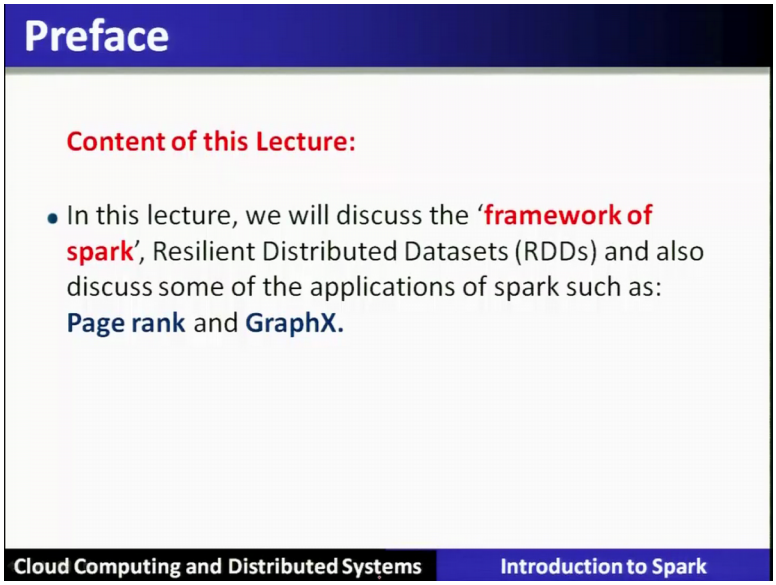
Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture-20
Introduction to Spark

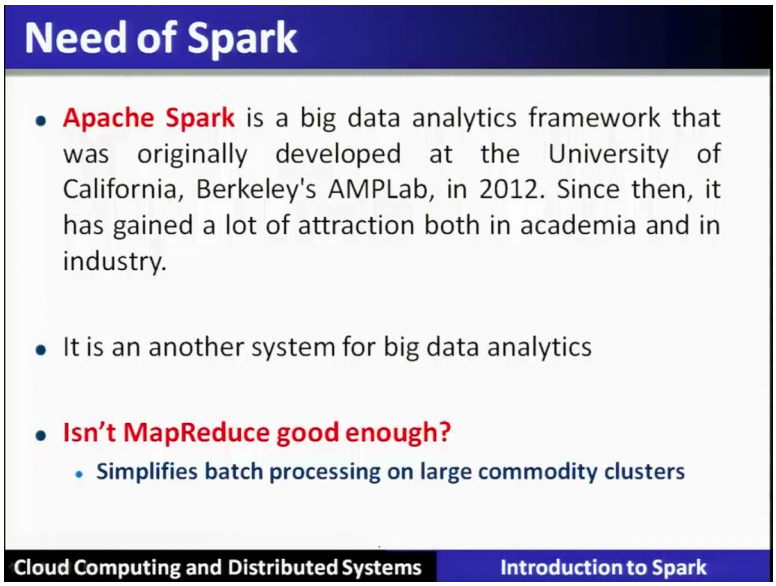
Introduction to Spark.

(Refer Slide Time: 00:21)



Preface content of this lecture, we will cover framework of a spark that is also having the resilient distributed datasets. And we will discuss some of the applications of a spark such as page rank and GraphX.

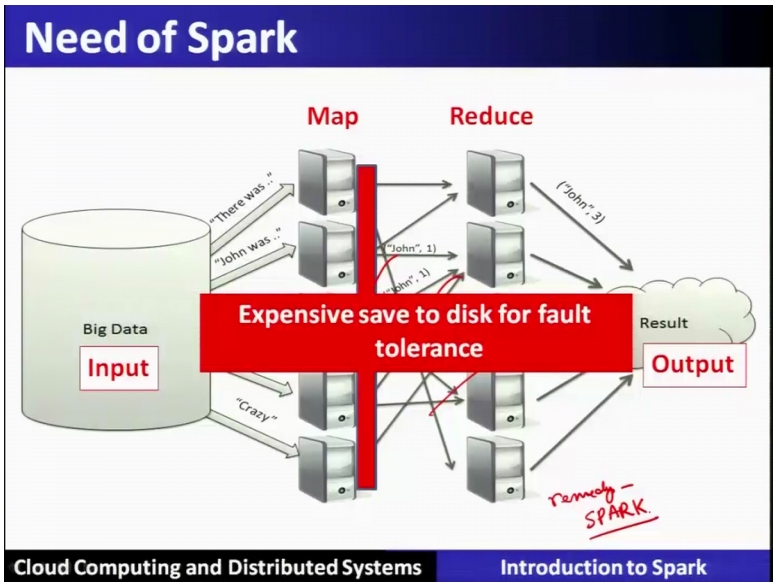
(Refer Slide Time: 00:47)



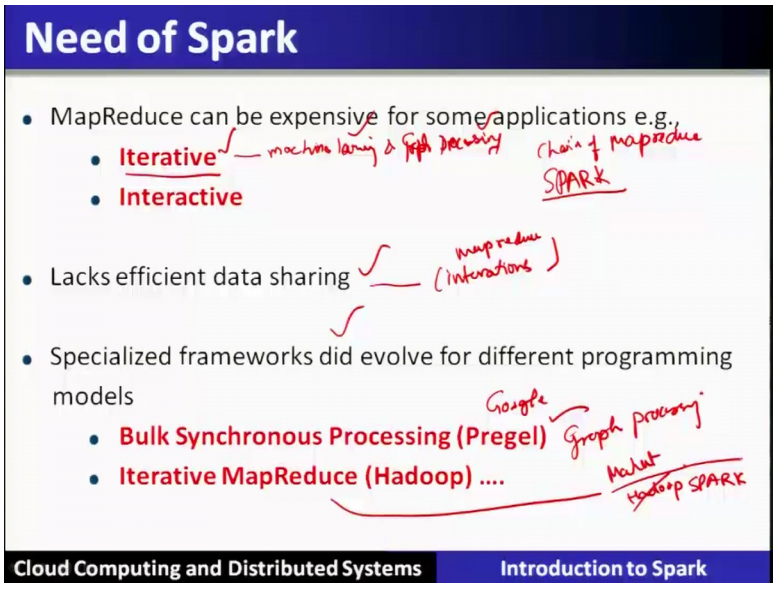
Need of a spark. Apache spark is a big data analytics framework that was originally developed at university of California Berkeley, in 2012. Since then it has gained a lot of attraction both academia and the industry and lot of development is going on in this particular platform.

This particular platform is another system alternative to the Hadoop MapReduce that we have seen in the previous lectures. So, why not the MapReduce still continuing or is not good enough. So, we have seen that MapReduce simplified the batch processing on a large commodity clusters.

(Refer Slide Time: 01:49)



(Refer Slide Time: 01:51)



But it suffers from some of the problems such as the iterative applications, for example, which requires lot of iterations such as machine learning and graph processing. This particular MapReduce performs very badly in these 2 different type of applications. Also in the interactive applications also MapReduce becomes very costly, in the since it slows down.

Let us analyze why this problem is there in the MapReduce. So, let us see that a typical example of a word count. The input data file contains, let us say it is a Wikipedia file. If

it is input, then this will be read into the system that is the MapReduce system and this particular file is divided into the splits. And each split is given to different workers. These workers will then apply the map function. And this map function will output the intermediate results in the form of key value pair.

For example, here, john 1, it will be emitted out of the MapReduce and this particular output will be stored in HDFS file system. Which in cause lot of I O's; that means, the intermediate data which will be emitted out of map function, will be stored in a file system like HDFS. So, the drawback of storing this file means that lot of I O's are involved, which will slow down the entire computing.

The next phase which is called the reduced phase will again read from these intermediate key values from the file system. Then it will read from the file system and again incur more your overhead. And this input will be given to the workers which will apply the reduce function on it and the result will be given back. Again through the HDFS using file system. This particular advantage of store is that the intermediate values are made persistent hence achieves the fault tolerance.

Although fault tolerance will be achieved there is no data loss in between the map to the reduce function, but it incurs lot of overhead and most of the applications which basically runs over several iterations of MapReduce again will be sloped down very much. Here it is written that this expensive, a storage to the disk will achieve the fault tolerance, but will be losing in the terms of time the processing takes a lot of time.

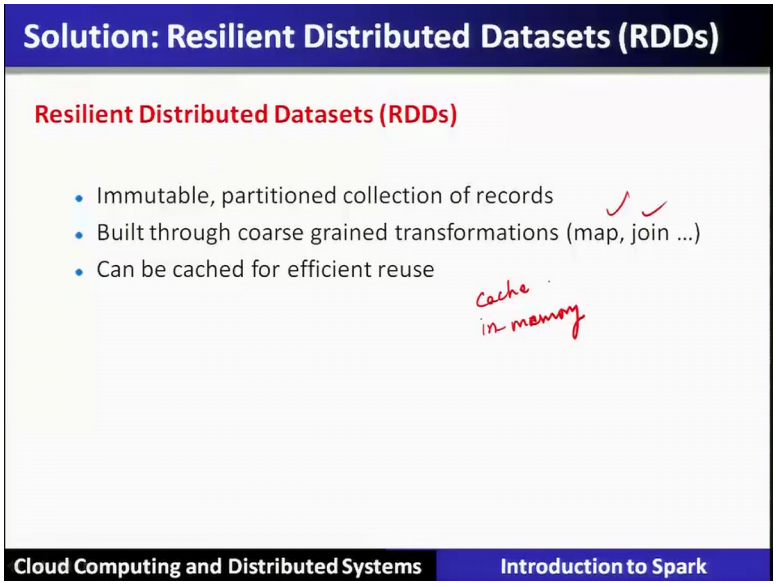
So, what is the remedy? Remedy is the emergence of that is spark system for such large scale computing on the cluster. Yet it is going to achieve the speed of in terms of in memory storage between map and reduce or across the iterations. So, with this particular idea in place the need of the spark has given the place. Therefore, the application such as iterative application which require several chain of MaoReduce jobs such as the machine learning or the graph processing applications which are called iterative applications MapReduce can be expensive therefore, alternative is the spark.

This is spark is required because the mapreduce lacks the efficient data sharing in the iterations or it is also called the MapReduce iterations. Therefore, a different paradigm than MapReduce is very much needed at this point of time. Before the emergence of a spark and the issues of expensiveness in iterative applications, lot of in between lot of

framework were evolved having different data model such as Googles Pregel data system, which is meant for the graph processing. And the iterative MapReduce they were all created to overcome this particular problem of MapReduce.

Now, once this spark is being available or is being developed all this particular applications are now shifted, for using the spark system for example, to run the mahout for machine learning iterative machine learning in the MapReduce mahout uses earlier the Hadoop file system, but now with the emergence of a spark they are using the spark framework even with the mahout.

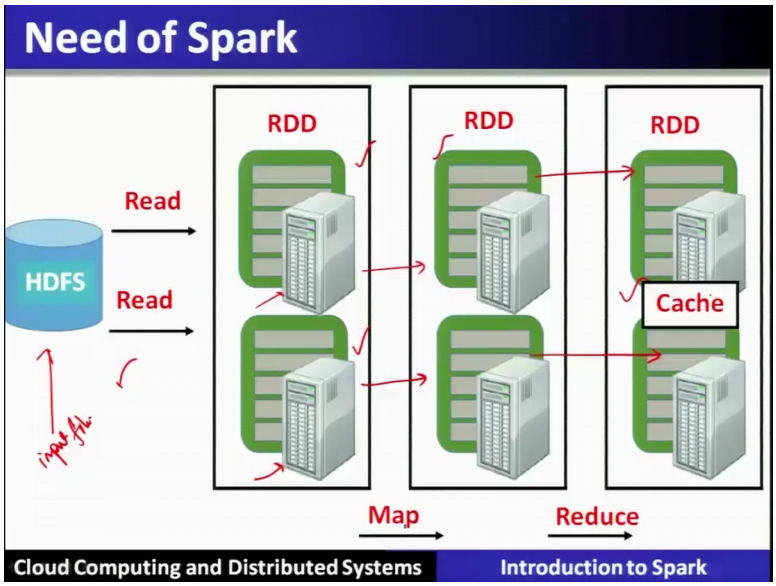
(Refer Slide Time: 11:17)



So, the solution which was available in the spark is in the form of resilient distributed datasets. So, resilient distributed data set that is called RDDs they are immutable partitioned collection of records.

They are built through the course grade transformations such as map and join and can be cached for efficient reuse. So; that means, this particular resilient distributed datasets can be there in memory operation or can be cached. Therefore, it is able to solve the problem of storing the intermediate results into the file system.

(Refer Slide Time: 12:21)

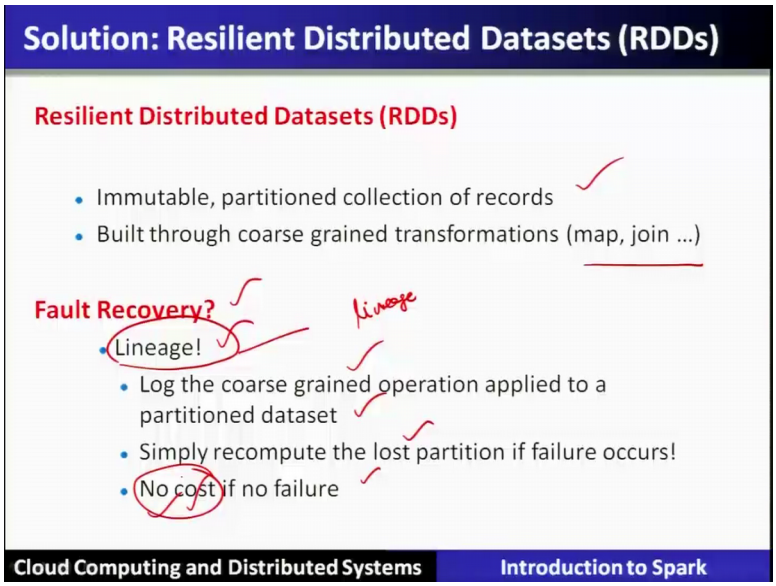


Let us understand the need of file spark in more detail. So, when we give the input file in the form of HDFS, this will split and these splits are read into the RDDs which will run on different clusters.

Then the map function will be applied and on this particular RDDs, the data will be converted in the form of RDDs and the map function will be applied to the RDDs. It will be transformed using map function and this RDDs will remain in cache in memory; that means, does not require the file system to store after the transformation using map function. Similarly, the data is in the memory there is no use of the file system. Only at the time of reading the file system is being used, then applying the reduce function using in memory will become quite efficient and these situations are speed up 100 times.

Because I O's used to slow down these operations with in memory operations using RDDs this particular operation becomes very efficient. The output of the MapReduce still will be remain available in the cache. So, that further queries can be on the result can be made efficiently if they are access through the cache. So, this is how the spark system is able to solve this computation over the clusters and also able to compute in a very fast speed.

(Refer Slide Time: 15:01)

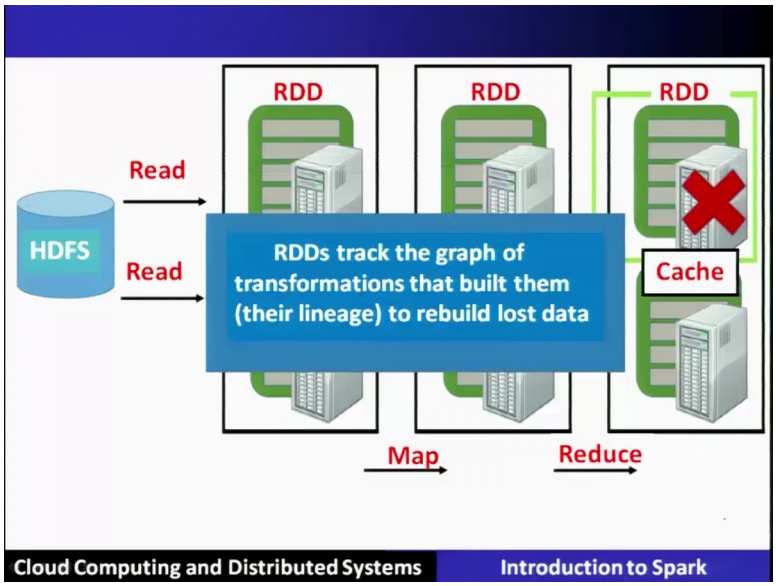


So, resilient distributed data set RDDs. They are immutable partition collection of records and built through the coarse grained transformation some of them we have seen like map join and so on.

This particular RDDs also ensures fault tolerance. Like we have seen that it will be an in memory computations of RDDs across map and reduce function. So, there is a possibility of failures. Let us see how the failure recovery is supported here in a spark, the failure recovery is supported through an operation which is called lineage. To support the lineage, it will not do the check pointing, but it will do the it will record the logs of the coarse grained operations applied to the partition data sets. So, whenever there is a failure it will simply re compute those lost partitions if the failure occurs. Hence it will not in current any overhead cost of check pointing if there is no failure.

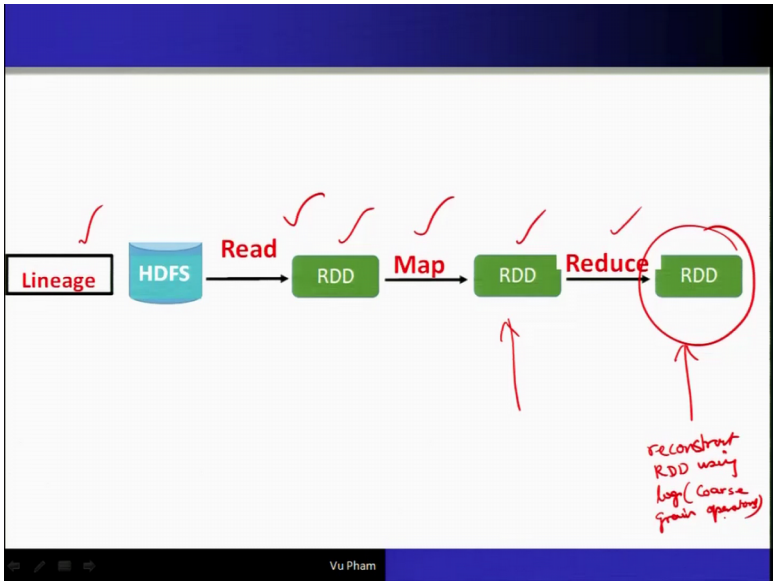
Therefore, the failure recovery in this spark is through the lineage. And also this will be not having any cost. So, basically speed is an important and also it ensures using lineage to reconstruct that RDD and it can recover from the failure also.

(Refer Slide Time: 17:03)



.

(Refer Slide Time: 17:08)



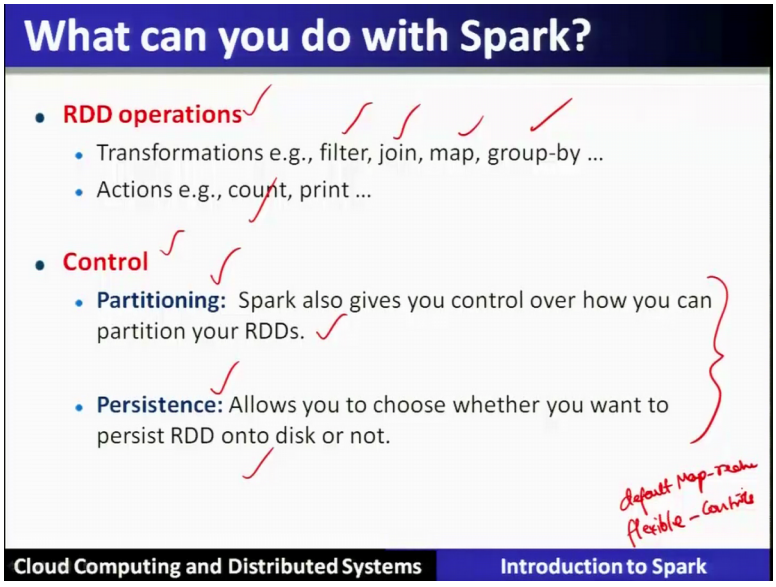
Let us see in this particular example let us say here this particular RDD is crashed or failed. If it is failed and the previous stage that is the map phase through the log, it will re construct the data that is RDDs in this particular form using that particular operation on that specific RDDs not all RDDs are affected.

So, lot of coarse grained operations are good enough to reconstruct the RDDs in case of the failures and the failures are not happening if the failure if there is no failures then

basically there is no overhead hence the competition will be speeded up in this particular scenario. Let us see the lineage how the lineage will be done. So, the data will be read and will be in the form of RDDs. Then the map will be applied and again a reduce will be applied to generate the final RDDs. So, at any stage if it if there is a failure so, from that stage we can reconstruct the RDDs.

So, here it is shown that the RDDs track the graph of the transformations that build them their lineage to rebuild the lost data here that is shown in this particular picture. So, the failure trans failure tolerant or a fault tolerance is also supported using lineage in spark.

(Refer Slide Time: 19:50)



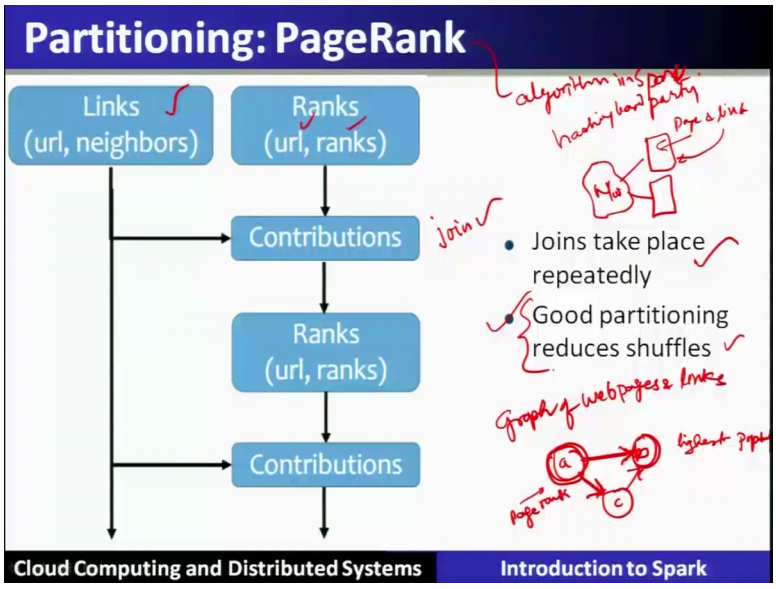
Now let us see what we can do in the spark what are the operations which are supported for RDDs. So, RDDs different operations spark supports for the transformation of RDDs such as it will filter; that means, remove some of the data and whatever is data required that is in the form of filter, it will also allow the join operation map and group by all these transformations are possible in the RDDs.

There are certain actions also, which are supported here in RDD operations such as count and so on. Besides these RDD operations the control of these RDDs are also available. So; that means, for partitioning it will not be automatically done as in case of MapReduce, but here it will be available in the form of the control. So, the partitioning whatever is required in the applications can be controlled and is done. So, spark also

gives you control over how you can partition your RDDs. So, that all is available and it is done in the form of control.

Similarly, the persistence also is available in the form of controls, super systems allows you to choose whether you want to persist the RDDs on the disk or not. So, these 2 things were default embedded into the MapReduce, whereas, here it is in the form of flexible in the form of controls.

(Refer Slide Time: 22:03)



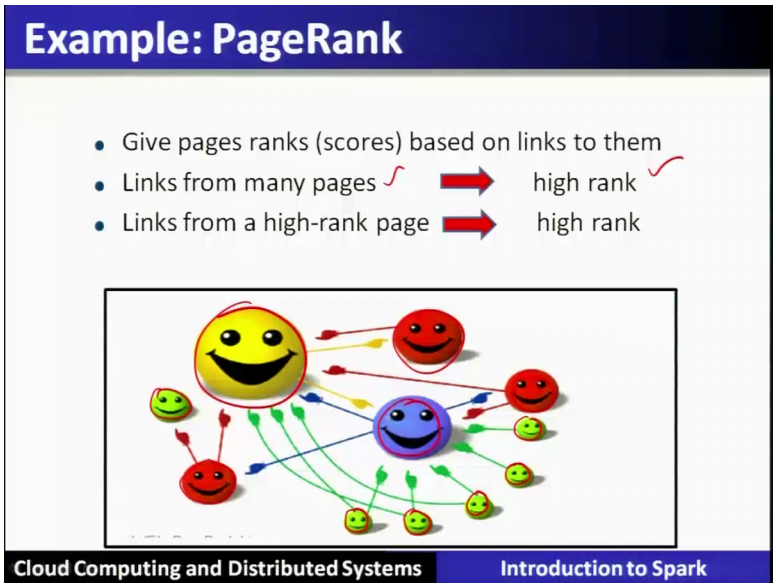
So, let us take an example of partitioning in an application which is called a page rank. Now you know that page rank algorithm requires a graph of web pages and their links. Take this particular example, a b c they represents the web pages and the webpage a refers to the webpage.

Similarly, a refers c and c refers b and so on. So, here we have to find out that page which is having highest popularity in this particular page rank algorithm. This is widely used algorithm in the different search engine like Google uses this PageRanks algorithm. So, this PageRanks algorithm requires these urls and also the links to the neighbours as the input data. And it will compute the ranks that is called a PageRanks using a particular formula, which require these information that is url and their neighbours and compute the rank this is an iterative algorithm; that means, every iteration it computes these PageRanks using join operation; that means, here different links which are referring to a particular webpage will basically a join.

So, join will take repeatedly across all the iterations. Now if these joins requires the partitioning in different clusters. Then lot of network access is required during the join operations. So, if there is a good partitioning so, that all the links which are required in the join operations they can be communicating within a particular cluster. Than that is called a good partitioning which can be done using the hash based partitioning. So, in the hash based partitioning; that means, the page and all it is consultants will be mapped to the same cluster to the same node in the cluster. Hence this will reduce automatically the number of shuffles.

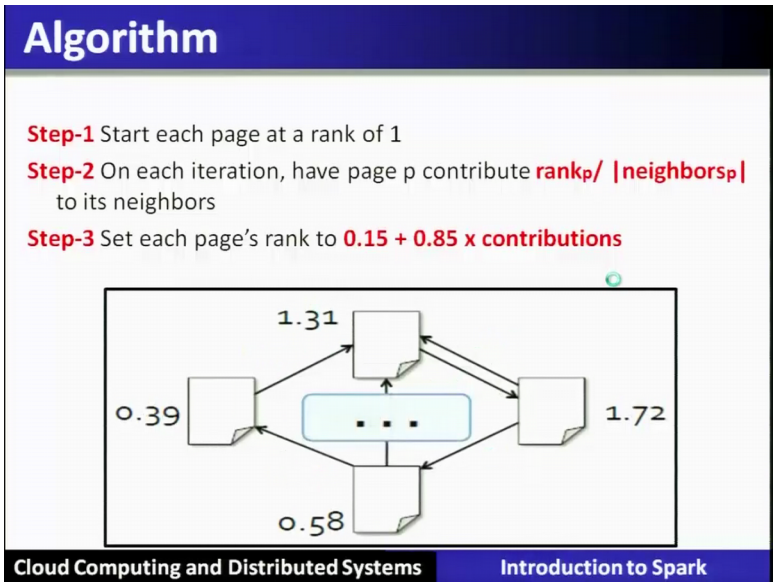
So, the number of shuffles across over the iterations using this join operations can be possible to be computed in memory hence the pagerank algorithm, if being implemented on spark will be more efficient in comparison to the MapReduce operations.

(Refer Slide Time: 25:59)



The in comparison to the MapReduce framework. Let us see the example of a page rank. Here, you can see this is the most popular page why because everyone is pointing towards that particular page and a small circle shows that they are not that popular why because nobody is having the reference to that particular page. So, the sizes of this page shows the rank that is the one which is having the highest number of links from many different pages having the highest rank. So, we want to rank all these pages using this particular information.

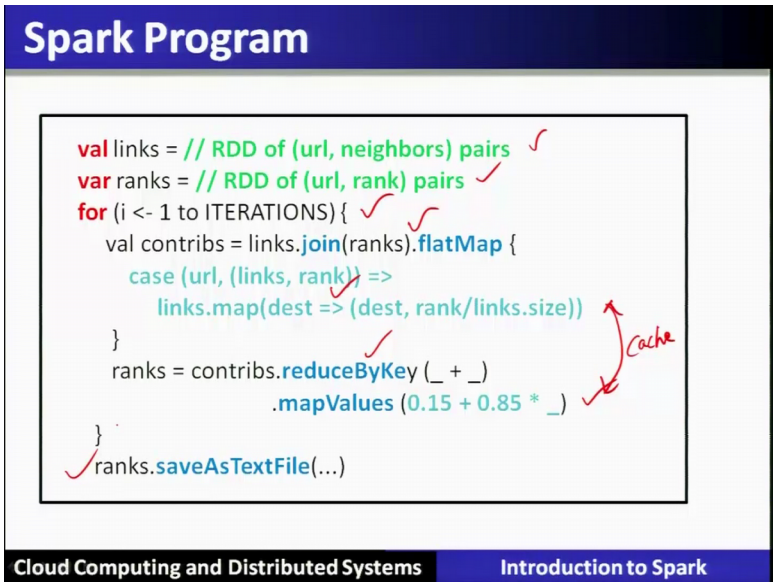
(Refer Slide Time: 26:54)



So, here this particular algorithm will have several steps, let us understand the those steps then we will explain the PageRank algorithm, step one will start each page with the rank one on each iterations each page will contribute the rank divided by number of neighbors to that neighbor and third step will set the page rank to 0.15 plus 0.85 into contributions. So, for example, this node to find out the page rank of this particular node which is being referred by this page let us say A, B and C they are referring to this particular page. So, PageRank of let us say D is equal to the page rank of a divided by the number of outgoing links here it is 1 plus, this is the page rank in the next iteration.

This is the bageth iteration similarly as far as it will also contribute the page rank of B. So, the page rank of B at the ith iteration divided by how many outgoing links are there this is 1 and this is 2 plus PageRank of C, divided by how many number of outgoing links 1 and 2. So, in every iteration this particular page rank will be calculated till it converges.

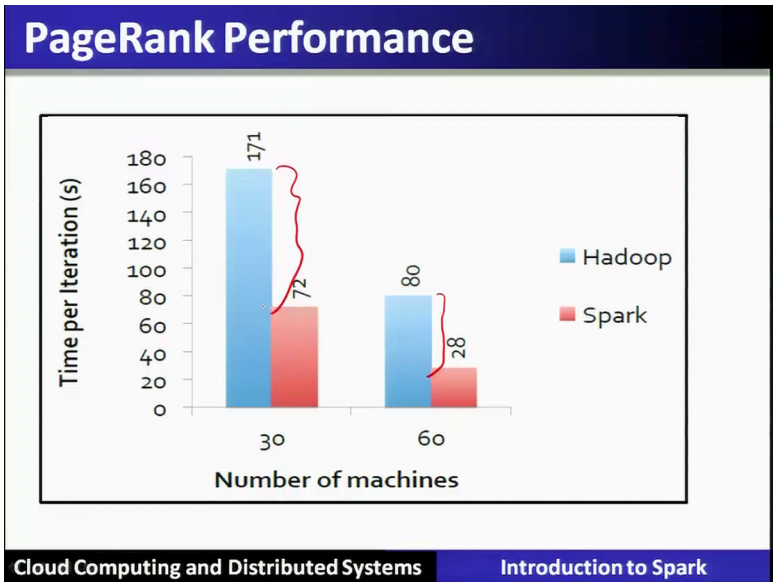
(Refer Slide Time: 29:08)



So, let us see how the spark will be used in this programming of the PageRank, here we have the links that is were which are the outgoing links url and the neighbor pairs and also the previous rank of the previous iterations that also is the url and the ranks these 2 are the inputs for the iterations to be.

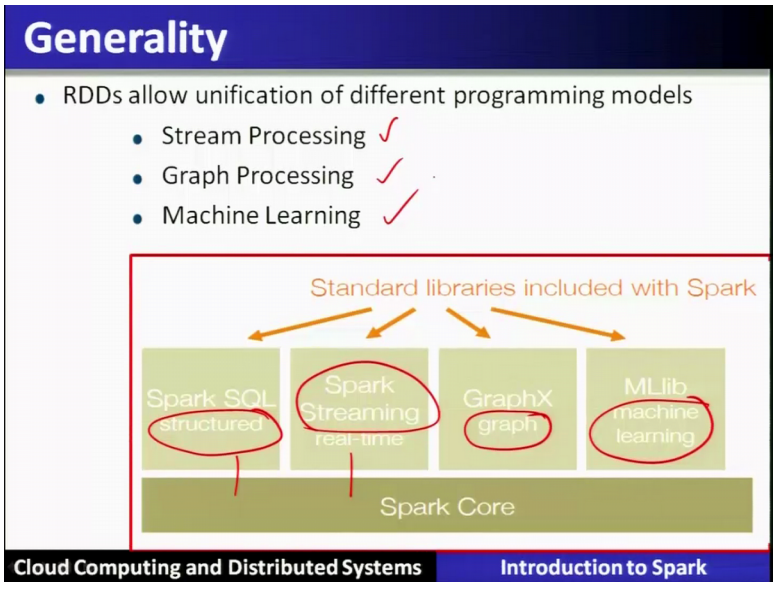
Now, the iterations will begin till it converges. Now as per the joints are concerned it will be done using flat map. And using map function easily this particular computations can be done and it will be reduced and computed the map values in this particular scenario. So, between the map and reduce, it will be the data will be in cache, hence this particular map and reduce function will share the data across iterations or in an any iterations. The final ranks will be saved in a text file in between there is no use of file system.

(Refer Slide Time: 30:58)



If you compare the performance of same algorithm using Hadoop and spark, we can see the difference is quite huge.

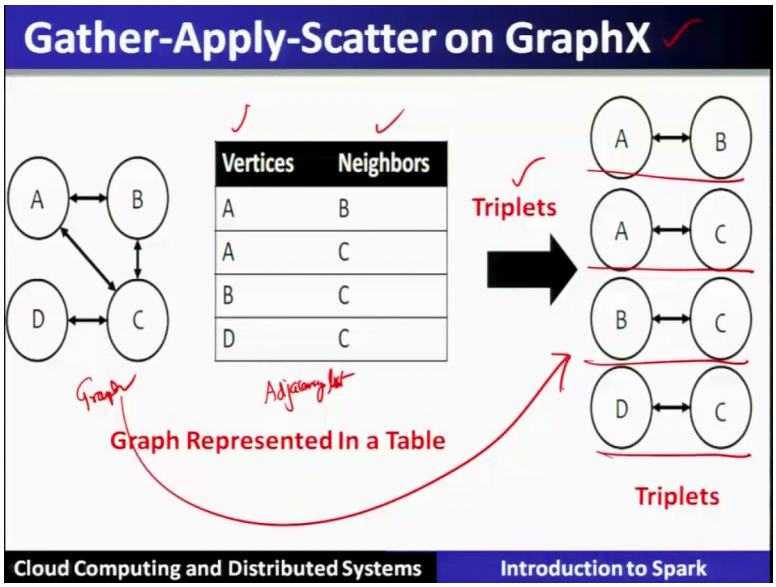
(Refer Slide Time: 31:22)



So, the spark, which supports the RDDs is many times faster than the MapReduce. Hence, it allows many different applications to be supported. There for a spark supports when use different type of libraries which is built over the spark. They are called spark streaming that is a real time streaming applications, then GraphX for graph applications and MLlib for machine learning applications. And structured SQL for database

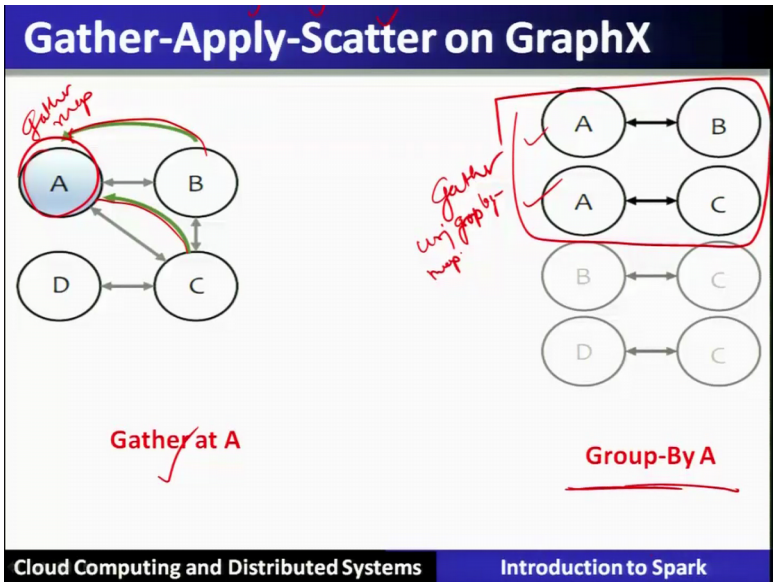
application these are different kind of so, we will. So, then; that means, there are stream processing graph applications and machine learning can easily are available using the libraries over the spark.

(Refer Slide Time: 32:34)



Let us see how the GraphX provides support for graph applications. Let us assume a the graph which is represented by these vertices and their corresponding neighbors, in the form of adjacency list. In a spark this graph can be represented as the triplets. So, triplet means A B C it is a triple. So, it will be represented as A B then A C then B C and then C D these triplets are represented. So, graph is represented in the form of this particular triplets. Having partition, the graph into the triplets, now it will be easier to do the graph processing in this particular triplet forms.

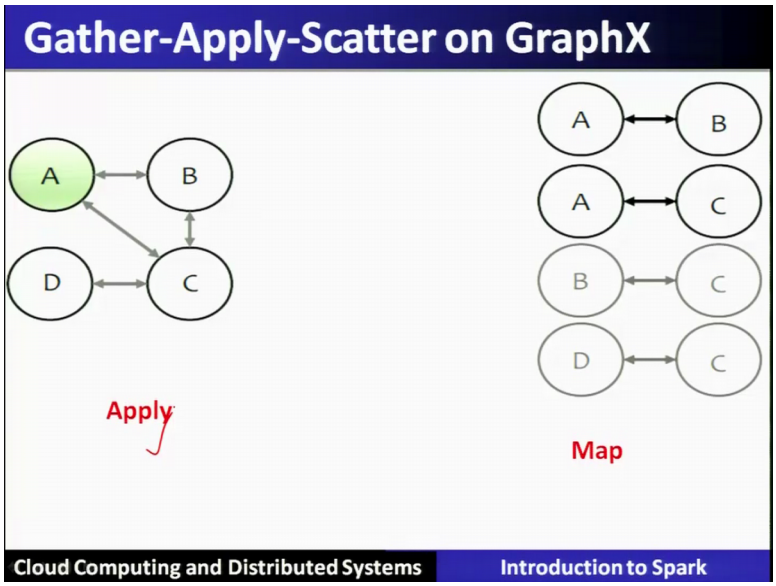
(Refer Slide Time: 33:54)



So, for that it supports GraphX supports the operation which is called gather apply scatter, let us see how this particular triplet supports this gather apply in the scatter. So, gather let us understand for the node A. So, the node A will collect the data from the triplets B and C.

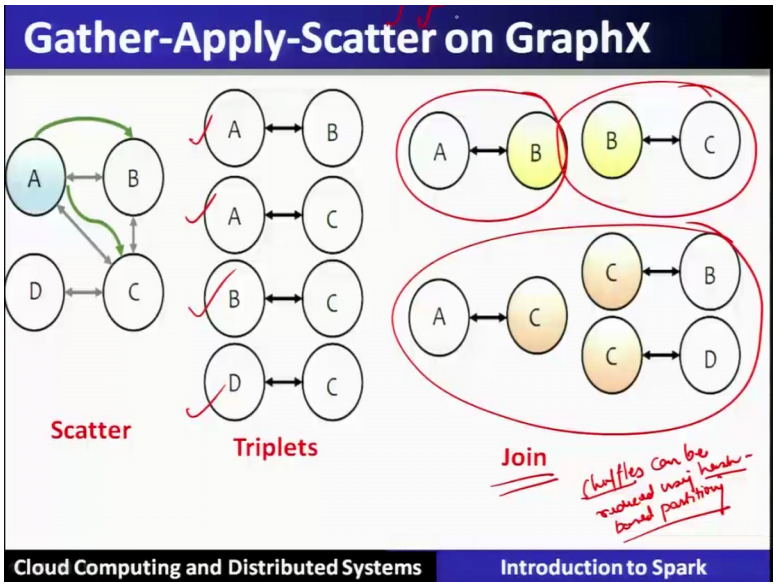
Then it will perform the apply will be quite easy because once the data is collect gathered then it will apply the map function on the data and by scatter; that means, the result of the map function has to be scattered across all different triplets. So, gather is easy using the group by operation which are supported in a spark. So, group by will basically identify these 2 different triplets, and the gather operation using, group by is easy to operate and then will perform the map function on it.

(Refer Slide Time: 35:39)



So, the next phase is called apply. So, once the group by and that is called gather is completed then applying map will be quite easy.

(Refer Slide Time: 35:53)

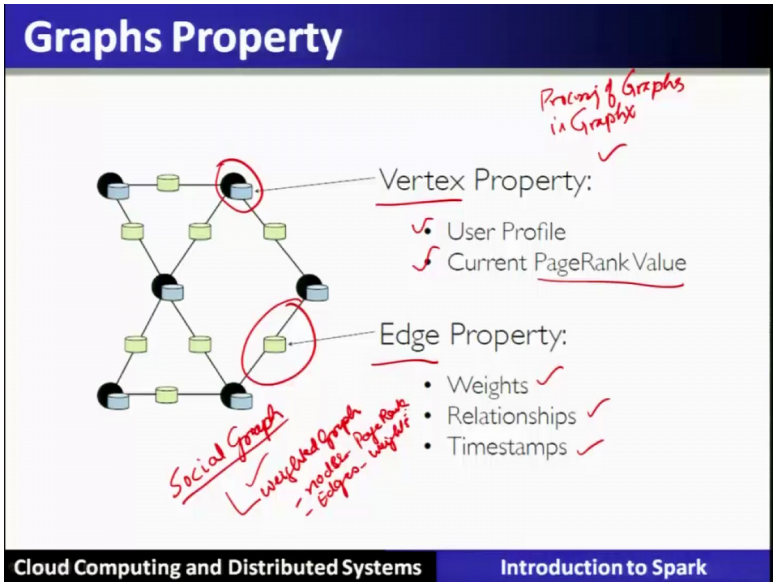


Now, then scatter that is that particular value out of the map has to be given to all it is corresponding neighbors using scatter. Now here we see that when this scatter will happen. So, it will basically affect these 2 triplets. And also it will affect the other triplets also in this particular form. So; that means, if the data is stored across several partitioned

across different clusters. Then this particular operation which is called scatter that is implemented through join and this becomes costlier in that case.

So, let us see that the scatter between A and B, and B and C is quite different than this particular scatter between A and C and C and B and C and D. So, therefore, the join operation will require lot of scatter requires lot of shuffles and here requires the use of good partitioning algorithm which will reduce the number of shuffles in it is scatter operation. So, using hash based partitioning this number of shuffles when this is scatter is applied can be reduced. So, here lot of design controls are available. So, that can be still controlled and good performance can be achieved. So, let us go and more detail of GraphX APIs.

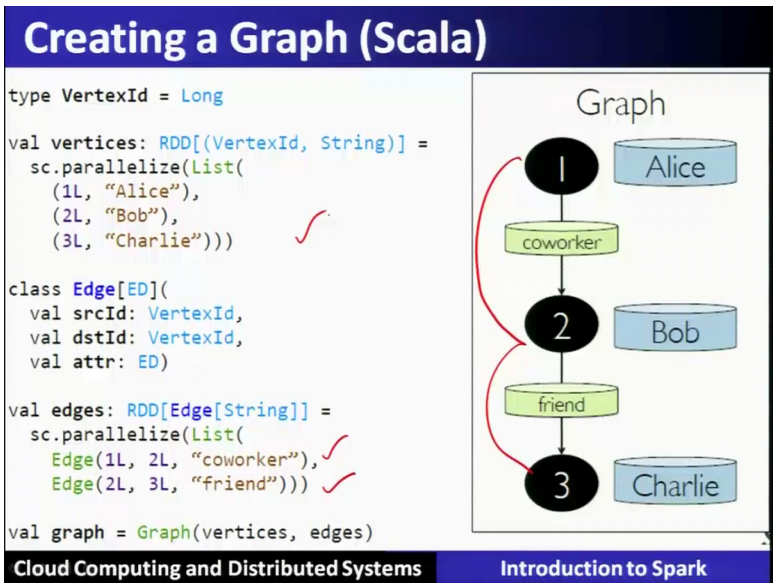
(Refer Slide Time: 38:43)



The graphs APIs let us go through the GraphX APIs which are useful for the graph processing. Now, every graph has some properties, with graph has access supported in the form of APIs which can be used to design the algorithms and form the graph based competitions of the data set. So, different data sets can be represent in the form of a graph and using GraphX they can be easily processed and computed. Let us see some of the properties so, the graph comprises of the vertices and the edges. The vertices are having the properties such as if this particular graph represents the people and their relationship or it is basically called a social graph. Then this vertex will nothing, but it will be having the user profile of a particular person.

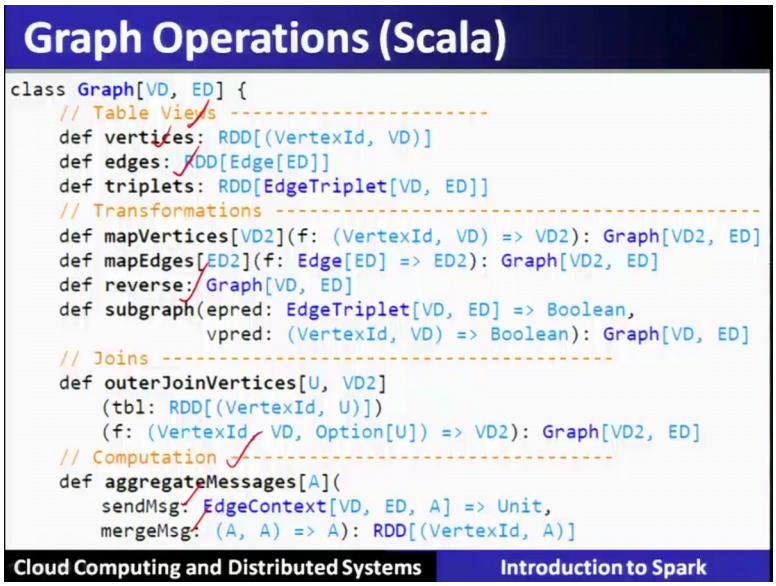
And also another important value is about the current PageRank value in this particular scenario; that means, the links to that particular vertex is having how many what is the importance of that particular vertex is it is PageRank value. Similarly, the edges will have the weights on that particular edge. It also represents the relationship between these 2 connections and also the timestamp when these weights are being computed. So, this is a kind of weighted graph, where nodes are also having the PageRank that is the weights and edges also having the weights defined. So, having defined the graph of this particular nature.

(Refer Slide Time: 41:40)



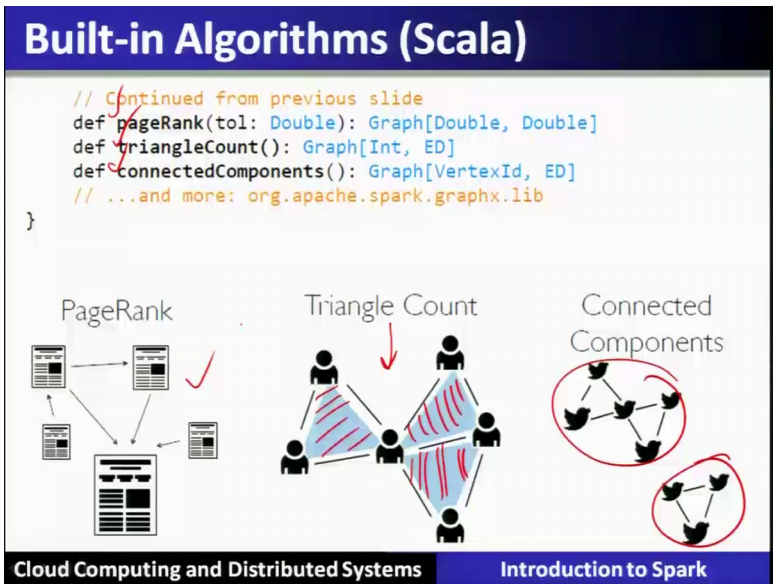
Different relationship can be defined here in the GraphX and then RDDs can be evolved and these relationships will exploit the controls and the transformations to process the graph. So, for example, here Alice and Bob their relationship is co worker. So, most of the time they will be together working or basically having the relation of co worker, which will be defined here in the edge. Similarly expires between Bob and Charlie the relationship is friend. So, this can be also defined. So, RDDs will use this particular information in the graph to create the graph and also define all these particular relationships.

(Refer Slide Time: 43:05)



In the GraphX, using the language which is called a scalar. So, scalar provides different apis, where in the vertices edges triplets and different sub graphs then different computations of the sub graph communicate across all this edges all this particular operations are well defined in the form of API's in GraphX, which can be used to program it.

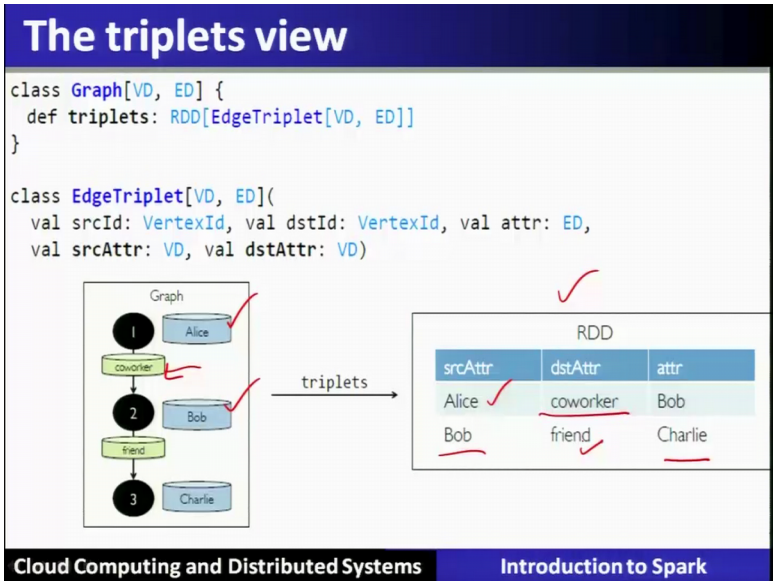
(Refer Slide Time: 43:46)



So, there are some built in algorithms also which can directly be used. For example, the PageRank algorithm is already built in the GraphX package and will be used directly as an API.

Similarly, to count how many triangles are there for example, here to count how many triangles are there which will give the relationship values. And also in a graph what are the connected components if a graph is disconnected then finding out the connected components all this particular algorithm graph algorithms are already implemented and can be used by just calling the API's.

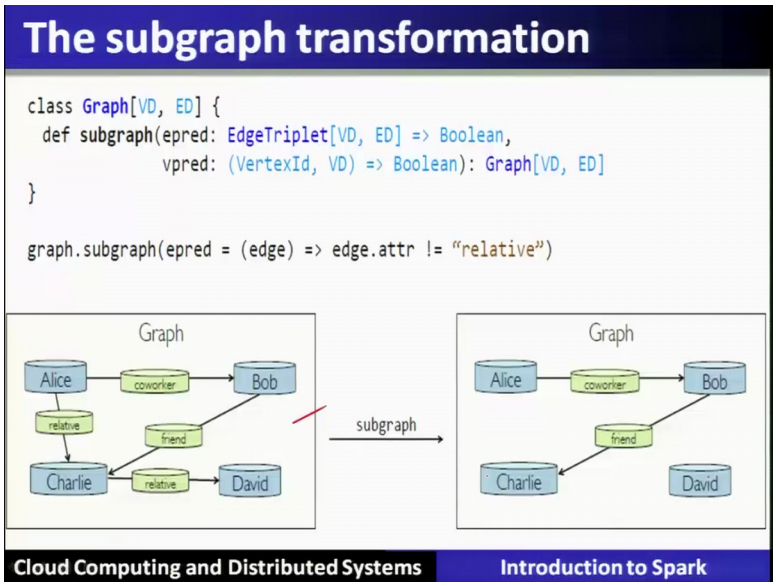
(Refer Slide Time: 44:33)



Now, as for as the triplet views are concerned, they can be using triplets the can be modeled in the form of RDDs. For example, the Alice in it is RDD the source and destination that is this is Alice and Bob and their attribute is the coworker; that means they have the relationship which is of type coworker.

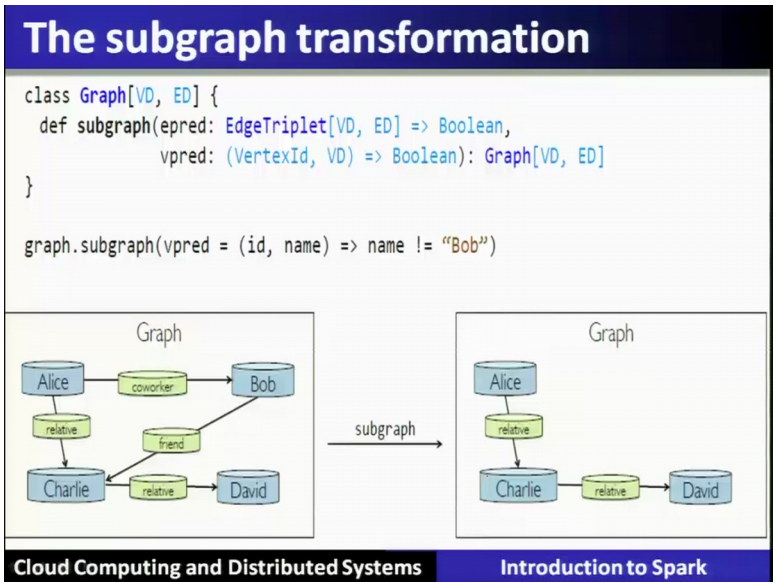
Similarly, bob and Charlie there are 2 different nodes in the graph connected with an edge which is having defined this particular relation friend. So, this way the triplets can be defined in the form of RDDs.

(Refer Slide Time: 45:34)



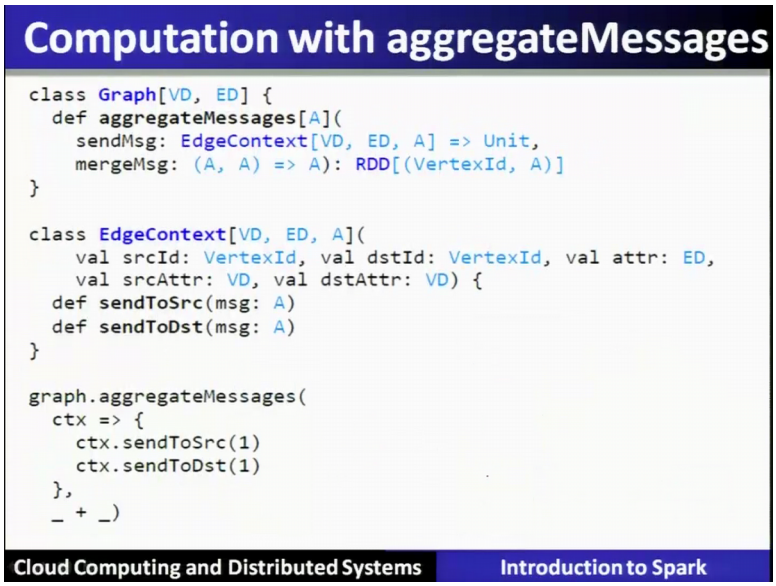
And then once the triplets are defined and various graphs sub graph transformations are also supported in the form of a GraphX.

(Refer Slide Time: 45:54)



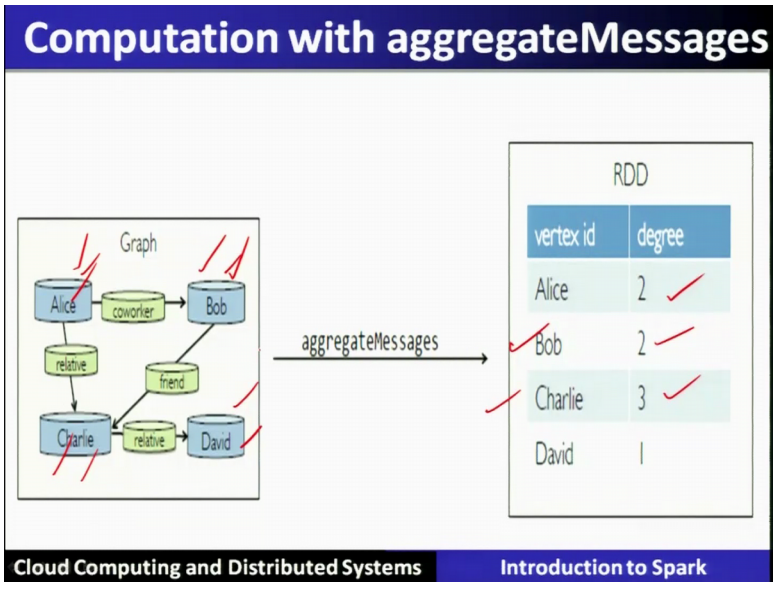
So, for example, out of this complete graph a subgraph can be partitioned and being operated on that part also using subgraph transformations.

(Refer Slide Time: 45:58)



Similarly, the computation with aggregated messages are also supported, where in the triplets can communicate using send message, send to the source send to the destination.

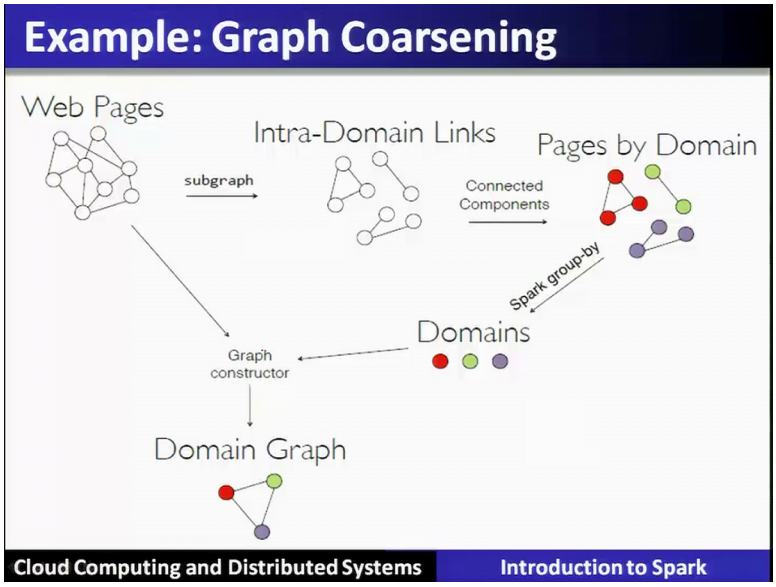
(Refer Slide Time: 46:22)



For example, here see the computation with the aggregated messages. So, let us see this particular Alice has the degree 2. In 2 degree means it is having the relationships with Charlie and Bob in the triplets similarly, as far as the Bob is concerned it is also having the relations with Alice and Charlie and Charlie has 3 relations with Alice Bob and David, David has only one relation with Charlie.

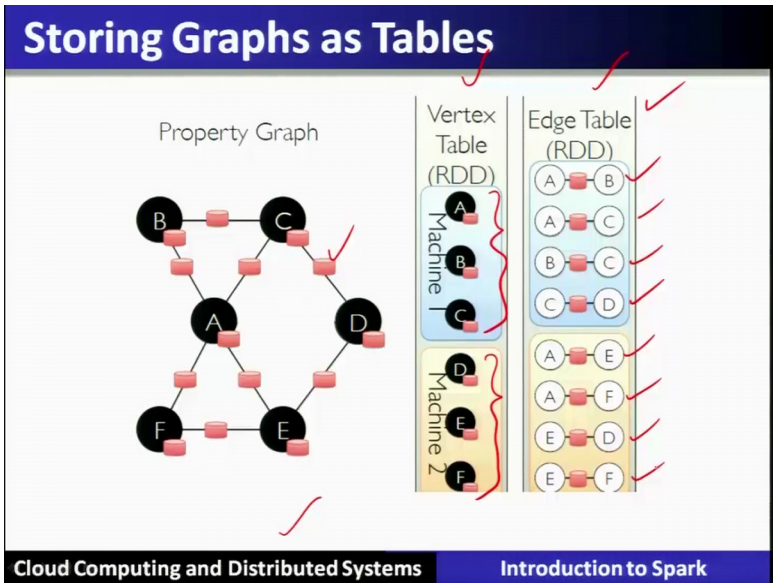
So, these aggregated messages can also be used in the computation, another example of graph coarsening.

(Refer Slide Time: 47:16)



Hear the different type of graphs and they can be operated using different graph coarsening.

(Refer Slide Time: 47:25)

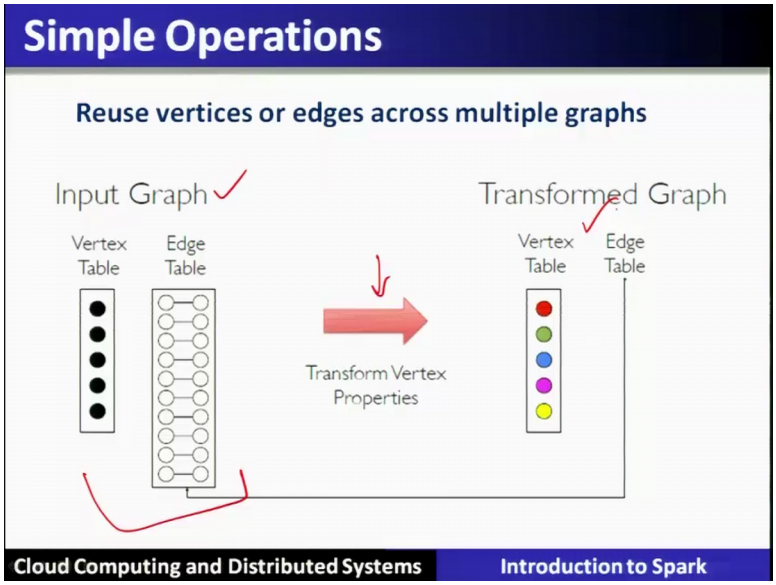


So, let us see how the GraphX works. So, the graph is now stored as a table in GraphX. For example, the example is given as the property graph it will be stored as 2 tables the vertex table and the edge table. So, here the list of vertices which are stored in one

machine and further list of 3 vertices are stored another machine. So; that means, these particular vertices are also partitioned across different machine so, that a throughput can be achieved or when a large scale graph is there then it can be computed over a cluster.

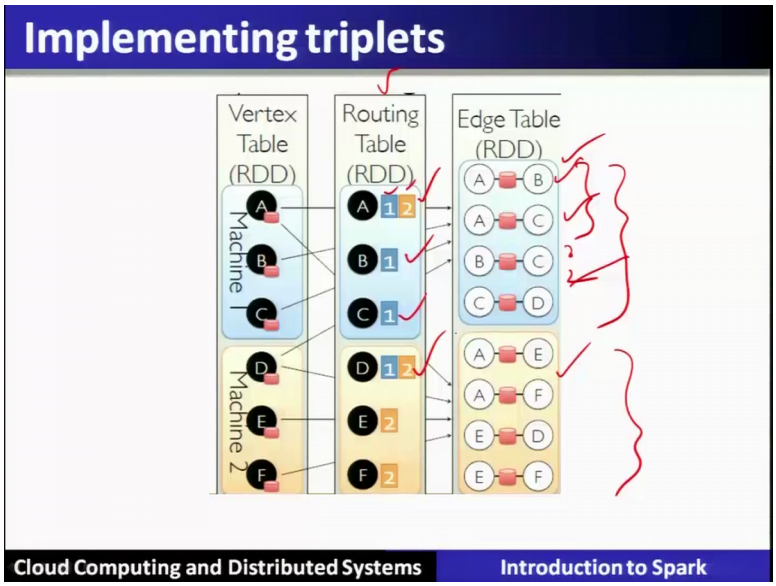
The edges are also stored in another table which is called an edge table they are all RDDs. For example, between A and B this particular triplet is stored A C and B C and C D. They are stored in one machine and A E, A F, E D and E F they are stored another machine. So, we have seen that the graph can be represented in the form of tables vertex table and the edge table they are nothing but RDDs in GraphX.

(Refer Slide Time: 48:57)



Having define these 2 different type of tables this will become an input a graph and then different transformations can be applied on the vertex properties. Similarly, it can be applied later on the graph properties. So, this is shown that if some properties are applied on the vertex table this is the transformed graph. So, there are simple transformations which can be applied.

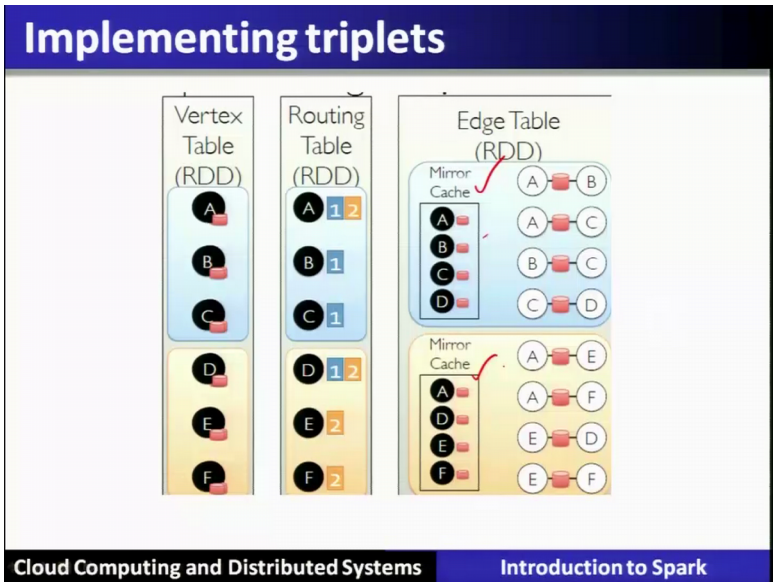
(Refer Slide Time: 49:33)



Now, implementing the triplets to implement the triplets there is one more table is involved that is called a routing table that is also an RDD. This will give an information that let us say the vertex A is having the 2 triplets A B and A C. So, A B and A C they are lying in 2 different segments. For example, this particular a is appearing here and also as far as A B is concerned, B is appearing only once, C is also appearing in only one of these clusters machines.

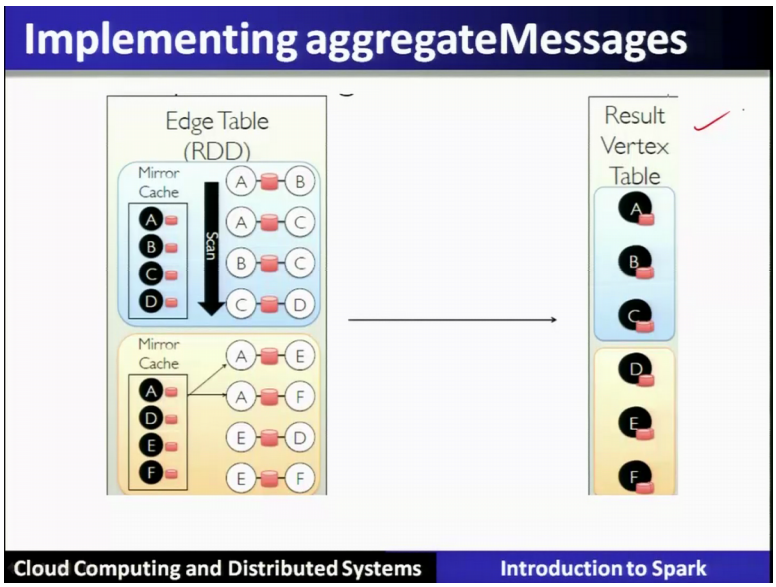
So, all that routing table will give the information that this triplets are either fully stored in one cluster or in many more than one cluster and this particular information is stored in the routing table.

(Refer Slide Time: 50:57)



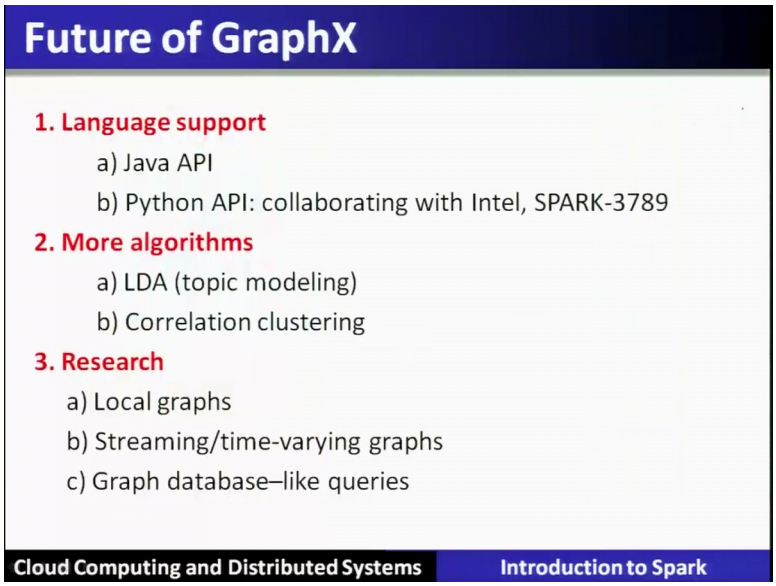
To implement further the triplets, the edge table will mirror this particular cache of this vertex table RDDs. Similarly, here also it will mirror the cache over here and this will be used in working with the triplets.

(Refer Slide Time: 51:25)



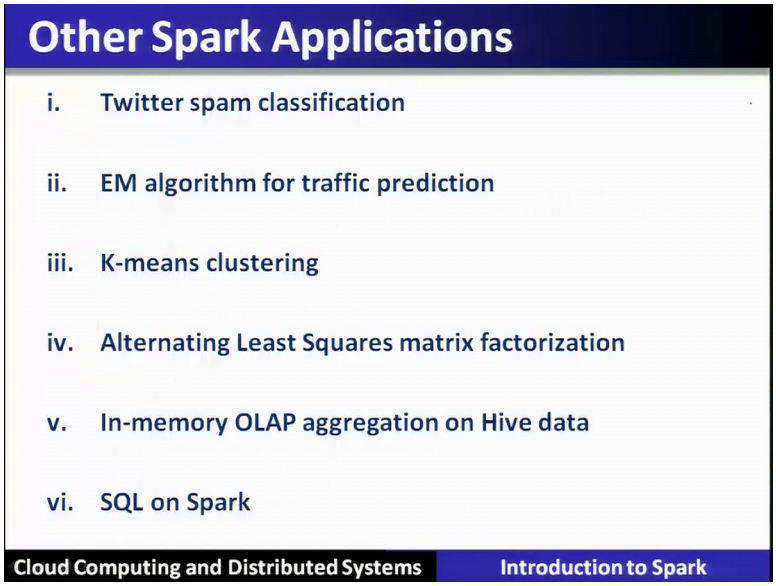
Similarly, the aggregate messages can be easily computed by a scan operations. So, it will be having a mirror cache and the edge table which will scan all these things and it will give the result vertex in another form of table.

(Refer Slide Time: 52:01)



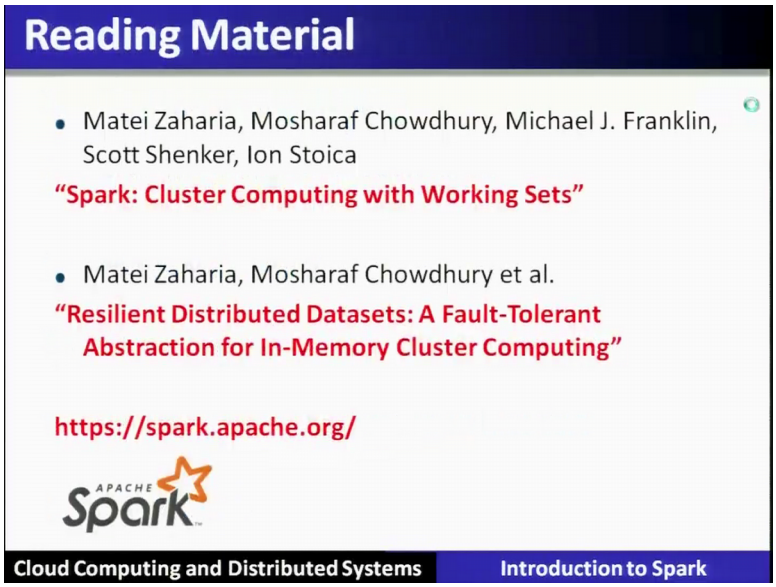
So, having seen the GraphX applications, different operations which supports the graph computations. Many new languages are coming up to support in the form of API's and more new algorithms are, also being built in the GraphX package also lot of research is going on in this particular support of graphs in graphics such as local graph stream time varying graphs graph database which supports the query operations.

(Refer Slide Time: 52:47)



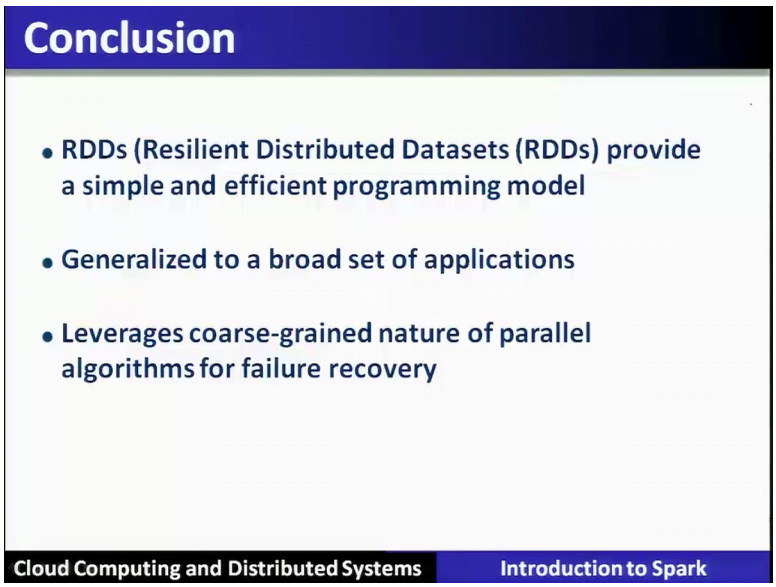
There are many other applications which basically uses the spark such as twitter spam classification EM algorithm for traffic prediction K means alternative least square in memory OLAP aggregation and SQL on.

(Refer Slide Time: 53:07)



So, there are many further reference materials we have included here the spark cluster computing with working sets and resilient distributed datasets a fault tolerant abstraction in memory cluster computing.

(Refer Slide Time: 53:27)

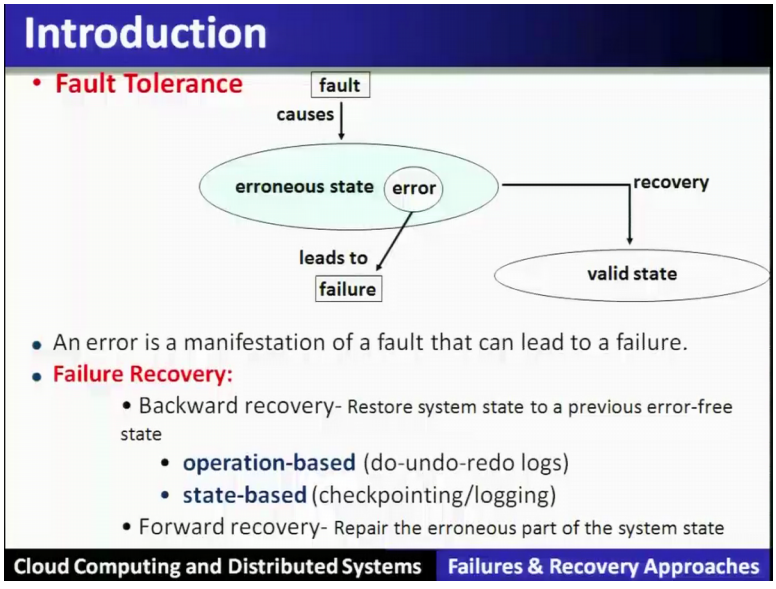


So, we have seen how the spark has overcome from the problems of MapReduce using an RDD that is a resilient distributed datasets, which provide simple and efficient programming model is, spark is 100 times faster than MapReduce operations. Because most of the operations across the iterations it performs in memory without using the distributed file system or basically overcome using the IO's.

We have also seen that this is spark core is used in building various libraries such as for machine learning, it is MLlib for graph processing it is GraphX and so on. We have seen that this is park leverages coarse grained nature of parallel algorithms for failure recovery. So, failure recovery is not that having an overhead and easily implementing it.

Thank you. Thank you erroneous states is a state where the errors are basically reaching to that state due to the error. So, error is a manifestation of a fault that will lead to a failure, here we will see that in that particular state which is called erroneous state the recovery is required to reach to a valid state, which is an error free state.

(Refer Slide Time: 55:26)

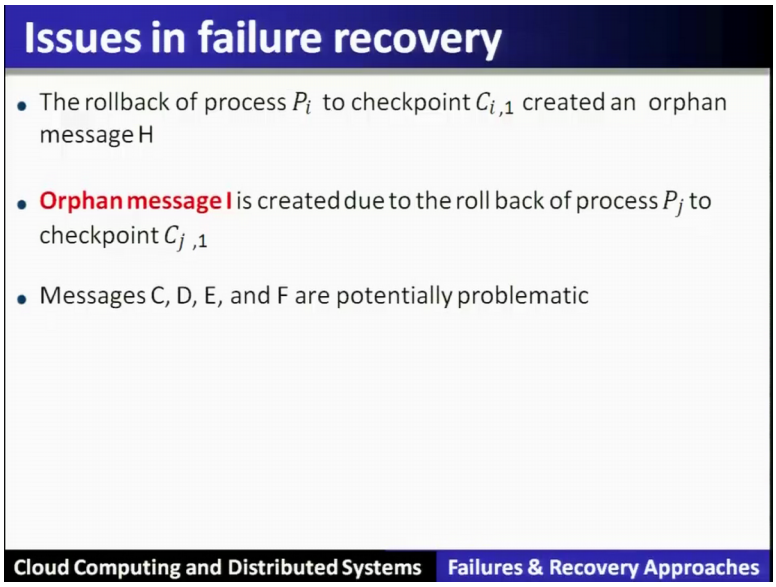


So, the fault tolerance is achieved using recovery of a failure using 2 methods the first one is called forward recovery method where the repair of the erroneous part of a system state is known and it will be carried out to reach or to recover the system from failed state to the varied state that is called forward recovery method. There is another failure recovery method, which is called a backward recovery method. It will restore the system state to a previous error free state. It does not require the knowledge of the erroneous part

of the system state. The backward recovery is of 2 types, that is operation based backward recovery method in which all the coarse grained operational like do undo redo they are logged in a file and they are being operated on. That is why it is called operation based the second type of backward recovery is called is state based here the states are stored or a logged they are also called checkpointing.

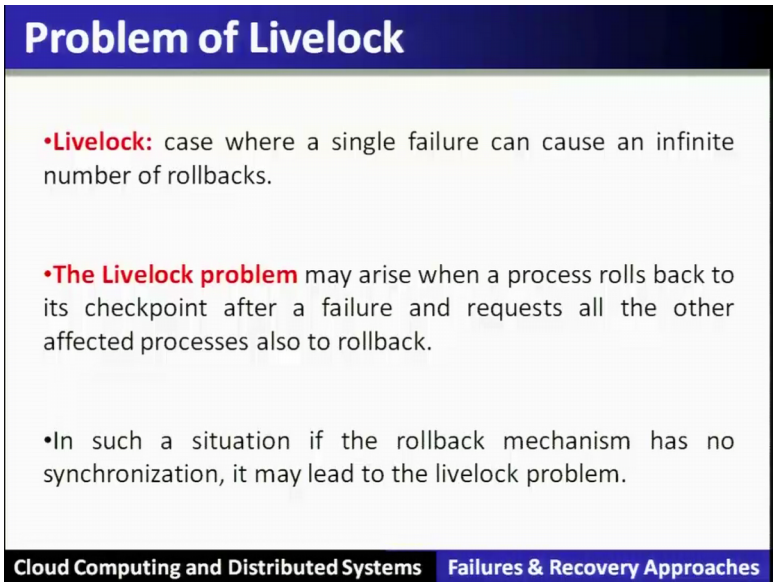
So, when a failure happens using this particular checkpoint or a log based logging method it will try to recover to a restore the previous valid state. Interaction with outside world the distributed system of an interacts with the outside world to receive the input data or deliver the outcome of the computation.

(Refer Slide Time: 57:47)



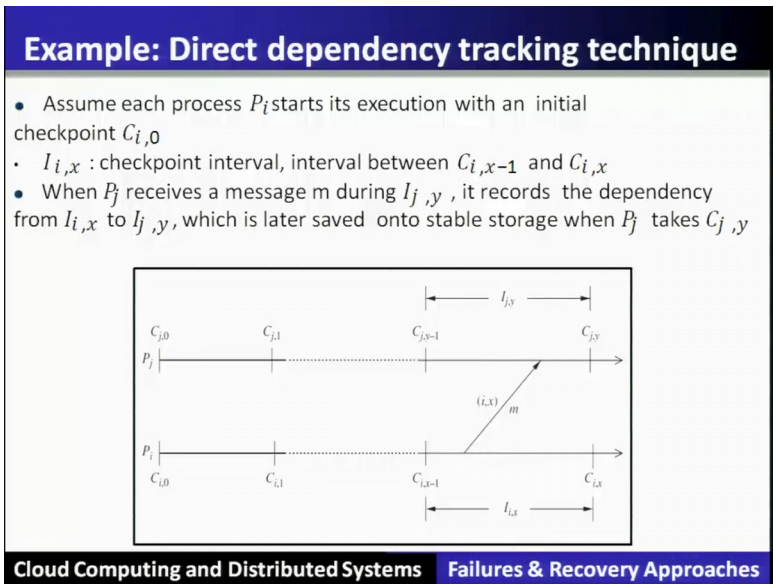
Issues in failure recovery; the rollback of a process P i to a check point C i coma 1 created an orphan message H. Orphan message is created due to the rollback of a process p j to a previous to a checkpoint that is c j 1. So, the messages here we are seeing in the example c d e f they are basically the problematic in that sense that they have become the orphan message and it requires a rollback to undo that affect.

(Refer Slide Time: 58:33)



Problem of livelock; the livelock case where the single failure can cause an infinite number of rollback is called a livelock. The livelock problem may arise when a process roll back to it is checkpoint after the failure and request all other affected processes also to rollback. In such a situation if rollback mechanism has no synchronization, which will lead to infinite number of rollback this may lead to the livelock problem.

(Refer Slide Time: 59:08)



Different rollback recovery schemes; example of a direct dependency tracking technique. Here we assume that each process P i starts the execution where the initial checkpoint C i coma 0. Then we will compute the check point interval i capital I, i coma x.

That is nothing but an interval between 2 checkpoints that is C i x minus 1 and C i x. So, when P j process receives a message and during that the interval that is j, in j process the interval is y. It records the dependency from the interval of a process i except interval 2 the interval of j or yth interval and which is later stored into the stable storage when P j takes it is checkpoint C j y. So, along with the check points these intervals which are called dependency tracking is also stored.