Data 604: Data Management

Jyotsna Potarazu, Adjunct Professor

Data 604: Data Management

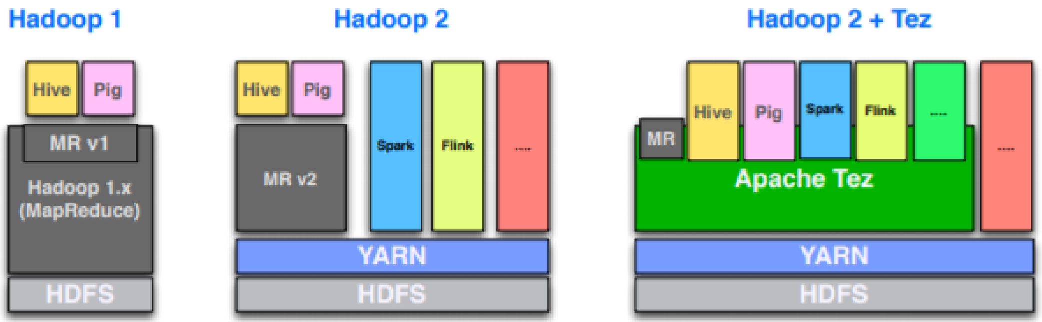
Lecture topics:

- Evolution of Hadoop
- Apache Spark
- Final Exam Topics Review

Jyotsna Potarazu, Adjunct Faculty

Evolution of Hadoop

- Hadoop 1
- Hadoop 2 with YARN
- TEZ with YARN



Reference from article: Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications

Tez

- Successor to MapReduce, part of Hadoop ecosystem https://tez.apache.org/
- It breaks a data process into tasks using the directed-acyclic-graph(DAG) framework.
- Apache projects Pig, HIVE, and Cascade can run on Tez and jobs leveraging TEZ have shown tremendous performance improvements.
- Hortonworks (and Microsoft?) project. Now that Cloudera and Hortonworks merged, expect wider adoption.

In Figure 9 we show a comparative scale test of Hive on Tez, with a TPC-H derived Hive workload [35], at 10 terabytes scale on a 350 node research cluster with 16 cores, 24Gb RAM and 6 x 2Tb drives per node. This was presented at Hadoop Summit 2014, San Jose. This shows that Tez based implementation outperforms the MapReduce based implementation at large cluster scale.

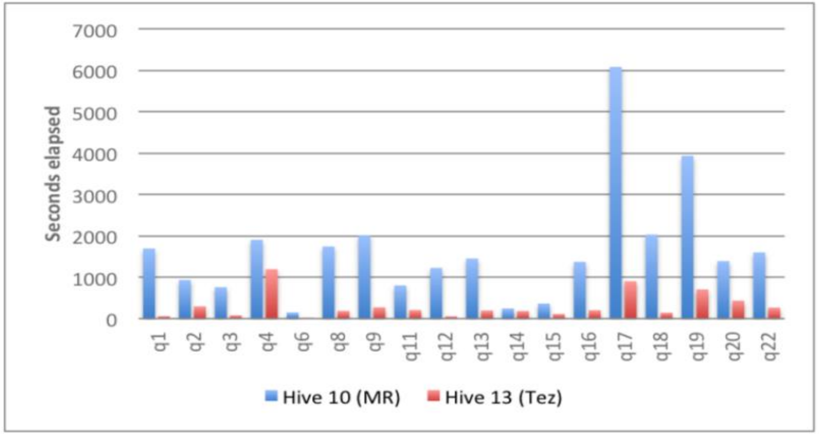
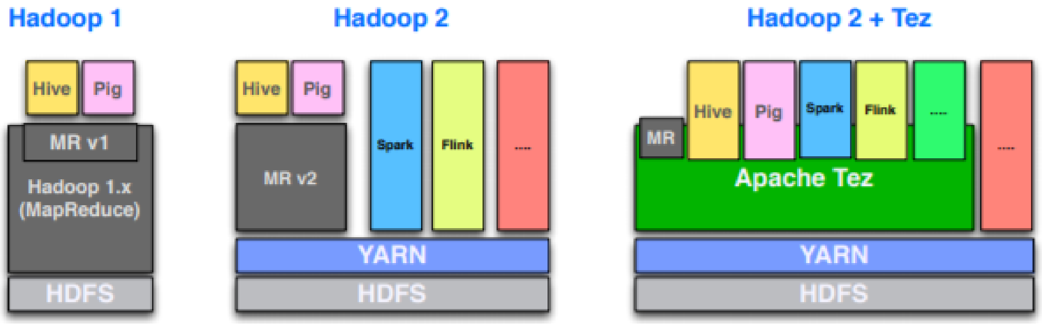


Figure 9: Hive: TPC-H derived workload at Yahoo (10TB scale)

Image copied from paper: Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications

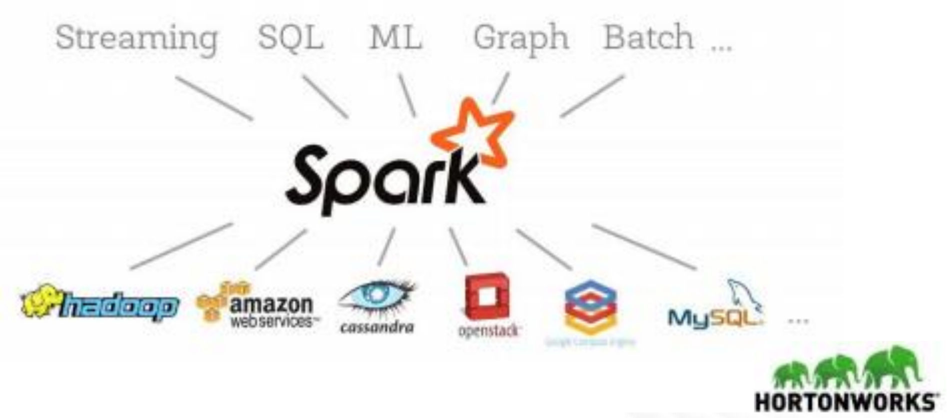
Spark



- https://spark.apache.org/
- Built as an option to address MapReduce shortcomings. Supports batch, in process, and streaming processes
- Can use it's own standalone framework or can leverage YARN for resource management
- Use PySpark libraries to program from Python https://pypi.org/project/pyspark/.

Apache Spark

- Similar to Hadoop, Spark works with HDFS and requires a cluster manager (e.g. YARN)
- Key components
  - Spark Core
  - Spark SQL
  - MLib
  - Spark Streaming
  - GraphX

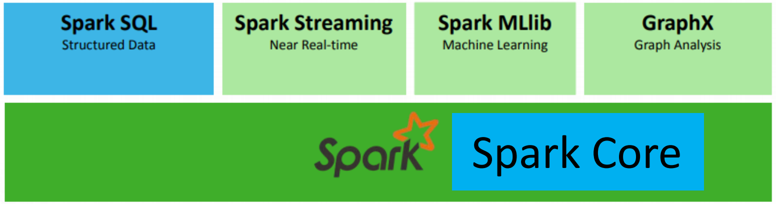


Apache Spark

- Open-source alternative for MapReduce
- New programming paradigm centered on a data structure called the resilient distributed dataset (RDD) which can be distributed across a cluster of machines and is maintained in a fault tolerant way
- RDDs can enable the construction of iterative programs that have to visit a data set multiple times, as well as more interactive or exploratory programs
- Many orders of magnitude faster than MapReduce implementations
- Rapidly adopted by many Big Data vendors

Spark Core

- Foundation for all other components
- Provides functionality for task scheduling and a set of basic data transformations that can be used through many programming languages (e.g., Java, Python, Scala, and R)
- RDDs are the primary data abstraction in Spark
  - designed to support in-memory data storage and operations, distributed across a cluster



Spark Core

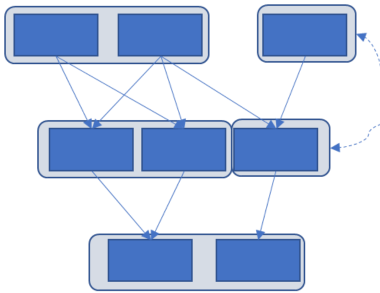
- Once data is loaded into an RDD, two basic types of operations can be performed:
  - Transformation which creates a new RDD through changing the original one
  - Actions which measure but do not change the original data
- Transformations are lazily evaluated
  - executed when a subsequent action has a need for the result
- RDDs will also be kept as long as possible in memory
- A chain of RDD operations gets compiled by Spark into a directed acyclic graph but which is then spread out and calculated over the cluster

Spark Core

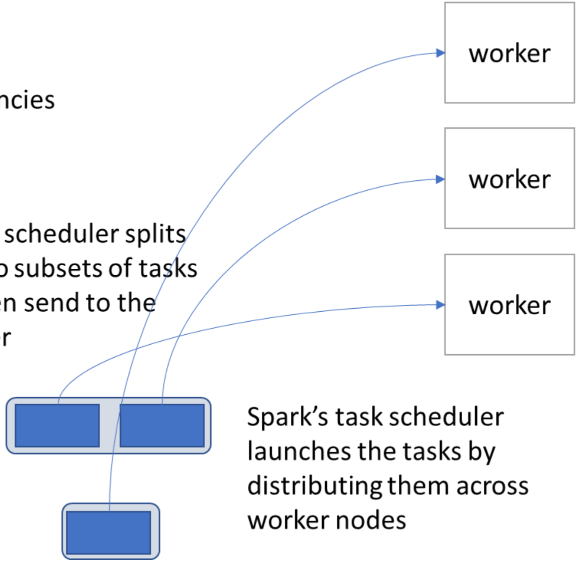
A programmer writes a Spark program using its API:

```
rdd1.join(rdd2).groupBy(…).filter(…)
```

Based on this, Spark builds a directed acyclic graph of operations with their dependencies



Spark's graph scheduler splits the graph into subsets of tasks which are then send to the task scheduler



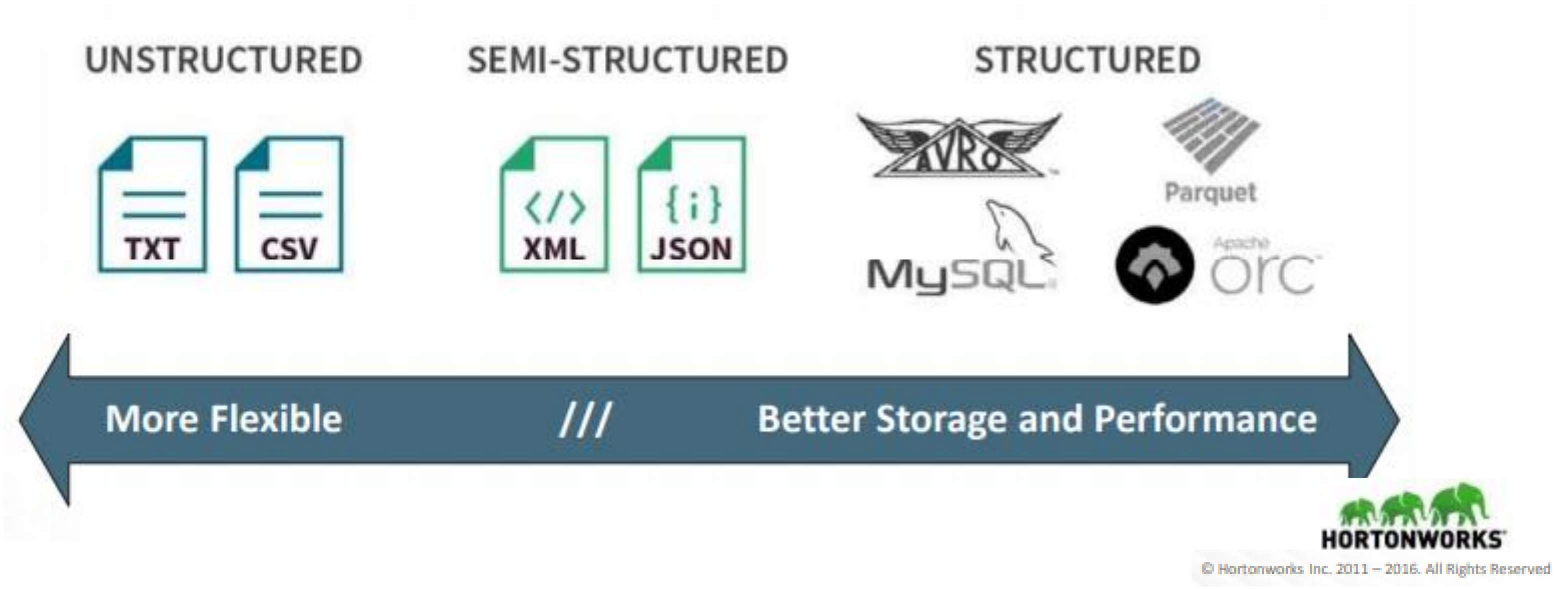
Spark's task scheduler launches the tasks by distributing them across worker nodes

Spark Core

- Spark's RDD API is relatively easy to work with compared to writing MapReduce programs

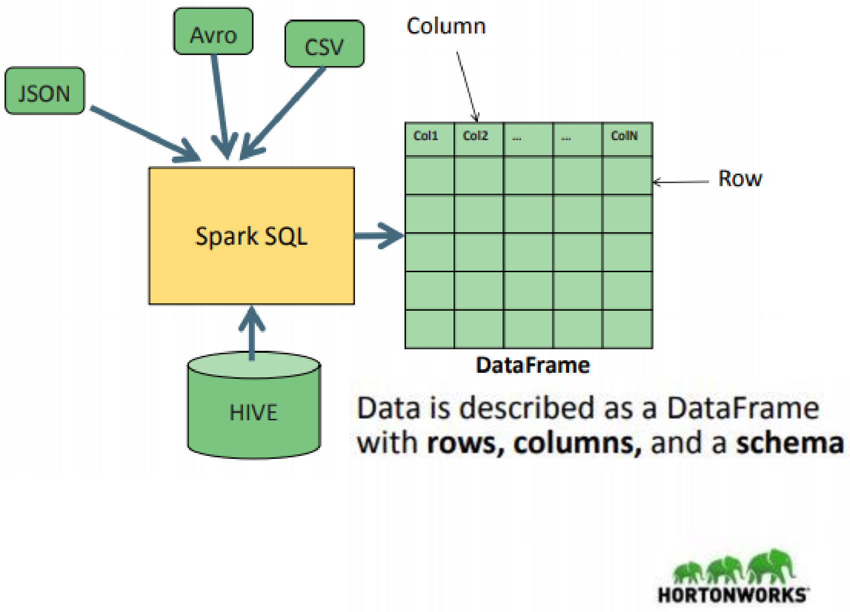
```
# Set up connection to the Spark cluster
sconf = SparkConf()
sc = SparkContext(master='', conf=sconf)

# Load in an RDD from a text file, the RDD will represent a collection of
# text strings (one for each line)
text_file = sc.textFile("myfile.txt")

# Count the word occurrences
counts = text_file.flatMap(lambda line: line.split(" ")) \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b)

print(counts)
```

Spark SQL


UNSTRUCTURED
SEMI-STRUCTURED
STRUCTURED
TXT
CSV
XML
JSON
AVRO
Parquet
MySQL
Apache
orc
More Flexible
///
Better Storage and Performance
HORTONWORKS
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Spark SQL

- Spark SQL runs on top of Spark Core and introduces another data abstraction called DataFrames
- DataFrames can be created from RDDs by specifying a schema on how to structure the data elements in the RDD, or can be loaded in directly from various sorts of file formats
- Even although DataFrames continue to use RDDs behind the scenes, they represent themselves to the end user as a collection of data organized into named columns



Spark SQL

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Spark example").getOrCreate()

# Create a DataFrame object by reading in a file
df = spark.read.json("people.json")

df.show()
# | age|    name|
# +----+--------+
# |null|   Seppe|
# |  30|Wilfried|
# |  19|    Bart|
# +----+--------+

# DataFrames are structured in columns and rows:
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)
```

Spark SQL

```
df.select("name").show()
# +--------+
# |    name|
# +--------+
# |   Seppe|
# |Wilfried|
# |    Bart|
# +--------+

# SQL-like operations can now easily be expressed:
df.select(df['name'], df['age'] + 1).show()
# +--------+---------+
# |    name|(age + 1)|
# +--------+---------+
# |   Seppe|     null|
# |Wilfried|       31|
# |    Bart|       20|
# +--------+---------+
```

Spark SQL

```
df.filter(df['age'] > 21).show()
# +---+--------+
# |age|    name|
# +---+--------+
# | 30|Wilfried|
# +---+--------+

df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |null|    1|
# |  30|    1|
# +----+-----+
```

Spark SQL

Spark implements a full SQL query engine which can convert SQL statements to a series of RDD transformations and actions

```
# Register the DataFrame as a SQL temporary view df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people WHERE age > 21")
sqlDF.show()

# +---+--------+
# |age|    name|
# +---+--------+
# | 30|Wilfried|
# +---+--------+
```

API Examples: DataFrame and SQL APIs

DataFrame API

```
flights.select("Origin", "Dest", "DepDelay")
       .filter($"DepDelay" > 15).show(5)
```

SQL API

```
SELECT Origin, Dest, DepDelay
FROM flightsView
WHERE DepDelay > 15 LIMIT 5
```

Results

```
+------+----+--------+
|Origin|Dest|DepDelay|
+------+----+--------+
|   IAD| TPA|      19|
|   IND| BWI|      34|
|   IND| JAX|      25|
|   IND| LAS|      67|
|   IND| MCO|      94|
+------+----+--------+
```

HORTONWORKS

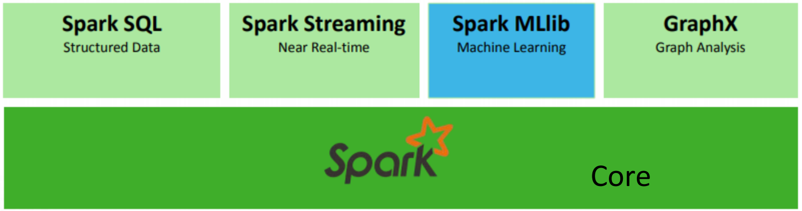
MLlib, Spark Streaming and GraphX

Example: Word counting

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
sc = SparkContext("local[2]", "StreamingWordCount")
ssc = StreamingContext(sc, 1)
 # Create a DStream that will connect to server.mycorp.com:9999 as a source
lines = ssc.socketTextStream("server.mycorp.com ", 9999)
 # Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
# Print out first ten elements of each RDD generated in the wordCounts Dstream
wordCounts.pprint()
# Start the computation
ssc.start()
ssc.awaitTermination()
```

MLlib

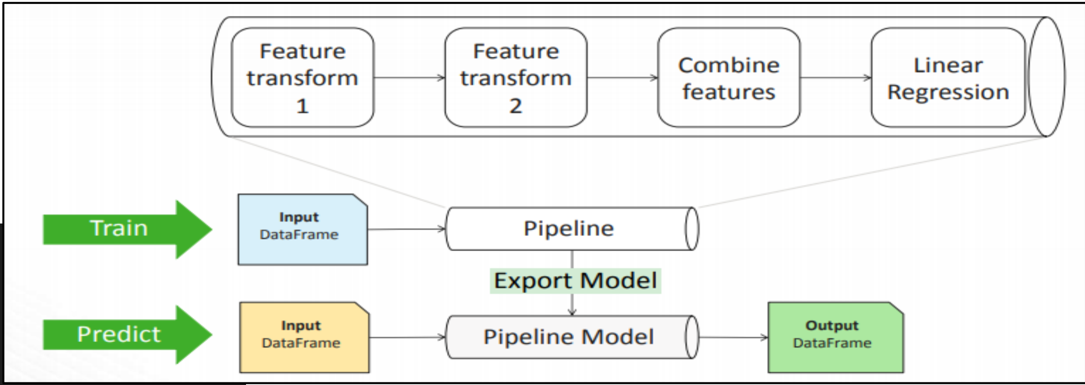
- MLlib is Spark's machine learning library
  - offers classification, regression, clustering, and recommender system algorithms
- MLlib was originally built directly on top of the RDD abstraction
- New MLlib version works directly with SparkSQL's DataFrames based API

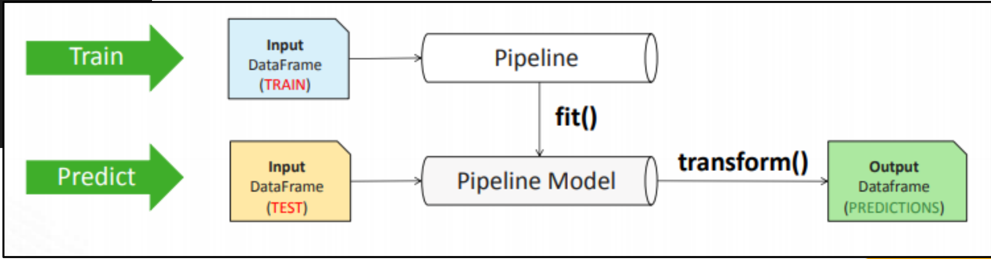


Spark ML Pipeline

Sample Spark ML Pipeline

```
indexer = …
parser = …
hashingTF = …
vecAssembler = …

rf = RandomForestClassifier(numTrees=100)
pipe = Pipeline(stages=[indexer, parser, hashingTF, vecAssembler, rf])

model = pipe.fit(trainData)              # Train model
results = model.transform(testData)      # Test model
```

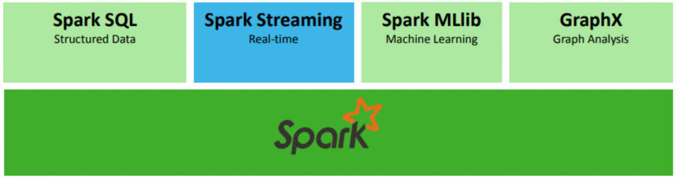


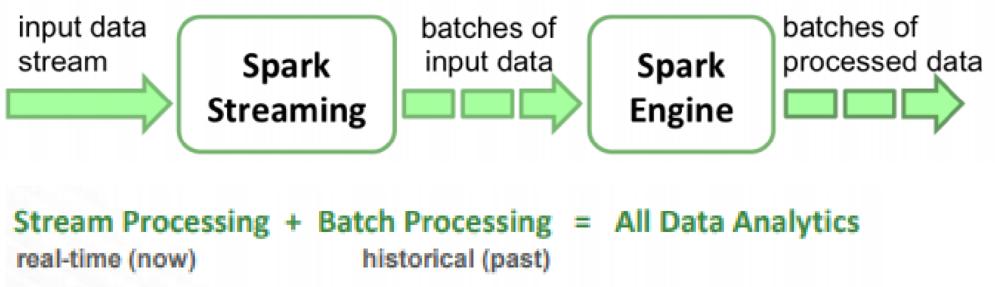


HORTONWORKS

MLlib, Spark Streaming and GraphX

- Spark Streaming leverages Spark Core and its fast scheduling engine to perform streaming analytics
- Spark Streaming provides another high-level concept called the DStream, which represents a continuous stream of data
    - represented as a sequence of RDD fragments
- DStreams provide windowed computations



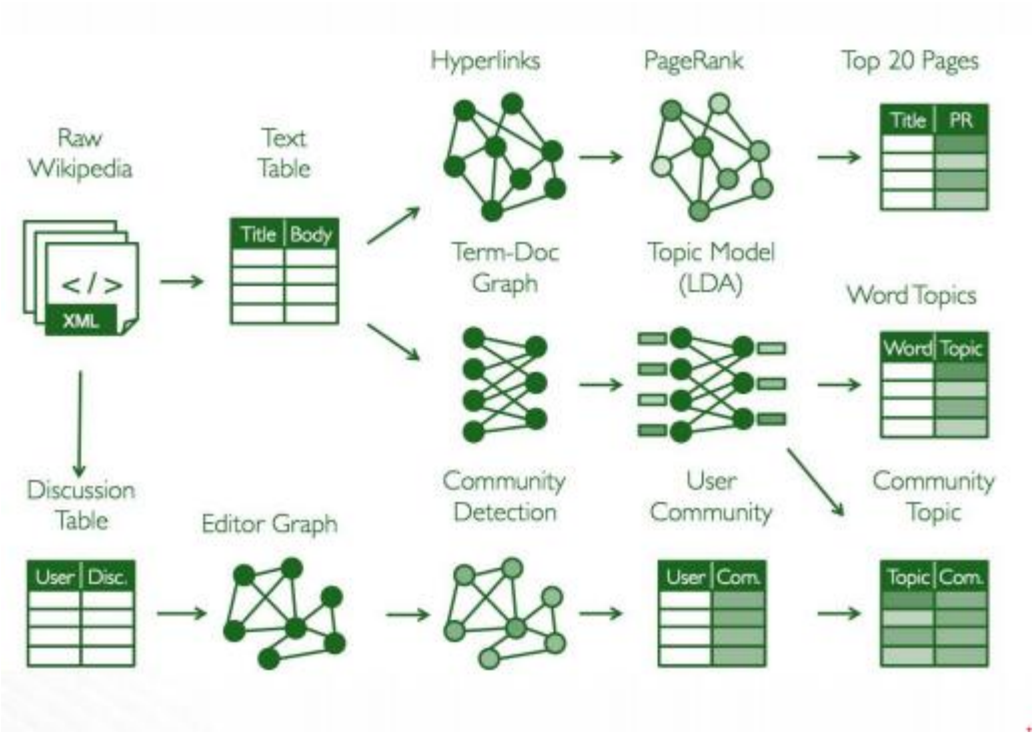


HORTONWORKS

GraphX

- GraphX is Spark's component implementing programming abstractions to deal with graph based structures, again based on the RDD abstraction
- GraphX comes with a set of fundamental operators and algorithms to work with graphs and simplify graph analytics tasks

GraphX



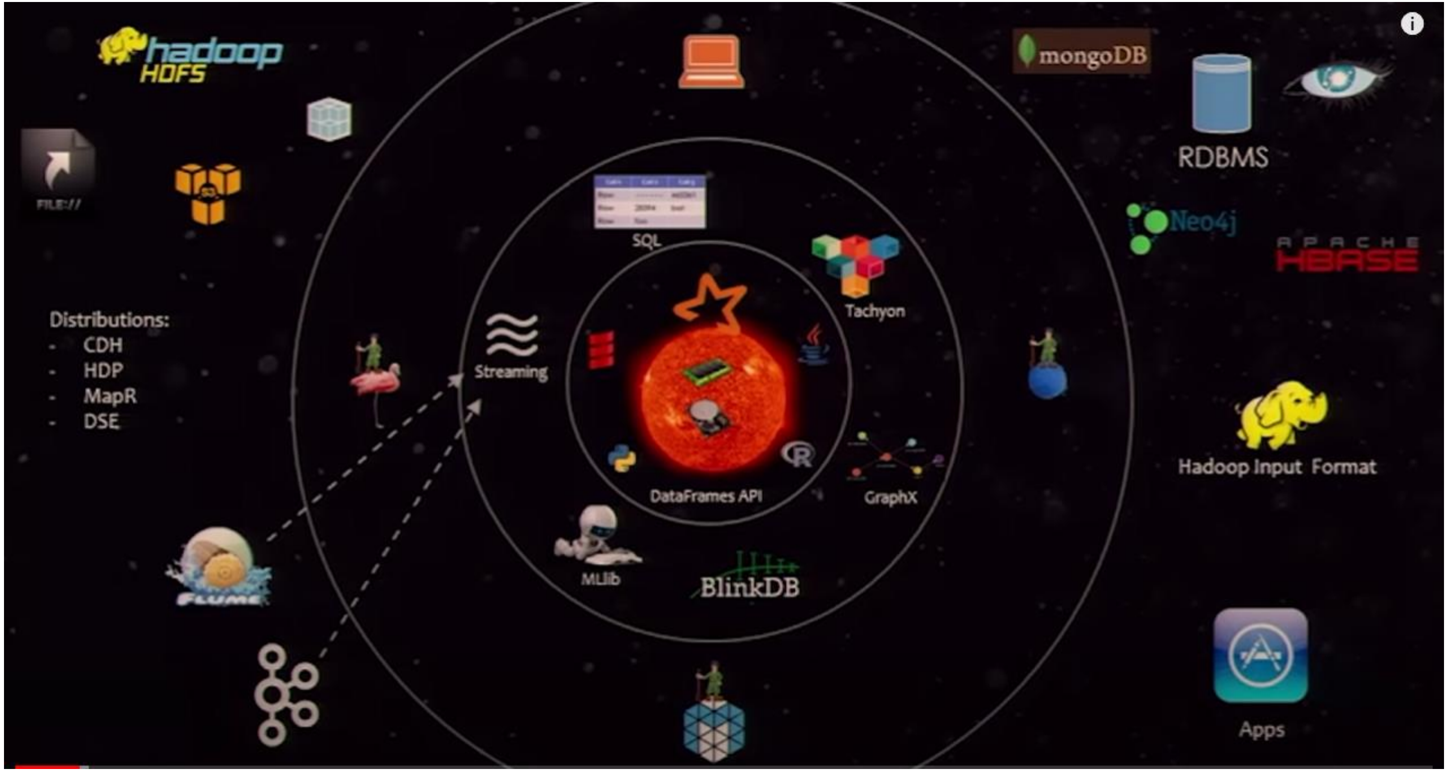
Algorithms:

- Page Rank
- Topic Modeling (LDA)
- Community Detection

Source: ampcamp.berkeley.edu

HORTONWORKS

The Big Picture …



Spark Summit

More information?