Architectures for massive data management

Apache Spark

Albert Bifet
albert.bifet@telecom-paristech.fr

October 20, 2015

Spark Motivation

Apache Spark

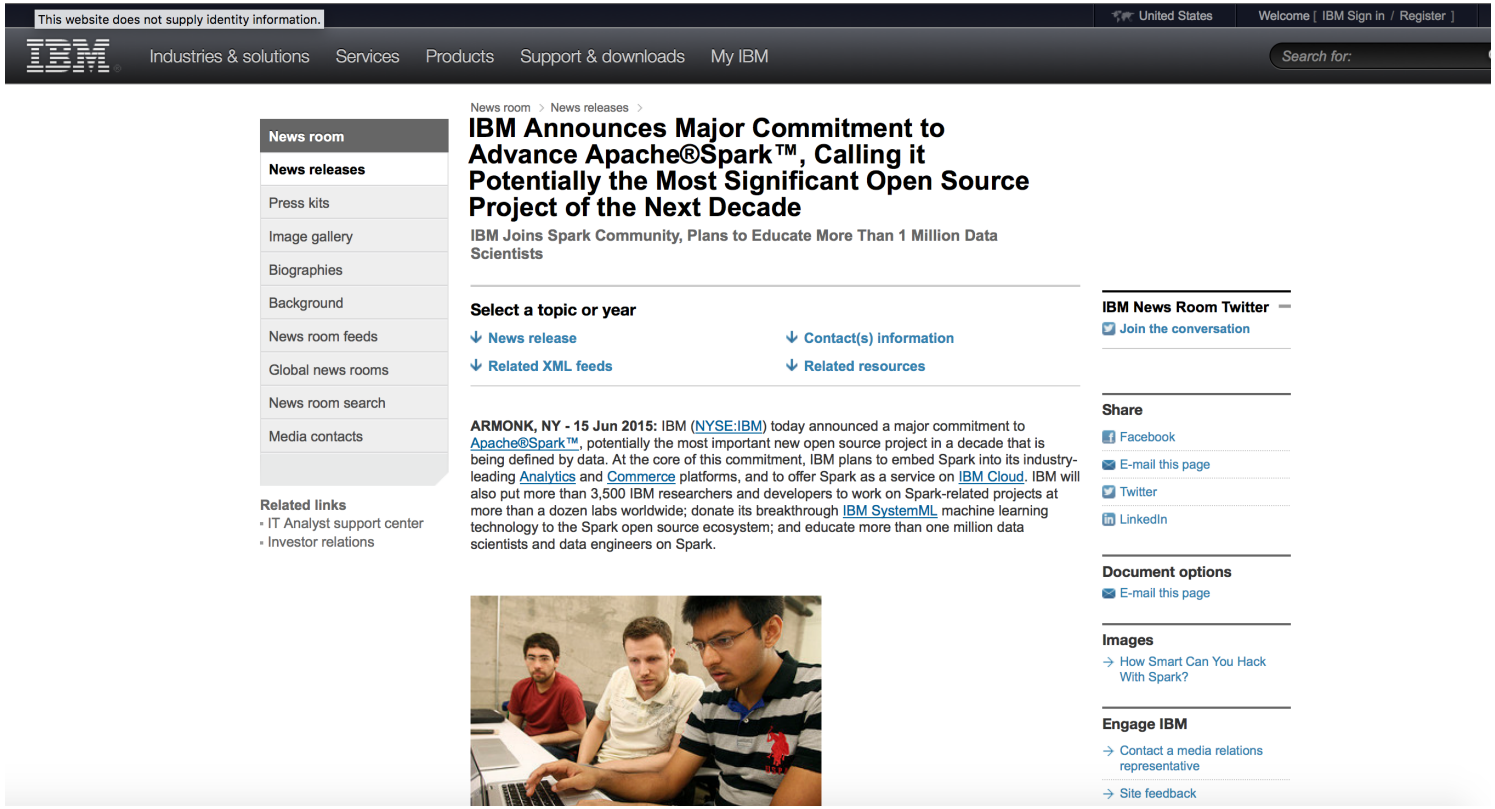


Figure: IBM and Apache Spark

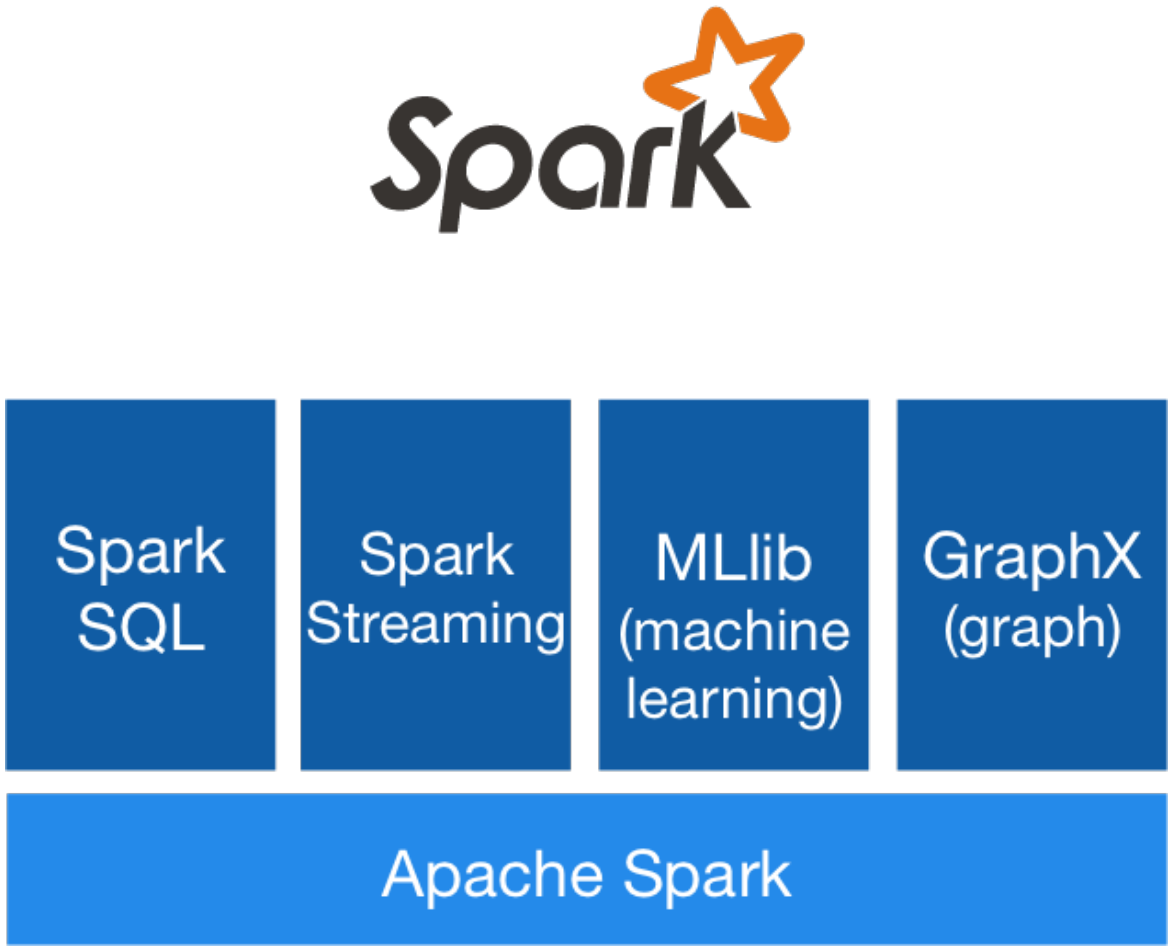
What is Apache Spark

Apache Spark is a fast and general engine for large-scale data processing.

- **Speed**: Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- **Ease of Use**: Write applications quickly in Java, Scala, Python, R.
- **Generality**: Combine SQL, streaming, and complex analytics.
- **Runs Everywhere**: Spark runs on Hadoop, Mesos, standalone, or in the cloud.

`http://spark.apache.org/`

Spark Ecosystem



Spark API


```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

```
val f = sc.textFile(hdfs://...")

val wc = f.flatMap(l => l.split(" "))
          .map(word => (word, 1))
          .reduceByKey(_ + _)
```

Word count in Spark's Scala API

Apache Spark

Apache Spark Project

- Spark started as a research project at UC Berkeley
  - Matei Zaharia created Spark during his PhD
  - Ion Stoica was his advisor
- DataBricks is the Spark start-up, that has raised $46 million

Resilient Distributed Datasets (RDDs)

- An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- RDDs are created :
  - parallelizing an existing collection in your driver program, or
  - referencing a dataset in an external storage system

Spark API: Parallel Collections


```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

Spark's Python API

```
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

Spark's Scala API

```
 List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
JavaRDD<Integer> distData = sc.parallelize(data);
```

Spark's Java API

Spark API: External Datasets


```
>>> distFile = sc.textFile("data.txt")
```

Spark's Python API

```
scala> val distFile = sc.textFile("data.txt")
distFile: RDD[String] = MappedRDD@1d4cee08
```

Spark's Scala API

```
JavaRDD<String> distFile = sc.textFile("data.txt");
```

Spark's Java API

Spark API: RDD Operations


```
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Spark's Python API

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

Spark's Scala API

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

Spark's Java API

Spark API: Working with Key-Value Pairs


```
lines = sc.textFile("data.txt")
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

Spark's Python API

```
val lines = sc.textFile("data.txt")
val pairs = lines.map(s => (s, 1))
val counts = pairs.reduceByKey((a, b) => a + b)
```

Spark's Scala API

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaPairRDD<String, Integer> pairs =
                        lines.mapToPair(s -> new Tuple2(s, 1));
JavaPairRDD<String, Integer> counts =
                        pairs.reduceByKey((a, b) -> a + b);
```

Spark's Java API

Spark API: Shared Variables


```
>>> broadcastVar = sc.broadcast([1, 2, 3])

>>> broadcastVar.value
[1, 2, 3]
```

Spark's Python API

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))

scala> broadcastVar.value
res0: Array[Int] = Array(1, 2, 3
```

Spark's Scala API

```
Broadcast<int[]> broadcastVar = sc.broadcast(new int[] {1, 2, 3});

broadcastVar.value();
// returns [1, 2, 3]
```

Spark's Java API

Spark Cluster

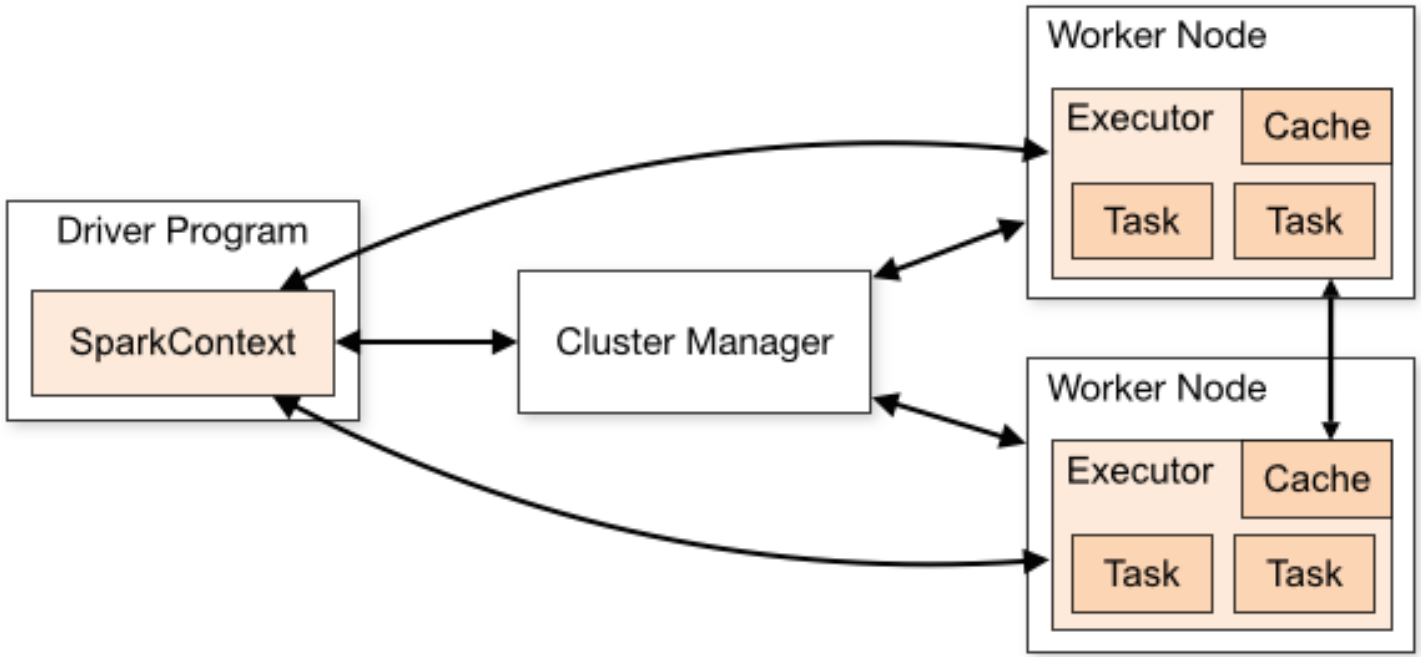
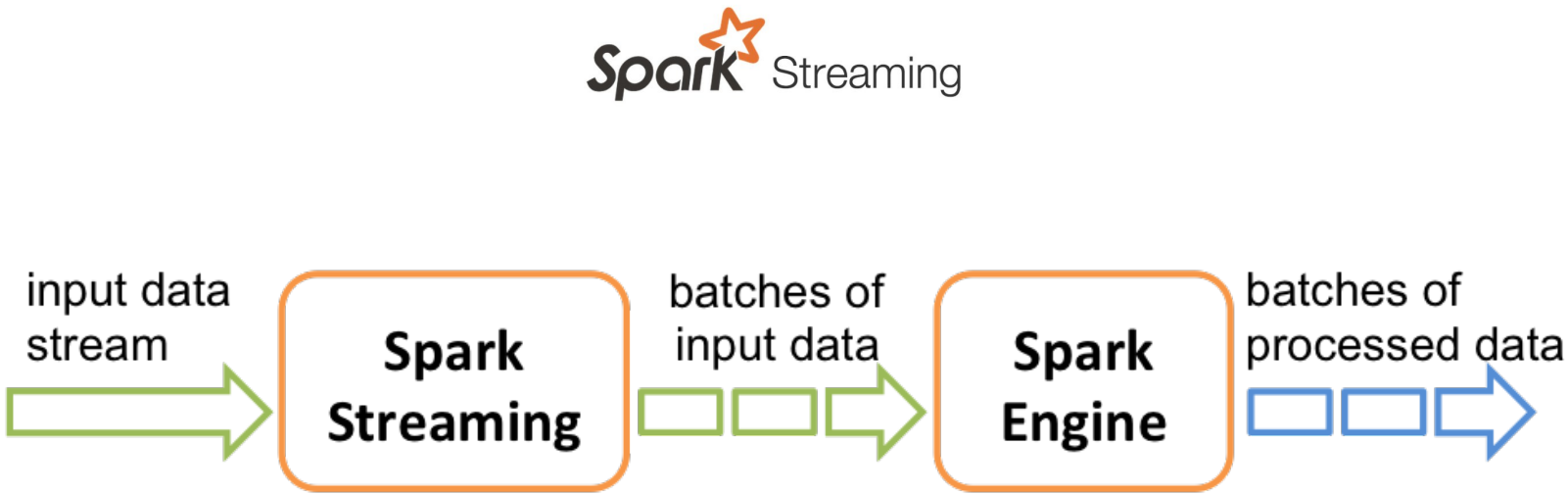


Figure: Cluster Components

Spark Cluster

- Spark is agnostic to the underlying cluster manager.
- The spark driver is the program that declares the transformations and actions on RDDs of data and submits such requests to the master.
- Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. Each driver schedules its own tasks.
- The drivers must listen for and accept incoming connections from its executors throughout its lifetime
- Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network.

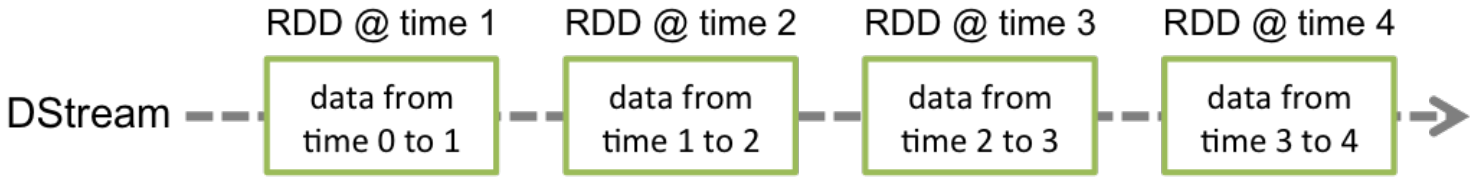
Apache Spark Streaming



Spark Streaming is an extension of Spark that allows processing data stream using micro-batches of data.

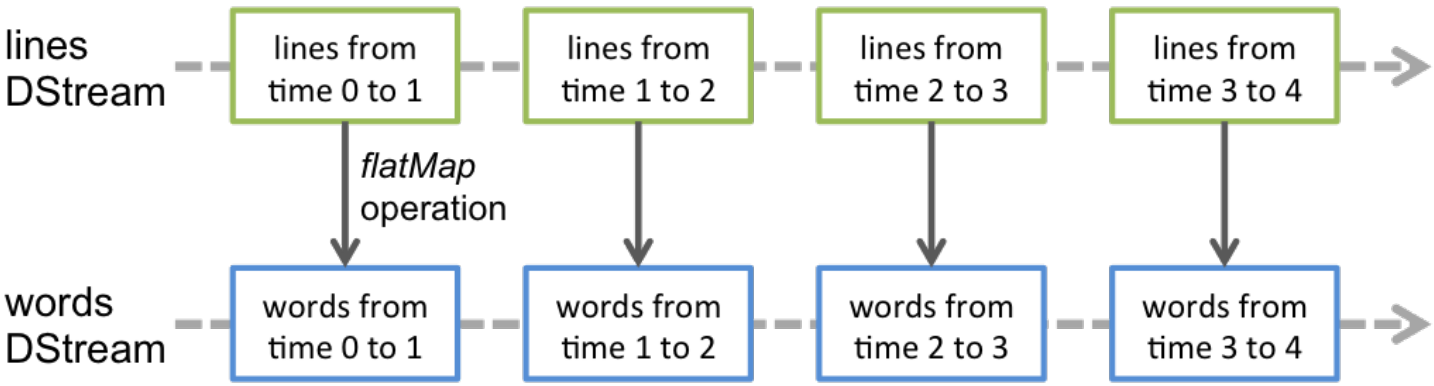
Discretized Streams (DStreams)

- Discretized Stream or DStream represents a continuous stream of data,
  - either the input data stream received from source, or
  - the processed data stream generated by transforming the input stream.
- Internally, a DStream is represented by a continuous series of RDDs



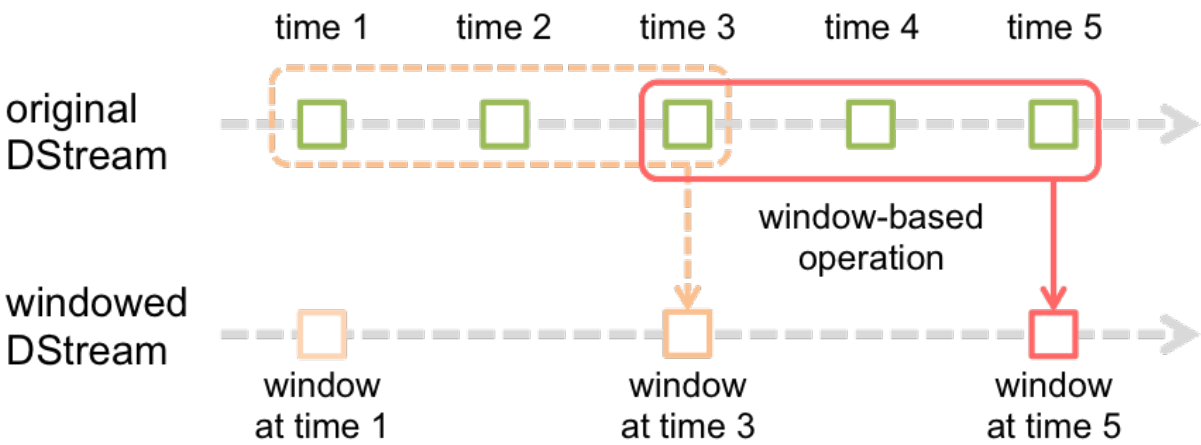
Discretized Streams (DStreams)

- Any operation applied on a DStream translates to operations on the underlying RDDs.



Discretized Streams (DStreams)

- Spark Streaming provides windowed computations, which allow transformations over a sliding window of data.



Spark Streaming

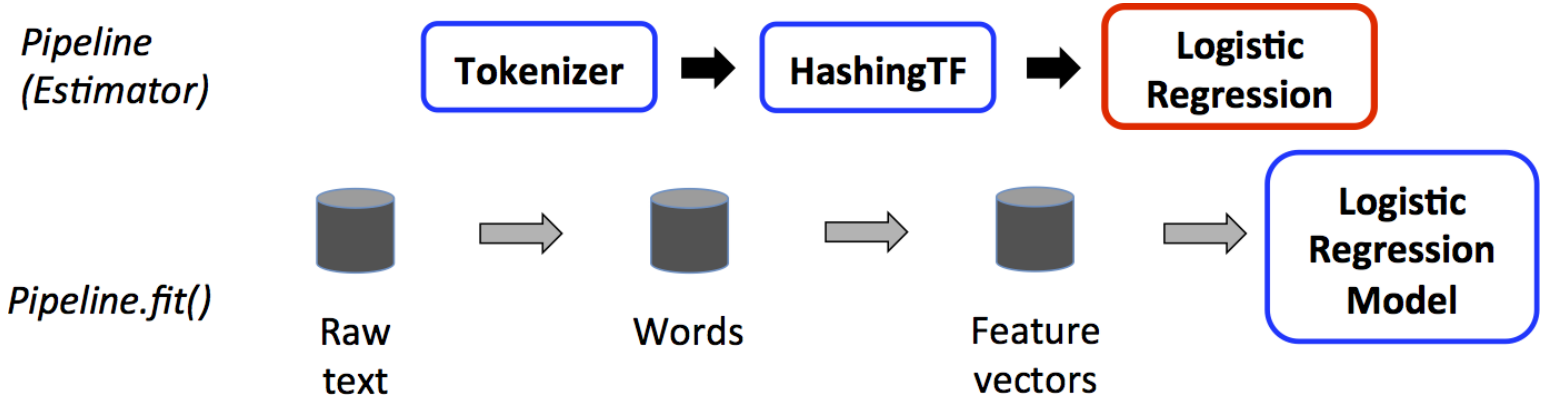

```scala
val conf = new SparkConf().setMaster("local[2]").setAppName("WCount")
val ssc = new StreamingContext(conf, Seconds(1))

// Create a DStream that will connect to hostname:port, like localhost:9999
val lines = ssc.socketTextStream("localhost", 9999)

// Split each line into words
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the con
wordCounts.print()

ssc.start()             // Start the computation
ssc.awaitTermination()  // Wait for the computation to terminate
```

Spark SQL and DataFrames

- Spark SQL is a Spark module for structured data processing.
- It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine.
- A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database .

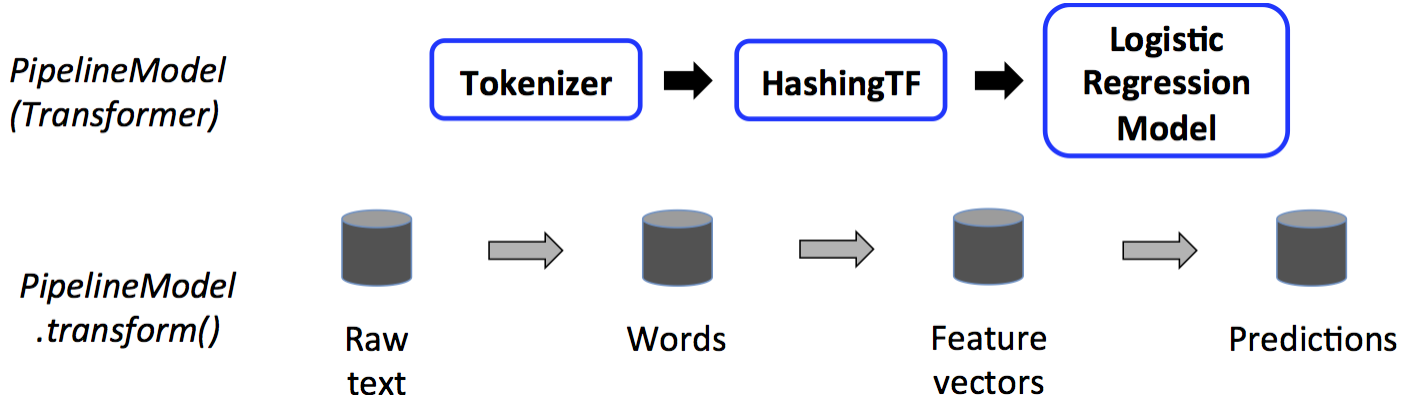
Spark Machine Learning Libraries

- MLLib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.

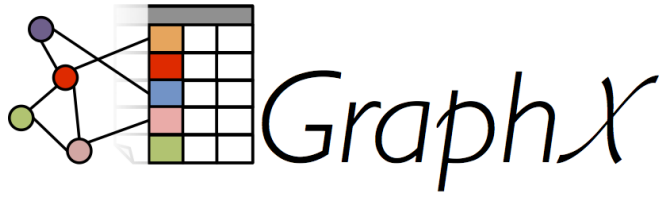


Spark Machine Learning Libraries

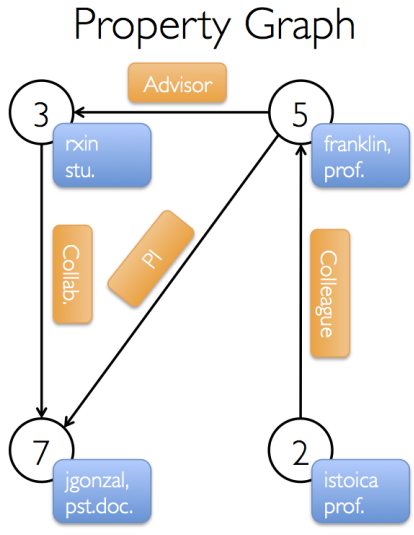
- MLLib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.



Spark GraphX



- GraphX optimizes the representation of vertex and edge types when they are primitive data types
- The property graph is a directed multigraph with user defined objects attached to each vertex and edge.



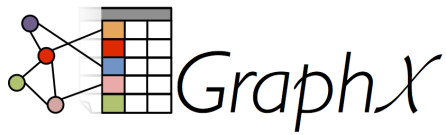
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Spark GraphX




```
// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                       (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                       Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
```

Apache Spark Summary

Apache Spark is a fast and general engine for large-scale data processing.

- **Speed**: Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- **Ease of Use**: Write applications quickly in Java, Scala, Python, R.
- **Generality**: Combine SQL, streaming, and complex analytics.
- **Runs Everywhere**: Spark runs on Hadoop, Mesos, standalone, or in the cloud.

`http://spark.apache.org/`