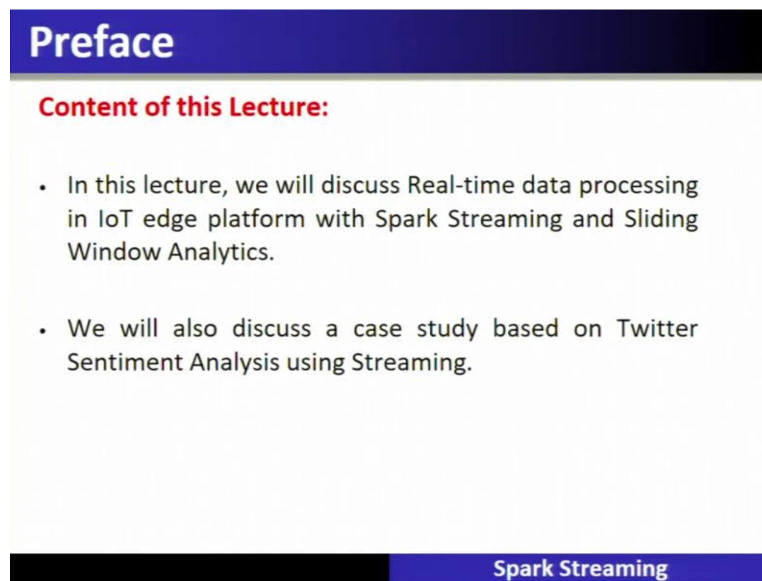
Foundation of Cloud IoT Edge ML
Professor Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna
Lecture - 16
Hot Data Analytics for Real Time Streaming in IoT Platform

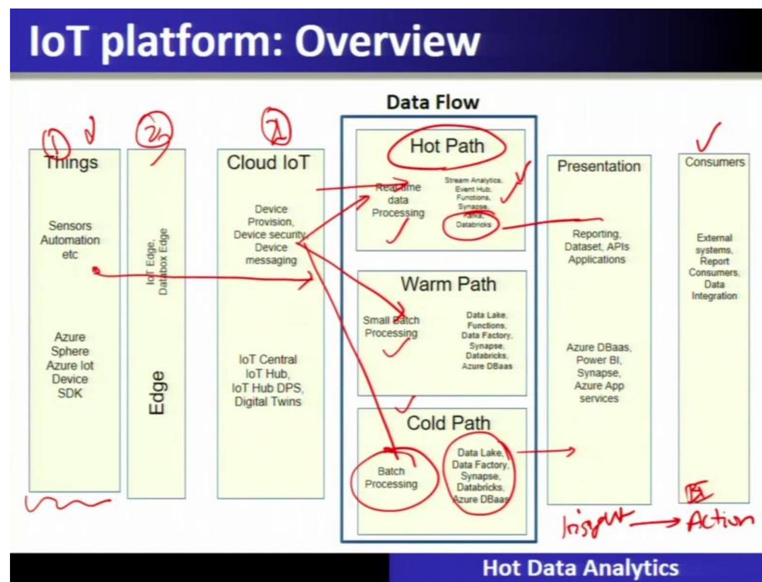
I am Doctor Rajiv Misra from IIT Patna. The topic of this lecture is Hot Data Analytics for Real Time Streaming in IoT Platform.

(Refer Slide Time: 0:24)



In this lecture, we will discuss real time data processing in IoT Edge platforms with Spark Streaming and Sliding Window Analytics, you will also discuss the case studies based on a particular use case that is with Twitter sentiment analysis using the streaming.

(Refer Slide Time: 0:46)



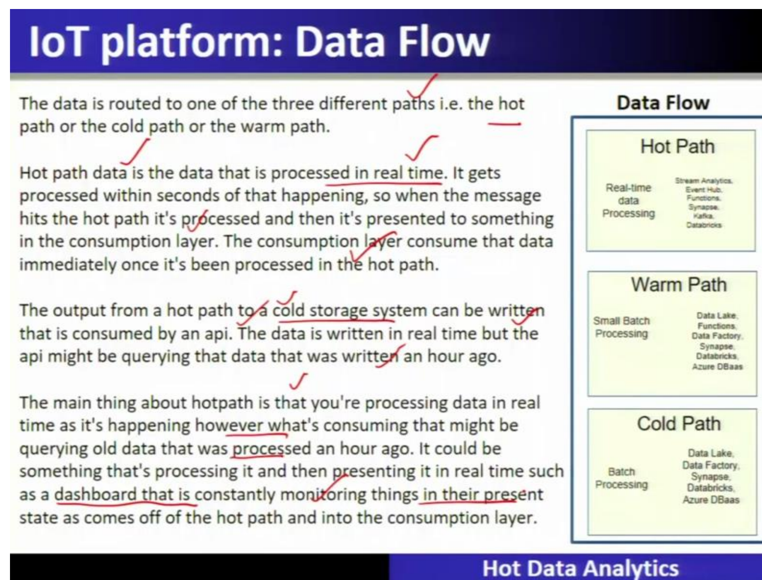
Let us give you IoT platform and overview which we have already covered. So, it is a recap that this particular IoT platform is divided into the things that is then coming the IoT platform and after that this edge on IoT Edge is being added. So, therefore, these sensors and actuators are part of these things and when they will ingest the data it goes into the cloud through the edge. So, the connectors which are there for data ingestion, so, data ingestion will happen using the Kafka connector or MQTT streams, this particular data stream is put into the system that is the cloud IoT platform.

Now, then that this is the routine that is called Real Time Data Processing, if it is the hot path data analysis, it will be performed using the stream analytics which we are now covering up. So, for this there will be the tools which are called Kafka, which we have covered. So, this will ingest the data for an enabling this real time data processing. So, we are going to cover for this part, which will be performed in the cloud that is using the Spark Streaming this data analysis will give the insight and this in turn further be used by business intelligence for the action part.

So, therefore, finally the end use is with the consumers who will be the beneficiary for making all the decisions which is taken by. So, therefore, there are three types of data flow which we will cover here. The first one is called hot path data analysis. So, data streams as it comes, it will put in the real time data processing using Spark Streaming which we will cover besides this hot path, let me tell you about that, we are also having two more type of data analysis that is called cold path that is a form of batch processing.

When you say batch processing that means, data will be put at the rest in one of these database technologies. And then, using this particular data at a later point of time, we will be performing various kind of insights and business intelligence called cold path. Warm path is to do the small batch processing in the batches of a small part will be taken up for this particular analysis comes in between the hot path and the cold path analysis.

(Refer Slide Time: 3:57)



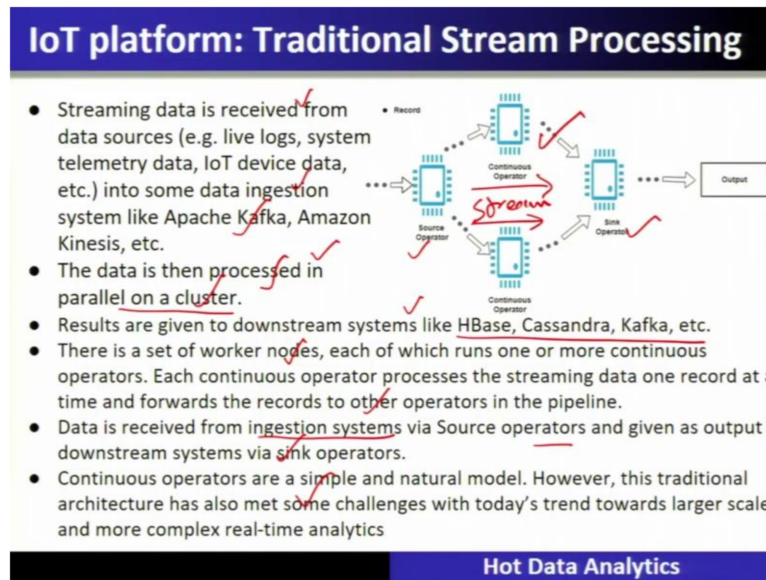
So, the topic which is kept here in this lecture is dealing with the hot path analysis. So, data is routed to three different paths. One is called hot path or cold path or a warm path. So, hot path data is the data that is processed as it is ingested into the system, that is the stream analysis is to be done for hot path data. And that is to be done in the real time. So, as it gets processed within the seconds of it is happening. So, when the message hits the hot path, it is processed and is presented to something in the consumption layer.

So, consumption layer will consume that processed information and this is called the hot path, the output of the hot path to the cold storage system or let us say that storage system to make it persistent and is consumed by these API's. So, the data which is written in the real time, but API's might be coding that particular data that was written an hour ago in the batch mode for this hot path analysis.

So, the main thing about the hard part is that when you are processing the data in the real time it is happening however, when it is consuming that might be acquiring old data that was processed an hour ago. So, that means the processing is to be done in the real time and that

result will be displayed into the dashboard, that is the insight will be available and will be given to the consumption layer.

(Refer Slide Time: 5:46)



So, let us discuss about the traditional stream processing and then we will go and discuss more advanced versions. So, streaming data that is received from various data sources such as live log and system telemetry data, IoT device data. So, these streaming data now is ingested using some of these tools, which we have already covered that is Apache Kafka or Amazon kinases. So, this particular is streaming so, that is ingested into the system and is then processed in parallel on the cluster.

So, you can see that this particular source will give the continuous stream of data into the system, which is then put in the process under this platform which is called a cluster. So, the details are given to the downstream systems like HBase or Cassandra. And so, here you can see that the results are now put in the data base like HBase, Cassandra and Kafka et cetera. So, now, there are set of worker nodes, each of which runs one or more continuous operations that is what is shown. So, each continuous operator processes the streaming data one record at a time and forwards the record to the other operators in the pipeline that is the sink operator.

So, data is received from that ingestion system via the source operators and given as the output to the downstream systems via these sync operators. So, continuous operators are simple and natural model however, this traditional architecture also has met challenges with today's trend towards larger scale and more complex real time analytics.

(Refer Slide Time: 7:45)

Traditional Stream Processing: Limitations

- **Fast Failure and Straggler Recovery** In real time, the system must be able to fastly and automatically recover from failures and stragglers to provide results which is challenging in traditional systems due to the static allocation of continuous operators to worker nodes.
- **Load Balancing** In a continuous operator system, uneven allocation of the processing load between the workers can cause bottlenecks. The system needs to be able to dynamically adapt the resource allocation based on the workload.
- **Unification of Streaming, Batch and Interactive Workloads** In many use cases, it is also attractive to query the streaming data interactively, or to combine it with static datasets (e.g. pre-computed models). This is hard in continuous operator systems which does not designed to new operators for ad-hoc queries. This requires a single engine that can combine batch, streaming and interactive queries.
- **Advanced Analytics with Machine learning and SQL Queries** Complex workloads require continuously learning and updating data models, or even querying the streaming data with SQL queries. Having a common abstraction across these analytic tasks makes the developer's job much easier.

Hot Data Analytics

So, therefore, this traditional stream processing which we have covered in the previous slide has the following limitations. First is that it is having the fast failure and the straggler recovery. So, in the real time system must be able to fastly and automatically recover from the failure and stragglers to provide the result which is challenging in the traditional system due to the static allocation of continuous operators to the worker nodes. So, due to these issues, this is the limitation of traditional stream processing.

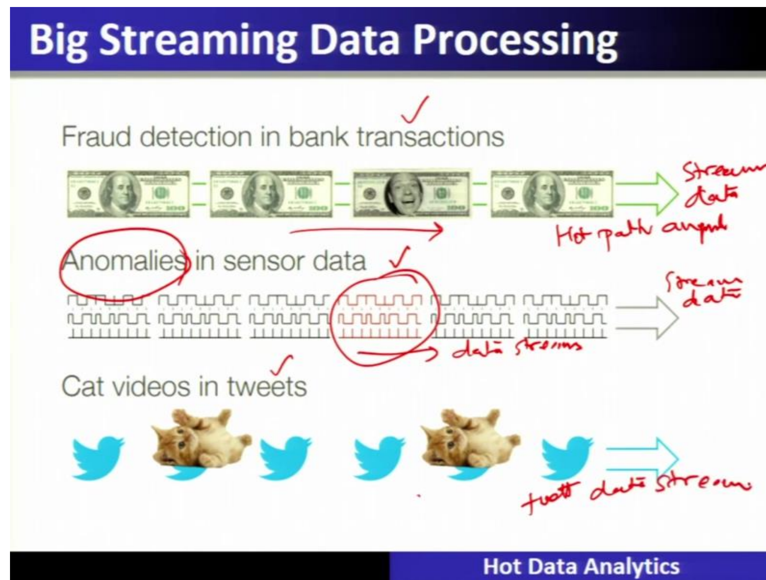
The other limitation is in the form of load balancing. So, in a continuous operating system, uneven allocation of the processing load between the worker can cause the bottlenecks and the system need to be able to dynamically adapt to the resource allocation based on the workload. So, therefore, load balancing is again another limitation of traditional stream processing system.

Third one is unification of streaming batch and interactive workloads. So, in many cases, it is also attractive to query the streaming data interactively or to combine it with a static data set in most of the application that is called pre-computed model. But this is hard in continuous operator system, which does not designed to the new operator for ad-hoc queries. So, this requires a single engine that can combine both batch streaming an interactive queries and that is unification of streaming batch and interactive workload is again a limitation of traditional systems.

Then the fourth one is advanced analytics with the machine learning and SQL queries. So, complex workloads requires continuous learning and updating data models or even querying

the streaming data using SQL queries however in the common abstraction across these analytics makes the developer job much easier.

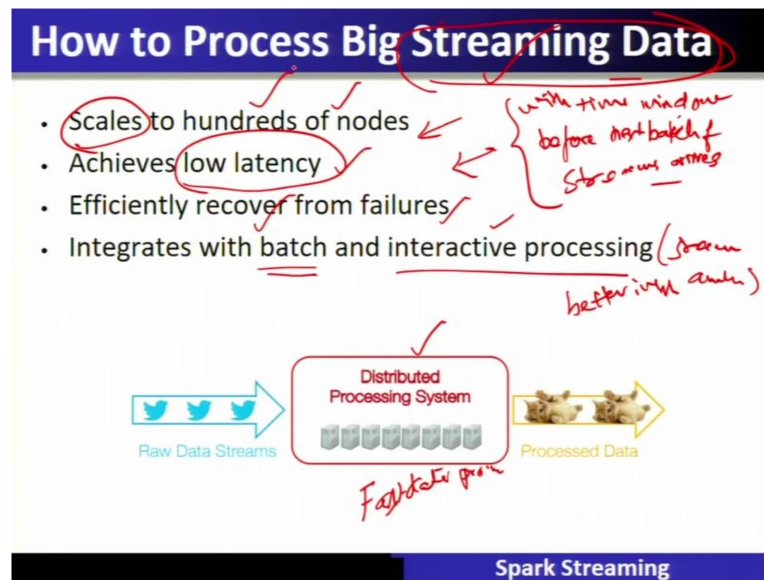
(Refer Slide Time: 9:45)



So, let us take the example of the streaming data processing. So, let us say that the bank transactions are continuously flowing and coming in the form of the streams and this particular stream data need to be processed as the data is entered into the system that is called hot path analysis. Now, similarly the sensor data now sensor data also comes in the form of a data streams.

So, this streaming data need to be analyzed to find out the anomalies which is arising in this particular sensor data. So, anomalies is the deviation from the specified state of the system. So, this particular anomaly is to be detected in the streaming data in the real time. Similarly, the tweet data is also coming in the streams. So, tweet data is also come in the streams need to be analyzed as it is ingested into the system.

(Refer Slide Time: 11:03)



Now, the question is how to process this streaming data? Now, this processing of the streaming data requires to be completed within a particular time window before the next batch of the extremes arrives in the system. So, therefore this particular processing which is in the real time needs to scale even to the hundreds of the nodes so, that in parallel they will be able to complete the execution of the streaming data otherwise, the application may miss the insights which is there inside that streaming data which is ingested into the system.

Similarly, this is one aspect called scale to hundreds of the nodes. Second is that it achieves a low latency, low latency means that this particular processing of the streaming data has to be completed very fast. So, it has to achieve the low latency otherwise, it might miss the batch and a new batch of the data will come and the previous batch will remain unprocessed.

Third one is to efficiently recover from the failures. So, failures in the sense if the nodes are down even then the system has to work irrespective of the node failures. So, therefore redundancy and all that are becoming a norm and you know that in a big data system, the number of clusters where the nodes are maybe considered as a commodity nodes they may fail. So, the system has to ensure that it will efficiently recover from the failures.

Fourth important thing is the requirement that sometimes it has to be integrated with the batch and the interactive processing. So, your stream analytics or analysis has to be integrated also with the batch and interactive processing for better insights sometimes. So, all these four requirements is bringing up or is making the specification for streaming data analytics.

And you can see that as you have seen the first requirement that it has to be scalable, therefore the streaming processing system, it has to be a distributed processing system where it can be scaled to hundreds of the nodes depending upon the pace in which the data is coming. So, this will have something called a fast data processing. So, fast data processing requires the skills to the hundreds of the nodes here in this case.

(Refer Slide Time: 14:04)

What people have been doing?

- Build two stacks – one for batch, one for streaming
  - Often both process same data
- Existing frameworks cannot do both
  - Either, stream processing of 100s of MB/s with low latency
  - Or, batch processing of TBs of data with high latency

Hot Data Analytics

Now, what people have been doing so, far, let us see how these particular four requirements will be able to solve. Let us people use it, they build two different stacks, one for the batch and one for the streaming often both process the same data. So, existing framework cannot do both, because either the stream processing of hundreds of megabytes with the low latency or the batch processing of terabytes of data with a high latency do both cannot exist in the system.

(Refer Slide Time: 14:36)

What people have been doing?

- Extremely painful to maintain two different stacks
  - Different programming models
  - Doubles implementation effort
  - Doubles operational effort

Hot Data Analytics

So, it is extremely painful to maintain two different stacks and different programming models doubles the implementation cost so much issues are there in this mode of combining batch and stream with the two different stacks.

(Refer Slide Time: 14:52)

Fault-tolerant Stream Processing

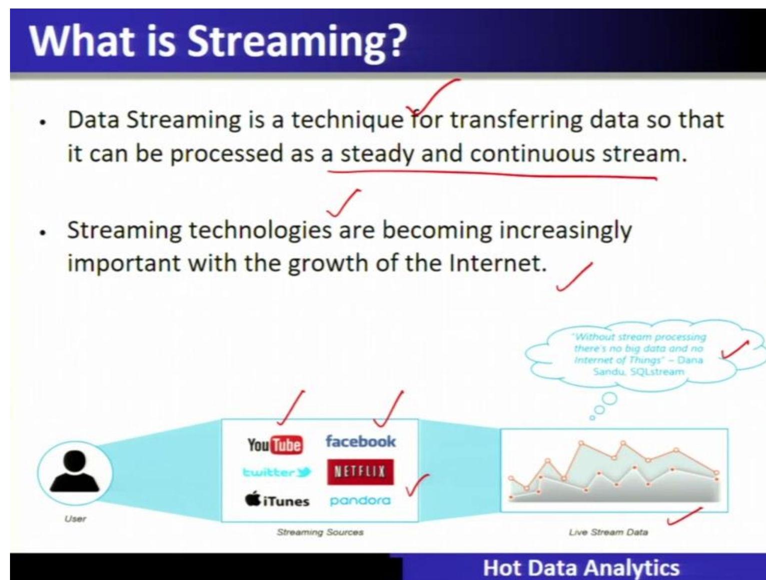
- Traditional processing model
  - Pipeline of nodes
  - Each node maintains mutable state
  - Each input record updates the state and new records are sent out
- Mutable state is lost if node fails
- Making stateful stream processing fault-tolerant is challenging!

mutable state
input records
node 1
input records
node 2
node 3

Hot Data Analytics

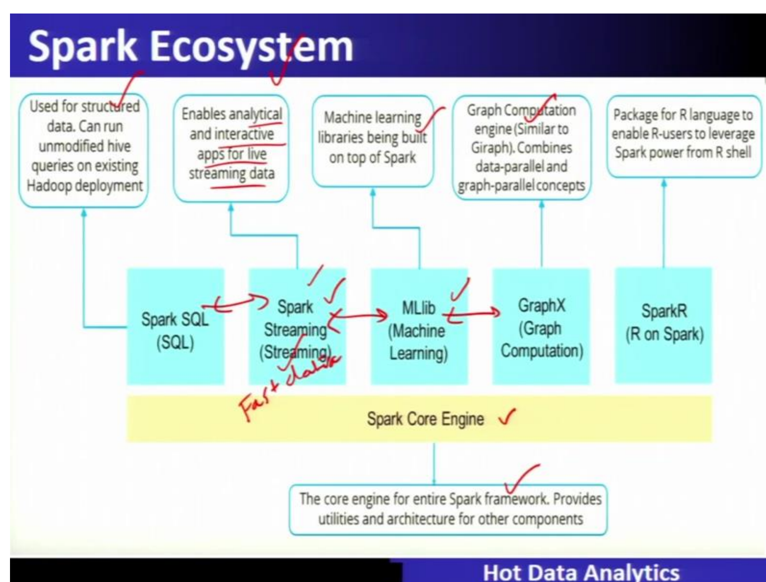
Therefore, next issue is the fault tolerant stream processing how that is being done. In the traditional processing models so, there is a pipeline of nodes and each node maintains a mutable states and each state records the updates the state and a new records are being sent out. So, now maintaining the states in this fault tolerant stream processing in a traditional mode, where the mutable state is lost if the node is failed making this stateful stream processing fault tolerant quite challenging in this particular case.

(Refer Slide Time: 15:30)



Now, let us understand about the data streaming. So, data streaming is a technique for transferring the data so, that it can be processed as a steady and continuous stream. So, example is that streaming technologies are becoming increasingly important with the growth of the internet. So, here you can see that the streaming sources such as the YouTube, Netflix Twitter will give a live stream of data and without stream processing, there is no big data and no internet of things. So, people have to now deal with this streaming system.

(Refer Slide Time: 16:05)



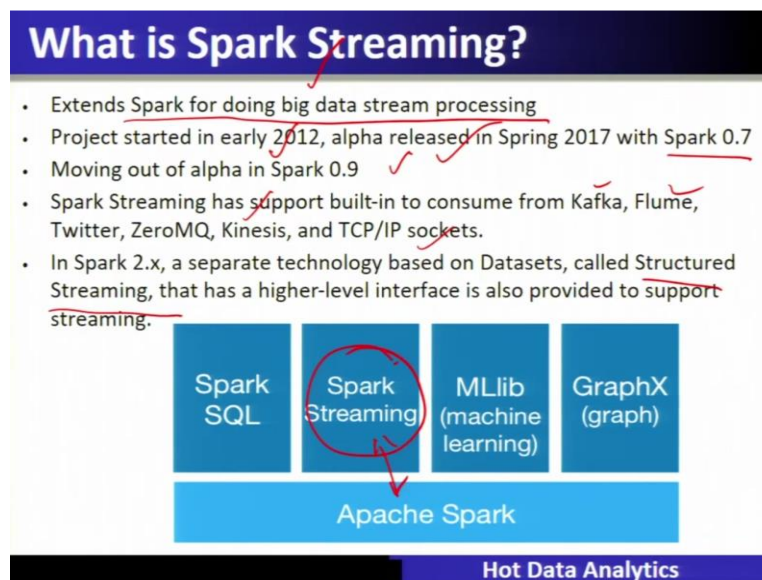
Now, let us see about the Spark ecosystem. In the Spark ecosystem, you have to now see that there is a Spark core engine and over the Spark core engine there are different libraries for different types of application. And one such application is called Spark Streaming for

streaming data analytics, and this is also called the fast data processing. So, this particular Spark is streaming enables the analytical and interactive applications for the live streaming data.

Now, this particular platform which is provided by the Spark ecosystem is integrated over the Spark core. So, the Spark is streaming that is the streaming data analytics, you can apply the machine learning library, which is also supported in the Spark core over the Spark Streaming and therefore, you can gain insight by applying the machine learning libraries which are built on top of that Spark. Sometimes, using this particular Spark Streaming and the graphics you can construct a graph computation engine and combine the data parallel in the graph parallel operations.

Similarly, the Spark Streaming also can be converted or can be put into that format called Spark SQL and it can be used for the structured data and various queries thereafter. So, the core engine for the entire Spark framework provides the utilities and architecture for different components.

(Refer Slide Time: 17:41)



So, let us go in more detail about this component which is called the Spark Streaming supported over the Spark core engine. So, this particular Spark Streaming this extends the Spark for doing big data stream processing. So, this particular Spark Streaming project is started in 2012 with alpha released in spring 2017 with Spark 0.7. Moving out of this alpha Spark 0.9 is now used. So, the Spark Streaming has support for built in and they will consume from Kafka, Flume, Twitter and other socket for data ingestion into the system. So,

Spark 2.x is separate technology based on the data sets called structured streaming. So, that has a higher level of interface and is also provided to support the streaming.

(Refer Slide Time: 18:41)

What is Spark Streaming?

- Framework for large scale stream processing
  - Scales to 100s of nodes
  - Can achieve second scale latencies
  - Integrates with Spark's batch and interactive processing
  - Provides a simple batch-like API for implementing complex algorithm
  - Can absorb live data streams from Kafka, Flume, ZeroMQ, etc.

Hot Data Analytics

Now, the framework for the large scale is stream processing. Now, we required to scale to hundreds of nodes and achieve a second scale latency and has to integrate with the Sparks batch and interactive processing provides a simple batch like API for implementation of the complex algorithms and can absorb the live data streams from Kafka, Flume, ZeroMQ and so on.

(Refer Slide Time: 19:08)

What is Spark Streaming?

- Receive data streams from input sources, process them in a cluster, push out to databases/ dashboards
- Scalable, fault-tolerant, second-scale latencies

Kafka
Flume
HDFS
Kinesis
Twitter

Spark Streaming

HDFS
Databases
Dashboards

Hot Data Analytics

So, let us see about the Spark Streaming. So, a Spark Streaming technologies will receive the data stream from the input source. So, that is what is shown were here, a process done in the cluster that is called the Spark Streaming which is running under the Spark core and the Spark core runs on the cluster, push out to the database here that is all shown.

So, therefore, it supports scalable, fault-tolerant and second-scale latencies that is the requirements we have seen set for stream processing. So, therefore Spark Streaming engine will now support the data ingestion from different sources, Kafka, Flume, HDFS, kinases and Twitter. Now, this will be processed in the Spark Streaming and then it will be stored either in HDFS, database or a dashboard.

(Refer Slide Time: 20:07)

Why Spark Streaming ?

- Many big-data applications need to process large data streams in realtime

Website monitoring

Fraud detection

Ad monetization

Hot Data Analytics

So, many big data application need to process the large data streams in the real time for example, the IoT applications. So, sometimes it is also used for website monitoring, fraud detection and ad monetization, which is happening on the fly.

(Refer Slide Time: 20:24)

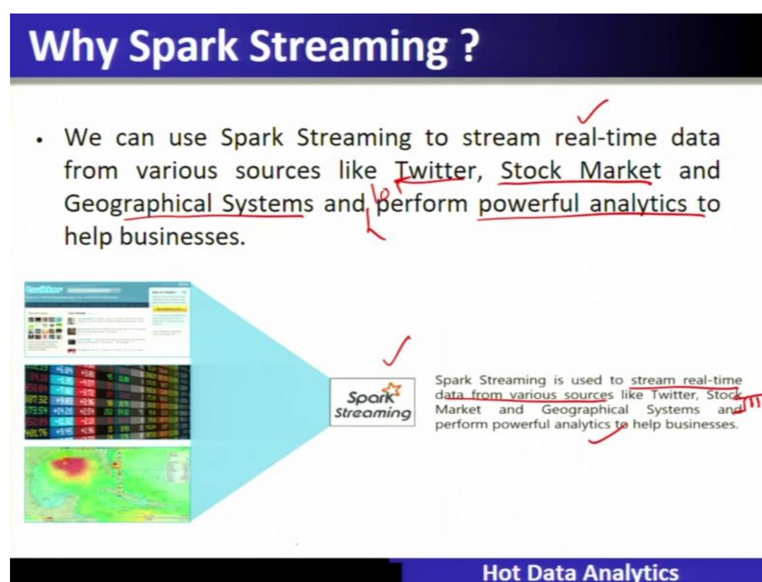
Why Spark Streaming ?

- Many important applications must process large streams of live data and provide results in near-real-time
  - Social network trends
  - Website statistics
  - Intrustion detection systems
  - etc.
- Require large clusters to handle workloads
- Require latencies of few seconds

Hot Data Analytics

So, many important application that process large streams of live data and provides the results in near real time. So, when we say near real time is that that means, the data comes in the batches and the processing takes a little delay, but it is almost real time for example, social network trend, website statistics, intrusion detection system and they require large clusters to handle the workloads and require latencies of the few seconds in this particular example.

(Refer Slide Time: 20:57)



So, let us see why Spark Streaming? So, we can use the Spark Streaming to stream the real time data from various sources like Twitter, Stock Market, Geographical Systems and some IoT data and IoT and perform the powerful analytics to help the business. So, Spark

Streaming that is used to stream the real time data from various sources, where IoT is one of that source and it will perform the powerful analytics to help the business by gaining the insight from that particular data into the system.

(Refer Slide Time: 21:33)

Why Spark Streaming?

Need a framework for big data stream processing that

Scales to hundreds of nodes

Achieves second-scale latencies

Efficiently recover from failures

Integrates with batch and interactive processing

Hot Data Analytics

So, there is a need for the framework for this kind of analytics, that is the big data or stream processing that can scale to hundreds of the nodes, achieve second latencies and efficiently recover from and it has to integrate with the batch and interactive processing.

(Refer Slide Time: 21:49)

Spark Streaming Features

- **Scaling:** Spark Streaming can easily scale to hundreds of nodes.
- **Speed:** It achieves low latency.
- **Fault Tolerance:** Spark has the ability to efficiently recover from failures.
- **Integration:** Spark integrates with batch and real-time processing.
- **Business Analysis:** Spark Streaming is used to track the behavior of customers which can be used in business analysis

Efficiently recover from failures
Fault Tolerance
Achieves low latency
Speed
Integration
Integrates with batch and real-time processing
Scales to hundreds of nodes
Scaling
Spark Streaming
Business Analysis
Used to track behaviour of customers

Hot Data Analytics

So, let us see the Spark Streaming features. It is scaling. Scaling is easy to scale speed, it achieves low latency, fault tolerance, integration and business analytics. So, all these are the features of Spark Streaming.

(Refer Slide Time: 22:06)

Batch vs Stream Processing

Batch Processing

- Ability to process and analyze data at-rest (stored data)
- Request-based, bulk evaluation and short-lived processing
- Enabler for Retrospective, Reactive and On-demand Analytics

Stream Processing

- Ability to ingest, process and analyze data in-motion in real- or near-real-time
- Event or micro-batch driven, continuous evaluation and long-lived processing
- Enabler for real-time Prospective, Proactive and Predictive Analytics for Next Best Action

Stream Processing + Batch Processing = All Data Analytics

real-time (now) historical (past)

Hot Data Analytics

Now, let us consider the stream processing versus the batch processing. So, batch processing is the ability to process the data, analyze the data at the rest that is the data is stored in the database is and it will be retrieved the data from the database. So, the data is first stored in the database and then it will be analyzed at a later point of time and this is called the stored data processing or the batch data processing.

So, this batch data processing is request based, data is available or at the rest into the database or in the storage system and whenever the request comes a bulk evaluation is to be performed on the stored data and this particular type of processing called a batch processing is a short-lived processing on demand or a request based.

Now, this batch processing enables for retrospective, reactive and on-demand analytics. Now, other types of processing is called a stream processing. So, stream processing is the ability to ingest, process and analyze the data in motion or in the real or a near real time. So, that is called the stream processing. So, events or the micro batch driven continuous evaluation and long-lived processing. So, stream processing is the enabler for real time prospective, proactive and predictive analytics for the next best action. Therefore, if you combine this stream processing and batch processing together, then it will become the all data analytics.

(Refer Slide Time: 23:44)

Integration with Batch Processing

- Many environments require processing same data in live streaming as well as batch post-processing
- Existing frameworks cannot do both
  - Either, stream processing of 100s of MB/s with low latency
  - Or, batch processing of TBs of data with high latency
- Extremely painful to maintain two different stacks
  - Different programming models
  - Double implementation effort

Hot Data Analytics

Modern Data Applications approach to Insights

Traditional Analytics
Structured & Repeatable
Structure built to store data

HYPOTHESIS → QUESTION → DATA → ANSWER

Start with hypothesis
Test against selected data

Analyze after landing...

Next Generation Analytics
Iterative & Exploratory
Data is the structure

DATA → EXPLORATION → CORRELATION → ACTIONABLE INSIGHT

Data leads the way
Explore *all* data, identify correlations

Analyze in motion...

Hot Data Analytics

Now, let us see that how you can integrate your stream processing with the batch processing. So, many environments require the processing the same data in live streaming as well as in the batch post processing. So, existing framework cannot do both and therefore requires a new kind of modern applications. So, let us see there are two ways of doing the analytics. One is that the traditional way of doing analytics structured and repeatable structure being to store the data that is you should having the question or the queries which need to be performed on the data.

So, therefore the hypothesis will pose the number of questions and based on these questions, the entire analytics pipeline is designed. So, you have to start with a hypothesis then you have

to test against the selected data and analyze after the landing. So, this kind of technique is called traditional analytics.

Whereas the next generation analytics starts with the iterative and exploratory data that is the structure. So, data is now processed and a correlation or a linear regression is established out of that particular pattern or data and then using this particular model later on this (())(25:00) of explore all the data and identity correlation and analyze the data in the motion that is the next generation analytics, which we will talk about.

(Refer Slide Time: 25:10)

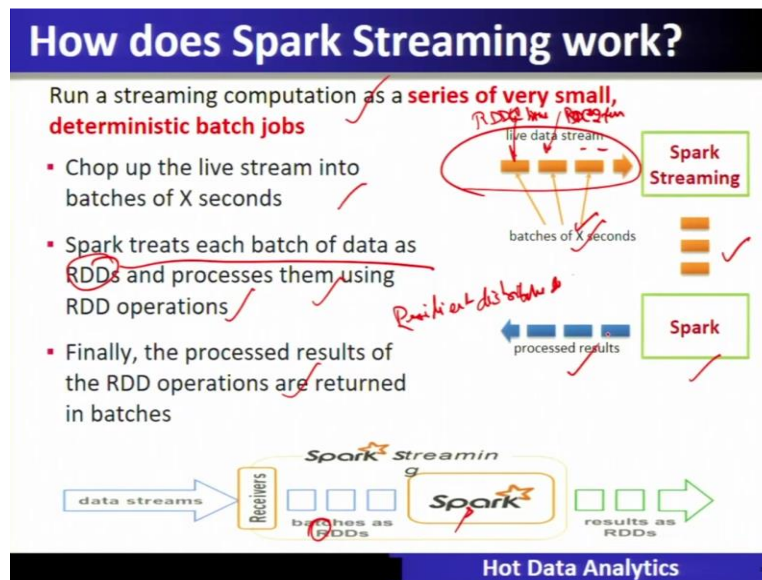
Existing Streaming Systems

- Storm
  - Replays record if not processed by a node
  - Processes each record *at least once*
  - May update mutable state twice!
  - Mutable state can be lost due to failure!
- Trident – Use transactions to update state
  - Processes each record *exactly once*
  - Per-state transaction to external database is slow

Hot Data Analytics

Now, existing streaming systems are storm and trident. So, these follow different kinds of semantics. So, at least one semantics is used in the storm. So, processes each record at least once and trident use the transaction to update the status and the processes and each record exactly once. So, both of them have a different kind of ways.

(Refer Slide Time: 25:37)



So, how does the Spark Streaming work? So, let us see that you have to run the streaming computation as a series of very small and deterministic batch jobs. So, livestream jobs will come into the system for that it divides the batches of X seconds and this particular live stream is divided into X seconds and Spark treats these each batch of data as RDDs and process them using RDDs. So, RDDs full form is Resilient Distributed Data. So, this is the unit of processing into the Spark core engine. So, Spark reads each batch of the data as RDDs and process them using RDD Operation. Finally, the process result of RDDs are returned in the batches.

So, you can see that this is the live stream. Live stream is divided into the batches of X seconds and each particular batch of X second is called an RDD at the rate let us say one-time unit, this is RDD at the rate two-time unit and so on. So, this particular batches of RDDs which is the batches of X second is given to the Spark Streaming for doing this computation. Now, Spark Streaming in turn, this gives these batches called RDDs to the Spark and they will process the result and give it back the results in this manner.

(Refer Slide Time: 27:12)

How does Spark Streaming work?

Run a streaming computation as a **series of very small, deterministic batch jobs**

- Batch sizes as low as ½ second, latency of about 1 second
- Potential for combining batch processing and streaming processing in the same system

live data stream

Spark Streaming

batches of X seconds

Spark

processed results

Spark Streaming

data streams

Receivers

batches as RDDs

Spark

results as RDDs

Hot Data Analytics

So, run the stream processing as the series of very small deterministic batch jobs. So, batch sizes are half a second, latency is about 1 second and potential for combining the back end the stream processing is possible in this platform.

(Refer Slide Time: 27:29)

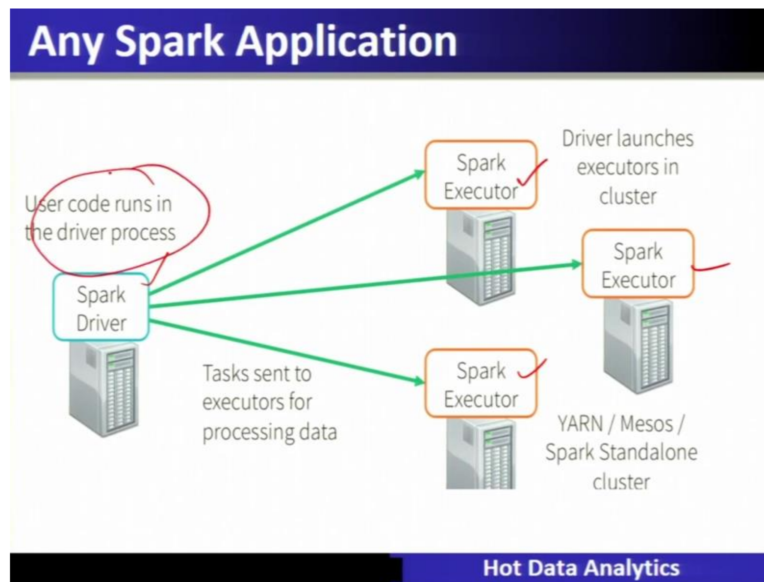
Word Count with Kafka

```
object WordCount {
  def main(args: Array[String]) {
    val context = new StreamingContext(new SparkConf(), Seconds(1))
    val lines = KafkaUtils.createStream(context, ...)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x,1)).reduceByKey(_ + _)
    wordCounts.print()
    context.start()
    context.awaitTermination()
  }
}
```

Hot Data Analytics

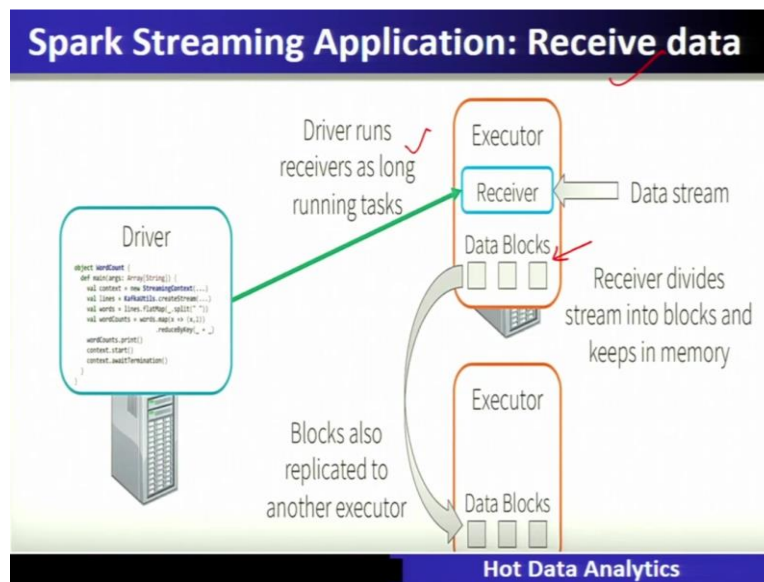
So, let us see how you can do the word count with the Kafka. So, you can see this Kafka will ingest take the data and perform this word count program in this particular case using map and reduce by the key.

(Refer Slide Time: 27:44)



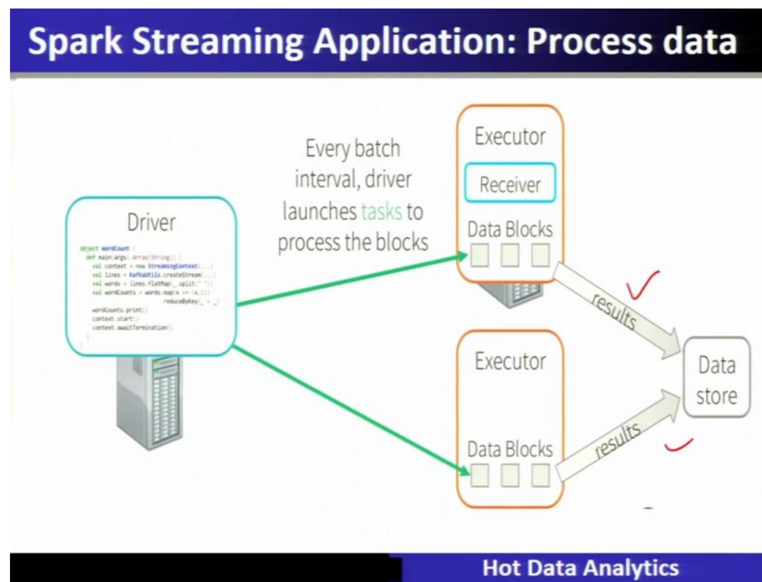
So, let us see any Spark application. It starts with the Spark driver. These tasks are sent for processing this particular data through the Spark execution. So, the user core will run in the driver process.

(Refer Slide Time: 27:59)



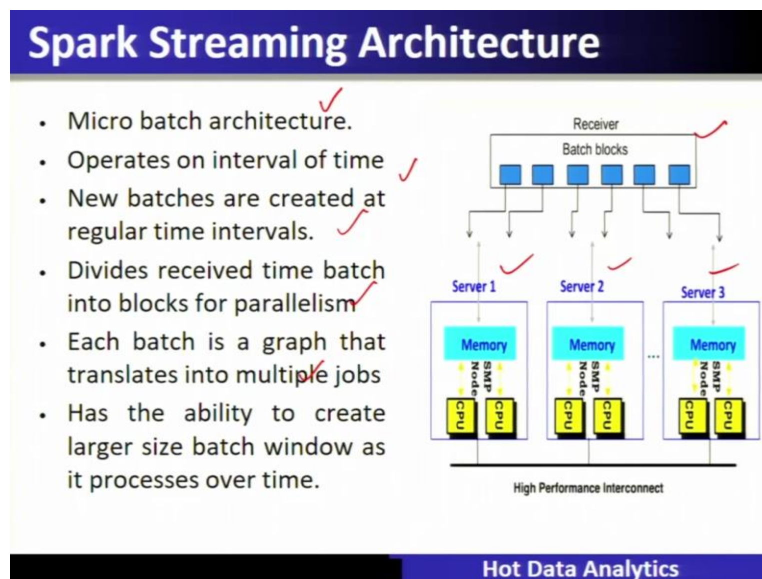
So, Spark Streaming application this example shows how to receive the data. So, data will be received by the driver. Driver runs the receiver as long as it is running the task through the executor will receive the data stream and it will divide the data blocks and give it to the other executors to execute in this manner and keep it into the memory after the execution.

(Refer Slide Time: 28:25)



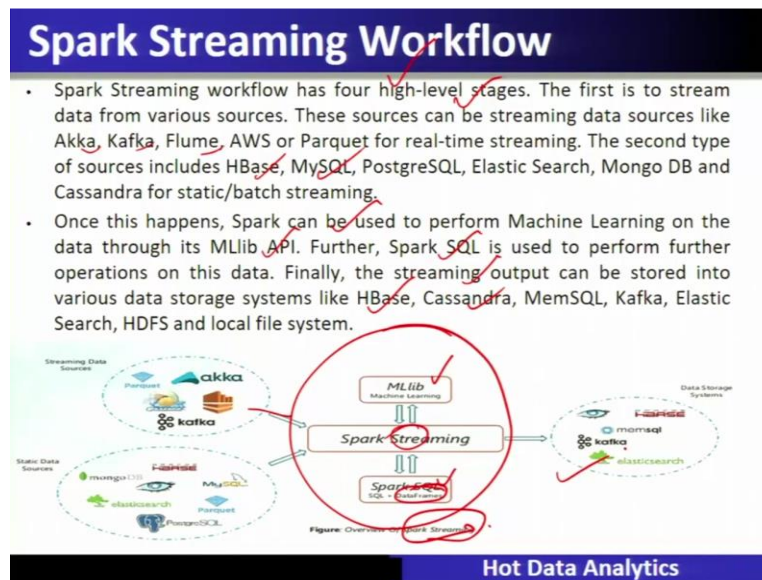
So, this is the processing. Once the executor completes the processing the results are given to the database and it will be stored in these particular batches.

(Refer Slide Time: 28:34)



So, Spark Streaming architecture is the following. So, it is called the micro batch architecture where the receiver will receive the batch blocks and these batch blocks are given to the executors that is different servers and these different servers will do the computation and that is shown what here. So, this particular micro batch architecture operates in the interval of time. New batches are created at regular time intervals divides the time batch into the blocks for parallelism. Each batch is a graph that translates into multiple job and is able to create large size batch window as it process over the time.

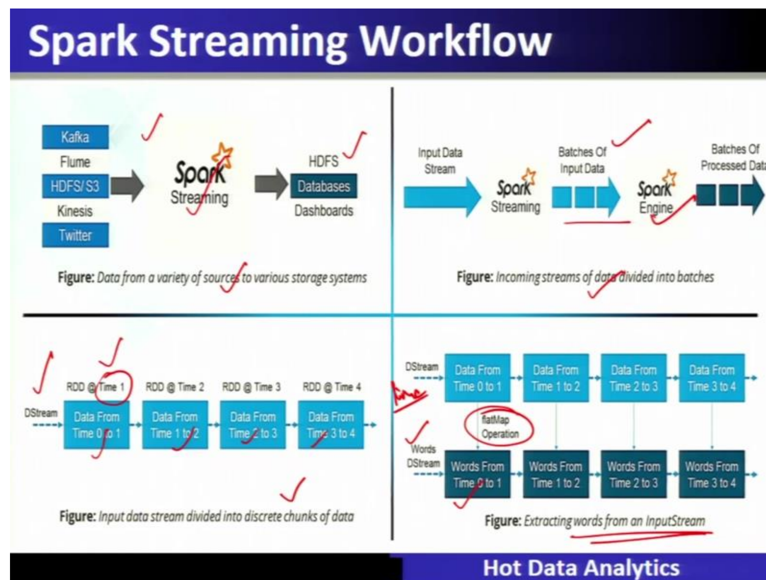
(Refer Slide Time: 29:12)



Now, Spark Streaming workflow has four high level stages. The first is to stream the data from various sources. These particular sources can be streaming data sources like Akka, Kafka, Flume, AWS, and so on. The second type of source includes HBase, MySQL, and so on. Once this happens, Spark can be used to perform machine learning on the data through the machine learning MLlib API. Further Spark SQL is used to perform further operations on this particular data. Finally, streaming output can be stored into the various data storage system like HBase, Cassandra and so on.

So, this is the typical overview which we have given you about the Spark Streaming processing. So, here you can see that once the data is ingested out of this Kafka and then through the Spark Streaming, what you can perform is the machine learning using MLlib or with using the Spark SQL you can convert into the data frame and then use it to the further level.

(Refer Slide Time: 30:17)

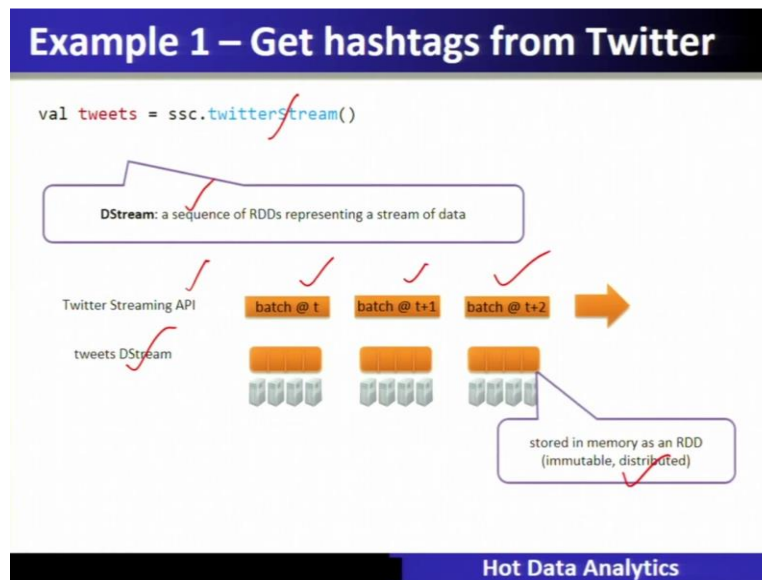


So, Spark Streaming workflow if you see that data from a variety of sources to the various storage systems, it will be transformed so, Kafka, Flume, they will ingest the data and put for the Spark Streaming and finally, the result will be put in different storage system like HDFS, databases dashboard and so on. So, here in this particular figure, it is shown that incoming stream of the data is divided into the batches and that will be done by the Spark Streaming and the batches of input data will be processed by the Spark engine and the batches of the process data will be put as a result.

So, this particular processing, which is shown what here is further detail is given. So, you can see that discretized stream is the proper data structure which is used. So, these are the batches are called D streams and D streams are nothing but the micro batch RDDs which are for a particular time interval they are being chopped off or being divided into that batches of different times. So, data of different timings is being broken up by Spark Streaming and give to the Spark engine for doing the computation.

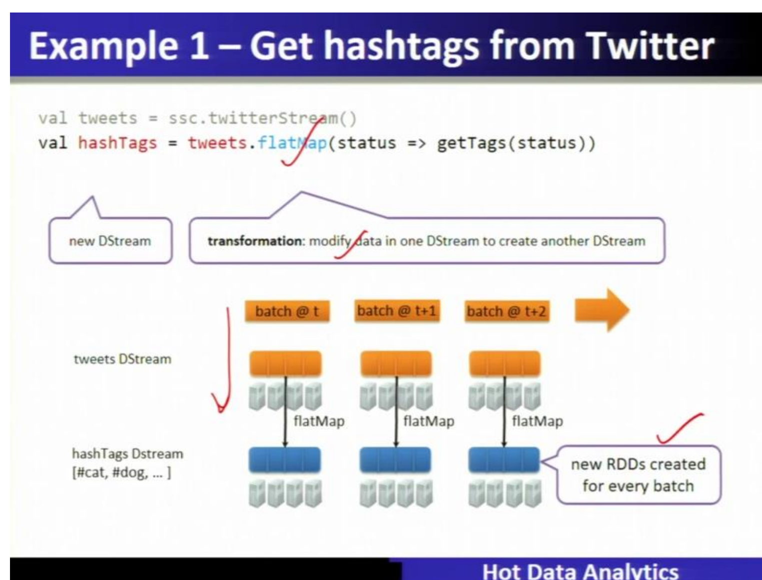
Now, once these D stream is given to the Spark engine, then different commands and the operations will be applied for example, flatMap, flatMap here in this example, a line in a in a particular document is now divided or is applied using a flatMap operation and from the line it will now it will divide into the words and the words are being extracted from the input streams between this so, this is how the Twitter data analytics can be performed.

(Refer Slide Time: 32:04)



So, you can see that in the tweets you can twitter stream will be nothing but in the form of D stream that is a sequence of RDDs representing the stream of the data. And this will be handled with the help of Twitter API. Now, this particular Twitter API will give the micro batches of different time intervals and they are called tweet D streams. Tweet D streams are stored in memory as an RDD that is immutable distributed data types.

(Refer Slide Time: 32:37)



Now, then on this particular micro batches, the operations like flatMap will be applied. So, flatMap will be a transformation will modify the data in one D stream to create another D stream. So, this particular transformation will happen using this new D stream it will create

using flatMap. So, new RDDs will be created for every batch. So, this is how the stream processing will take place.

(Refer Slide Time: 33:05)

Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.foreach(hashTagRDD => { ... })
```

foreach: do whatever you want with the processed data

tweets DStream

hashTags DStream

batch @ t

batch @ t+1

batch @ t+2

flatMap

foreach

Write to a database, update analytics UI, do whatever you want

Hot Data Analytics

So, the output of this particular transformation will be pushed to the external storage and so, that is what is shown over here and that will be stored either on HDFS or in some any other database.

(Refer Slide Time: 33:21)

Java Example

Scala

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Java

```
JavaDStream<Status> tweets = ssc.twitterStream()
JavaDstream<String> hashTags = tweets.flatMap(new Function<...> {  })
hashTags.saveAsHadoopFiles("hdfs://...")
```

Function object

Hot Data Analytics

Now, this particular use case, so, this particular Spark is streaming programming can be either done in a scalar or in the Java. So, you can see that flatMap and saving into the file, there are three lines of core written in the scala similarly, you can write in the Java also.

(Refer Slide Time: 33:41)

Fault-tolerance

- RDDs are remember the sequence of operations that created it from the original fault-tolerant input data
- Batches of input data are replicated in memory of multiple worker nodes, therefore fault-tolerant
- Data lost due to worker failure, can be recomputed from input data

tweets RDD
input data replicated in memory
flatMap
hashTags RDD
lost partitions recomputed on other workers

Hot Data Analytics

So, let us see about the other aspect that is the fault tolerance aspects. So, RDDs remember the sequence of operation that it created it from the original in original fault tolerant input data. So, these input data is now replicated in the memory like this, and when the flatMap is applied and if any data is lost, then you know that that replication will be used to process this. So, the batch of input data is replicated in the memory of the multiple worker node therefore, it is fault tolerant. So, data loss due to the worker failure can be re-computed from the input data.

(Refer Slide Time: 34:23)

Key concepts

- **DStream** – sequence of RDDs representing a stream of data
  - Twitter, HDFS, Kafka, Flume, ZeroMQ, Akka Actor, TCP sockets
- **Transformations** – modify data from on DStream to another
  - Standard RDD operations – map, countByValue, reduce, join, …
  - Stateful operations – window, countByValueAndWindow, …
- **Output Operations – send data to external entity**
  - saveAsHadoopFiles – saves to HDFS
  - foreach – do anything with each batch of results

Hot Data Analytics

So, these principles are already well established in the Spark core same is used for the fault tolerance. So, let us see the key concepts which we have quickly used up let us review that.

So, D stream is a sequence of RDD is representing the stream of data such as Twitter, Kafka, HDFS, Flume and so on. Transformation is to modify the data from D stream to another standard RDD functions and stateful functions such as windows and count by value and a window output operations is to send the data to the external entities such that HDFS.

(Refer Slide Time: 35:02)

Example 2 – Count the hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```

tweets, hashTags, tagCounts [(#cat, 10), (#dog, 25), ...]

batch @ t, batch @ t+1, batch @ t+2; flatMap, map, reduceByKey

Hot Data Analytics

Another example is to count the hashtags. So, this particular example is shown that hashtags are counted by the value. So, here after the flatMap, another operation is to be performed the map and reduced by the key, which will be doing that particular count operation that is reduced by key.

(Refer Slide Time: 35:25)

Example 3 – Count the hashtags over last 10 mins

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation, window length, sliding interval

window length, DStream of data, sliding interval

Hot Data Analytics

So, third operation is to count the hashtag over the last 10 minutes. So, you can see that you have to create a sliding window of 1-minute time. So, this is the 1-minute time, 1-minute interval of a sliding window and that particular window will now shift after every 5 seconds. So, this is called sliding window and it now, this particular hashtag counting count the hashtag will be performed inside this particular window of 1-minute time so, sliding window you know that it is window operation window length and sliding interval that I have already explained. So, after 5 seconds, it will shift to another shift to the right in this manner over the data.

(Refer Slide Time: 36:31)

Example 3 – Counting the hashtags over last 10 mins

```
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```

t-1 t t+1 t+2 t+3

hashTags

countByValue

tagCounts

count over all the data in the window

Hot Data Analytics

Smart window-based countByValue

```
val tagCounts = hashtags.countByValueAndWindow(Minutes(10), Seconds(1))
```

t-1 t t+1 t+2 t+3

hashTags

countByValue

tagCounts

subtract the counts from batch before the window

add the counts from the new batch in the window

Hot Data Analytics

So, therefore, if you see this kind of complete details, so, this is the hashtag using the sliding window of 10 minutes it can have 1, 2, 3, 4, 4 different micro batch RDDs and after 1 second

it has to shift. So, it will count over the data in that sliding window and this will be performed an action and after 1 second you know that it will be now shifting its values.

(Refer Slide Time: 37:00)

Smart window-based *reduce*

- Technique to incrementally compute count generalizes to many reduce operations
  - Need a function to "inverse reduce" ("subtract" for counting)
- Could have implemented counting as:
  hashTags.reduceByKeyAndWindow(_ + _, _ - _, Minutes(1), ...)

Hot Data Analytics

Arbitrary Stateful Computations

Specify function to generate new state based on previous state and new data

- Example: Maintain per-user mood as state, and update it with their tweets

```
def updateMood(newTweets, lastMood) => newMood

moods = tweetsByUser.updateStateByKey(updateMood _)
```

Hot Data Analytics

So, smart window based reduce operation is being supported here in Spark Streaming and also will be arbitrary stateful computation is also supported here, which is nothing but update state by the key.

(Refer Slide Time: 37:16)

Arbitrary Combinations of Batch and Streaming Computations

Inter-mix RDD and DStream operations!

- Example: Join incoming tweets with a spam HDFS file to filter out bad tweets

```
tweets.transform(tweetsRDD => {
    tweetsRDD.join(spamHDFSFile).filter(...)
})
```

Hot Data Analytics

So, arbitrary combination of batch and stream computation is shown here in this particular example, the joint incoming tweet with the spam HDFS file to filter for this is you know that it will be joined here with the tweet data that is the stream data.

(Refer Slide Time: 37:33)

Spark Streaming-Dstreams, Batches and RDDs

input Streaming data

Spark Streaming

rdd-3 @ time-3

data from time 2 to 3

rdd-2 @ time-2

data from time 1 to 2

rdd-1 @ time-1

data from time 0 to 1

Spark Engine

processed batch data

- These steps repeat for each batch.. Continuously
- Because we are dealing with Streaming data. Spark Streaming has the ability to "remember" the previous RDDs...to some extent.

Hot Data Analytics

So, Spark Streaming, D streams and batch and RDDs are the different data type or data structures of a Spark Streaming is being followed here in this Spark Streaming. So, input streaming after going through Spark Streaming it will divide into the different micro batches and these are all RDDs of different time duration and they have to be processed by the Spark engine in the batch. So, these particular it will repeat for each batch and because we are

dealing with a Spark Streaming data the streaming has the ability to remember the previous RDDs to some extent.

(Refer Slide Time: 38:11)

DStreams + RDDs = Power

- Online machine learning
  - Continuously learn and update data models (*updateStateByKey* and *transform*)
- Combine live data streams with historical data
  - Generate historical data models with Spark, etc.
  - Use data models to process live data stream (*transform*)
- CEP-style processing
  - window-based operations (reduceByWindow, etc.)

Hot Data Analytics

From DStreams to Spark Jobs

- Every interval, an RDD graph is computed from the DStream graph
- For each output operation, a Spark action is created
- For each action, a Spark job is created to compute it

DStream Graph
In
In
Union
U
Transform
T
Action
Ac
T
T
Ac
Ac

Block RDDs with data received from the last batch interval

RDD Graph
B
B
Block RDDs
U
T
Ac
T
T
Ac
Ac
3 Spark jobs

Hot Data Analytics

So, D streams plus RDDs will become the power of computation for doing the analytics and you can see that from D stream to the Spark jobs, you can create the, for each interval RDD graph is computed for D stream graph. So, this is called a D stream graph and this D stream graph will contain all the operations which are to be performed on the micro batch RDDs, and the output operation Spark action is created based on this particular RDDs.

(Refer Slide Time: 38:49)

Input Sources

- Out of the box, we provide
  - Kafka, HDFS, Flume, Akka Actors, Raw TCP sockets, etc.
- Very easy to write a *receiver* for your own data source
- Also, generate your own RDDs from Spark, etc. and push them in as a "stream"

Hot Data Analytics

So, input source are that out of the box, the input sources which are being supported here in Spark Streaming is Kafka, HDFS, Flume, Akka Actors, Raw TCP sockets. So, input streams input sources, it is very easy to write a receiver for your own data source also generate your own RDDs from the Spark and push them in the stream.

(Refer Slide Time: 39:12)

Current Spark Streaming I/O

- Input Sources
  - Kafka, Flume, Twitter, ZeroMQ, MQTT, TCP sockets
  - Basic sources: sockets, files, Akka actors
  - Other sources require receiver threads
- Output operations
  - Print(), saveAsTextFiles(), saveAsObjectFiles(), saveAsHadoopFiles(), foreachRDD()
  - foreachRDD can be used for message queues, DB operations and more

Kafka
Flume
HDFS/S3
Kinesis
Twitter
Spark Streaming
HDFS
Databases
Dashboards

Hot Data Analytics

So, the current Spark Streaming input output, you can see that it is supporting various input sources like Kafka, Flume, Twitter, ZeroMQ and MQTT and so on. And the basic sources are the socket, files, akka actors and so on. Output operations are also supported over here.

(Refer Slide Time: 39:30)

Dstream Classes

- Different classes for different languages (Scala, Java)
- Dstream has 36 value members
- Multiple types of Dstreams
- Separate Python API

org.apache.spark.input hide focus
PortableDataStream
org.apache.spark.serializer hide focus
DeserializationStream
org.apache.spark.streaming.api.java hide focus
JavaDStream
JavaDStreamLike
JavaInputDStream
JavaPairDStream
JavaPairInputDStream
JavaPairReceiverInputDStream
JavaReceiverInputDStream
org.apache.spark.streaming.dstream hide focus
ConstantInputDStream
DStream
InputDStream
PairDStreamFunctions
ReceiverInputDStream

Hot Data Analytics

So, D stream classes are different. That is different classes for different languages are being supported scalar, Java and D stream has 36 different values and multiple type of D streams are supported through and separate Python API is there.

(Refer Slide Time: 39:46)

Spark Streaming Operations

- All the Spark RDD operations
  - Some available through the transform() operation

map/flatmap	filter	repartition	union
count	reduce	countByValue	reduceByKey
join	cogroup	transform	updateStateByKey

- Spark Streaming window operations

window	countByWindow	reduceByWindow
reduceByKeyAndWindow	countByValueAndWindow	

- Spark Streaming output operations

print	saveAsTextFiles	saveAsObjectFiles
saveAsHadoopFiles	foreachRDD	

Hot Data Analytics

Now D stream Spark Streaming operations are listed over here. So, all the Spark RDD operations are applicable some available through the transform operation for example, map flatMap, count, join that we have already seen. Spark Streaming window, operations window, count by window, reduced by window all these operations are there Spark output operation sources that print and so on, save as a file.

(Refer Slide Time: 40:10)

Fault-tolerance

- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data
- All transformations are fault-tolerant, and *exactly-once* transformations

tweets RDD
input data replicated in memory
flatMap
hashTags RDD
lost partitions recomputed on other workers

Hot Data Analytics

So, fault tolerance, you can see here in this particular slide the batches of input data are replicated in memory for fault tolerance. So, data loss due to the worker failure can be recomputed from the replicated data. So, all the transformations therefore, are fault tolerant and exactly once transformations.

(Refer Slide Time: 40:29)

Fault-tolerance

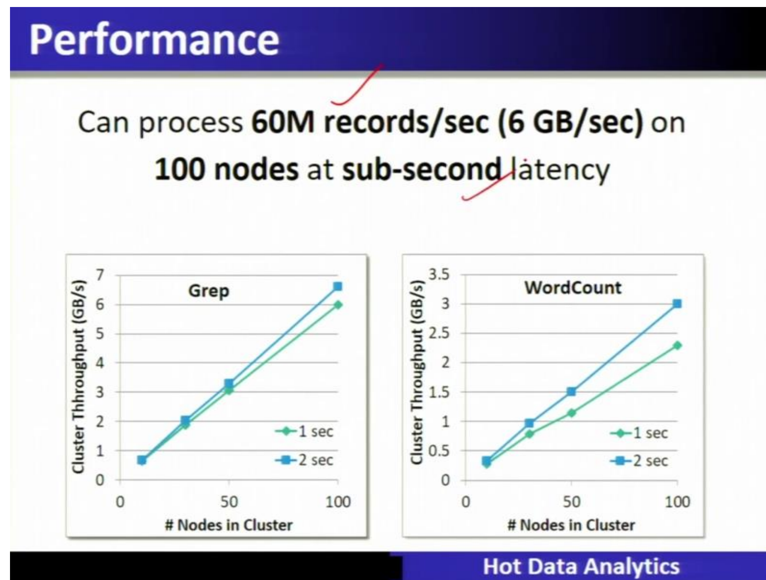
- Received data is replicated among multiple Spark executors
  - Default factor: 2
- Checkpointing
  - Saves state on regular basis, typically every 5-10 batches of data
  - A failure would have to replay the 5-10 previous batched to recreate the appropriate RDDs
  - Checkpoint done to HDFS or equivalent
- Must protect the driver program
  - If the driver node running the Spark Streaming application fails
  - Driver must be restarted on another node.
  - Requires a checkpoint directory in the StreamingContext
- Streaming Backpressure
  - spark.streaming.backpressure.enabled
  - spark.streaming.receiver.maxRate

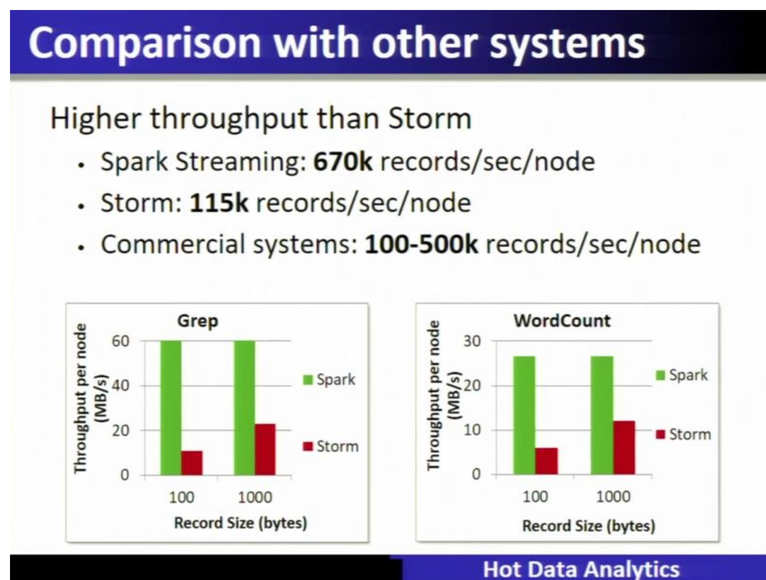
Hot Data Analytics

So, fault tolerance means receive the data, received data is replicated among multiple Spark execution default factor is 2. So, it must product the driver program and if the driver node running the Spark fails, then the driver must be restarted on another node requires the check pointing directory in the Spark context. So, check pointing saves the data, save the state on a regular basis that is typically every 5 to 10 batches of the data, a failure would have to replay

5 to 10 previous batches to recreate. The RDDs and checkpoints is done to HDFS or equivalent streaming back pressure is enabled.

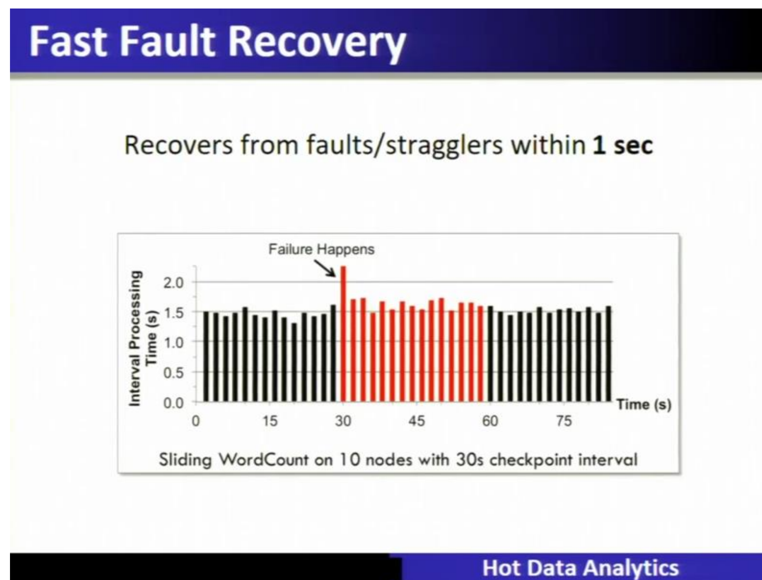
(Refer Slide Time: 41:08)





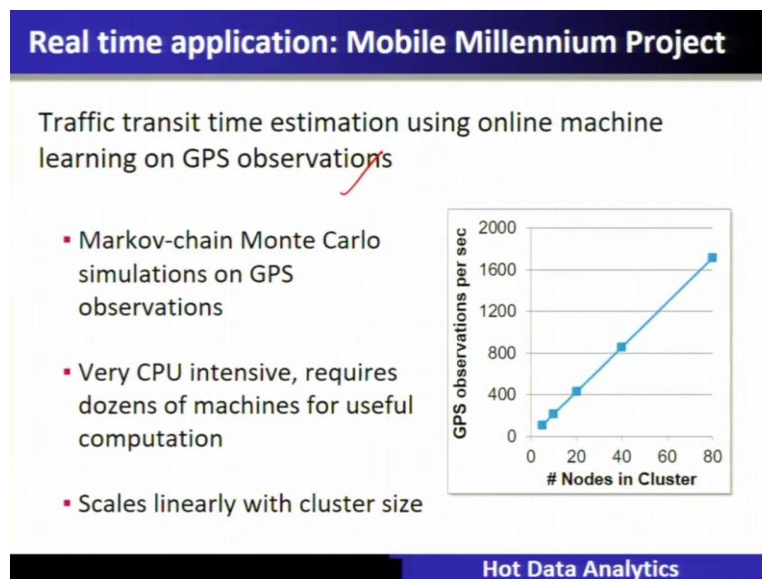
So, performance if you see it can process up to 6 million records per second that is 6 GB per second on 100 nodes at the sub-second latency that is shown here in this particular graph and comparing with the other systems is also shown that is Spark Streaming that is 670 k records per second per node is there compared to the storm is able to do only 115 k records per second per node and commercial systems often requires 100 to 500 k records per second per node.

(Refer Slide Time: 41:43)



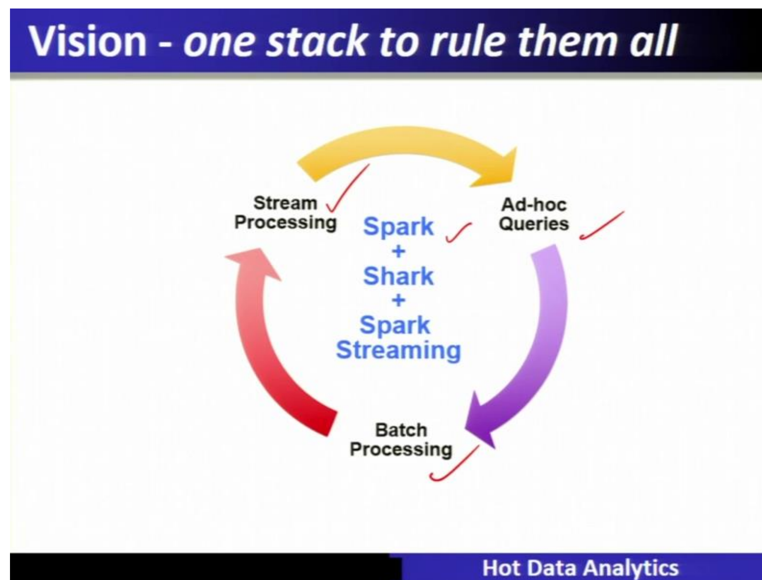
Fast fault recovery recovered from the faults and stragglers within one second that is shown here in these performance metrics.

(Refer Slide Time: 41:50)



So, real time applications, let us say that Mobile Millennium Project, so traffic transit time estimation using online machine learning on a GPS observation. So, you can see here these particular nodes in the cluster, which is also scalable and even very compute intensive CPU intensive also requires the machine and it scales linearly with the cluster size.

(Refer Slide Time: 42:18)



So, vision - one stack to rule them. So, you can see that the Spark Streaming supports also the ad hoc queries and also with the help of integration with a batch processing.

(Refer Slide Time: 42:33)

Spark program vs Spark Streaming program

Spark Streaming program on Twitter stream

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Spark program on Twitter log file

```
val tweets = sc.hadoopFile("hdfs://...")
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFile("hdfs://...")
```

Hot Data Analytics

So, a Spark program versus a Spark Streaming program. So, Spark Streaming on the Twitter data stream is shown over here and it applies the Spark Streaming function like flatMap and on a Twitter log.

(Refer Slide Time: 42:48)

Advantage of an unified stack

- Explore data interactively to identify problems
- Use same code in Spark for processing large logs
- Use similar code in Spark Streaming for realtime processing

```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
...
scala> val filtered = file.filter(_.contains("ERROR"))
...
scala> val mapped = filtered.map(...)
```

```
object ProcessProductionData {
  def main(args: Array[String]) {
    val sc = new SparkContext(...)
    val file = sc.hadoopFile("productionLogs")
    val filtered = file.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

```
object ProcessLiveStream {
  def main(args: Array[String]) {
    val sc = new StreamingContext(...)
    val stream = sc.kafkaStream(...)
    val filtered = stream.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

Hot Data Analytics

So, advantage of unified stack here is to explore the data interactively to identify the problem and same code the Spark for processing the large logs and use the similar code in the Spark Streaming for the real time processing.

(Refer Slide Time: 43:04)

Roadmap

- Spark 0.8.1
  - Marked alpha, but has been quite stable
  - Master fault tolerance – manual recovery
    - Restart computation from a checkpoint file saved to HDFS
- Spark 0.9 in Jan 2014 – out of alpha!
  - Automated master fault recovery
  - Performance optimizations
  - Web UI, and better monitoring capabilities
- Spark v2.4.0 released in November 2, 2018

Hot Data Analytics

So, the roadmap is that we have already covered that Spark 0.8.1 marked alpha and it is being stable and the master fault tolerance is a manual recovery is enabled.

(Refer Slide Time: 43:18)

Sliding Window Analytics

Hot Data Analytics

Spark Streaming Windowing Capabilities

- Parameters
  - Window length: duration of the window
  - Sliding interval: interval at which the window operation is performed
  - Both the parameters must be a multiple of the batch interval
- A window creates a new DStream with a larger batch size

time 1 time 2 time 3 time 4 time 5

original DStream

window-based operation

windowed DStream

window at time 1 window at time 3 window at time 5

Hot Data Analytics

Now, let us see the sliding window analytics. So, Spark Streaming, windowing capabilities are one of the most powerful way of performing the real time analytics or a stream analytics. So, for that, it requires the two parameters, one is the window length that is the duration of the window. So, here you can see that this is the duration of the window and then the sliding interval that is after what interval it has to move to the right over the stream data that window operation. Both the parameter must be in the multiple of batch intervals. So, a window creates a new D stream with a larger batch size.

So, here you can see that the original stream of the data is divided into the micro batches, time 1, time 2, time 3, time 5 and over which you have to now define your window operations windowing capabilities, so it is windowed D stream, you start getting as far as the

operation which you perform at time 3 and then this particular time interval will shift it and you will get the next particular result after this. This is called windowed D streams.

(Refer Slide Time: 44:33)

Spark Window Functions

Spark Window Functions for DataFrames and SQL

Introduced in Spark 1.4, Spark window functions improved the expressiveness of Spark DataFrames and Spark SQL. With window functions, you can easily calculate a moving average or cumulative sum, or reference a value in a previous row of a table. Window functions allow you to do many common calculations with DataFrames, without having to resort to RDD manipulation.

Aggregates, UDFs vs. Window functions

Window functions are complementary to existing DataFrame operations: aggregates, such as sum and avg, and UDFs. To review, aggregates calculate one result, a sum or average, for each group of rows, whereas UDFs calculate one result for each row based on only data in that row. In contrast, window functions calculate one result for each row based on a window of rows. For example, in a moving average, you calculate for each row the average of the rows surrounding the current row; this can be done with window functions.

Hot Data Analytics

So, Spark window functions for data frames and SQL is introduced in Spark 1.4, Spark window operations improved the expressiveness of Spark data frame and Spark SQL. So, with the window functions, you can easily calculate the moving average and cumulative sum or reference value in the previous row of the table. And the window function allows you to do many common calculations with the data frames without having to resort to RDD mapping. So, there are different functions which are supported in this window functions called aggregate UDF versus window functions.

So, window functions are complementary to the existing data frames, Operation aggregates sum as sum and average and UDF. To review the aggregate calculates a result and some or average for each group of the rows, where UDF calculates one result for each row on only the data in that row. So, therefore in contrast window functions calculate one result for each row based on the windows of the row. So, in the moving average you calculate each row the average of the rows surrounding the current row this can be done using the window function.

(Refer Slide Time: 45:44)

Moving Average Example

- Let us dive right into the moving average example. In this example dataset, there are two customers who have spent different amounts of money each day.
- // Building the customer DataFrame. All examples are written in Scala with Spark 1.6.1, but the same can be done in Python or SQL.

```
val customers = sc.parallelize(List(("Alice", "2016-05-01", 50.00),
                          ("Alice", "2016-05-03", 45.00),
                          ("Alice", "2016-05-04", 55.00),
                          ("Bob", "2016-05-01", 25.00),
                          ("Bob", "2016-05-04", 29.00),
                          ("Bob", "2016-05-06", 27.00))).
                     toDF("name", "date", "amountSpent")
```

Hot Data Analytics

So, let us dive into the into the moving average example. In this example, the data set there are two different customers who have spent different amounts of money each day. So, building the customer data frame, all the examples are written in scalar. So, here you can see that the customers one is the Alice and the other one is called Bob. So, these two customers have spent different amounts of money each day. So, you can see that Alice is spending 50, 45, 55. Bob is spending 25, 29, 27. So, the date and the amount is spent is given in the data set.

(Refer Slide Time: 46:22)

Moving Average Example

```
// Import the window functions.
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._

// Create a window spec.
val wSpec1 =
Window.partitionBy("name").orderBy("date").rowsBetween(-1, 1)
```

- In this window spec, the data is partitioned by customer. Each customer's data is ordered by date. And, the window frame is defined as starting from -1 (one row before the current row) and ending at 1 (one row after the current row), for a total of 3 rows in the sliding window.

Hot Data Analytics

Now, as far as moving average example, you have to create the window that is window partition by name and the date and the row between minus 1 and 1. So, in this window the

data is partitioned by the customers and each customer data is ordered by the date. So, window frame is then defined starting from minus 1 that is one row before the current row and ending at 1 that is one row after the current row and for a total of three rows in the sliding window.

(Refer Slide Time: 46:52)

Moving Average Example

```
// Calculate the moving average
customers.withColumn( "movingAvg",
                avg(customers("amountSpent")).over(wSpec1) ).show()
```

This code adds a new column, "movingAvg", by applying the avg function on the sliding window defined in the window spec:

name	date	amountSpent	movingAvg
Alice	5/1/2016	50	47.5
Alice	5/3/2016	45	50
Alice	5/4/2016	55	50
Bob	5/1/2016	25	27
Bob	5/4/2016	29	27
Bob	5/6/2016	27	28

Hot Data Analytics

So, let us see that this particular moving average therefore, will create using the data frame by applying the average functions on the sliding window define in the window a specification. So, just see that three different users so, you can see that sliding window operations is being established here with the moving average of Alice 47.5, then 50, then 50 and 27, 27 and 28.

(Refer Slide Time: 47:23)

Window function and Window Spec definition

- As shown in the above example, there are two parts to applying a window function: (1) specifying the window function, such as avg in the example, and (2) specifying the window spec, or wSpec1 in the example. For (1), you can find a full list of the window functions here:
- https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$
- You can use functions listed under "Aggregate Functions" and "Window Functions".
- For (2) specifying a window spec, there are three components: partition by, order by, and frame.
  1. "Partition by" defines how the data is grouped; in the above example, it was by customer. You have to specify a reasonable grouping because all data within a group will be collected to the same machine. Ideally, the DataFrame has already been partitioned by the desired grouping.
  2. "Order by" defines how rows are ordered within a group; in the above example, it was by date.
  3. "Frame" defines the boundaries of the window with respect to the current row; in the above example, the window ranged between the previous row and the next row.

Hot Data Analytics

So, as shown in the above example, two parts to applying the window functions specify the window functions such as the average in the example and specify the window specifications for in the example you can find out all these details in further examples.

(Refer Slide Time: 47:40)

Cumulative Sum

Next, let us calculate the cumulative sum of the amount spent per customer.

```
// Window spec: the frame ranges from the beginning (Long.MinValue) to
the current row (0).
val wSpec2 =
Window.partitionBy("name").orderBy("date").rowsBetween(Long.MinValue, 0)
// Create a new column which calculates the sum over the defined window
frame.
customers.withColumn( "cumSum",
 sum(customers("amountSpent")).over(wSpec2) ).show()
```

name	date	amountSpent	cumSum
Alice	5/1/2016	50	50
Alice	5/3/2016	45	95
Alice	5/4/2016	55	150
Bob	5/1/2016	25	25
Bob	5/4/2016	29	54
Bob	5/6/2016	27	81

Hot Data Analytics

So, cumulative sum is another statistics which you can or another function you can perform in the sliding window over the same data set. So, cumulative, you can see that it will take data 50. And the next data when it comes it will do the cumulative sum and then finally, when 55 is added then it will add and for the next user when 25 is the amount which is spent with this only and when the next chunk it will be cumulative addition is done and finally, all that details are given over here.

(Refer Slide Time: 48:17)

Data from previous row

In the next example, we want to see the amount spent by the customer in their previous visit.

```
// Window spec. No need to specify a frame in this case.
val wSpec3 = Window.partitionBy("name").orderBy("date")

// Use the lag function to look backwards by one row.
customers.withColumn("prevAmountSpent",
 lag(customers("amountSpent"), 1).over(wSpec3) ).show()
```

name	date	amountSpent	prevAmountSpent
Alice	5/1/2016	50	null
Alice	5/3/2016	45	50
Alice	5/4/2016	55	45
Bob	5/1/2016	25	null
Bob	5/4/2016	29	25
Bob	5/6/2016	27	29

Hot Data Analytics

Rank

- In this example, we want to know the order of a customer's visit (whether this is their first, second, or third visit).

```
// The rank function returns what we want.
customers.withColumn( "rank", rank().over(wSpec3) ).show()
```

name	date	amountSpent	rank
Alice	5/1/2016	50	1
Alice	5/3/2016	45	2
Alice	5/4/2016	55	3
Bob	5/1/2016	25	1
Bob	5/4/2016	29	2
Bob	5/6/2016	27	3

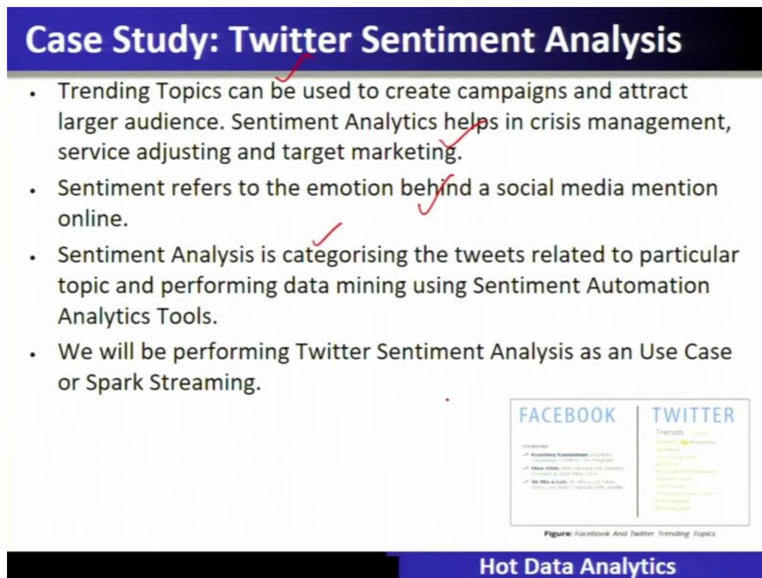
Hot Data Analytics

So, in the next example, you see that the amount is spent by the customer in their previous visit is automatically previous amount spent. So, with the 50 previous is not there then it will be null and for every data you see that previous values are given. So, this is our these are some of the operations which are there and then you can also do a ranking in this example, we want to show that the order of customers visit whether it is first second or third the rank function returns what you want over the window specifications. So, you rank this particular way in which the data is coming.

(Refer Slide Time: 48:57)

Case Study: Twitter Sentiment Analysis with Spark Streaming

Hot Data Analytics



So, let us see about the Twitter sentiment analysis using Spark Streaming. So, Twitter sentiment analysis is to do a trending topic can be used to create the campaign and attract the larger audience. So, sentiment analytics, helps in crisis management, and so on. Sentiments refers to the emotion behind the social media mentioned online and semantic sentiment analysis is categorizing the tweets related to a particular topic performing data mining and so on.

(Refer Slide Time: 49:30)

Problem Statement

- To design a Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing.

Sentiment Analysis is used to:

- Predict the success of a movie
- Predict political campaign success
- Decide whether to invest in a certain company
- Targeted advertising
- Review products and services

Hot Data Analytics

So, it can be done with this particular so, to design the Twitter sentiment analysis where you populate the real time sentiments for the crisis management and service adjustment and for the target marketing. So, Twitter so, sentiment analysis is used to predict the success of a movie predict the political campaign success and so on.

(Refer Slide Time: 49:52)

Importing Packages

```
//Import the necessary packages into the Spark Program
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext._
import org.apache.spark.streaming.twitter._
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark._
import org.apache.spark.rdd._
import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext._
import org.apache.spark.sql
import org.apache.spark.storage.StorageLevel
import scala.io.Source
import scala.collection.mutable.HashMap
import java.io.File
```

Hot Data Analytics

Twitter Token Authorization

```
object mapr {

 def main(args: Array[String]) {
 if (args.length < 4) {
 System.err.println("Usage: TwitterPopularTags <consumer key>
<consumer secret> " +
 "<access token> <access token secret> [<filters>]")
 System.exit(1)
 }

 StreamingExamples.setStreamingLogLevels()
 //Passing our Twitter keys and tokens as arguments for authorization
 val Array(consumerKey, consumerSecret, accessToken,
accessTokenSecret) = args.take(4)
 val filters = args.takeRight(args.length - 4)
```

Hot Data Analytics

DStream Transformation

```
// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret",
accessTokenSecret)

val sparkConf = new
SparkConf().setAppName("Sentiments").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(5))
val stream = TwitterUtils.createStream(ssc, None, filters)

//Input DStream transformation using flatMap
val tags = stream.flatMap { status =>
status.getHashtagEntities.map(_.getText)}
```

Hot Data Analytics

So, to do this, let us build the environment and then Twitter token authorization and then D stream transformation is being carried out.

(Refer Slide Time: 50:02)

Results

Positive

Neutral

Negative

Hot Data Analytics

Sentiment for Trump

'Trump' Keyword

Positive

Neutral

Negative

Hot Data Analytics

And this is the result of this particular sentiment for the product Trump which is shown over here.

(Refer Slide Time: 50:10)

Applying Sentiment Analysis

- As we have seen from our Sentiment Analysis demonstration, we can extract sentiments of particular topics just like we did for 'Trump'. Similarly, Sentiment Analytics can be used in crisis management, service adjusting and target marketing by companies around the world.

- Companies using Spark Streaming for Sentiment Analysis have applied the same approach to achieve the following:
1. Enhancing the customer experience
2. Gaining competitive advantage
3. Gaining Business Intelligence
4. Revitalizing a losing brand

Hot Data Analytics

And applying the sentiment analysis, as we have seen from the sentiment analysis demonstration that we can extract the sentiments of a particular topic. For example, it is a Trump similarly the sentiment analytics can be used for the crisis management and so on. So, companies using the Spark Streaming for sentiment analysis have applied the same approach to the following enhancing the customer experience, gaining the competitive advantage, gaining the business intelligent, revitalizing a losing band.

(Refer Slide Time: 50:41)

Conclusion

- Stream processing framework that is ...
  - Scalable to large clusters
  - Achieves second-scale latencies
  - Has simple programming model
  - Integrates with batch & interactive workloads
  - Ensures efficient fault-tolerance in stateful computations

Hot Data Analytics

So, let us conclude this particular lecture here. So, we have seen the stream processing framework, which is for hot data analytics, hot data path analytics, which is also called as a real time analytics or the streaming analytics. It is also called the fast data processing. So,

stream processing frameworks, like Spark Streaming, we have seen an important way of doing these streaming analytics that is called windowing-based methods, we have seen, what we have seen is that this particular method to do this stream processing within a particular time and need to be scaled.

Therefore, there is a proper framework which is required to do this kind of stream processing which is scalable to a large number of cluster. We have also seen that the methods that is in the form of scale, we have seen that it can achieve seconds scale latencies and also has a simple programming model such as a scala and Java we have seen. Also, we have seen how to integrate with the batch and the interactive workloads for better insight when doing this particular real time analytics. It also ensures efficient fault tolerance in a stateful computation. Thank you.