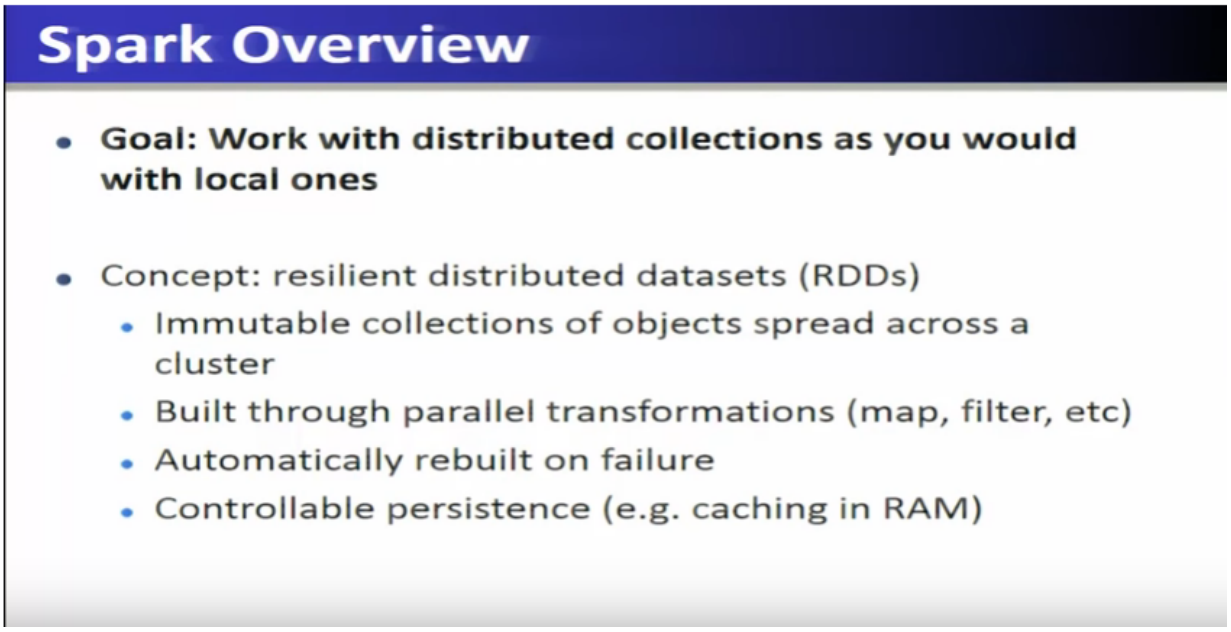
Inside another function, there similarly there are other collection, methods which are available in the scalar. So, a scalar collection provides many other functional methods, for example as Google for a scalar sequence we can see on it. So, map function is can be applied and this will pass each element, through a function f and similarly flat map one-to-many map function, it will apply similarly for the filter it will keep the element passing F and exist means a true if one element passes, for all reduce group by and sort by, all different methods are available.

Refer slide time :(12:35)



Now let us see the Spark concepts.

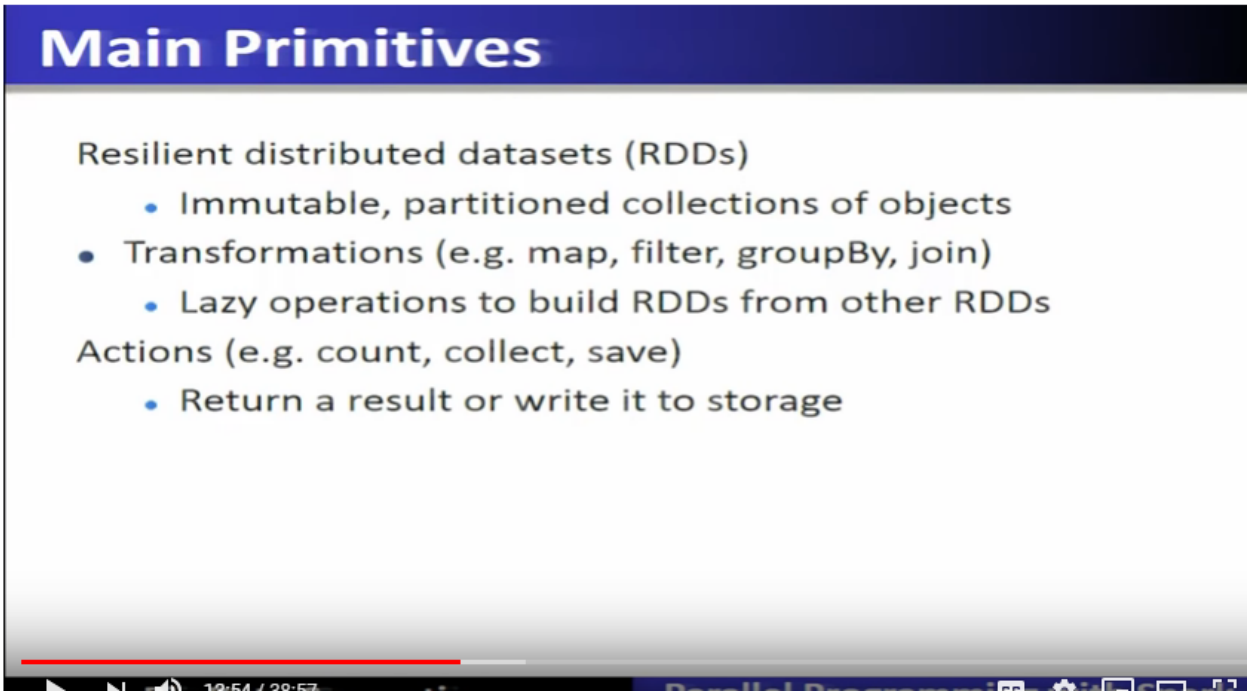
Refer slide time :(12:40)



So, here the goal of Spark is to provide a distributed, collection and here the concept of Spark ,is to support, this distributed computation in the form of resilient, distributed data sets and RDDs are immutable collection of objects which are spread across the clusters and these are d DS they are built through a transformation, such as map filter etcetera and these particular, RDDs they are automatically

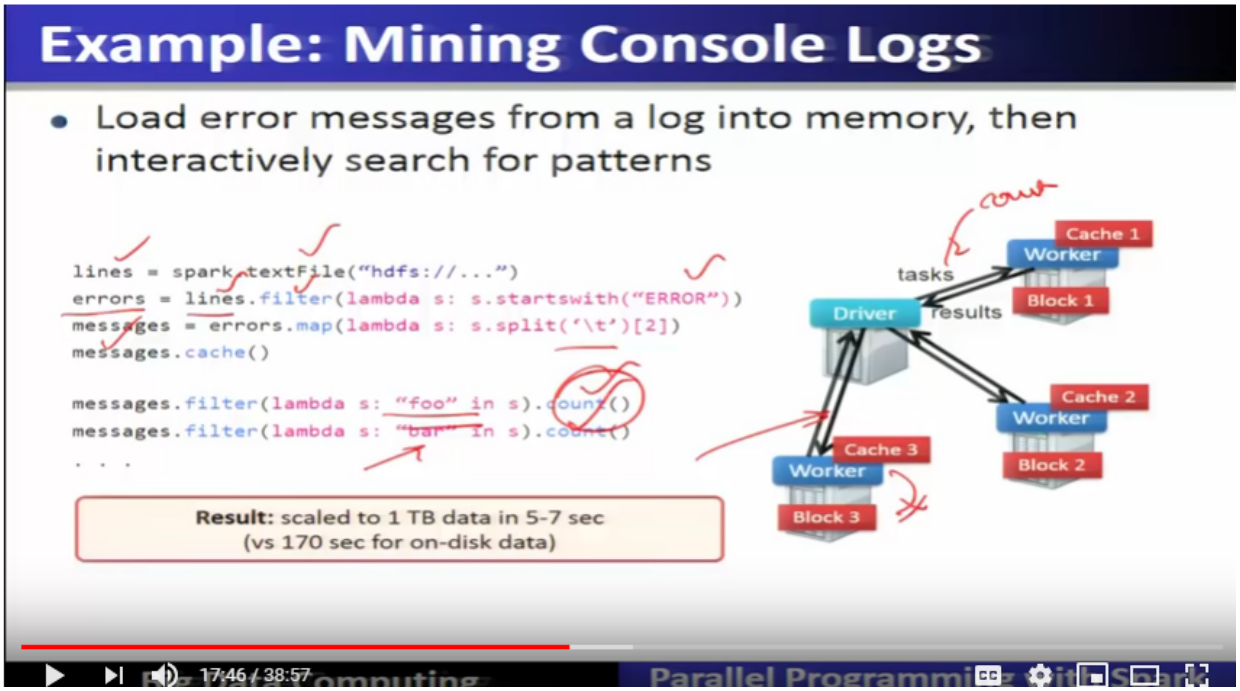
rebuilt, on the failure, that means a lineage is automatically generated and whenever there is a failure it will be reconstructed, similarly it is also controllable, persistence that is the cache.

Refer slide time :(13:29)



In the rhyme is being done. So, the main primitives is RDDs which are immutable partition collection, of objects various transformations are applied on the RDDs such as, map filter and group by join and these are all lazy operations to build RDDs from, other RDDs and besides transformations, actions also can be defined on our RDD such as count collect, save and it will return the values or it will write it on the under disk.

Refer slide time :(13:58)

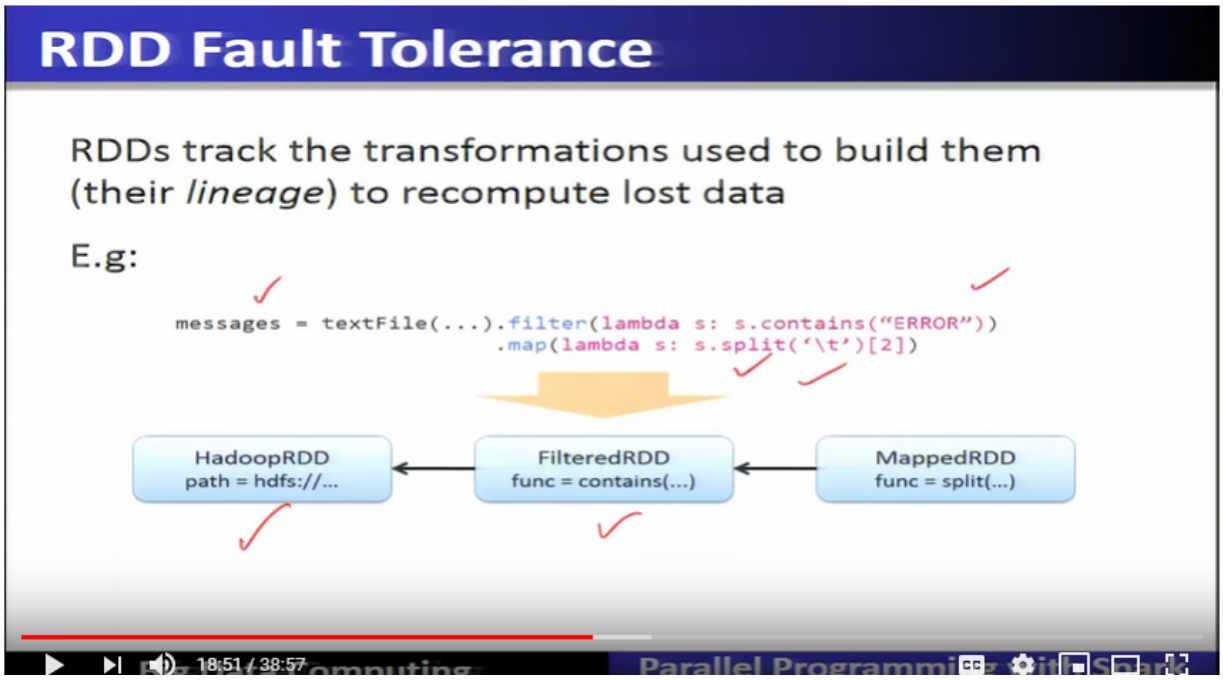


Now let us see one example of, of mining the console logs, using the scalar program written, in Spark and here, it what it will do it will load the error messages, from a log into the memory and then interactively search for a pattern. So, let us see that, the Spark program now you, you, you know that it runs in a form of a master that is the driver and the worker, system and the first line written is to read this particular log files, that is in the from the HDFS, it will read the text file and this particular lines after reading it, it will become in memory .so, it will generate the, the base RDD after reading the file and then what it does is apply the filter, on the lines which are now red, into the in memory ,it will apply the filter operation that we have seen, on the lines and in this filter operation, what it does is it will identify the, the, the errors which are appearing, in the in the in the log files and this will perform a transformation, after filtering and they are collected in the form of, another RDD that is errors RDD and using this errors, RDD down we can apply, the map function and which says that using this map function. Now you can split this, s to the wherever the taps are there and they will be now dividing, these into the different error messages and these error messages are now cached.

So, apply the filter on these error messages that wherever this, foo is available or appearing, in the message and you have to make a count, of it how many such foo messages are appearing, in this error message. So, here the count is the action and before that all were transformations. So, let us see how it will be done. So, wherever these messages are restoring on that, this foo will be counted so, just see that the driver will perform a task, to count. So, task is to count, this particular task will be communicated by the driver to all the workers and they will count and then return the result back to the driver and also will be in the cache, these messages will be in the cache. Now the next time when you want to filter the, the particular string that is called a,' Bar', from, the message then, it will be performed in the cache itself it will not go and look up into the into, the disk it will not be done in the disk, it will be now returned back from the from the cache itself. So, the results will be quickly returned back because so, therefore the full, text search of a Wikipedia can be done, in a less than one second if it is in the cache or, or it is 20 second

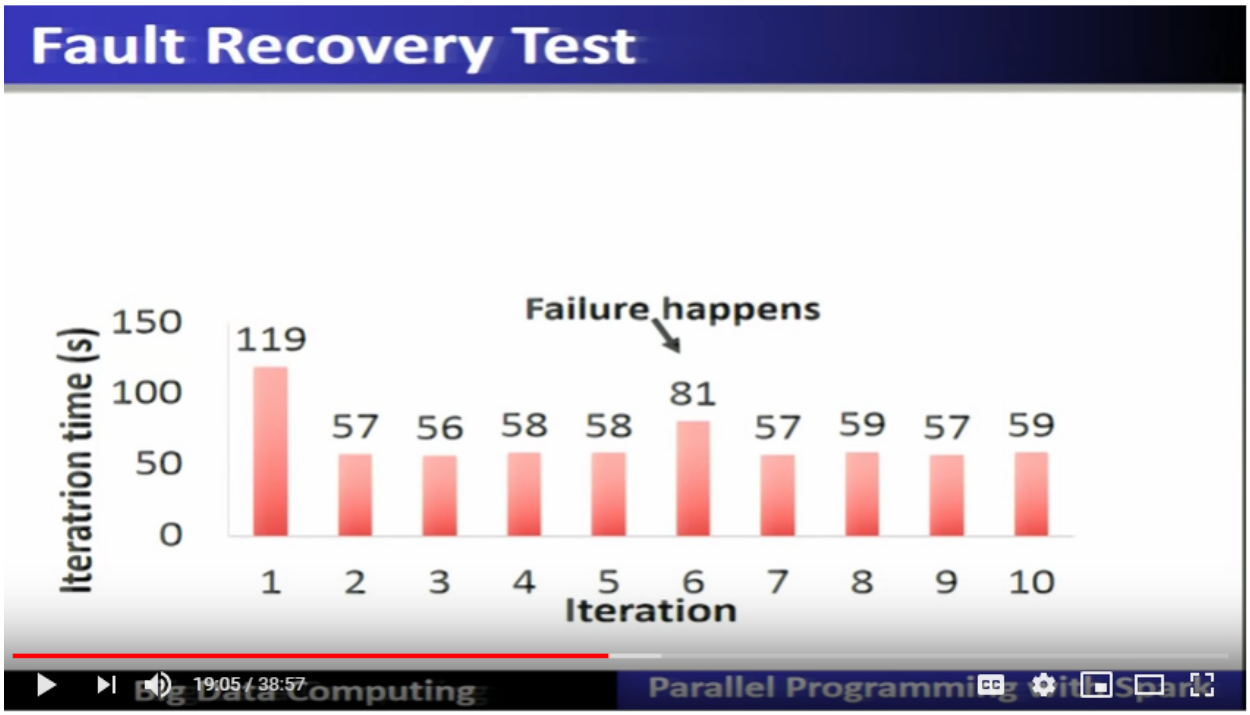
if it is done through the disk access. So, therefore if the data is scaled to one terabyte, then it will take five to seven seconds, if it is to be accessed through the cache or it is 170 seconds if it is on disk data

.
Refer slide time :(17:49)



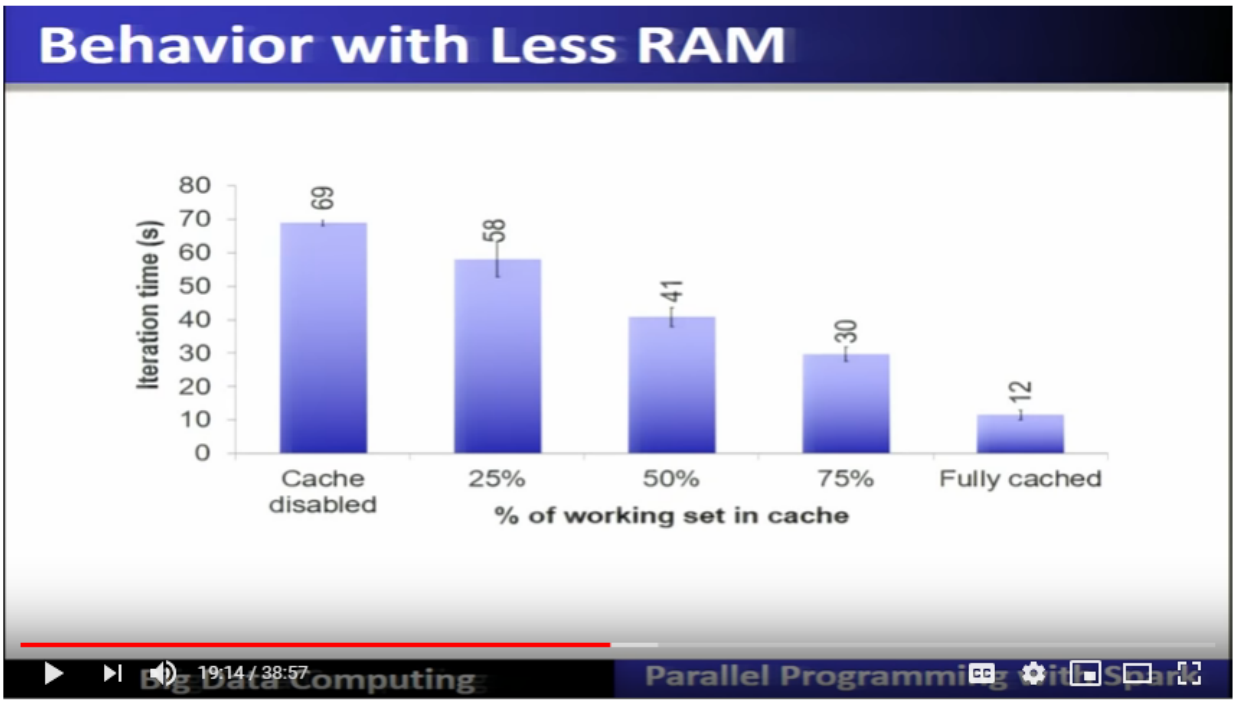
So, in all cases this entire big corpus of terabyte of data can be processed very, efficiently and in a very quickly. Now let us see the RDDs fault-tolerance. So, RDDs will track the transformations, used to build them through the lineage to recompute the last data. So, here we can see that, once we specify these filter that is using, filter which contains the error and then it will split, all those messages, which are tab separated and this will be collected in the form of messages. So, let us see that, this particular filtered RDDs, will contain the information and they are now stored in the HDFS file system. So, RDDs keep track of the transformations, used to build them, their lineage to recompute the last data.

Refer slide time :(18:55)



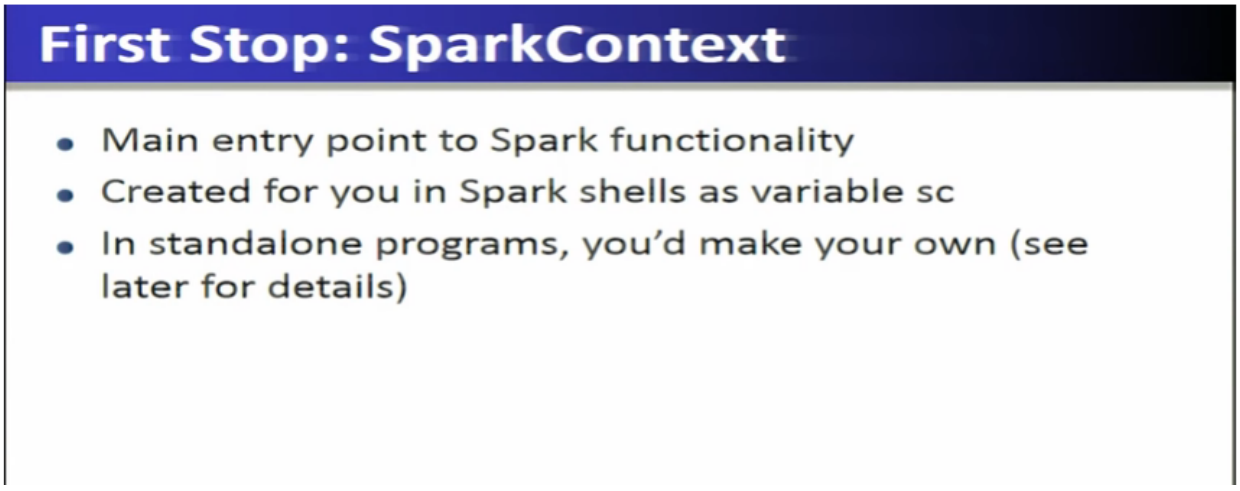
So, fault recovery, test if we see here is that so, the iteration time and the failure happens, is recovery is quite efficiently and done.

Refer slide time :(19:09)



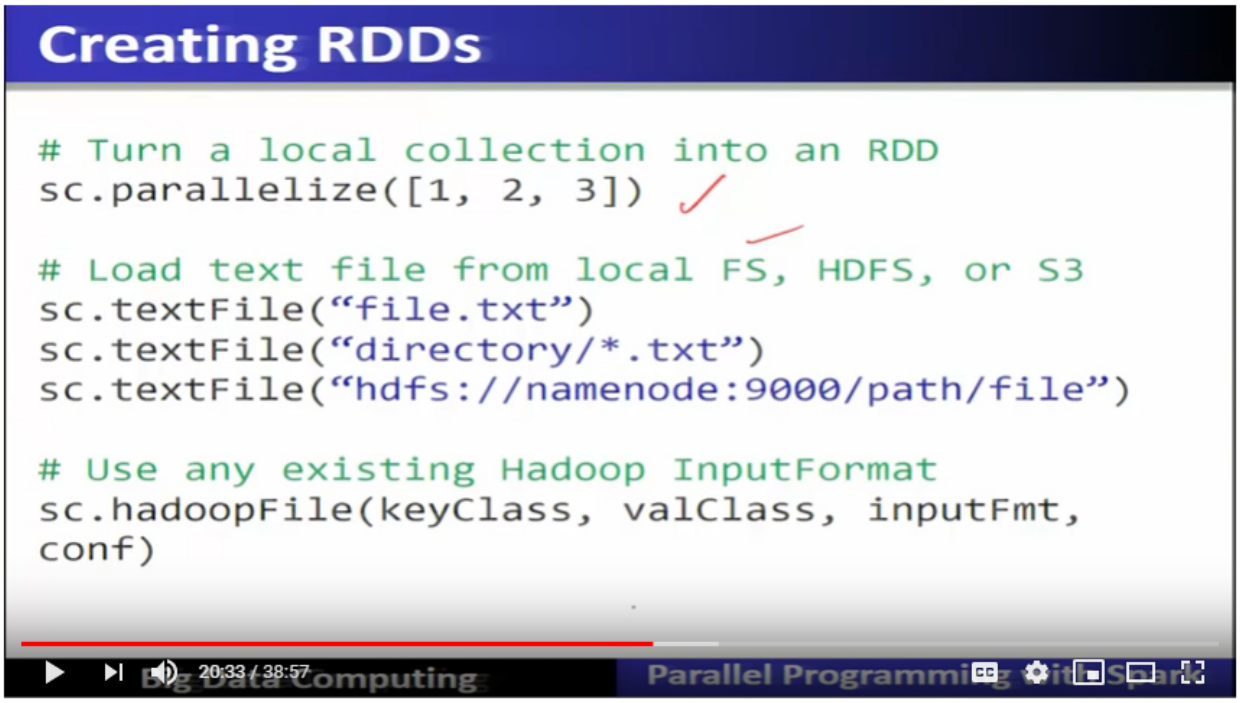
Now behavior with less RAM, can see that if it is fully cached, then it will iteration time will be quite less here in this case.

Refer slide time :(19:18)



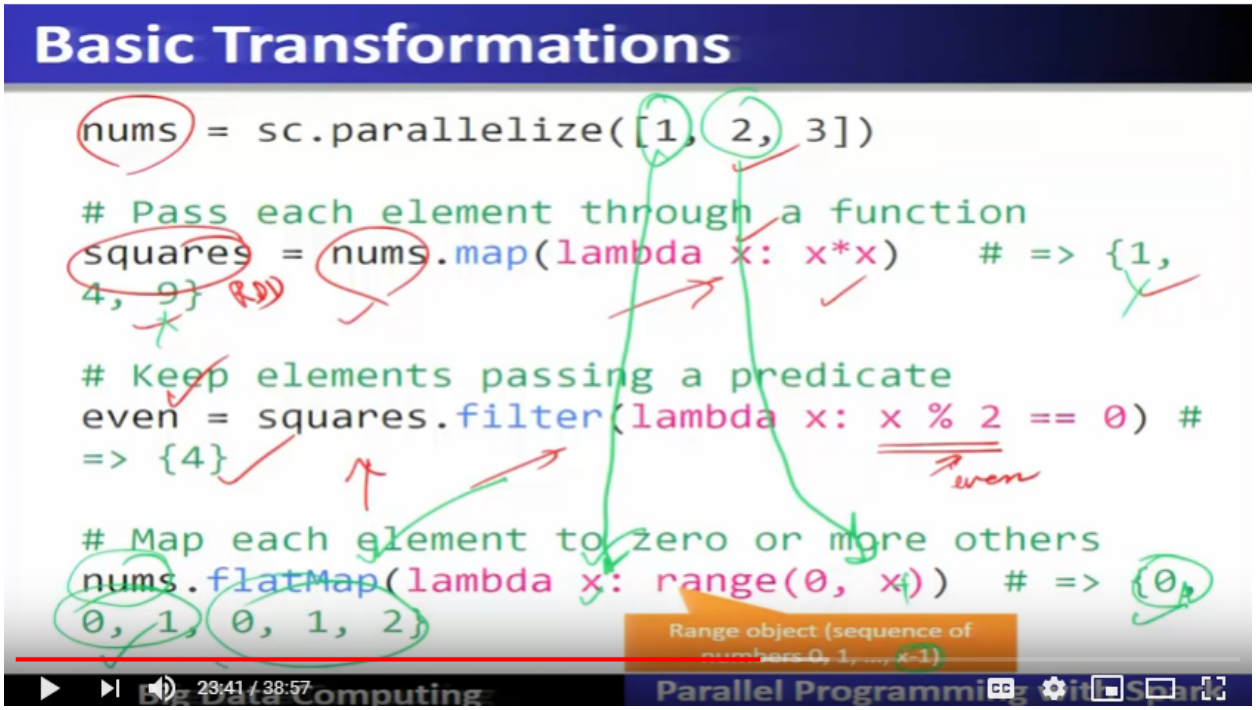
Now the question is what language you can use, obviously a scalar will be performing, better one let us see the two of Spark further operations. So, easiest way, to use the Spark is by the interpreter over, the

shell and it runs in a local mode with, only one thread by default what control, can be with the master one and cluster. So, the first stop is through, the to the Spark context, is the main entry point to the spar functionality which is created through the Spark shell.
Refer slide time :(19:18)



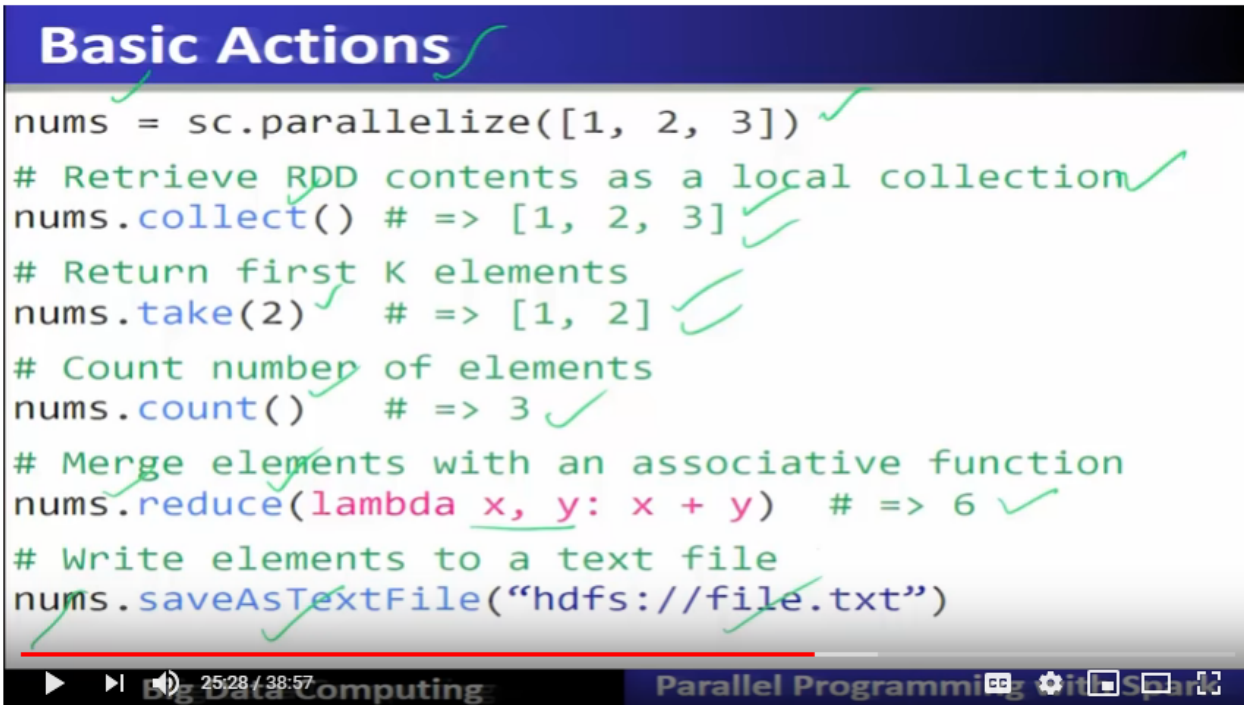
And let us see how to create, this entire operation that is so, first thing is we have to process, this particular list and Spark context will turn the local collection into an RDD and then it will note the text file from local file system, HDFS or s3. So, that is called,' Text File SC', dot txt file similarly we can also use the existing Hadoop input, format for the creating RDDs that is reading.

Refer slide time :(20:37)



The data and creating the RDDs now the basic transformations are shown over here, which are provided in the form of a Spark, is not only we can create the RDDs, now we can perform, the functions on these RDDs using map function and we can apply, this operator operations and these operations are this saying that for every element of these numbers, we have to apply X star X means multiplication operation, has to be done. So, just see that 1 2 3 is basically the, the numbers. So, the so if we can take the square. So, it will become n square RDD, similarly we can keep elements passing as a as a predicate and we can see that we can apply the filter, on the square, saying that if it is even number that is X modulo 2 becomes, 0 that means it is checking whether it is even number then only it will, output in the even number, so you see here out of 1 4 9 what is even number? Is the four and nine and one will not be will be filtered out, only four will be passed. Finally the map function on each element, each element to generate zero or more others using flat map operations. So, flat map on the numbers so, the number says one two three four one two three if we apply the flat map which says that the lambda X is in the range 0 to X and 0 to X it will. So, X means for example the number 1 from the list if it is taken. So, it will generate the number in the range 0 to X means 0 to 1. So, it will generate 0 it will generate 0 and then, then the next element is 2 it will generate 0 1 and 2 0 and 1 it will generate and finally 3 means 0 1 & 2 it will generate 0 and X minus 1. So, the flat map will be, now is able to generate more than 1 elements, in this particular manner. So, the range of the values, is nothing but the sequence, of the numbers from 0 to X minus 1 it will generate and so here it will generate 1 4 ,4 1 it will generate 0 & 4 2 it will generate 0 & 1 & 4 3 it will generate 0 1 & 2 So, that is called,' Flat Map Operation'.

Refer slide time :(23:45)



And this is the basic transformation and now we are going to see some of the actions, we have seen transformations now we have to see some actions. So, actions means let us assume that we are given the list and this list, is now taken up in the RDDs which are called,' Numbers'. And it will retrieve the RDD content as a local connection, now in this particular, when we say collect it will now in store as a local connection and when we say that, return first K elements. So, it will take, two elements out of this particular, three elements and now when we say count, count is an action all these are action, when we say a count. So, it will output the value 3, in this case take 2 means it will produce a list, of two elements collect means it will print all the collection of all the elements, similarly when we say, the merge the elements with the associative function and we apply the reduce function, on the number wherein it says that XY is equal to X plus y that is all the numbers are to be added up and the value of 6 will be returned back by the reduce function, similarly we can also do, a save the elements, to the file save the S the text file and we have to specify the name of the file. So, all the all these numbers will be now saved this list that is RDDs in the form of numbers it will be saved into a file.
Refer slide time :(25:29)

Working with Key-Value Pairs

- Spark's "distributed reduce" transformations act on RDDs of *key-value pairs*
- Python:
```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```
- Scala:
```
val pair = (a, b)
pair._1 // => a
pair._2 // => b
```
- Java:
```
Tuple2 pair = new Tuple2(a, b);
// class scala.Tuple2
pair._1 // => a
pair._2 // => b
```

And now let us see how to work with key value pairs, now is part distributed reduced transformation X, on RDDs of the key value pairs and key value pairs means here, whether if the value if the pair is XY. So, we can specify and the pair with the first argument as a and the pair with the second argument as B. so, these are some of the operations.

Refer slide time :(25:59)

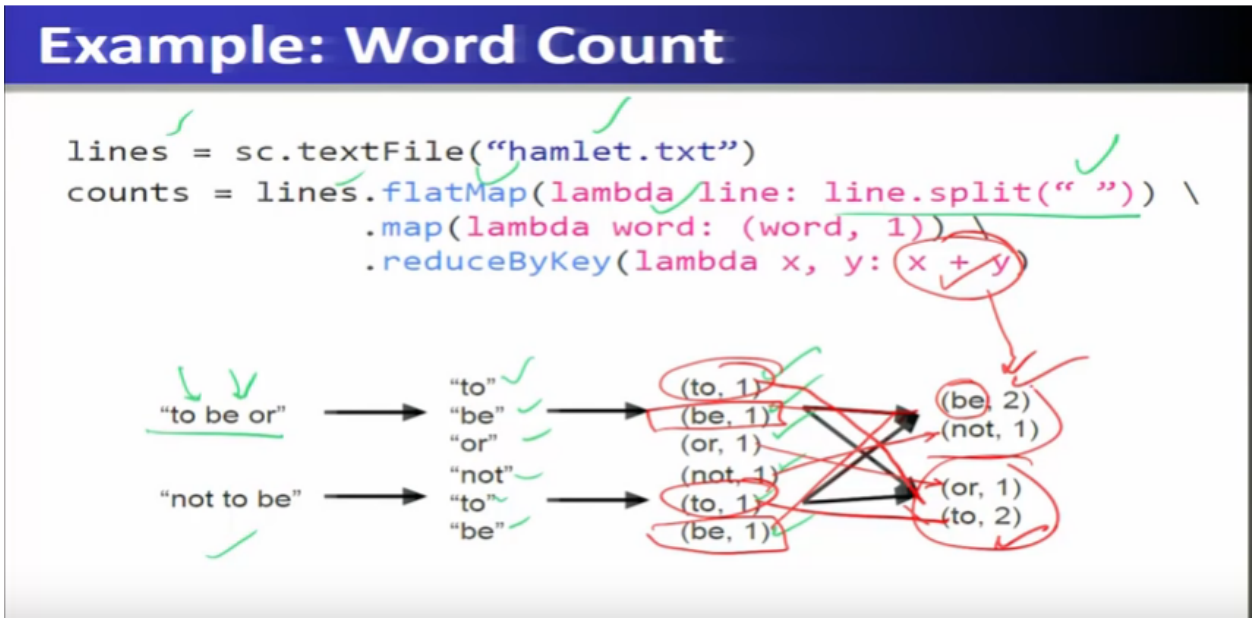
Some Key-Value Operations

```
pets = sc.parallelize([("cat", 1), ("dog", 1),
("cat", 2)])
pets.reduceByKey(lambda x, y: x + y)
# => {(cat, 3), (dog, 1)}
pets.groupByKey()
# => {(cat, Seq(1, 2)), (dog, Seq(1)}
pets.sortByKey()
# => {(cat, 1), (cat, 2), (dog, 1)}
```

reduceByKey also automatically implements combiners on the map side

Now let us see some more key value pair operations, for example we have the locally defined data, in the form of a key value pair cat with a value 1, dog with the value 1 and cat with a value 2 and this will be now generated as packs RDD and when we apply the reduced by key. So, therefore in the reduce by key, we will say that X Y is equal to X plus y. so, that means whenever given x and y. so, it will be reduced by X plus y. so, here in this case the, the cat will have two different objects so, he reduced by key, that is the cat is appearing twice and it will make the sum of these values X plus y for the values of the same thing key, similarly dog will be having only one key similarly now group by, key if group by key is there then it will say that, the cat will have the sequence, it will not add but it will generate the value, that is sequence one and two and dog will be having the sequence, one similarly when we say that sort by key then cat value one, will be sorted will be in this order and then cat - and then dog one. So, reduced by key also, automatically implements the combiner, on the map side.

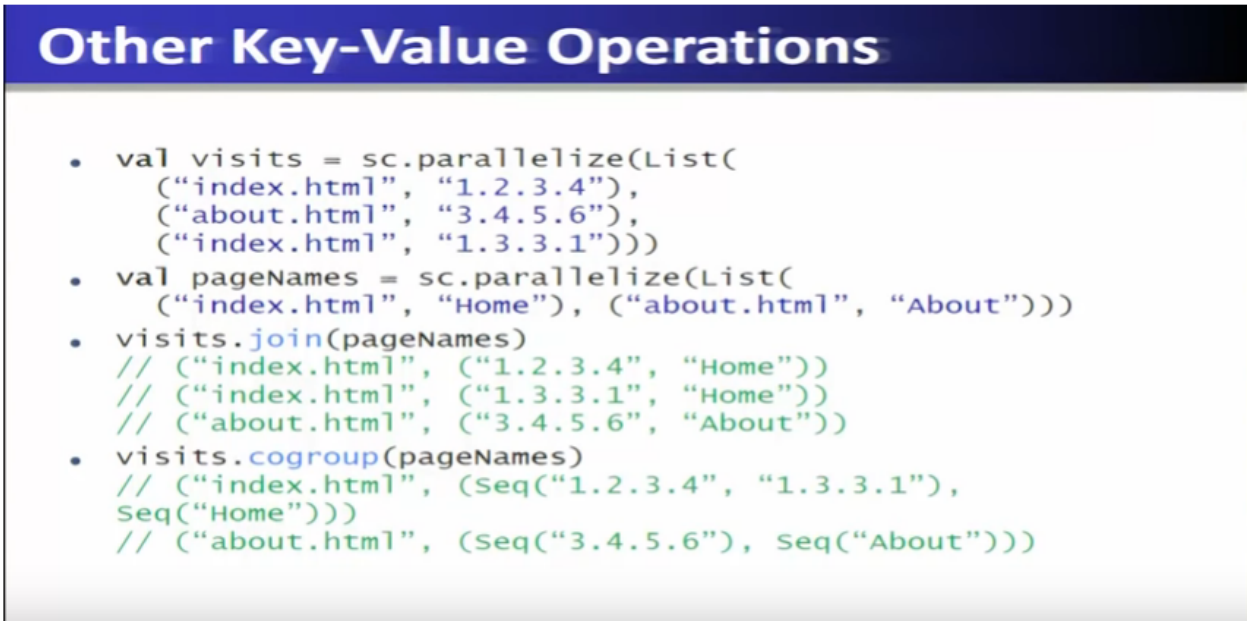
Refer slide time :(27:31)



Now let us see the full use of full execution of, the word count program. So, what count program means if let us say file contains all the lines and it will be read into, the lines as RDDs, then what we can do is when we apply the flat map operation, on the lines and this flat map will now split the line, based on the spaces. So, let us take this example to be or will have this space separated. So, what it will do is it will now, I split the line into the words. So, to be R will have three different words are generated similarly this another line will generate not to be, as three different words and then map function will be applied on these words. So, these words means map function, will what it will do it will for every word it will generate what comma 1. So, 2 it will generate 2 comma 1 B, B comma 1 or, or 1 not comma 1 and 2 comma 1 and B comma 1. So, after having generated this map function, now it will reduce by the key. So, reduced by the key in the sense it will say that for, a particular key it will do the summation. So, for the same keys for example B, is the key which is appearing in this particular, map output and here also this is the output, they will be collected back and that their values will be aggregated, according to the plus sign.

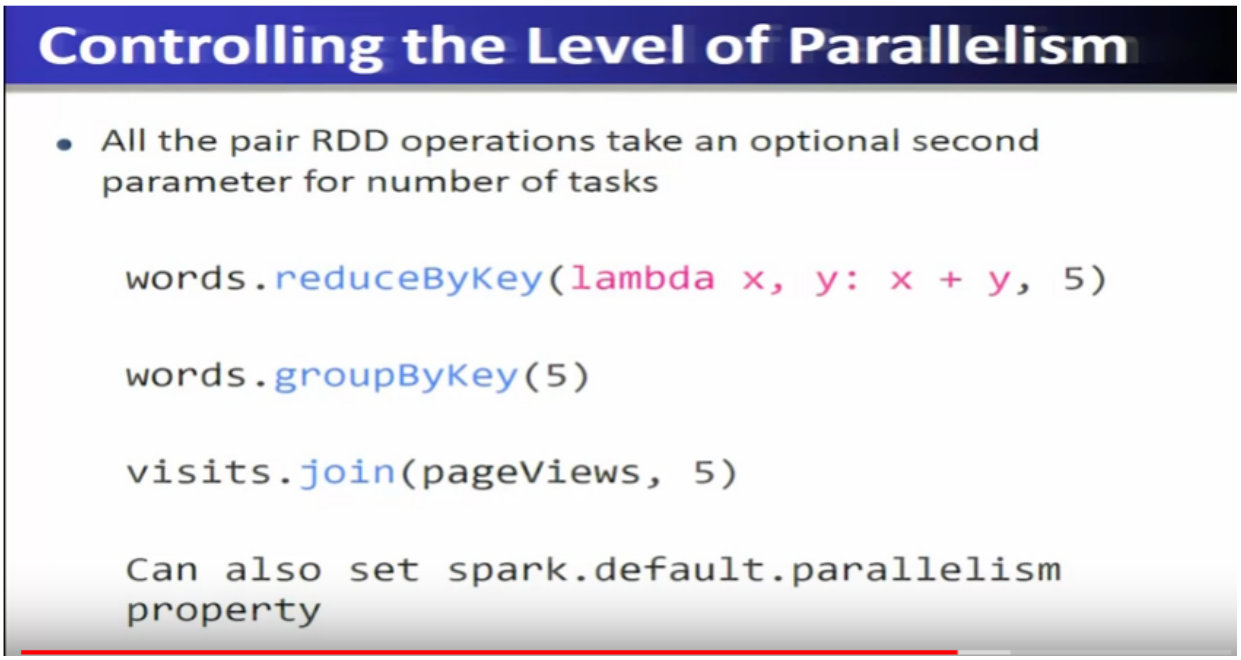
So, 2 will be there similarly, 2 is also appearing two times and 2 will be gathered over here and their values also will be, added up and whereas all others are appearing only once. So, naught will be collected over there and R will be connected over there. So, this is the reduced function, which will be applied.

Refer slide time :(29:41)



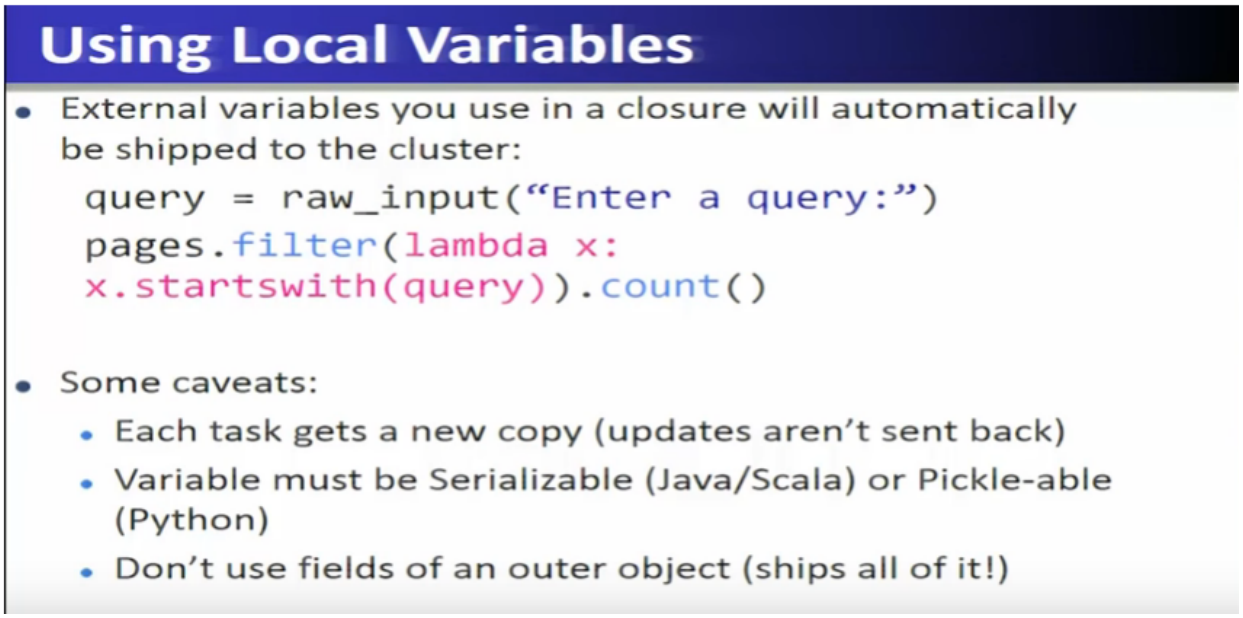
There are other key value operations and now we can see about these operations, which basically are mentioned, were here like, like join and Co group operations.

Refer slide time :(30:01)



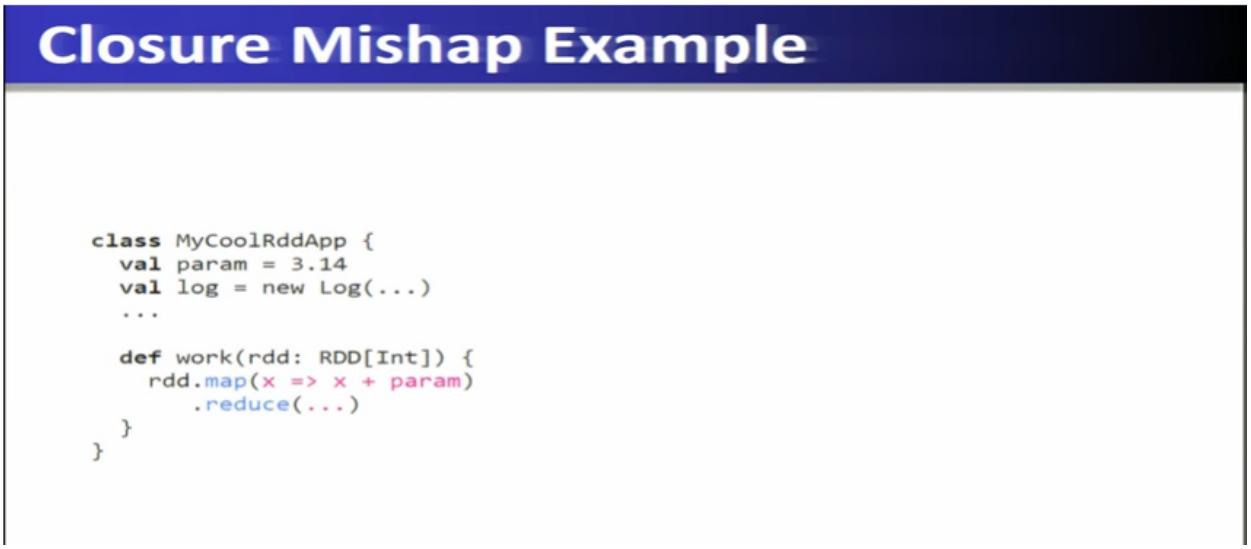
And now, we can all the pair's RDDs operations take the optional second parameter for the number of tasks and this we have shown over here.

Refer slide time :(30:13)



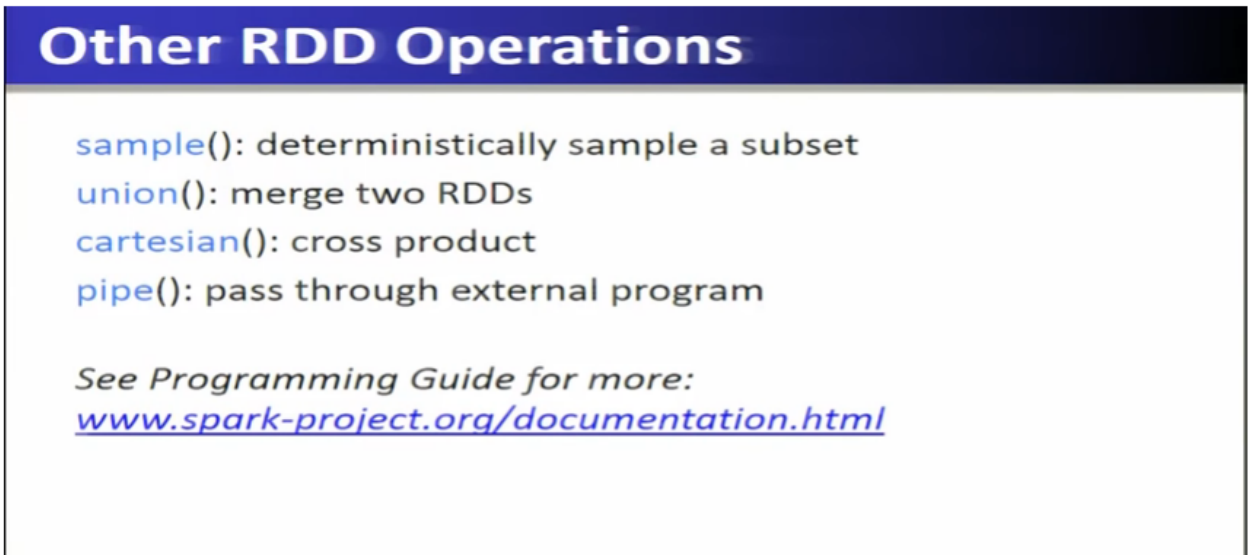
And external variables you can you use in the closure will automatically be shipped to the to the cluster.

Refer slide time :(30:24)



So, all these are the implementation issues.

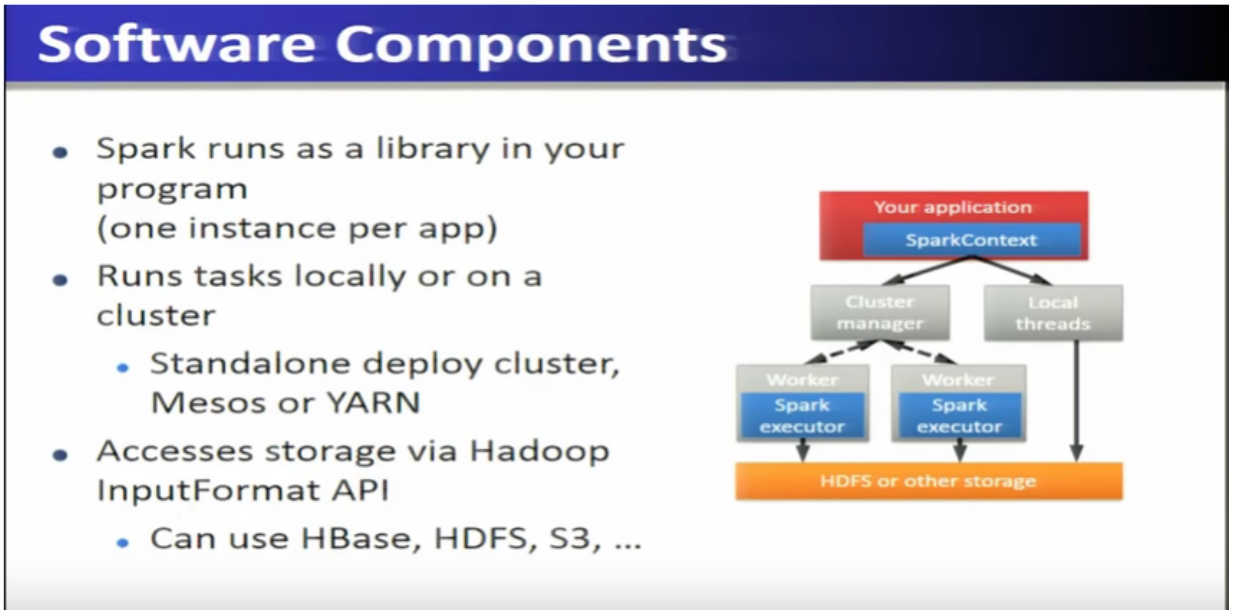
Refer slide time :(30:27)



Refer slide time :(30:29)

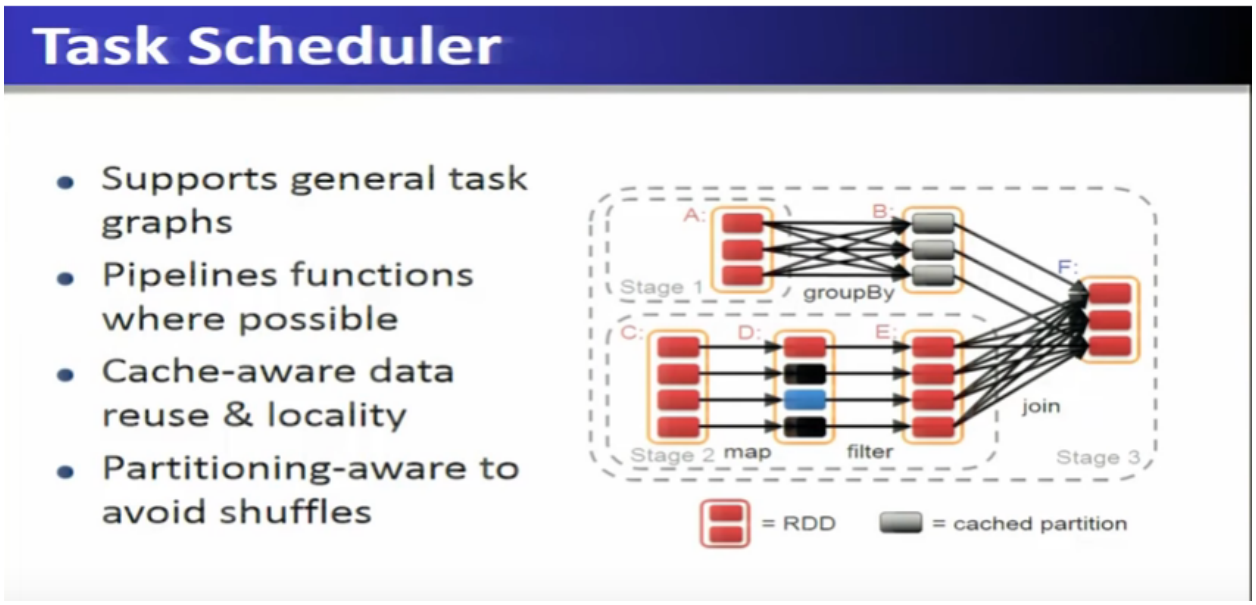
So, for other RDD operations the programming guide is there.

Refer slide time :(30:27)



And that can be seen now let us see about, the job execution in his Spark it has various software components and when you say Spark context, will be creating it will be created, in the master job master and then it will create the worker threads, in the form of power context and then executors will execute them.

Refer slide time :(30:56)

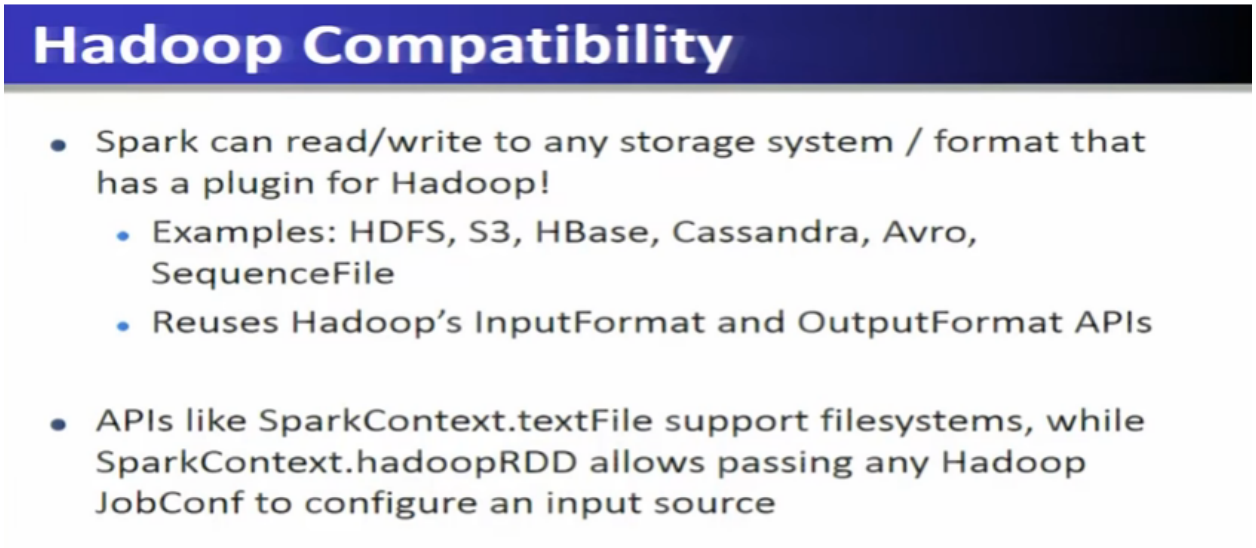


Similarly there will be at a scheduler Tasha, you will support the general tasks graphs, internally and pipelines will be also pipeline functions we are possible, they will be created by the task scheduler and cache aware data reuse and locality and partition aware things are done to avoid the shuffles.

Refer slide time :(31:21)

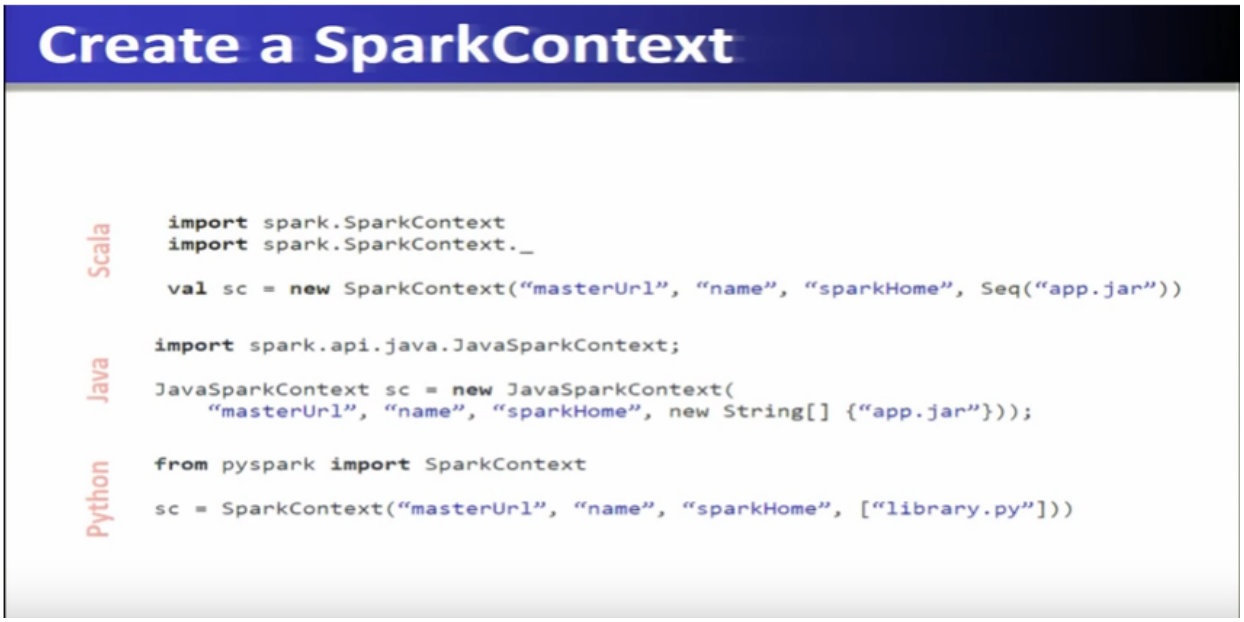
So, more information about resources, on Scylla is available.

Refer slide time :(31:28)



And let the Hadoop is Spark can read and write to any storage system format that has plug into the Hadoop. And API is like a Spark on text file supports, while the Spark context Hadoop RDD allows pass any Hadoop job, to configure the input file.

Refer slide time :(31:45)



And this is to create the Spark context, now Spark context when we say it has different arguments. So, the first one is called,' Master URL', is nothing but to create the URL or a local oblique in local node n. so, next one next thing is the application name and then, Spark will now install the path on the cluster, with the name is park home and finally the list of jar files also has to be given in the Spark context automatically it creates a Spark context using the Java jar file.

Refer slide time :(32:25)

Complete App: Scala

```
import spark.SparkContext
import spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "WordCount", args(0), Seq(args(1)))
    val lines = sc.textFile(args(2))
    lines.flatMap(_.split(" "))
         .map(word => (word, 1))
         .reduceByKey(_ + _)
         .saveAsTextFile(args(3))
  }
}
```

So, now this is the complete word count program, which we have discussed, shown over here as the local and what count is the name of the program arguments are there and this argument number one.

Refer slide time :(32:40)

Complete App: Python

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( "local", "WordCount", sys.argv[0], None)
    lines = sc.textFile(sys.argv[1])

    lines.flatMap(lambda s: s.split(" ")) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda x, y: x + y) \
         .saveAsTextFile(sys.argv[2])
```

And

Refer slide time :(32:41)

Now let us see an example

Refer slide time :(32:43)

Of a PageRank algorithm how we can execute in the

Refer slide time :(32:48)

Why PageRank?

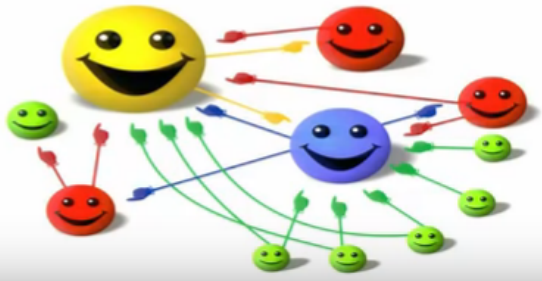
- Good example of a more complex algorithm
  - Multiple stages of map & reduce
- Benefits from Spark's in-memory caching
  - Multiple iterations over the same data

Refer slide time :(32:48)

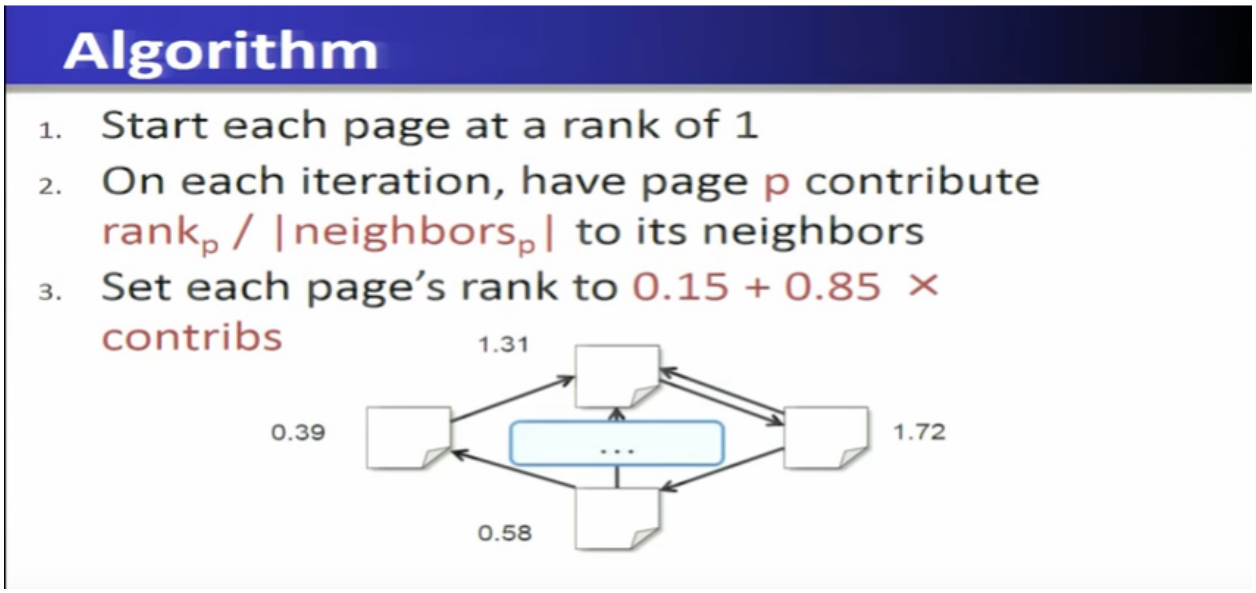
Spark system.

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



Refer slide time :(32:51)



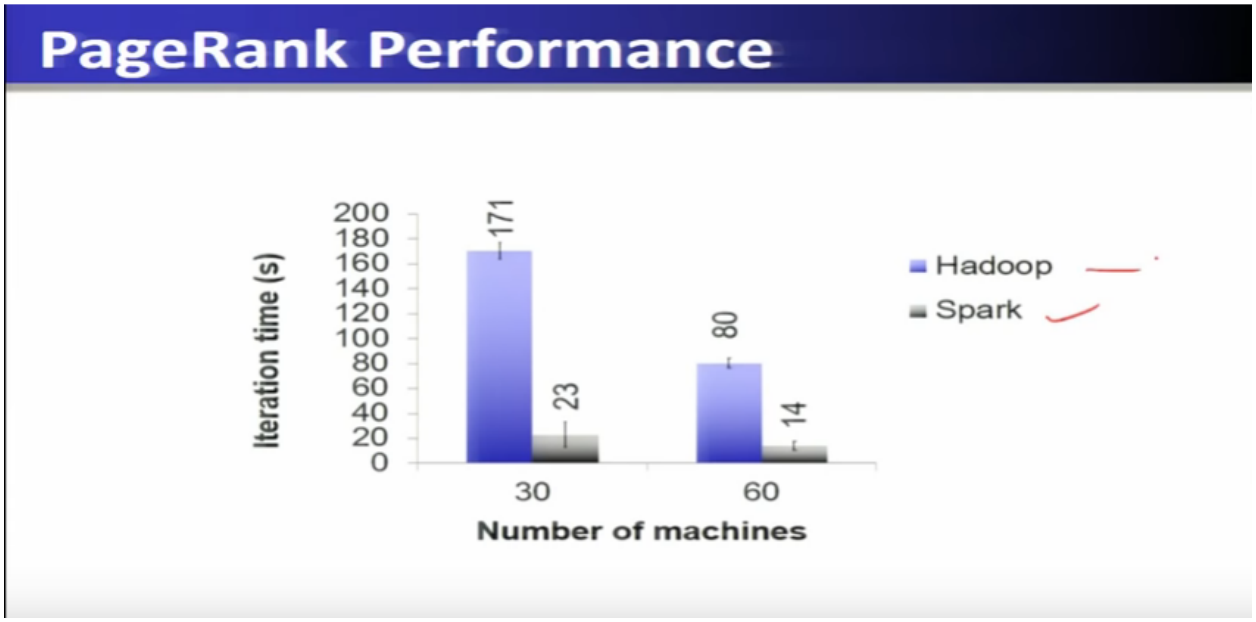
So, let us start let us see the algorithm, start at each page the rank of one. So, let us say these are different pages and we they are initialized with the rank one, at all the places the second step is that on each iteration, how the page P contribute the rank ask rank of page P divided, by the total number of neighbors and for each page rank, it will set as 0.15 plus, 0.85 times the contributions. So, here we can see that, here we can see that, this particular node, is basically this particular node has, two links out so, the contribution of one will be divided equally 0.5 and 0.5 similarly this page also has two outgoing lines links so, it will be also having a contributions of 0.5 and this will have only one. So, it will be having the contribution of entire 1 and this page is also having 1. So, it will be having contribution of 1, now let us see that this particular page, we can see that it is now incoming. So, it is in coming with 0.5 so, now we have to calculate according to this 0.15 plus, 0.85 multiplied by 0.5. So, that will be the new page rank of this and as far as this particular page, is concerned the incoming is once 0.85, multiplied by 1 plus 0.15. so, the page rank will become 1 in that case so, 1 will not be changed so, this will be the page rank 1 and how about this so, here 1 2 3 different links are coming so, with a with this, this link will be 1 plus this will be 1 and this will be 0.5 and this also will be the this also will be 0.5 so, this become 2 multiplied by 0.85 plus 0.15. And this also I will have the same so, let us see that after doing this particular iterations. So, the PageRank will now be changed, into as we have seen that it will be 1 it was pointed point, five eight point five eight point eight five one point eight five and this particular, iterations will continue and here it will stop ,why because it is not going to change.

Refer slide time :(35:52)

Scala Implementation

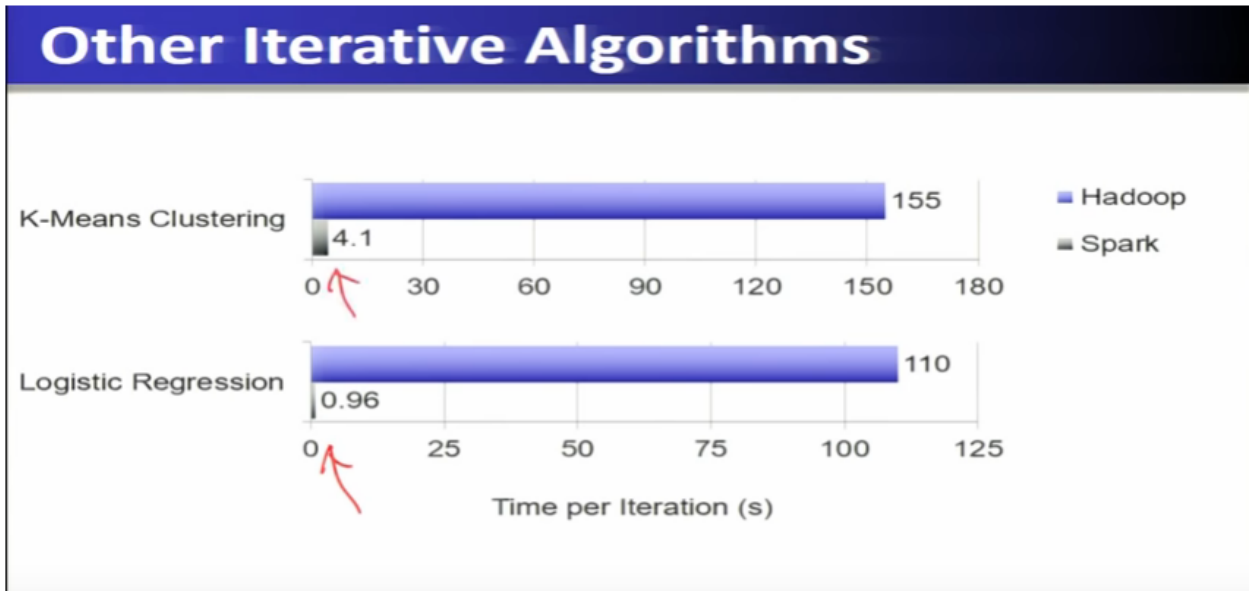
```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs

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

ranks.saveAsTextFile(...)
```

Further let us see how this entire PageRank algorithm can be implemented using scalar. So, here we have two RDDs one is in the form of a links, the other is in the form of the ranks. So, this rank means it is a page rank, URL in the page rank and the link means the, the URL and the link pairs will be there and now, iterations will now start and one to the maximum number of iterations we will contribute. Now as far as the contributions we have to, join with the links and the ranks, using flat map what we will do is we will start the case, as URL and links comma rank and this will do the flat map and destination will be now calculated as destination and rank divided by links, dot size and then what we will do we will reduce, by key the contributions and store in the in the ranks filled in the ranks are RDD and the map will be now 0.15 times 0.1 0.85 into the, the this one contributions which are now calculated, in the previous steps and this particular operations, will be saved in a file.
Refer slide time :(37:23)



So, we see that the page rank performance, with the Spark it is very efficient and very fast compared to The Hadoop here, if the number of machines are 16 and the iteration time is very less,

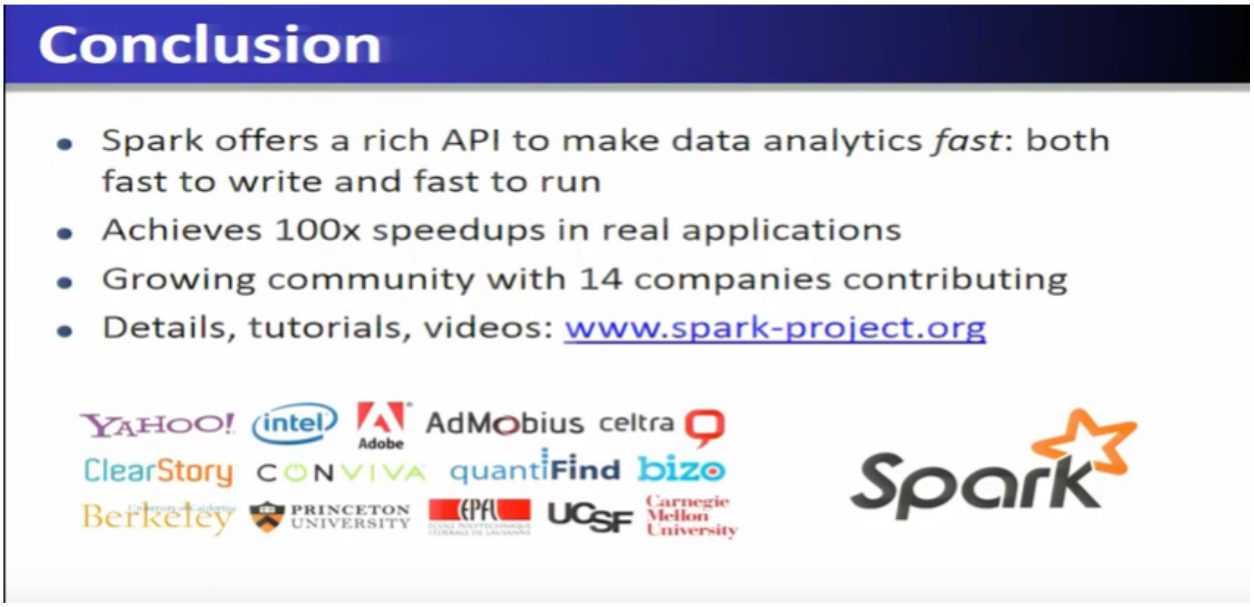
Refer slide time :(37:36)



There are other iterative algorithms which are implemented, in the SPARC such as k-means clustering logistic regression in all the cases you see that, Spark is very, very efficient compared to the Hadoop.
Refer slide time :(37:53)

In the iterations so, these are some of the references, for the Spark to be.

Refer slide time :(37:59)



So, conclusion is, Spark offers a rich API is, to make the data analytics fast. Both fast to write and fast to run. It achieves hundred times speed-up, in the real applications, the growing community with 14

companies are contributing to it and details tutorials are available in the website, www.farkyaratanlar.org. Thank you.