Big Data
(Spark)

Instructor: Thanh Binh Nguyen

September 1st, 2019

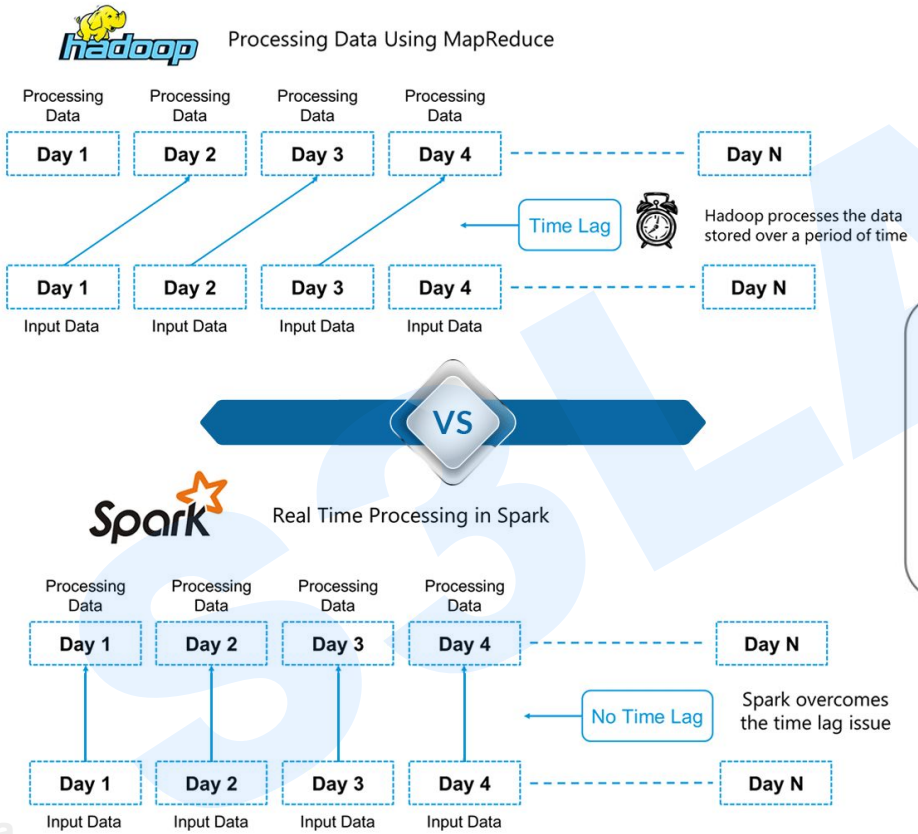
S3Lab
Smart Software System Laboratory

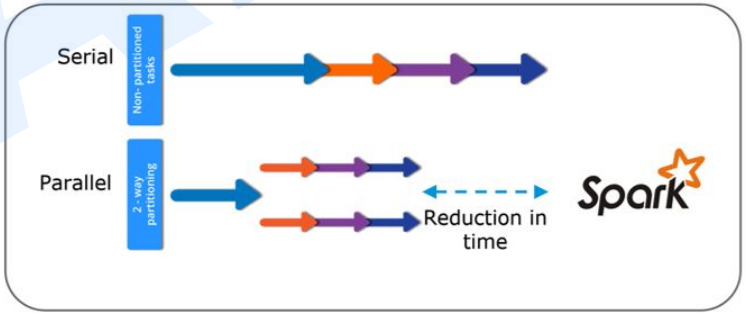

"Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming."
– Chris Lynch, Vertica Systems

Introduction

- Apache Spark is an open-source cluster computing framework for **real-time processing**
- Spark provides an interface for programming entire clusters with implicit data **parallelism** and **fault-tolerance**.
- Spark is designed to cover a wide range of workloads such as **batch applications**, **iterative algorithms**, **interactive queries** and **streaming**
- Spark is not a modified version of Hadoop.
- Since Spark has its own cluster management computation, it uses Hadoop for **storage** purpose only.

Introduction





Applications

Banking

Government

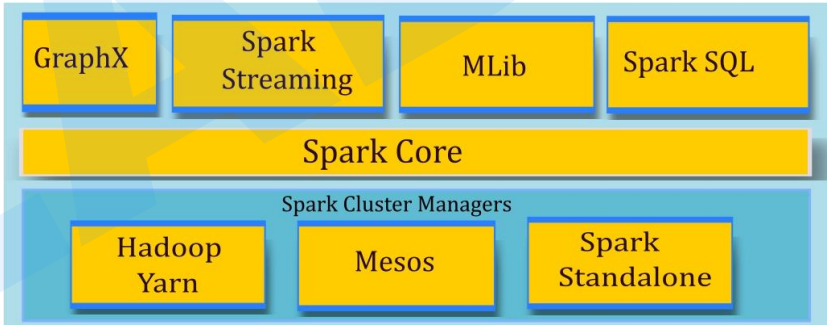
Healthcare

Telecommunications

Stock Market

Components

- **Spark Core** and Resilient Distributed Datasets (**RDDs**)
- **Spark SQL**
- **Spark Streaming**
- Machine Learning Library or **MLlib**
- **GraphX**



Components


Spark SQL
Spark streaming
Spark MLlib
Spark GraphX
Spark R
Spark
Scala
R

Components - Spark Core

- The base engine for **large-scale parallel** and **distributed data processing**. The core is the **distributed execution engine** and the **Java**, **Scala**, and **Python APIs** offer a platform for **distributed ETL** application development. Further, additional libraries which are built atop the core allow diverse workloads for **Streaming**, **SQL**, and **Machine Learning**. It is responsible for:
  - **Memory management** and **fault recovery**
  - **Scheduling**, **distributing** and monitoring jobs on a cluster
  - **Interacting** with **storage** systems

Components - RDD

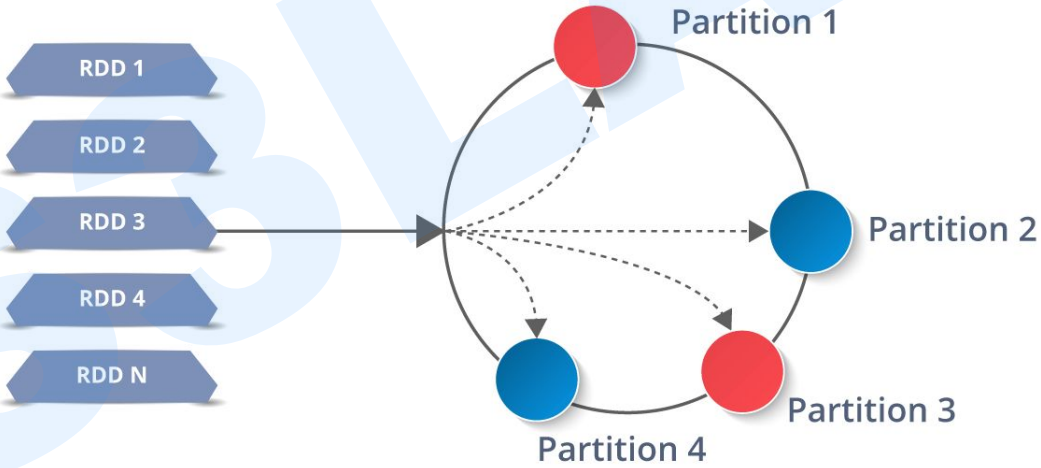
Definition

- **R**esilient **D**istributed **D**ataset
  - the fundamental data structure abstraction of Spark
  - a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. Ex. you can create an RDD of integers and these gets partitioned and divided and assigned to various nodes in the cluster for parallel processing.
- RDDs can contain any type of **Python**, **Java**, or **Scala** objects, including **user-defined** classes.

Components - RDD

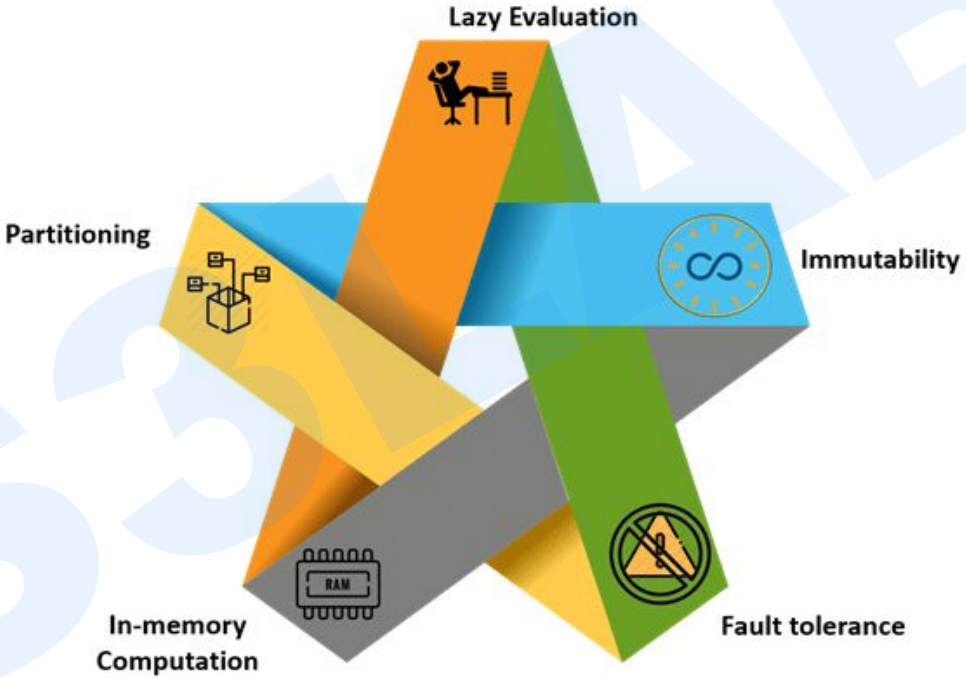
Definition

- **Dataset** - it can represent any form of data be it reading from a csv file, loading data from a table using rdbms, text file, json , xml.



Components - RDD

Features



Components - RDD

Features

- **Resilient ~ fault tolerant**, detect and recompute missing or damaged partitions of an RDD due to node or network failures.
- **Distributed**, Data present in an RDD resides on multiple nodes. It is distributed across different nodes of a cluster.
- **In-memory computation** - Mostly they are in memory so that iterative operations runs faster and performs way better than traditional hadoop programs in executing iterative algorithms

Components - RDD

Features

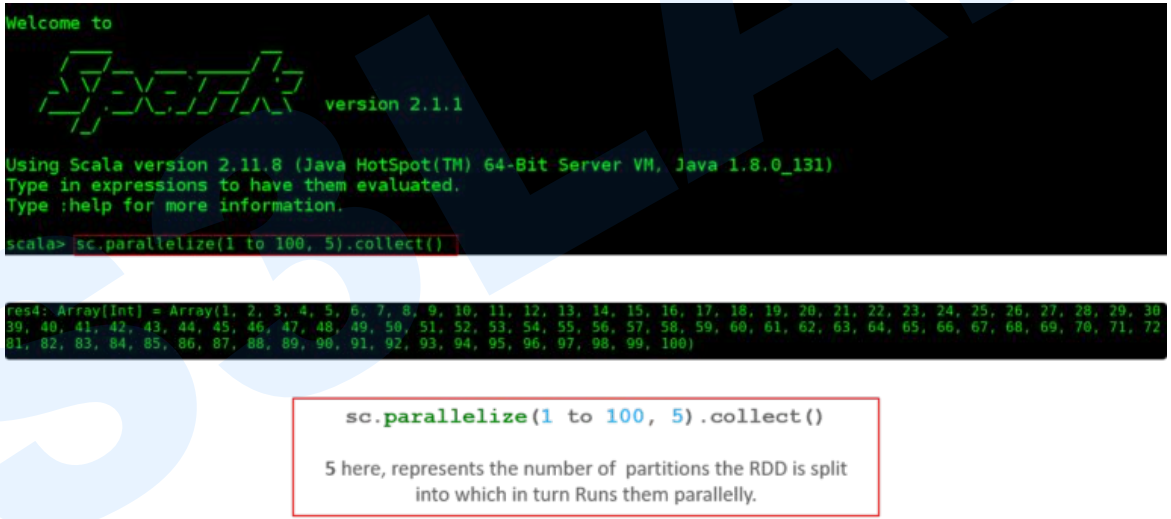
- **Immutability**: Data stored in an RDD is in the **read-only** mode.
- **Lazy evaluation**: Data does not get loaded in an RDD even if you define it. Transformations are actually computed when you call action, such as count or collect, or save the output to a file system.
- **Partitioning**: Partitions can be done on any existing RDD to create logical parts that are mutable. You can achieve this by applying transformations to the existing partitions.

Components - RDD

Features

- Partition - is a logical chunk of a large distributed data set. By default.

  Spark tries to read data into an RDD from the nodes that are close to it.



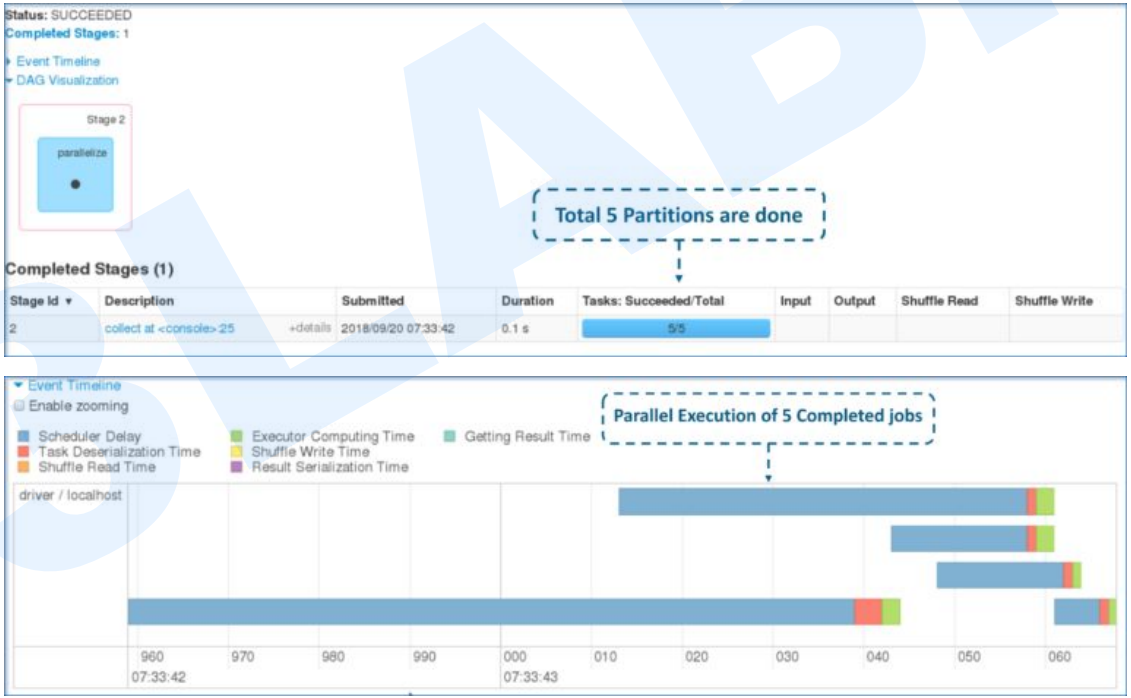

```
sc.parallelize(1 to 100, 5).collect()
```

5 here, represents the number of partitions the RDD is split into which in turn Runs them parallelly.

Components - RDD

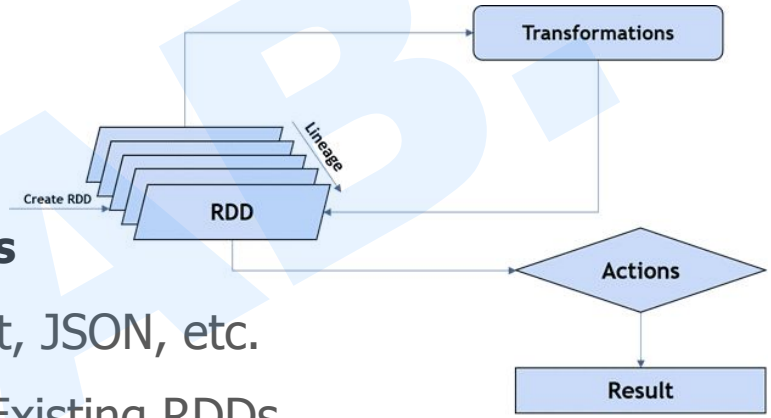
Features

Partition and Parallelism



Components - RDD

Workflow



- 3 ways to create RDDs
  - Parallelizing the Collection of Objects
  - Loading an External Dataset: csv, txt, JSON, etc.
  - Performing Transformations on the Existing RDDs
- 2 types of operations
  - **Transformation**: to create new RDD.
  - **Actions**: applied on an RDD to instruct Spark to apply computation and pass the result back to the driver.

Components - RDD

Workflow

Loading an External Dataset

```
scala> val fileData = sc.textFile("file:///home/training/NoMay.txt")
fileData: org.apache.spark.rdd.RDD[String] = file:///home/training/NoMay.txt Map
PartitionsRDD[1] at textFile at <console>:24

scala> fileData.foreach(println)
January
February
March
April
There is no May
```

Components - RDD

Workflow

Parallelizing

```
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_181)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val rdd1 = spark.sparkContext.parallelize(Array("jan","feb","mar","april"
,"may","jun"),3)
rdd1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize
 at <console>:23

scala> val result = rdd1.coalesce(2)
result: org.apache.spark.rdd.RDD[String] = CoalescedRDD[1] at coalesce at <conso
le>:25

scala> result.foreach(println)
mar
april
may
jun
jan
feb

scala>
```

Components - RDD

Workflow

Transformations

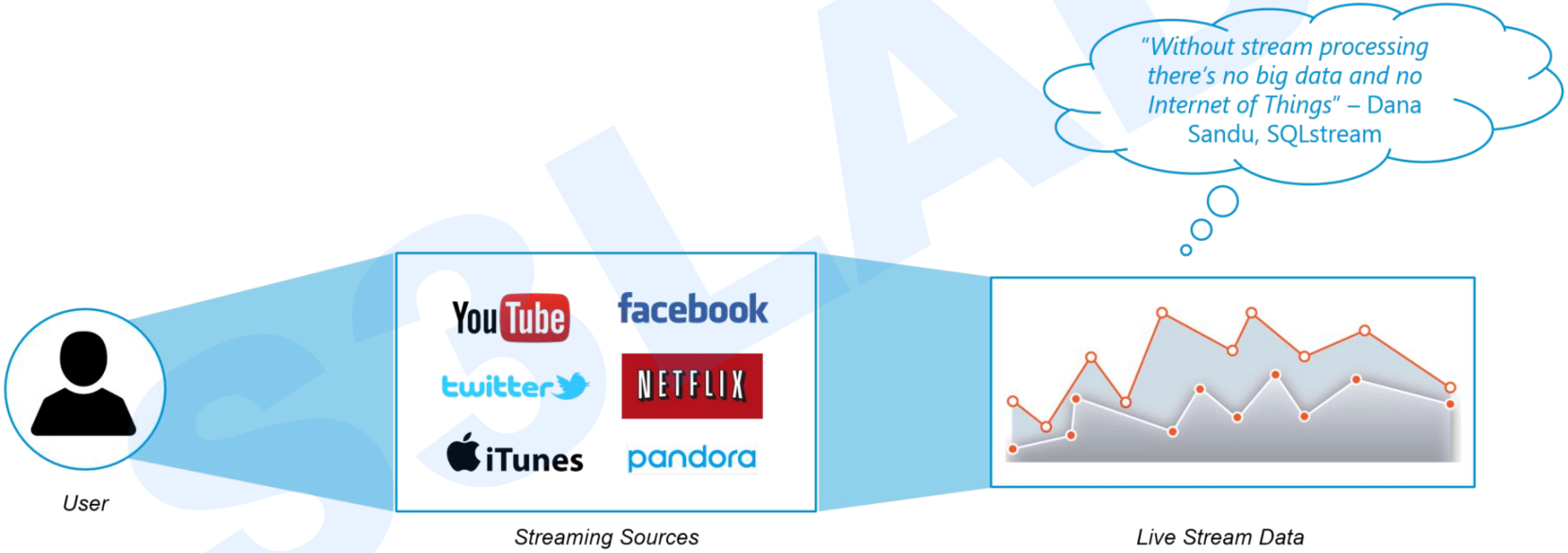
```
scala> val words=spark.sparkContext.parallelize(Seq("the","quick","brown","fox",
"jumps","over","the","lazy","dog"))
words: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[4] at paralleliz
e at <console>:23

scala> val wordPair = words.map(w => (w.charAt(0), w))
wordPair: org.apache.spark.rdd.RDD[(Char, String)] = MapPartitionsRDD[5] at map
at <console>:25

scala> wordPair.foreach(println)
(j,jumps)
(o,over)
(t,the)
(l,lazy)
(d,dog)
(t,the)
(q,quick)
(b,brown)
(f,fox)

scala> words.foreach(println)
the
quick
brown
fox
jumps
over
the
lazy
dog
```

Components - RDD

Workflow

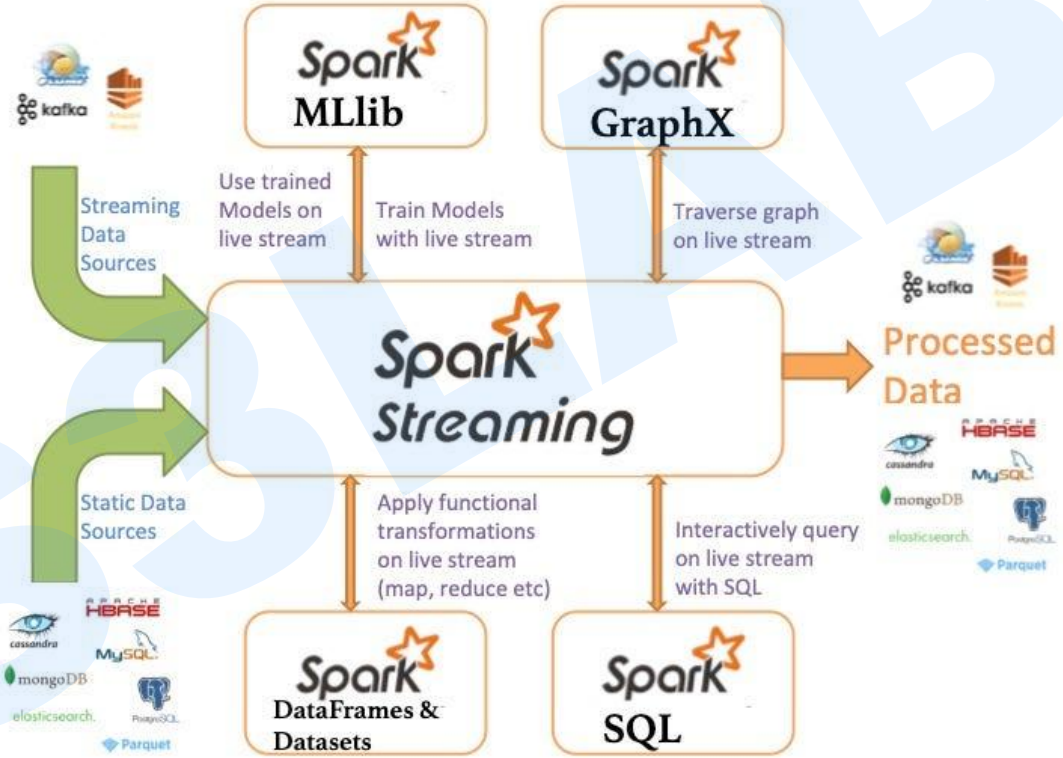
Components - Spark Streaming

Overview



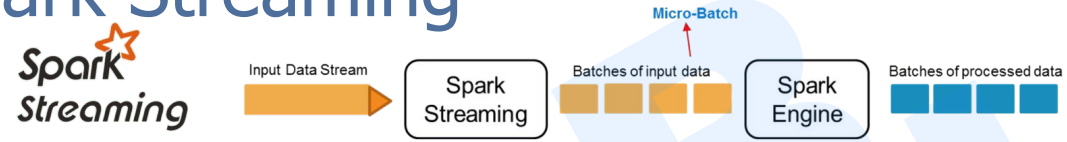
Components - Spark Streaming

Overview



Components - Spark Streaming

Overview



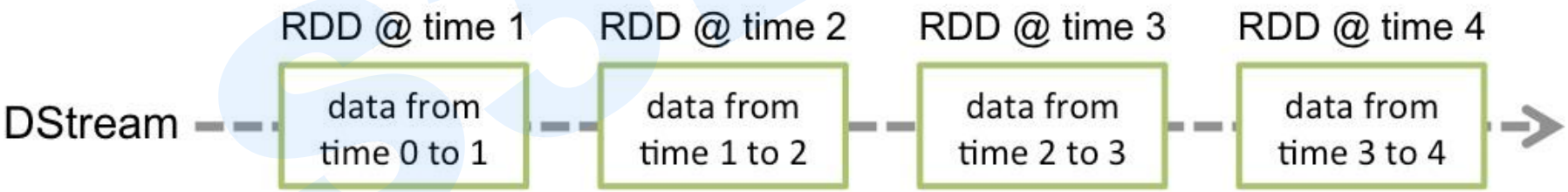
- Used to process real-time streaming data.
- It enables high-throughput and fault-tolerant stream processing of live data streams. The fundamental stream unit is **DStream** which is basically **a series** of **RDDs** (Resilient Distributed Datasets) to process the real-time data.

Spark Streaming is used to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

Components - Spark Streaming

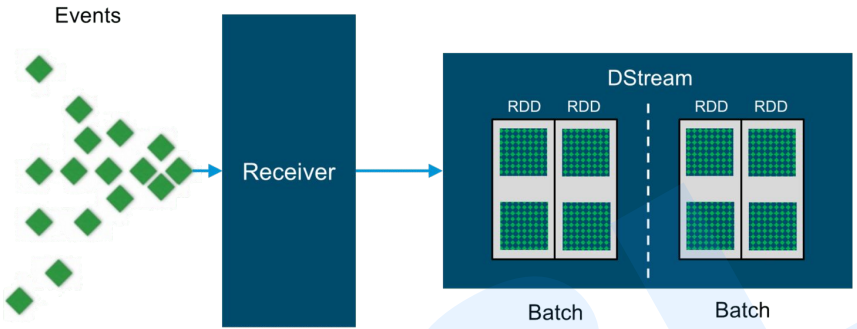
Fundamentals

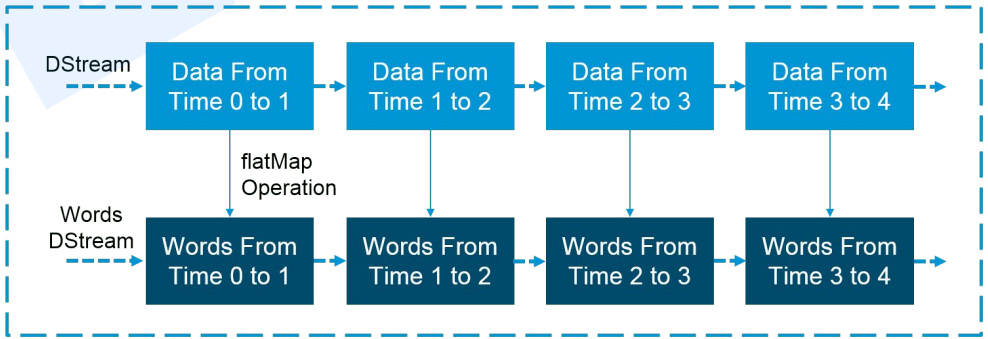
- Streaming Context
- DStream (Discretized Stream)
- Caching
- Accumulators, Broadcast Variables and Checkpoints



Components - Spark Streaming

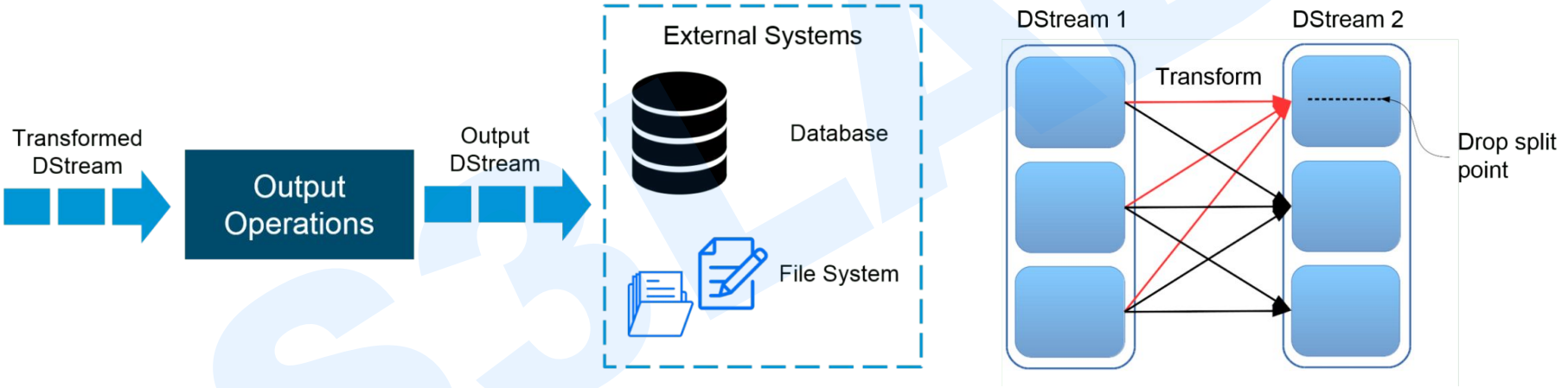
Fundamentals





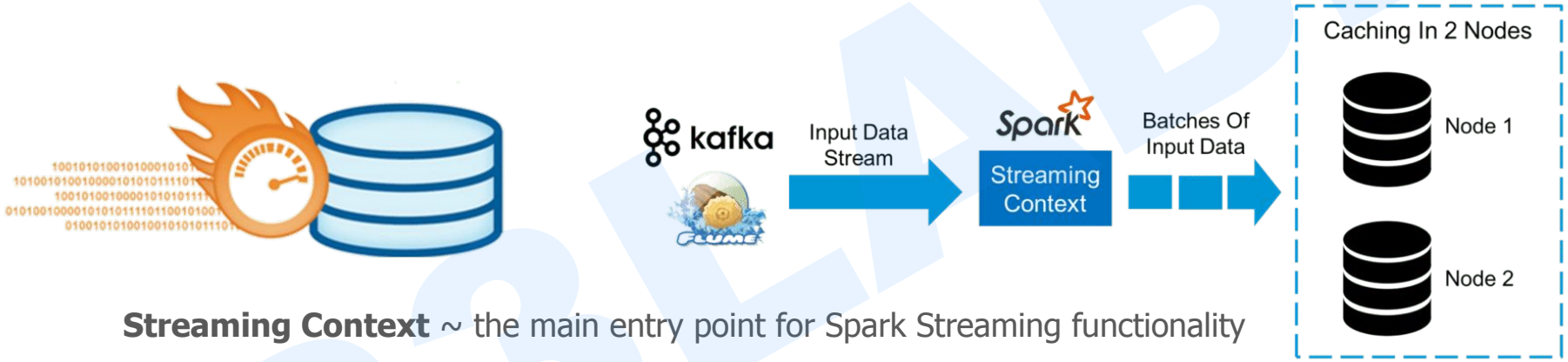
Components - Spark Streaming

Fundamentals



Components - Spark Streaming

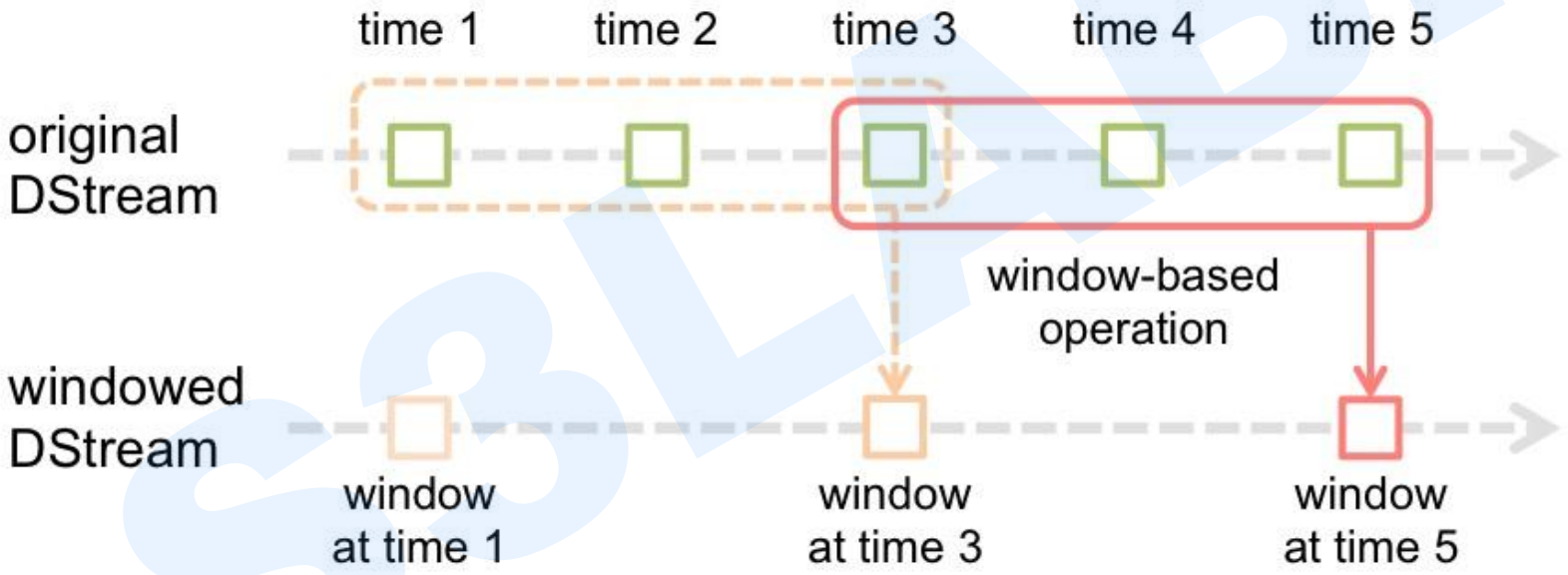
Fundamentals



Streaming Context ~ the main entry point for Spark Streaming functionality

Components - Spark Streaming

Fundamentals



window length & sliding interval

Components - Spark SQL

- Spark SQL integrates relational processing with Spark functional programming. Provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool. The following are the four libraries of Spark SQL.
  - Data Source API
  - DataFrame API
  - Interpreter & Optimizer
  - SQL Service

Components - Spark SQL

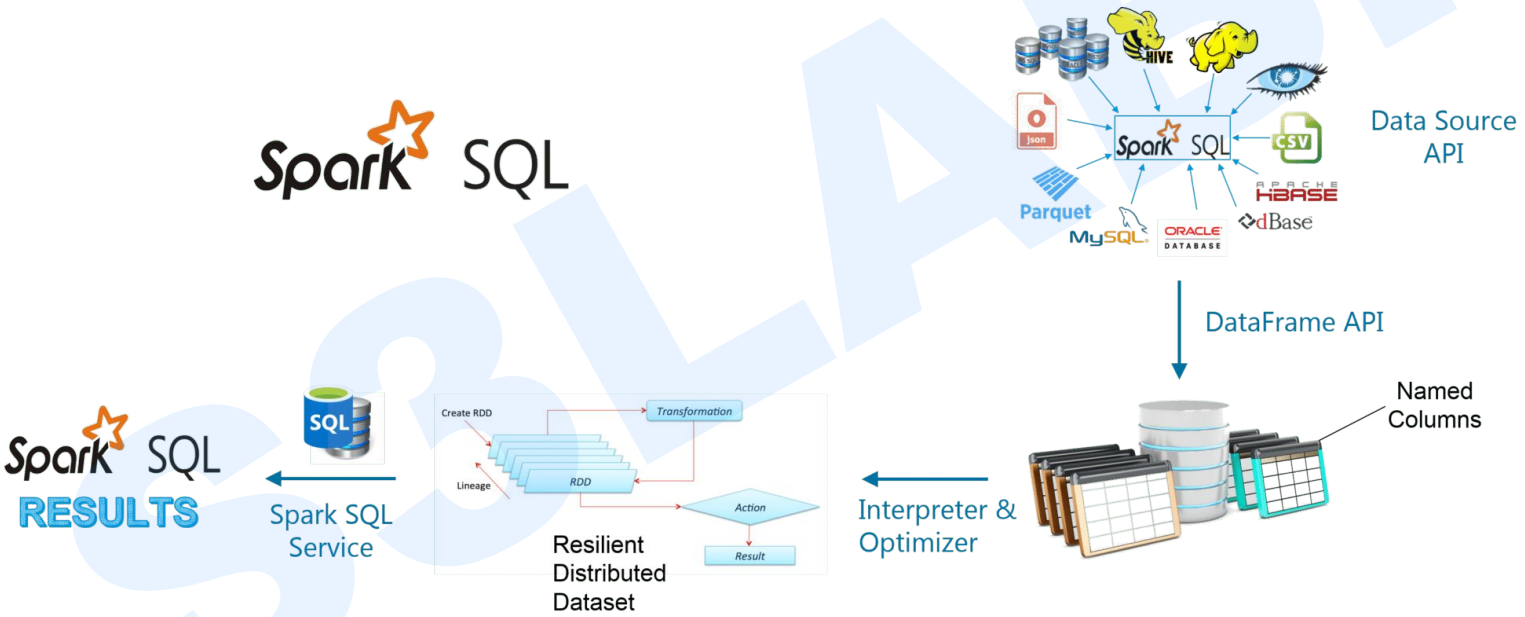


Figure: *The flow diagram represents a Spark SQL process using all the four libraries in sequence*

Components - Spark SQL

Data Source API

- Universal API for **loading** and **storing structured data**.
  - Built in support for Hive, JSON, Avro, JDBC, Parquet, ect.
  - Support third party integration through spark packages
  - Support for smart sources
  - Data Abstraction and Domain Specific Language (DSL) applicable on structure and semi-structured data
  - Supports different data formats (Avro, CSV, Elastic Search and Cassandra) and storage systems (HDFS, HIVE Tables, MySQL, etc.)
  - Can be easily integrated with all Big Data tools and frameworks via Spark-Core.
  - It processes the data in the size of Kilobytes to Petabytes on a single-node cluster to multi-node clusters.

Components - Spark SQL

DataFrame API

- A Data Frame is a distributed collection of data organized into named column. It is equivalent to a relational table in SQL used for storing data into tables.

Components - Spark SQL

SQL Interpreter And Optimizer

- based on functional programming constructed in Scala.
  - Provides a general framework for transforming trees, which is used to perform analysis/evaluation, optimization, planning, and run time code spawning.
  - This supports cost based optimization (run time and resource utilization is termed as cost) and rule based optimization, making queries run much faster than their RDD (Resilient Distributed Dataset) counterparts.

Components - Spark SQL

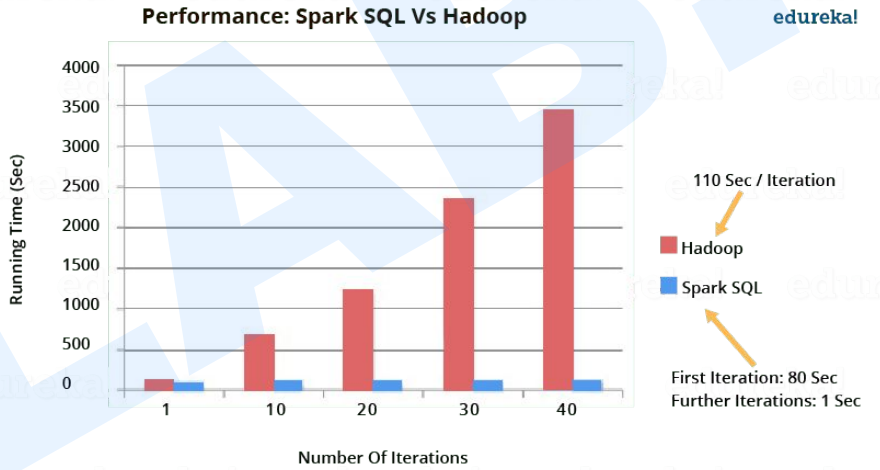
SQL Service

- SQL Service is the entry point for working along structured data in Spark. It allows the creation of DataFrame objects as well as the execution of SQL queries.

Components - Spark SQL

Features

- Integration With Spark
- Uniform Data Access
- Hive Compatibility
- Standard Connectivity
- Performance And Scalability
- User Defined Functions



Components - GraphX

- GraphX is the Spark API for graphs and graph-parallel computation.
  Thus, it extends the Spark RDD with a Resilient Distributed Property Graph.
  The property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. Here, the parallel edges allow multiple relationships between the same vertices.

Components - GraphX

- GraphX exposes a set of **fundamental operators** (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API.
- In addition, GraphX includes a growing collection of **graph algorithms** and builders to simplify **graph analytics tasks**.
- GraphX unifies **ETL** (Extract, Transform & Load) process, **exploratory analysis** and **iterative graph computation** within a single system.

Components - GraphX

Use cases

- Disaster Detection System

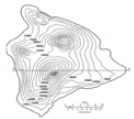
- Page Rank

- Financial Fraud Detection

- Business Analysis

UBER

  - Machine Learning, understanding the customer purchase trends
- Geographic Information Systems

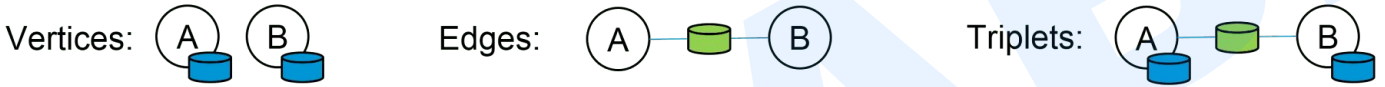


  - Watershed delineation and weather prediction
- Google Pregel

Google

Components - GraphX

Graph and Examples



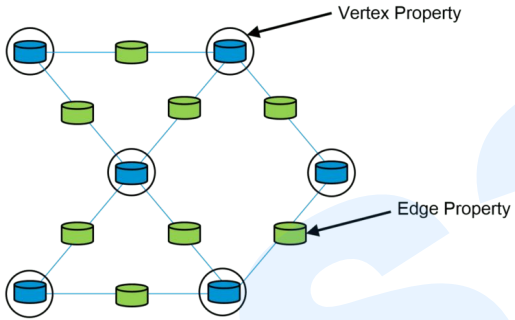


Figure: *Property Graph*

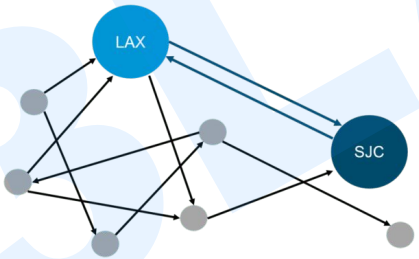
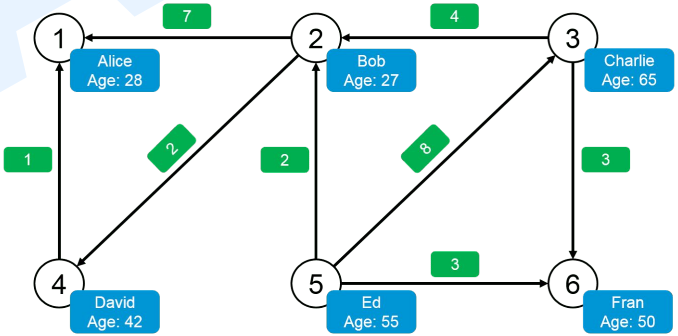


Figure: *An example of property graph*



Components - MLLib

- stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.

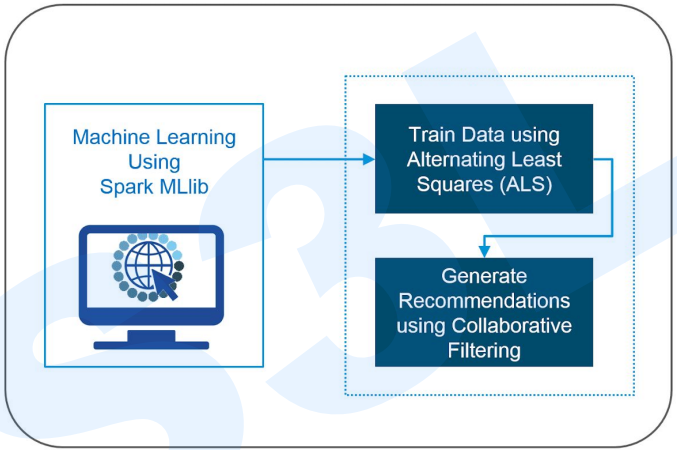


Figure: *Machine Learning Flow Diagram*

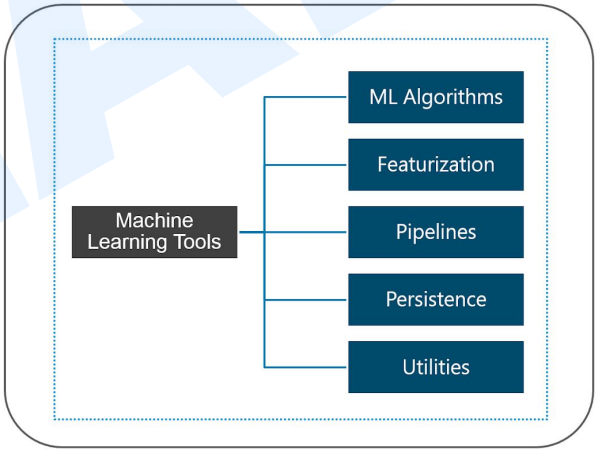
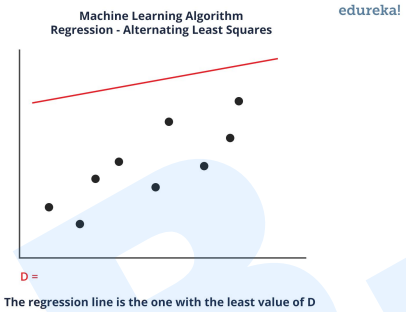


Figure: *Machine Learning Tools*

Components - MLLib



Algorithms

- **Basic Statistics**: Summary, Correlation, Stratified Sampling, Hypothesis Testing, Random Data Generation.
- Regression
- Classification
- Recommendation System: Collaborative, Content-Based
- Clustering
- Dimensionality Reduction: Feature Selection, Feature Extraction
- Feature Extraction
- Optimization

Challenges of Distributed computing

- How to divide the input data
- How to assign the divided data to machines in the cluster
- How to check and monitor a machine in the cluster is live and has resources to perform its duty
- How to retry or reassign failed chunks to another machine or worker

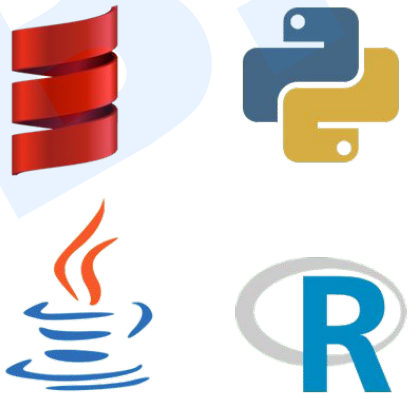
Challenges of Distributed computing

- If the computation involves any aggregation operation like a sum, how to collate results from many workers and compute the aggregation
- Efficient use of memory , cpu and network
- Monitoring the tasks
- Overall job coordination
- Keeping a global time

Spark's Features

Spark's Features

- **Multiple languages support** namely Java, R, Scala, Python for building applications. Spark provides **high-level APIs (>80 operators)** in **Java**, **Scala**, **Python** and **R**. Spark code can be written in any of these four languages. It provides a **shell** in **Scala** and **Python**. The Scala shell can be accessed through *./bin/spark-shell* and Python shell through *./bin/pyspark* from the installed directory.



Spark's Features

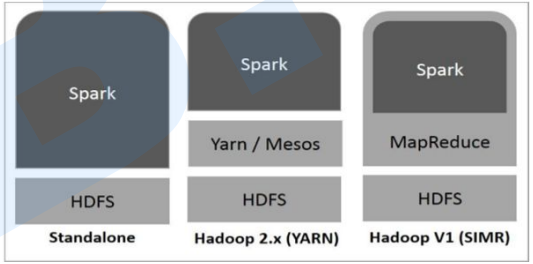
- **Fast Speed** in Data Processing, 10 times faster on Disk and 100 times swifter in Memory (compare to Hadoop). Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.
- Spark with abstraction-RDD provides **Fault Tolerance** with ensured Zero Data loss
- **In-Memory Processing** resulting in high computation speed and acyclic data-flow

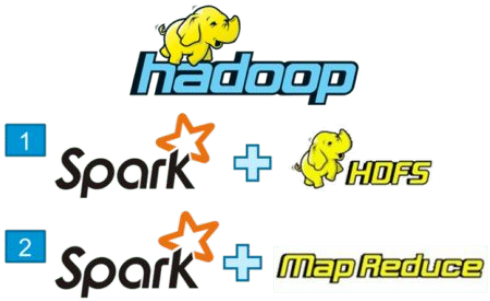
Spark's Features

- Increase in system efficiency due to **Lazy Evaluation** of transformation in RDD: Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For **transformations**, Spark adds them to a **DAG** (**Directed Acyclic Graph**) of computation and only when the driver requests some data, does this DAG actually gets executed.

Spark's Features

- Flexible to run **Independently** and can be **integrated with Hadoop** Yarn Cluster Manager. Apache Spark provides smooth compatibility with Hadoop. This is a boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.



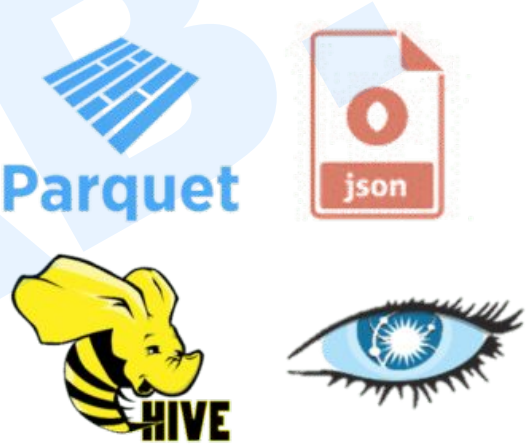


Spark's Features

- **Cost Efficient** for Big data as minimal need of storage and data center
- Futuristic analysis with **built-in tools** for machine learning (MLLib), interactive queries & data streaming
- **Persistence** and **Immutable** in nature with data paralleling processing over the cluster
- Graphx simplifies Graph Analytics by collecting algorithm and builders
- **Re-using Code** for batch processing and to run ad-hoc queries
- Progressive and expanding Apache community active for **Quick Assistance**

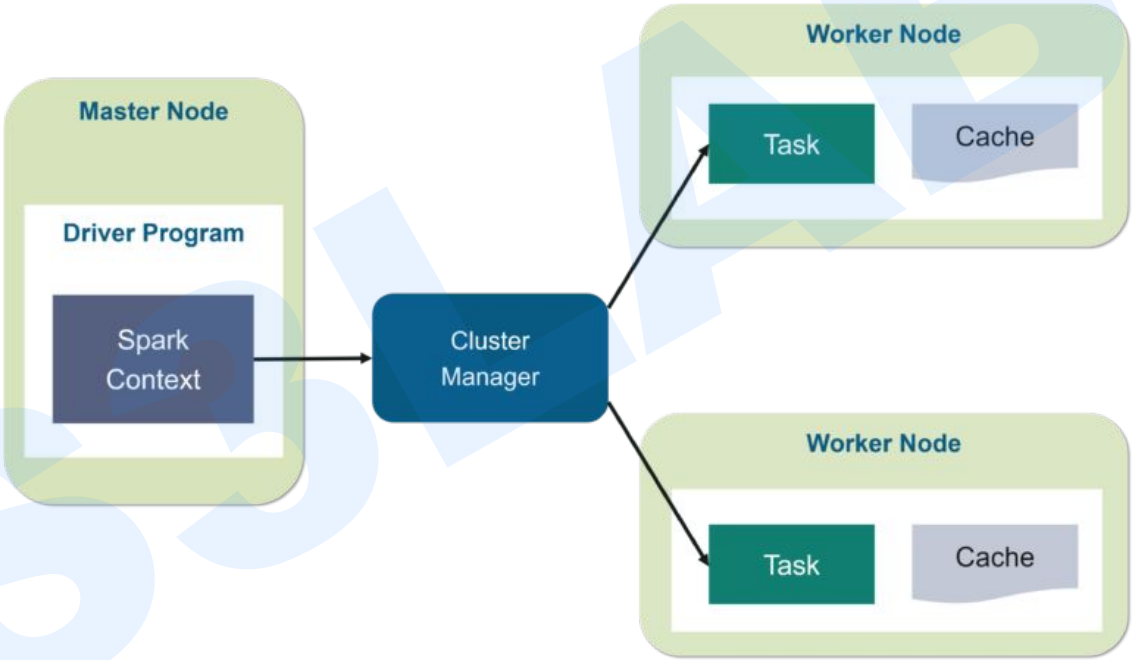
Spark's Features

- Spark supports **multiple data sources** such as **Parquet**, **JSON**, **Hive** and **Cassandra** apart from the usual formats such as **text files**, **CSV** and RDBMS tables. The Data Source API provides a pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.



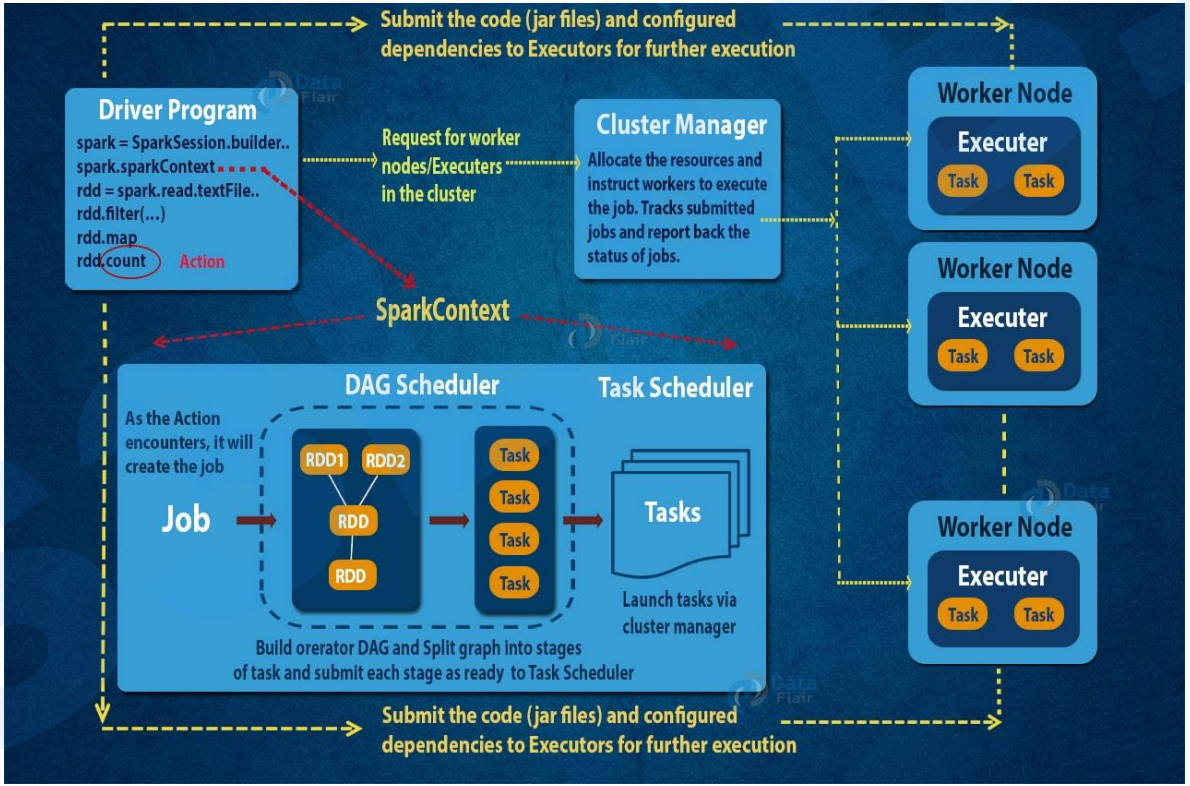
Spark's Program

Architecture



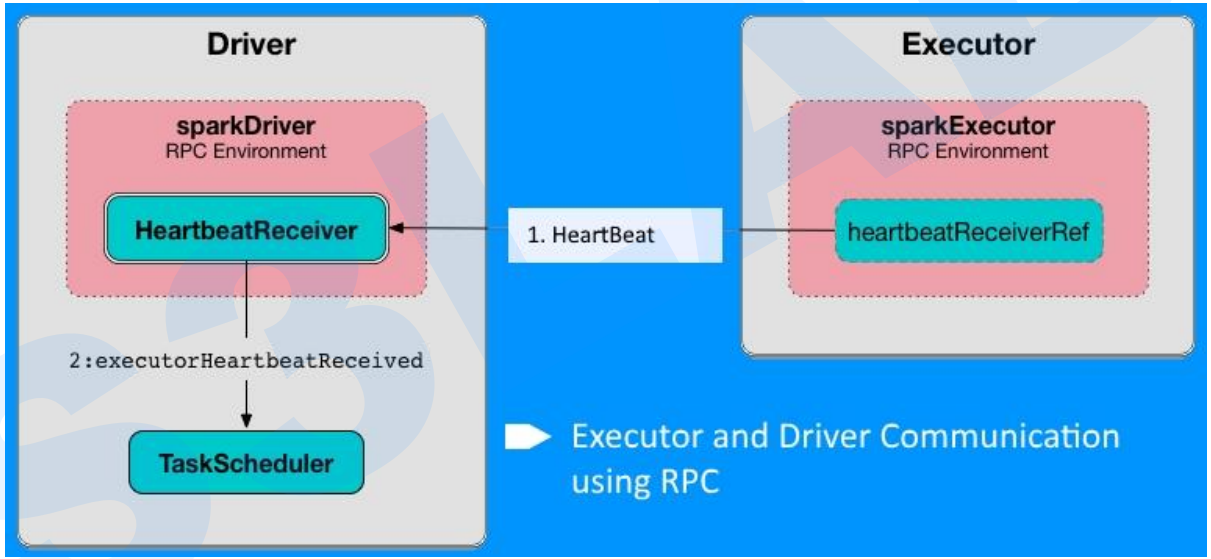
Spark's Program

Architecture



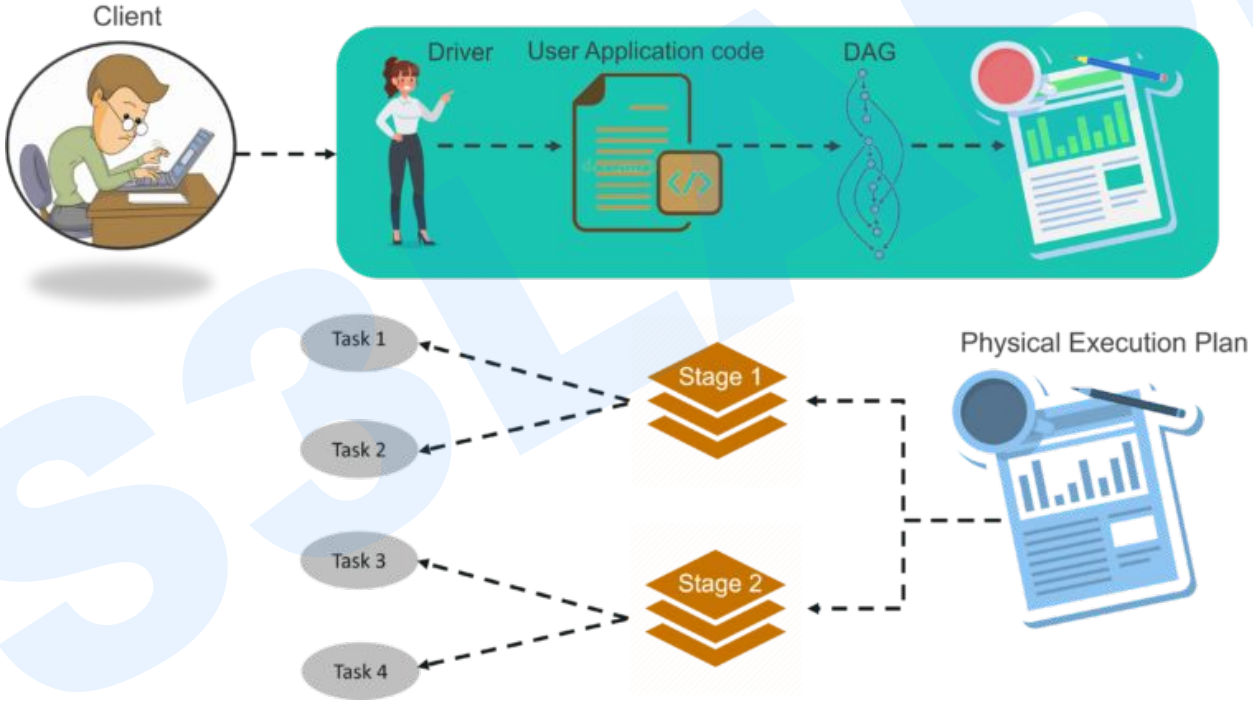
Spark's Program

Architecture



Spark's Program

Architecture

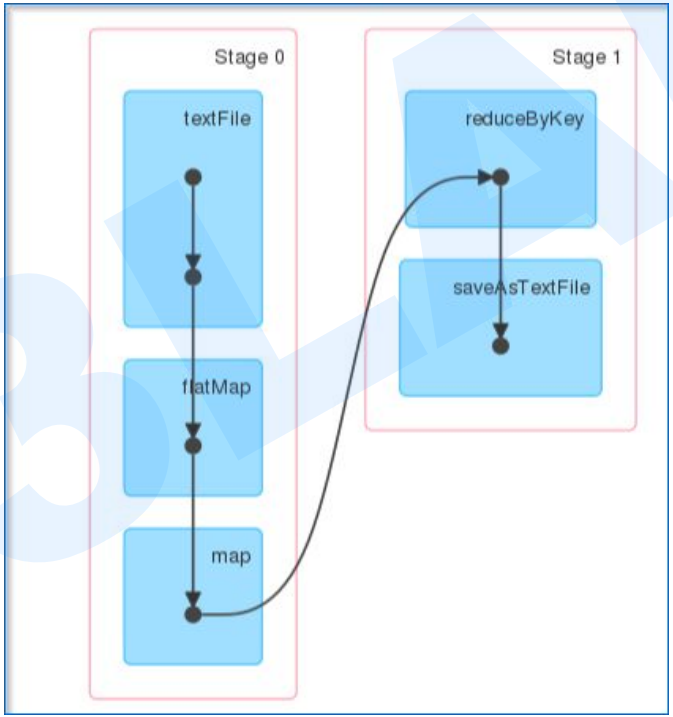


Spark's Program

Architecture

Spark's Program

Directed Acyclic Graph (DAG) Visualization



Spark Program

Word count demo

- Web UI port for Spark is **4040**.
- HDFS web browser port is **50040**

Spark Program

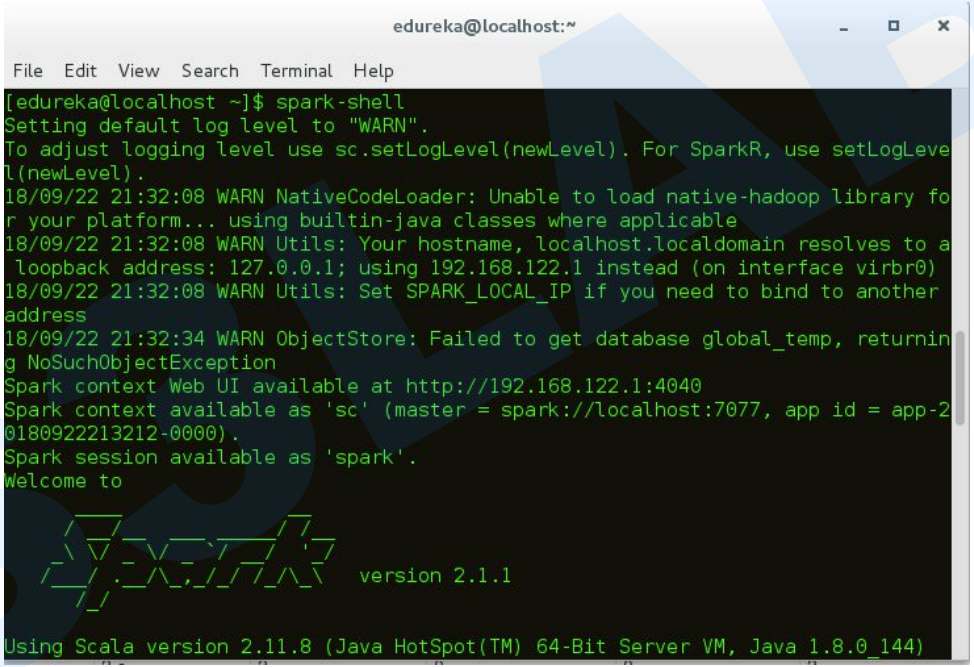
Word count demo

Run wordcount on cloudera

- hdfs dfs -mkdir /user/cloudera/input
- hdfs dfs -put /home/cloudera/wordcount.txt /user/cloudera/input
- hdfs dfs -ls /user/cloudera/input
- hdfs dfs -ls /user
- hdfs dfs -cat /user/cloudera/input/wordcount.txt
- ls -ltr /usr/lib/hadoop-mapreduce/
- hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
- hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/cloudera/input/wordcount.txt /user/cloudera/output
- hdfs dfs -ls /user/cloudera/output

Spark Program

Word count demo



Spark Program

Word count demo

- Create a text file and stored it in the hdfs directory.
- Create RDD with transformation **flatMap():**
  - scala> var map = sc.textFile("/user/cloudera/input/wordcount.txt").flatMap(line => line.split(" ")).map(word => (word,1));
- Apply action reduceByKey() to the created RDD.
  - scala> var counts = map.reduceByKey(_+_);
- save the output in a text file
  - scala> counts.saveAsTextFile("/user/cloudera/output1");

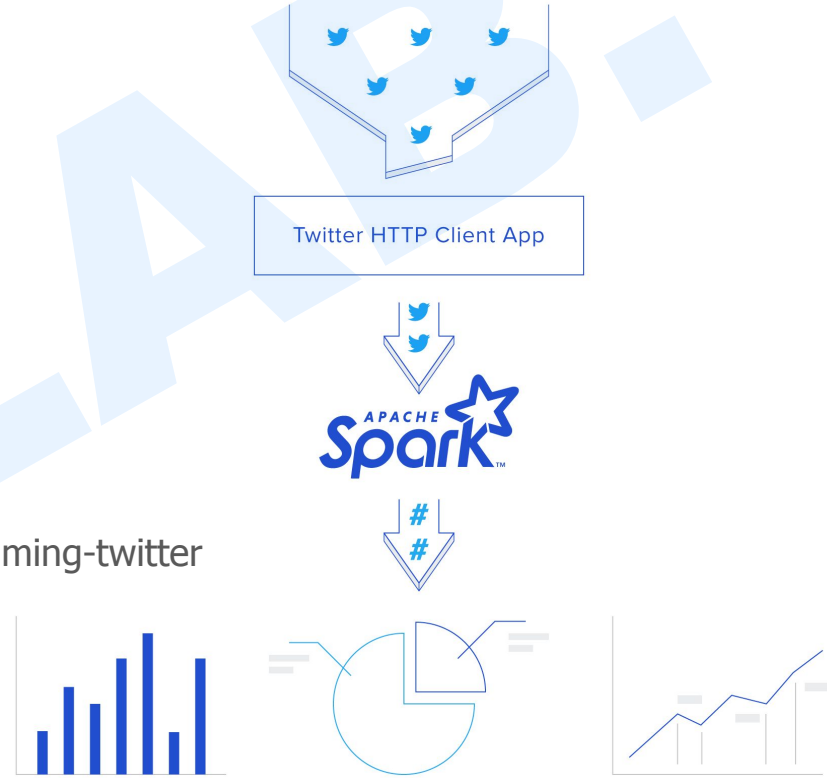
Usercases

- ETL
- Analytics
- Machine Learning
- Graph processing
- SQL queries on large data sets
- Batch processing
- Stream processing

Usercases - Twitter

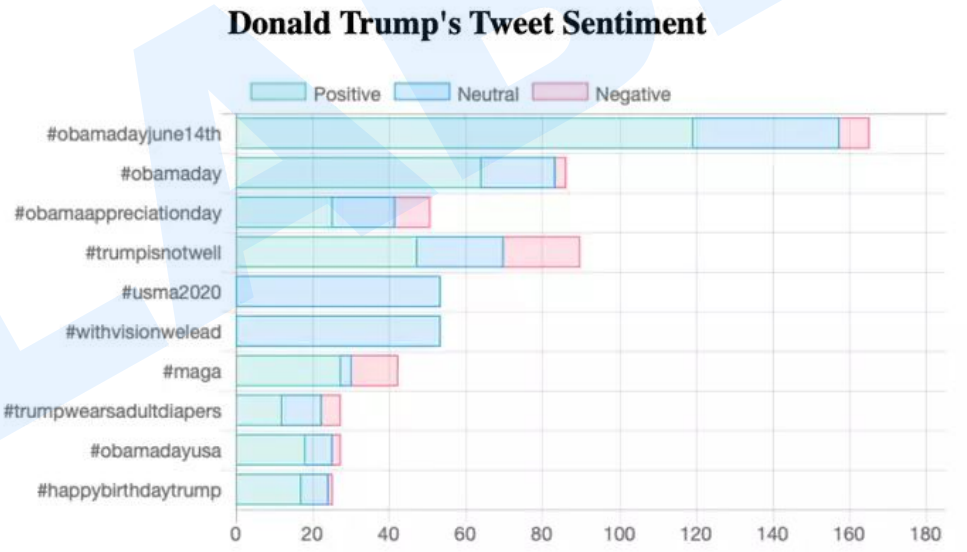
1. Twitter App Client (Python)
2. Spark App (Python)
3. Dashboard Web App
   ChartJS

https://www.toptal.com/apache/apache-spark-streaming-twitter



Usercases - Twitter

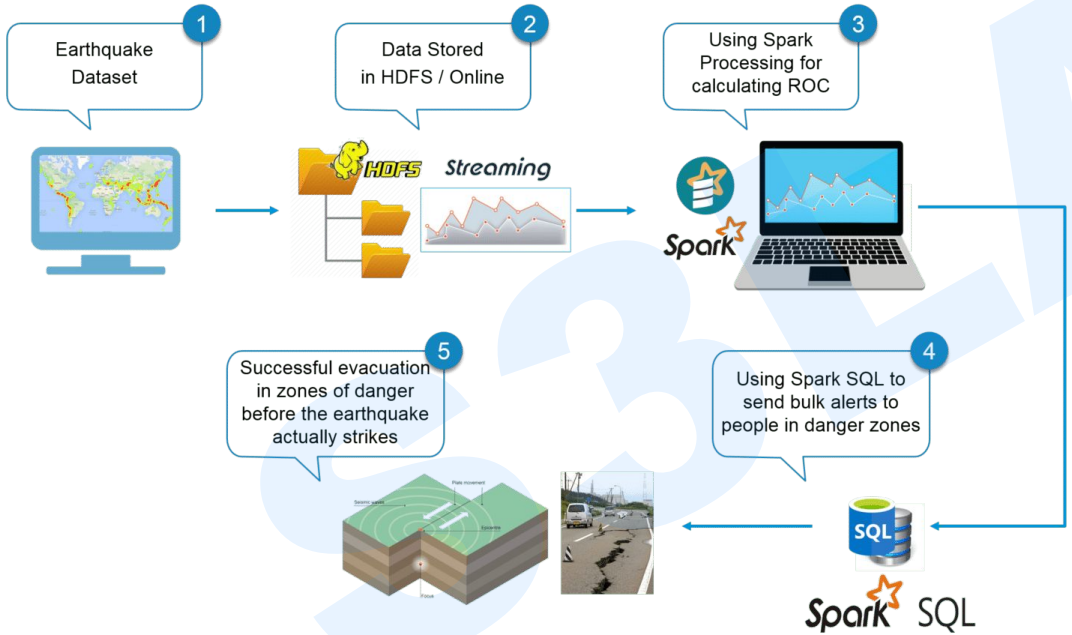
```
----------- 2020-06-14 18:44:30 -----------
+-------------------+-------------+---+---+---+
|            hashtag|hashtag_count|pos|neu|neg|
+-------------------+-------------+---+---+---+
|          #usma2020|            1|  0|  1|  0|
|  #withvisionwelead|            1|  0|  1|  0|
|       #feebleweeble|            1|  0|  1|  0|
|#allbirthdaysmatter|            1|  0|  1|  0|
+-------------------+-------------+---+---+---+

----------- 2020-06-14 18:44:32 -----------
+-------------------+-------------+---+---+---+
|            hashtag|hashtag_count|pos|neu|neg|
+-------------------+-------------+---+---+---+
|          #usma2020|            1|  0|  1|  0|
|  #withvisionwelead|            1|  0|  1|  0|
|       #feebleweeble|            1|  0|  1|  0|
|#allbirthdaysmatter|            1|  0|  1|  0|
+-------------------+-------------+---+---+---+
```

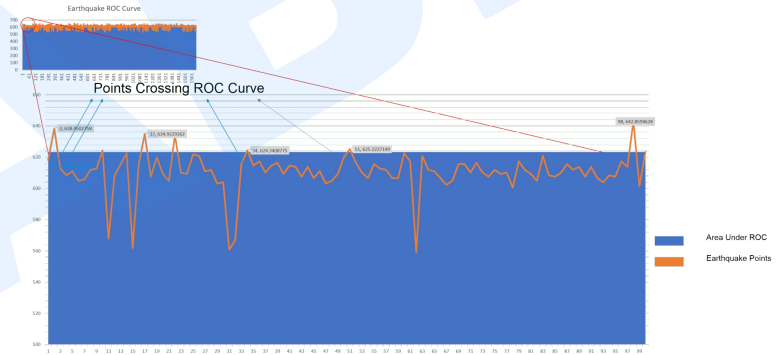


Usercases - Earthquake Detection System

EARTHQUAKE ROC DATASET																		
	S Wave							P Wave										
Classification Index	First Activity	Time Taken	Acceleration	Building Strength	Velocity	Sa	Sd	First Activity	Time Taken	Acceleration	Building Strength	Velocity	Sa	Sd	Total Weight	Sum * ROC	ROC	AVG * ROC
0	3	11	14	19	39	42	55	64	67	73	75	76	80	83	701	618.168	0.881837	623.2843
0	3	6	17	27	35	40	57	63	69	73	74	76	81	103	724	638.4503	0.881837	623.2843
0	4	6	15	21	35	40	57	63	67	73	74	77	80	83	695	612.877	0.881837	623.2843
0	5	6	15	22	36	41	47	66	67	72	74	76	80	83	690	608.4678	0.881837	623.2843
0	2	6	16	22	36	40	54	63	67	73	75	76	80	83	693	611.1133	0.881837	623.2843
0	2	6	14	20	37	41	47	64	67	73	74	76	82	83	686	604.9405	0.881837	623.2843
0	1	6	14	22	36	42	49	64	67	72	74	77	80	83	687	605.8223	0.881837	623.2843
0	1	6	17	19	39	42	53	64	67	73	74	76	80	83	694	611.9952	0.881837	623.2843
0	2	6	18	20	37	42	48	64	71	73	74	76	81	83	695	612.877	0.881837	623.2843
1	5	11	15	32	39	40	52	63	67	73	74	76	78	83	708	624.3409	0.881837	623.2843
0	5	16	30	35	41	64	67	73	74	76	80	83			644	567.9033	0.881837	623.2843
0	5	6	15	20	37	40	50	63	67	73	75	76	80	83	690	608.4678	0.881837	623.2843
0	5	7	16	29	39	40	48	63	67	73	74	76	78	83	698	615.5225	0.881837	623.2843
0	1	11	18	20	37	42	59	62	71	72	74	76	80	83	706	622.5772	0.881837	623.2843
1	5	18	19	39	40	63	67	73	74	76	80	83			637	561.7304	0.881837	623.2843

Usercases - Earthquake Detection System





https://github.com/AI4EPS/QuakeFlow

Usercases - Movie Recommendation System

Usercases - Movie Recommendation System

Figure: *Various File Formats*

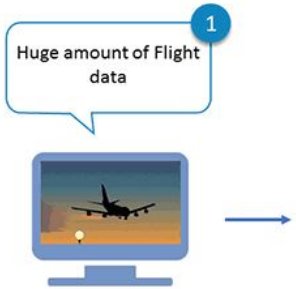
https://willmbennett.medium.com/how-to-build-a-recommendation-system-using-spark-38663f9017bf

Usercases - Flight Data Analysis

https://www.edureka.co/blog/spark-graphx/

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	dOfM	dOfW	carrier	tailNum	flNum	origin_id	origin	dest_id	dest	crsdeptime	deptime	depdelaymins	crsarrtime	arrtime	arrdelaymins	crselapsedtime	dist
2	1	3	AA	N338AA	1	12478	JFK	12892	LAX	900	914	14	1225	1238	13	385	2475
3	2	4	AA	N338AA	1	12478	JFK	12892	LAX	900	857	0	1225	1226	1	385	2475
4	4	6	AA	N327AA	1	12478	JFK	12892	LAX	900	1005	65	1225	1324	59	385	2475
5	5	7	AA	N323AA	1	12478	JFK	12892	LAX	900	1050	110	1225	1415	110	385	2475
6	6	1	AA	N319AA	1	12478	JFK	12892	LAX	900	917	17	1225	1217	0	385	2475
7	7	2	AA	N328AA	1	12478	JFK	12892	LAX	900	910	10	1225	1212	0	385	2475
8	8	3	AA	N323AA	1	12478	JFK	12892	LAX	900	923	23	1225	1215	0	385	2475
9	9	4	AA	N339AA	1	12478	JFK	12892	LAX	900	859	0	1225	1204	0	385	2475
10	10	5	AA	N319AA	1	12478	JFK	12892	LAX	900	929	29	1225	1245	20	385	2475

Usercases - Flight Data Analysis



Usercases - Flight Data Analysis

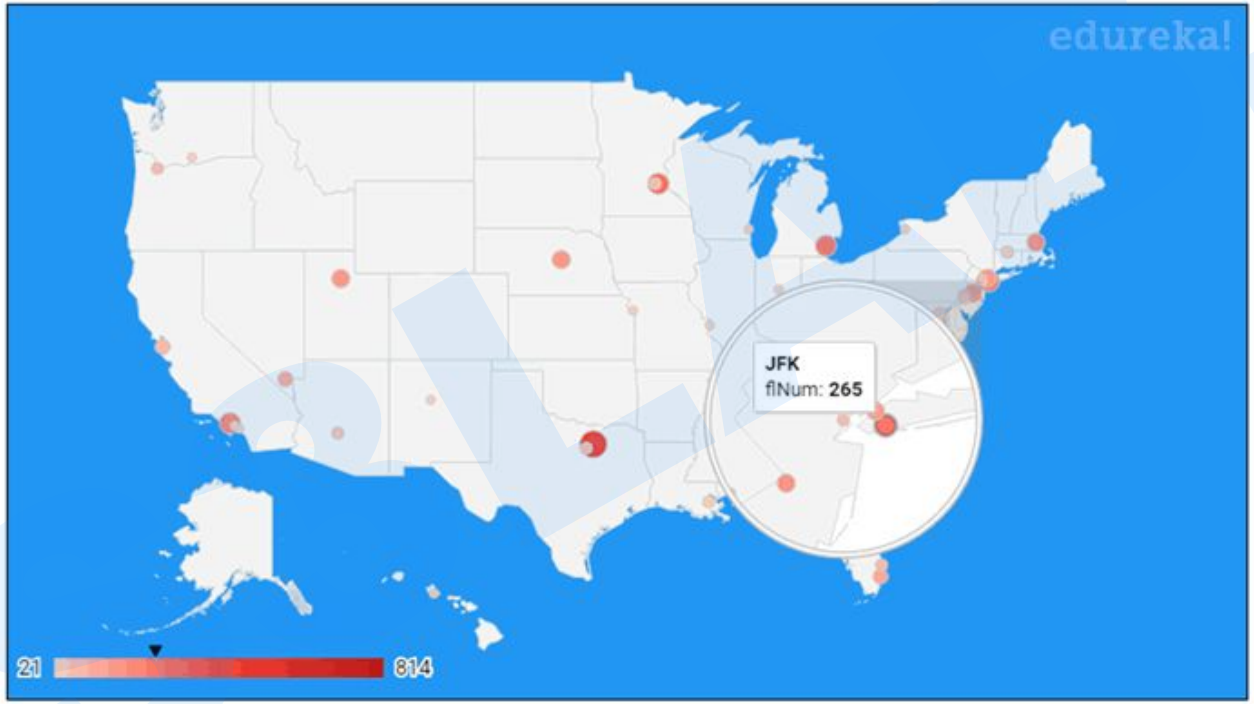


Figure: *Total Number of Flights from New York*

Q & A

Thank you for listening

"Coming together is a beginning;
Keeping together is progress;
Working together is success."
- HENRY FORD