Spark

5/30/2017

galvanize

- Define the pros/cons of Spark compared to Hadoop MapReduce
- Describe RDDs by their properties and operations
- Describe the difference betweens transformations and actions and how the affect the DAG
- Introduce Spark SQL and Spark DataFrames

Sources

Shout out to Erich and all of the people at Galvanize who helped put the Spark materials together

https://github.com/ewellinger/spark-talk

Spark

Data science friendly parallel computing

- Fast and general engine for large-scale data processing
- Highly efficient distributed operations
- Supports in-memory computing
- Supports Python, Scala, Java, & R

Apache Hadoop Integration

- Relatively easy integration into Hadoop ecosystem (HDFS)
- Scalability, reliability, and resilience

We can also do machine learning

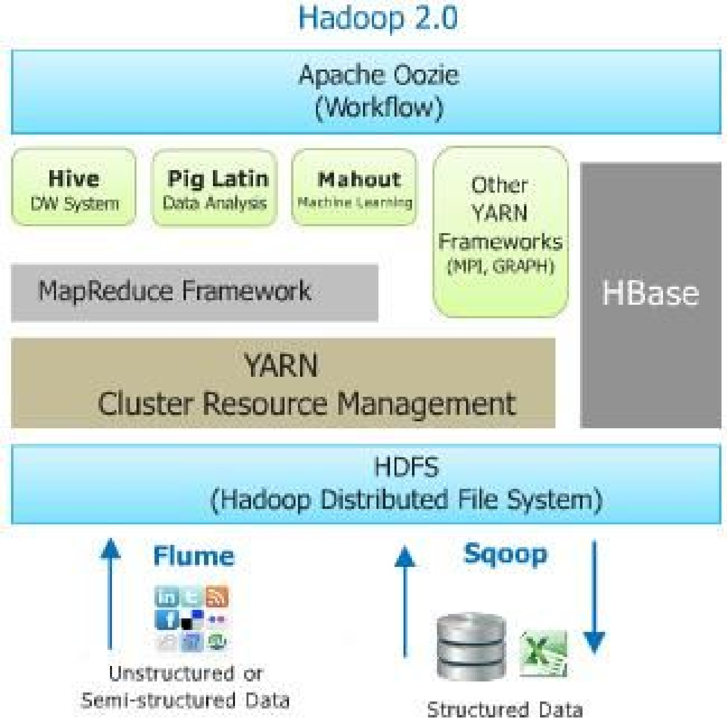
Downsides of Spark

- Under massive development (also a plus…)
- Can be a memory hog if jobs are not tuned well, resulting in frustrating out-of-memory errors
- Not all features are available in every API

Hadoop Ecosystem

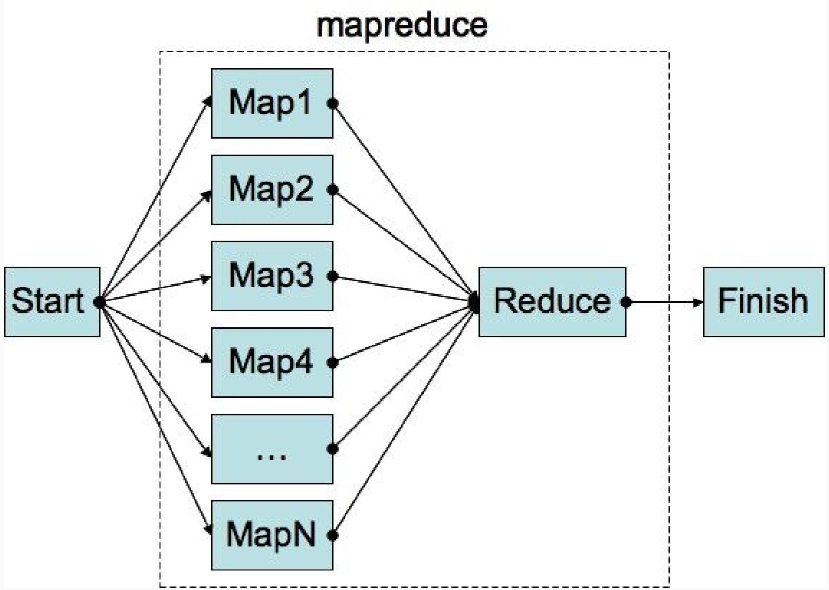
Hadoop consists of an entire ecosystem.

When we are comparing it to Spark we are talking about Spark vs. Hadoop MapReduce.



Hadoop MapReduce

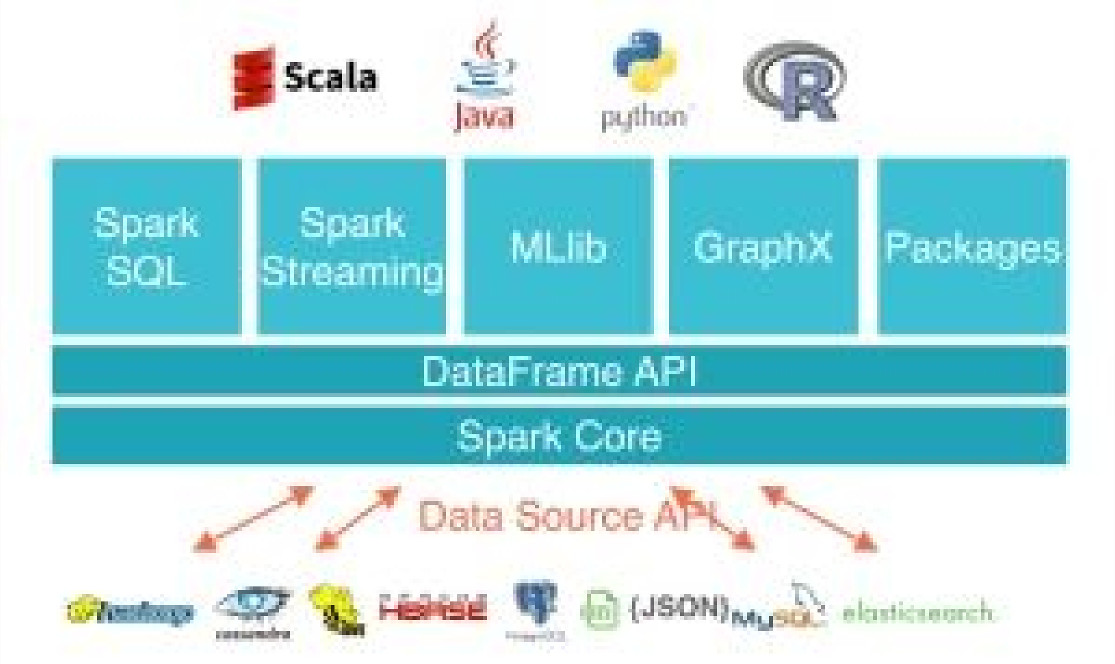
- Hadoop MapReduce is a batch processing engine that exclusively works from disk
  - i.e. MapReduce sequentially reads data from disk, performs an operation on the data, and writes the results back to the cluster



Spark

- Comes with a user-friendly API for Scala, Java, Python, R, and Spark SQL
- Can work with data in-memory leading to lightning fast data operations
    - Up to 100x faster than Hadoop MapReduce when performing in-memory operations (up to 10x faster when operating on disk)
- Capable of integrating with existing Hadoop Ecosystem
    - A Spark application can be run on Hadoop clusters through YARN (yet another resource negotiator)
    - A Spark application can read directly from HDFS

Spark Ecosystem



Resilient Distributed Dataset

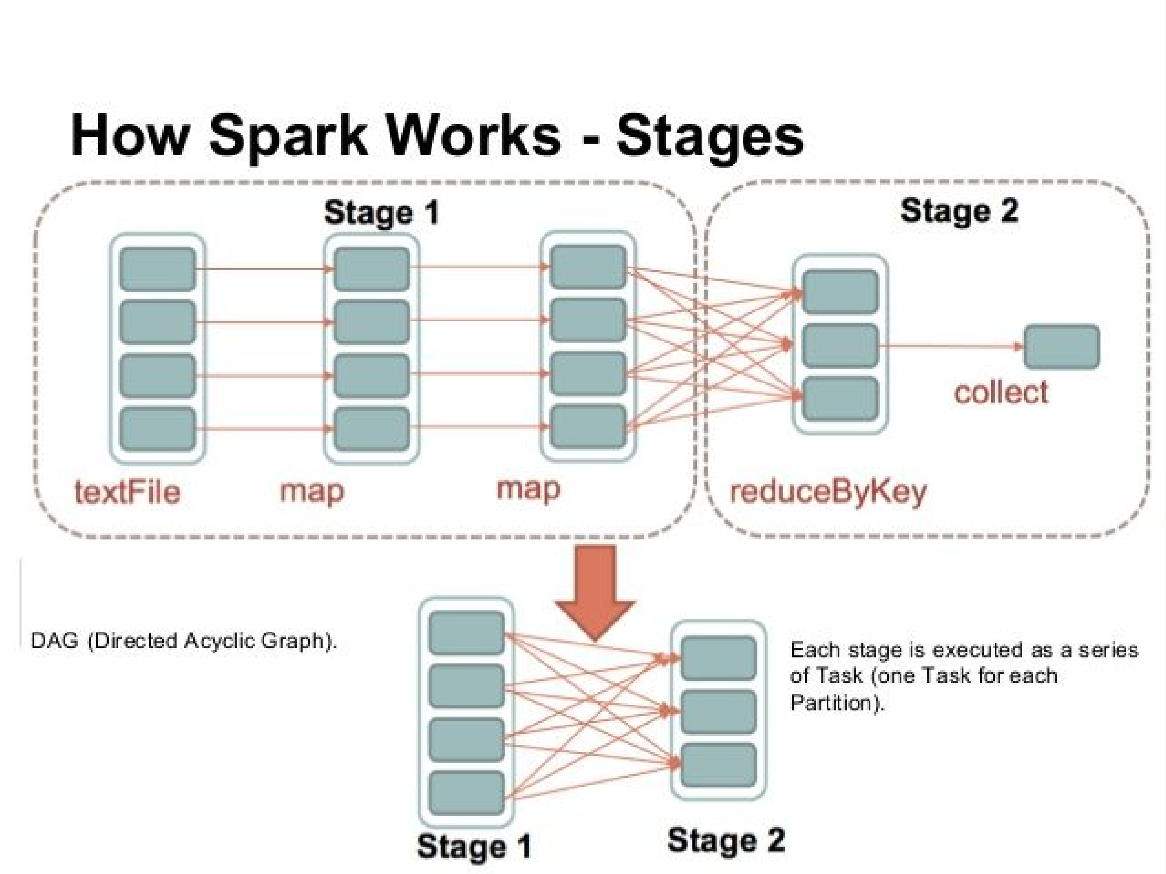
- created from many sources -- HDFS, S3, HBase, JSON, text, etc.
- distributed across the cluster as partitions (chunks of data)
- can recover from errors (node failure, slow process)
- traceability of each partition, can re-run the processing
- immutable : you cannot modify an RDD in place

Transformation and Actions

- Spark is lazy
- Two types of Spark operations
  - We make **transformations** to an RDD to make a new RDD
    - Stores it as step but doesn’t actually do anything
  - **Actions** are where everything really happens
    - We call an action and the transformations take place to return a result

Directed Acyclic Graph

- You construct your sequence of transformations in python.
- Spark functional programming interface builds up a DAG.
- This DAG is sent by the driver for execution to the cluster manager.



How Spark Works - Stages

DAG (Directed Acyclic Graph).

Each stage is executed as a series of Task (one Task for each Partition).

Individual Assignment

Use ---> link = 's3a://dsi-spark-day/airline_data.csv'

Jupyter Notebooks and Spark

Please make sure you do the following to launch your notebook:

- ***DO THIS:*** $ bash ~/scripts/jupyspark.sh
- Instead of this: $ jupyter notebook

Running Spark using Python locally

DO THIS: $ bash ~/scripts/localsparksubmit.sh my_script.py

Instead of this: $ python my_script.py

RDD Demo

galvanize

Spark SQL and DataFrames

- Unlike the traditional Spark RDD API, the Spark SQL module adds additional information about the schema of the data contained in an RDD, thereby allowing extra optimization
- But what is a schema?
  - Schemas are metadata about your data
  - Schema = Table Names + Column Names + Column Types
- What are the Pros of Schemas?
  - Schemas enable queries using SQL and DataFrame syntax
  - Schemas also make your data more structured

Pair Assignment

Use ---> link = ‘s3a://galvanize-ds-bak/transactions.txt’

Spark Structured API Demo

galvanize