Introduction to Spark I

Galvanize

A quick introduction

By way of introduction...

- Conor Murphy
- DSI Cohort 17
- 4 years with data in non-profits
- GalvanizeU faculty member
- Emerging from 8-month deep dive in data engineering

Introduction to Spark I

OBJECTIVES

- **Motivate** the use cases for Spark
- **Describe** the pros/cons of Spark compared to Hadoop MapReduce
- **Create** a SparkContext
- **Load** data using Spark
- **Define** what an RDD is, by its properties and operations
- **Explain** the different between transformations and actions on an RDD
- **Implement** the different transformations through use cases

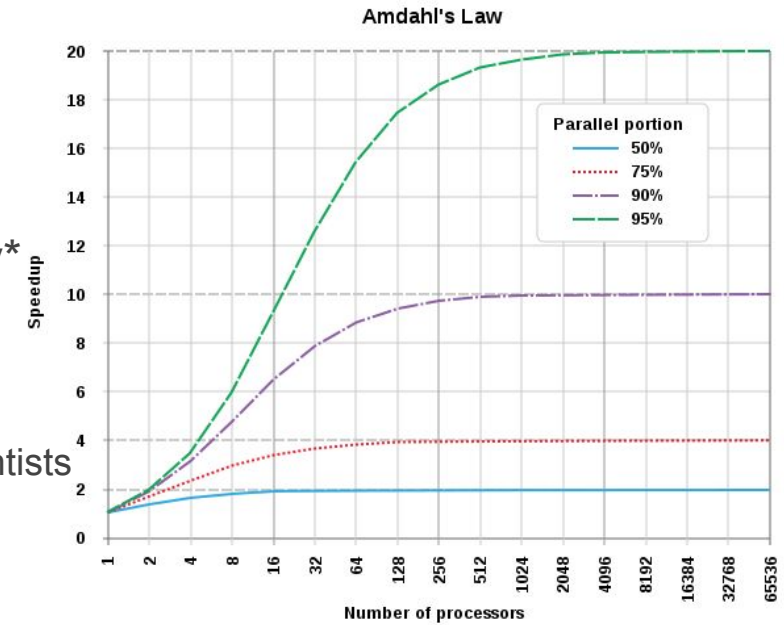
Motivation I: Distributed Thinking

Emphasize **paradigms over tools**

- Don’t learn stats, learn statistical inference
- Don’t learn to code, learn computational thinking

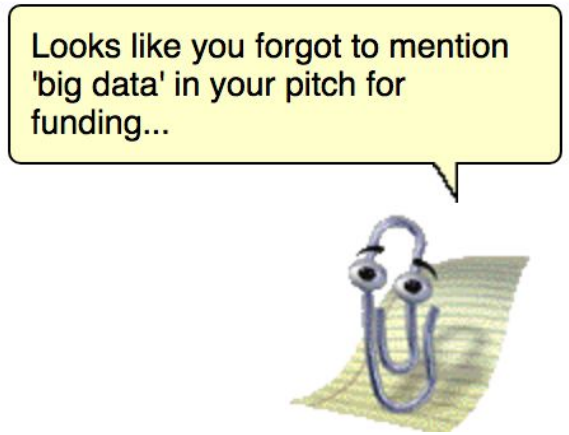
Distributed thinking allows you to scale infinitely*

- Without it, you’re relegated to one-off analysis
- You can only process data that fits on a single machine
- More data engineering mastery is demanded of data scientists
- There are significant costs associated with not scaling



*Well, theoretically at least. Remember Amdahl’s Law

Motivation II: The State of Data

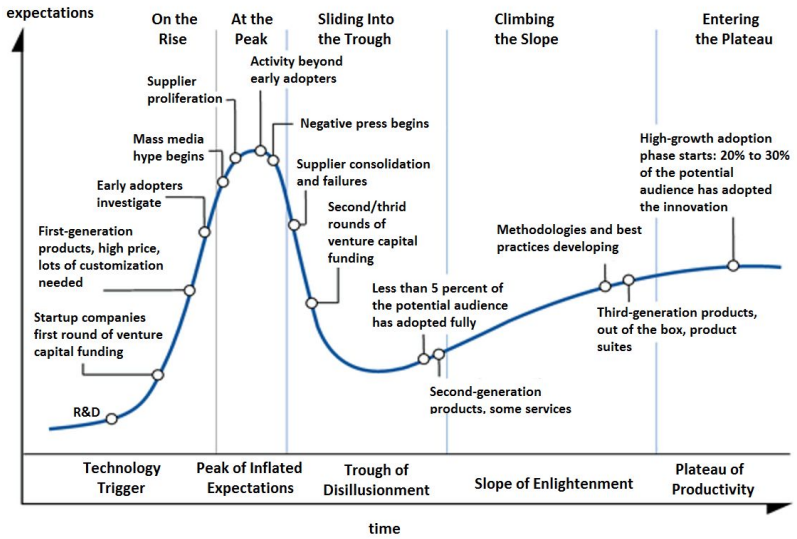


Where's data now?

- Shift from enterprise hardware to commodity hardware
- Memory is getting cheaper (and fast)
- Rise of cloud computing and clusters

What is big data?

- More data than you can fit on any one machine
- Four V's
  - Volume (scale)
  - Variety (type)
  - Velocity (speed)
  - Veracity (reliability)



Source: [The Hype Cycle]

Motivation III: Model Performance

Compare **smart and dumb models**

- Smart models are smarter, right?
- Model performance is a function of n
- Generally, more data is better than a “smarter” model

Consider **the Netflix Prize**

- We have a bias towards hyper-tuned models
- Kaggle competitions reinforce the mega-ensemble model
- These solutions often can’t run in production
- Think about runtime, and tools that can scale

Source: [Scaling to Very Very Large Corpora for Natural Language Disambigu

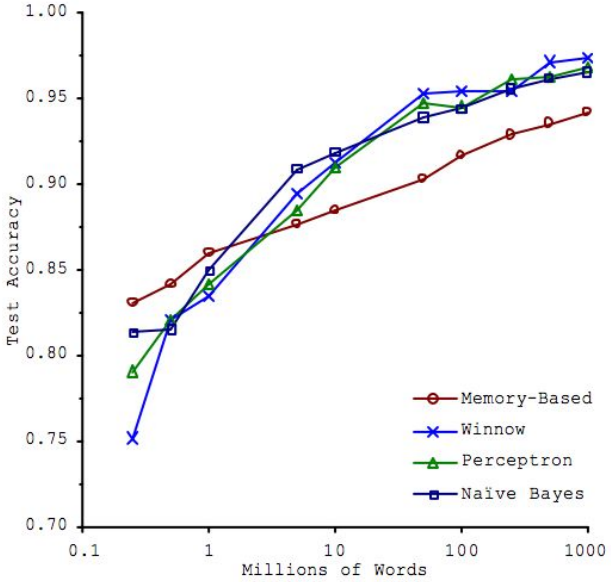


Figure 1. Learning Curves for Confusion Set Disambiguation

The what and why of Spark

At a high level, **what is Spark?**

- “A fast and general engine for large-scale data processing”
- A DAG scheduler
- A top-level Apache project
- In-memory computations (vs Hadoop, which is on disk)
- **Highly** optimized (Catalyst Optimizer, Project Tungsten, …)

Data science friendly parallel computing

- Processing massive data sets
- Highly efficient distributed operations
- More use cases than just MapReduce
- Python and SQL supported natively

Integration with other technologies

Apache Hadoop integration

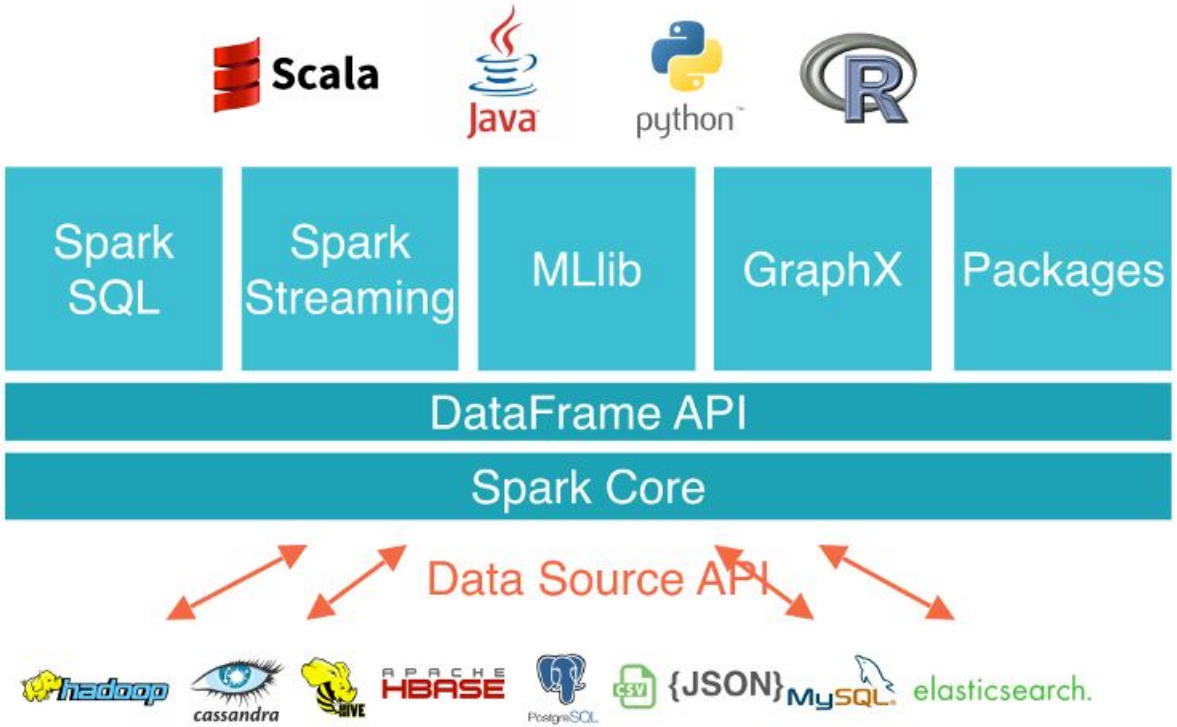
- ~~Seamless~~ relatively easy integration into existing eco-systems
- Spark quickly dominated the market because of HDFS integration
- Scalability, reliability, resilience
- Up to 100x faster than Hadoop

Plays nicely with other systems

- S3 and distributed databases like Cassandra (not its own file management system)
- Can integrate python tools (like grid search)
- New capacity for deep learning pipelines

And… machine learning functions available!

Spark Ecosystem



https://weidongzhou.wordpress.com/2015/09/08/hadoop-hdfs-mapreduce-and-spark-on-big-data/

The canonical big data benchmark

Limitations of MapReduce

- You can't interact with your data live
- It's a restrictive framework
- It's slow (I/O limitations)
- Not designed for OLAP (not for data science)

```
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

[Image Source]

Getting Started: a SparkContext

- Step 1: create a SparkContext object
- This tells Spark how to access a cluster
- To create a SparkContext you can build a SparkConf object that contains information about your application.
- `.setMaster()` indicates the number of threads to use
- This is the python API
    - Execute arbitrary python code in your script (will execute on the master node)
    - Use methods on `sc` to use Spark

```
from pyspark import SparkContext, SparkConf

conf = SparkConf() \
    .setAppName("My App") \
    .setMaster("local[*]")
sc = SparkContext(conf=conf)
```

Source: Spark Docs

Importing Data with Spark

- Toy problems: use `parallelize` to distribute a python data object
- Spark maps onto s3 like HDFS, making it easy to prototype in the AWS ecosystem
- `data` is on the master node, `distData` is distributed across the cluster
- Other options: `wholeTextFiles`, `pickleFile`

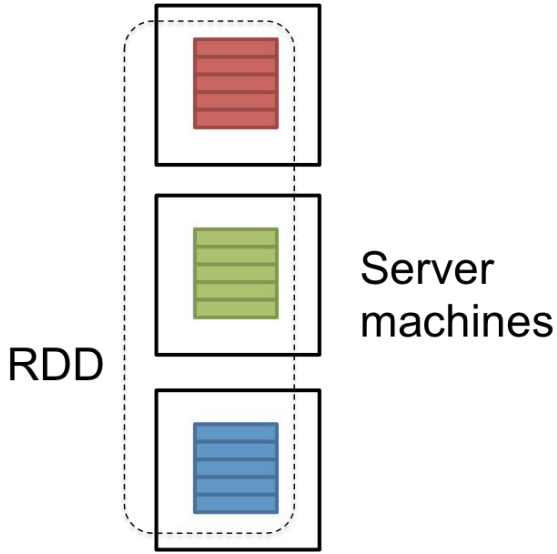
```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)

distData = sc.textFile("s3a://my_bucket/*")
```

Source: Spark Docs

Resilient Distributed Datasets (RDDs)

- **RDDs are the backbone of Spark**
- Created from HDFS, S3, HBase, JSON, text, local
- Distributed across the cluster as partitions (atomic 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
- Java Virtual Machine (JVM) objects



[Image Source]

Transformations

- Transformations return pointers to new RDDs
- They are *lazily evaluated*, waiting until an action is called to perform a computation
- They do map, reduce, and much more

Not an exhaustive list

Method	Type	Category	Description
`.map(func)`	transformation	mapping	Return a new RDD by applying a function to each element of this RDD.
`.flatMap(func)`	transformation	mapping	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
`.filter(func)`	transformation	reduction	Return a new RDD containing only the elements that satisfy a predicate.
`.sample()`	transformation	reduction	Return a sampled subset of this RDD.
`.distinct()`	transformation	reduction	Return a new RDD containing the distinct elements in this RDD.
`.keys()`	transformation	<k,v>	Return an RDD with the keys of each tuple.
`.values()`	transformation	<k,v>	Return an RDD with the values of each tuple.
`.join(rddB)`	transformation	<k,v>	Return an RDD containing all pairs of elements with matching keys in self and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in self and (k, v2) is in other.
`.reduceByKey()`	transformation	<k,v>	Merge the values for each key using an associative and commutative reduce function.
`.groupByKey()`	transformation	<k,v>	Merge the values for each key using non-associative operation, like mean.
`.sortBy(keyfunc)`	transformation	sorting	Sorts this RDD by the given keyfunc.
`.sortByKey()`	transformation	sorting/<k,v>	Sorts this RDD, which is assumed to consist of (key, value) pairs.

Actions

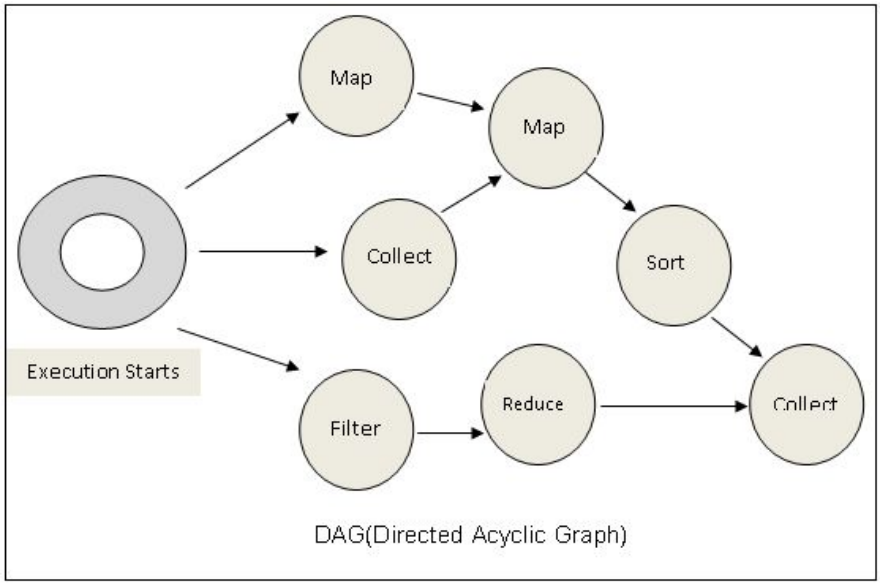
- Actions return values to the driver after running a computation
- Trigger the “execution” of transformations/DAG
- Be careful with `collect`! (return a python list)

Not an exhaustive list

Method	Type	Description
`.collect()`	action	Return a list that contains all of the elements in this RDD. Note that this method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver’s memory.
`.count()`	action	Return the number of elements in this RDD.
`.take(n)`	action	Take the first `n` elements of the RDD.
`.top(n)`	action	Get the top `n` elements from a RDD. It returns the list sorted in descending order.
`.first()`	action	Return the first element in a RDD.
`.sum()`	action	Add up the elements in this RDD.
`.mean()`	action	Compute the mean of this RDD’s elements.
`.stdev()`	action	Compute the standard deviation of this RDD’s elements.

A “Functional” Programming paradigm

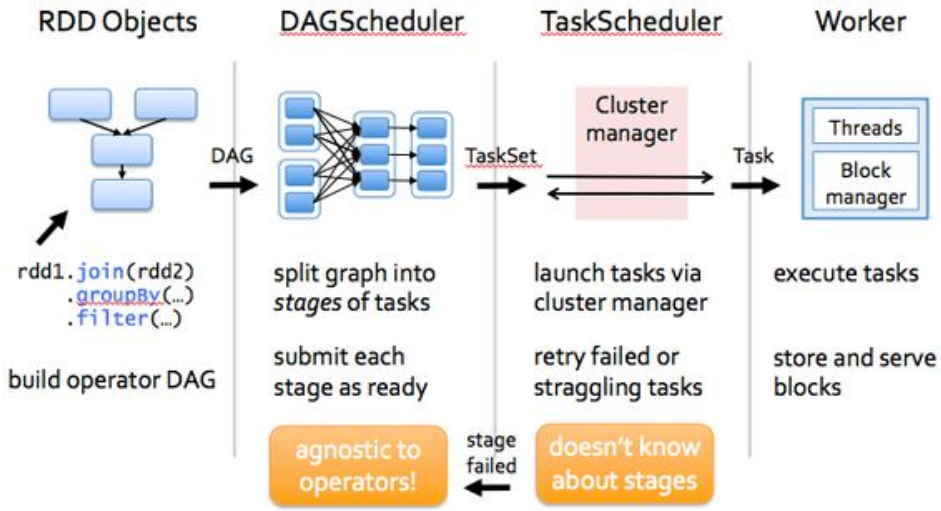
- RDDs are immutable !
  You can **only transform** an existing RDD into another one.
- Spark provides many **transformations functions**.
- Programming = construct a **Directed Acyclic Graph** (DAG).
- **Passed from the client to the master**, who then distributes them to workers, who apply them accross their partitions of the RDD.



[Image Source]

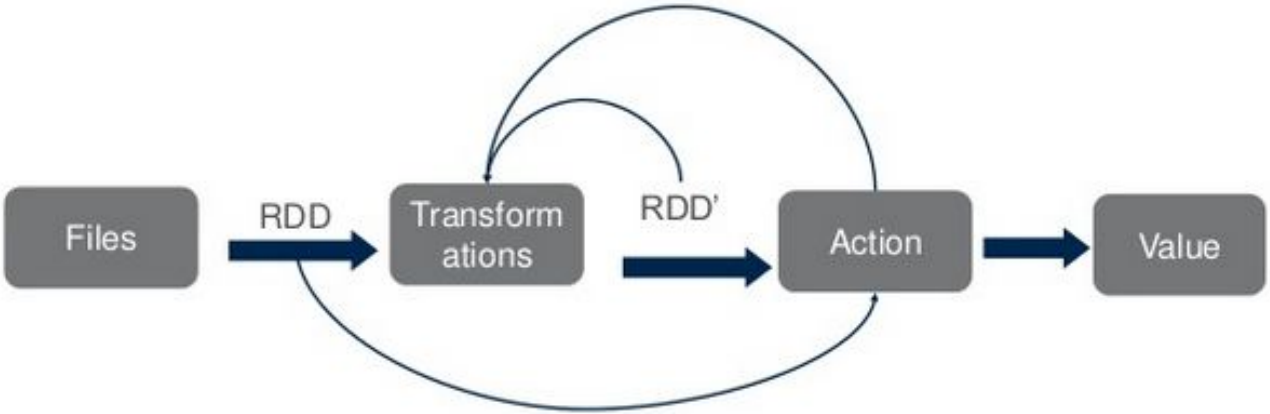
Directed-Acyclic-Graph

- You construct your sequence of transformations in python
- Spark functional programming interface builds up a DAG
- The DAGScheduler avoids **shuffling**, the most expensive operation (I/O)
- This DAG is sent by the driver for execution to the cluster manager



[Image Source]

Operational Spark Workflow



Brainstorming: So, let's suppose you have this thing called an RDD, which is just basically a dataset made of rows and values.

What are all the operations you'd like to do to that RDD ?

Resources

- Spark Documents: Best stop for the most up to date (and versioned) information
- JF's Walkthroughs: Galvanize instructor who made some helpful walkthroughs
- *Learning PySpark*: An up-to-date intro to the python API to Spark. Good treatment of machine learning
- *Learning Spark*: A good introduction to Spark written on Spark 1.X. Parts of it are still relevant for a high level overview, but most of it is outdated
- *High Performance Spark*: More recent publication by the author of *Learning Spark*
- Databrick's explanation of key terms

Questions?