Data Analytics with Apache Spark

MSDS 7330

Jacob Brionez, Damon Resnick, Trace Smith

Overview

Motivation:

- In an era of Big Data, what is the latest software being utilized in the industry to efficiently process and analyze large scale datasets
- Is there a significant advantage to running Spark on top of Relational Databases such as MySQL or Postgres from a standpoint of computation time?

Objective:

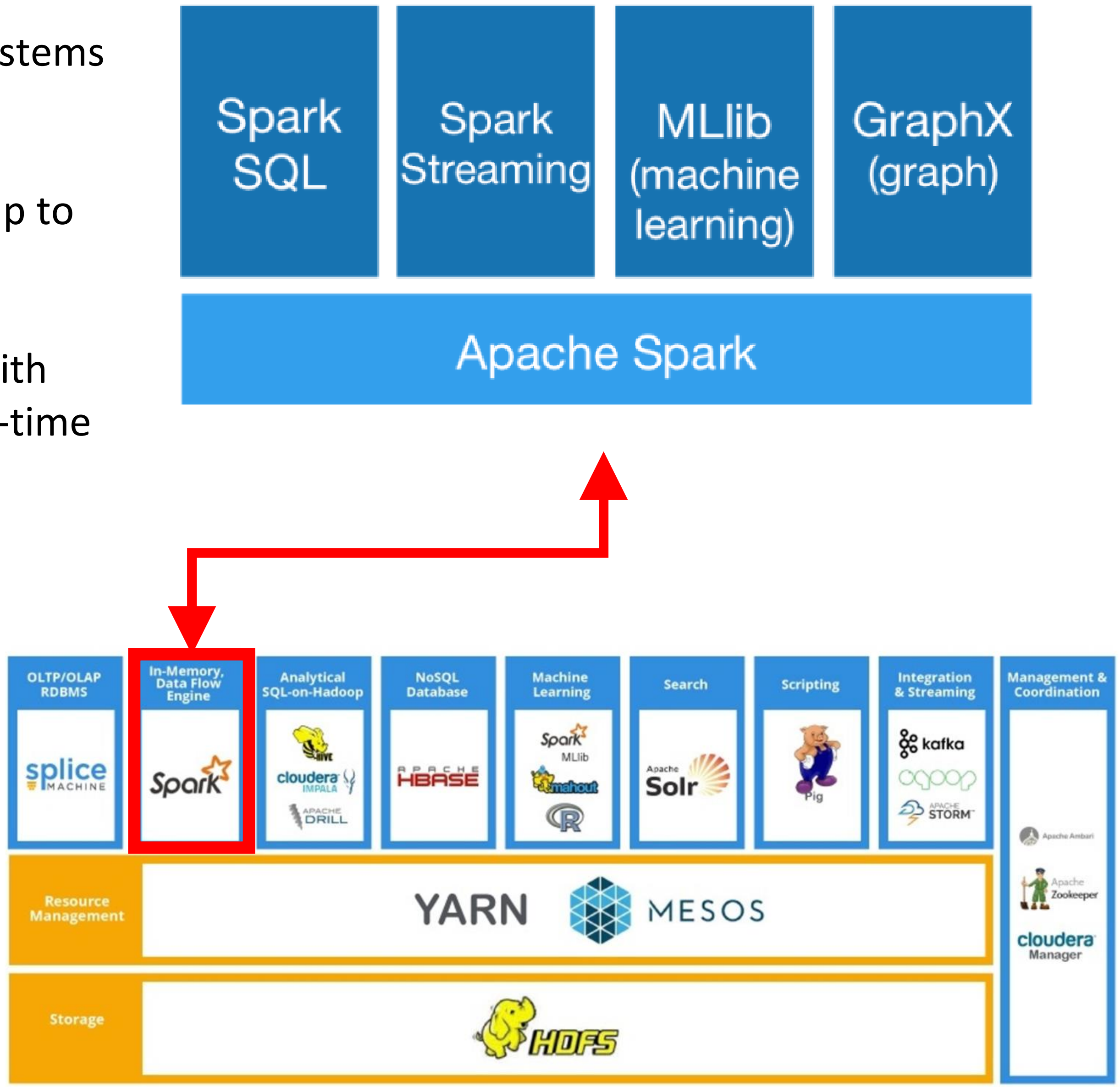
- Learn about the core components of the Apache Spark Ecosystem
- Explore how Spark differs from other big data processing methods such as MapReduce
- Understand at a high-level the functionality of Resilient Distributed Dataset
- Provide a working example of how Spark can be implemented in an analytics workflow (stand alone)

Python Demo

- Leveraging Spark for processing a large dataset (i.e. 7 million rows)
- Interface through Spark using the PySpark API (Python 3.5)
- Provide an in-depth demo in Jupyter Notebook on examples of how to get setup with Spark, querying the data set, creating DataFrames, and creating visualizations using PySpark and other Python libraries

Introduction to Spark

- High performance distributed clustering computing systems for large scale data processing
- Spark enables applications in Hadoop clusters to run up to 100x faster than MapReduce
- Less expensive shuffles in the data processing along with capabilities like in-memory data storage and near real-time processing
- Core APIs in R, SQL, Java, Scala, and Python
- Spark Ecosystem consists of:
  - Spark SQL + DataFrames
  - Spark Streaming
  - MLlib
  - GraphX



SMU

Spark Ecosystem

Spark Streaming:

API used for processing the real-time streaming data (i.e. Twitter)

Spark SQL:

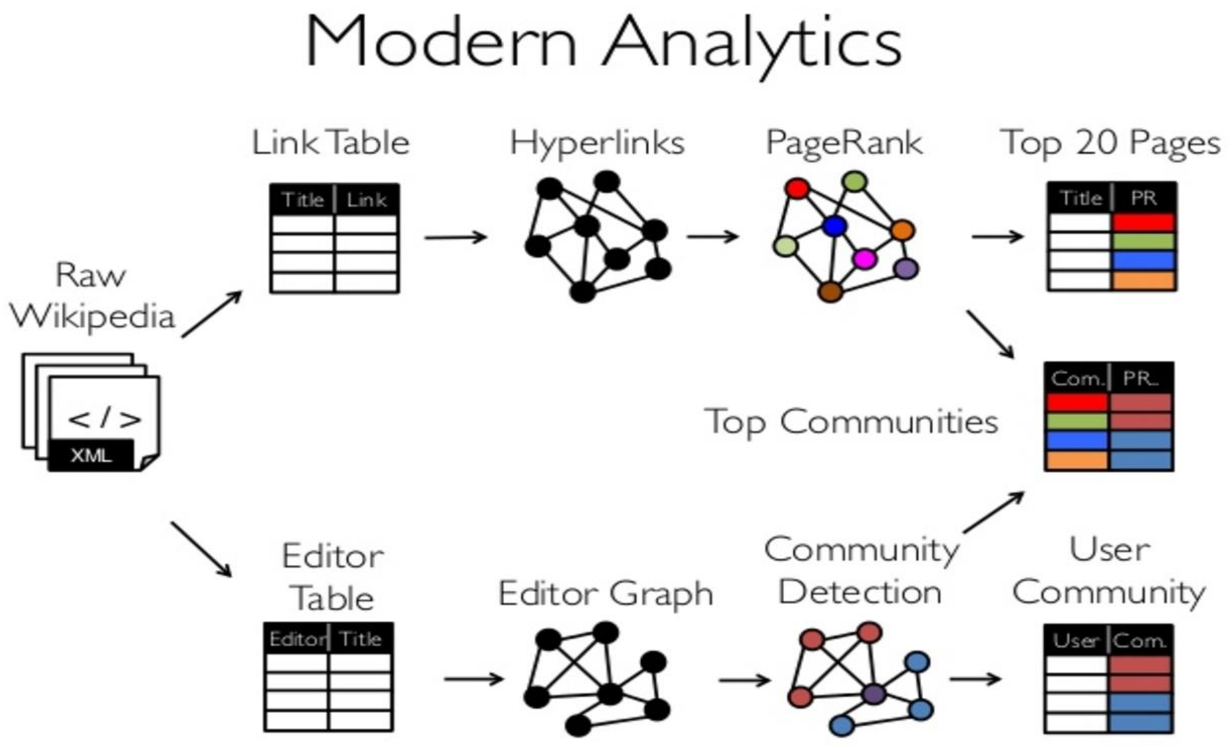
Provides the capability to expose the Spark datasets over JDBC API and allow running the SQL like queries on Spark data (also supports JSON and NoSQL databases)

Spark MLlib:

scalable machine learning library including classification, regression, clustering algorithms

Spark GraphX:

Spark API for graphs and graph-parallel computation (i.e. Page Ranking, Collaborative Filtering, Triangle Counting)

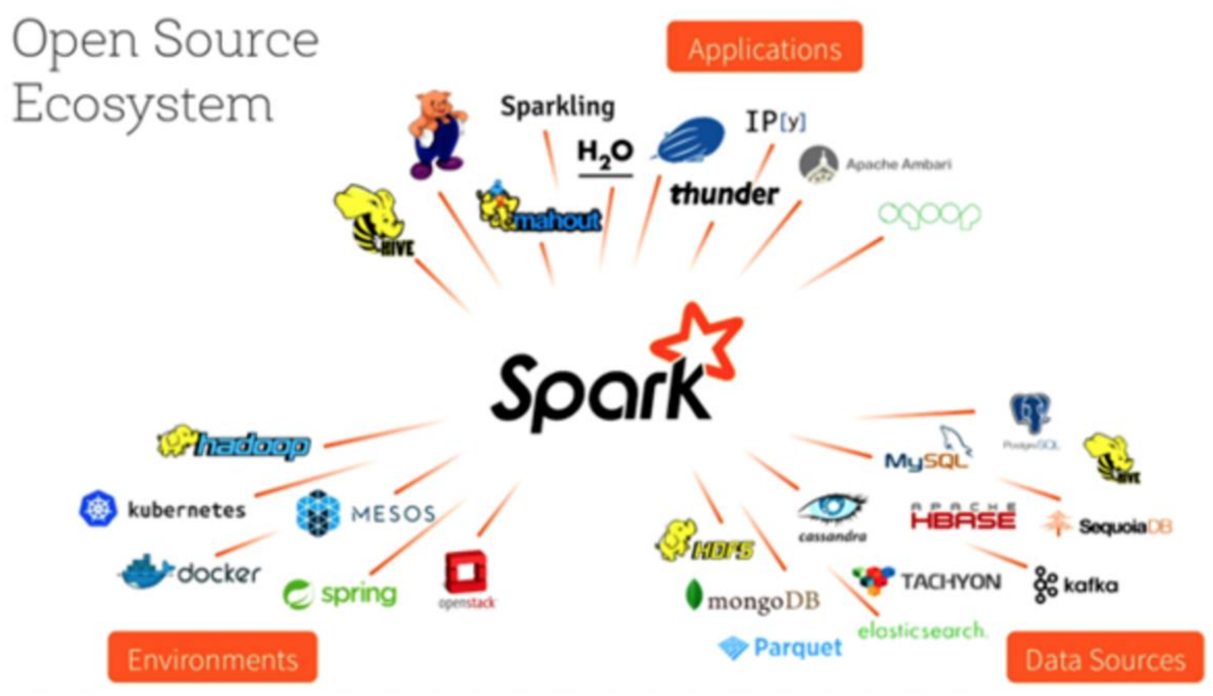


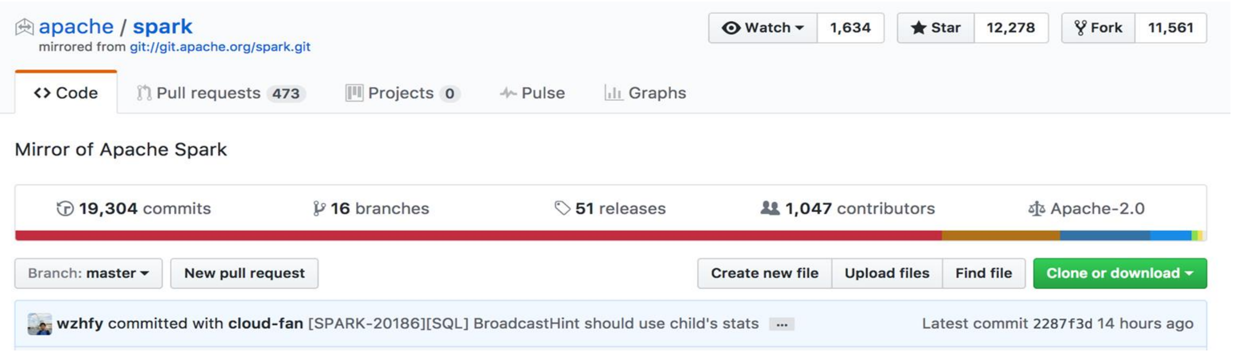
Spark GraphX

https://www.slideshare.net/ankurdave/graphx-ampcamp5

History of Apache Spark

- Started as a class project in 2009 at UC Berkley (Created by: Matei Zaharia)
- General idea was to build a cluster management framework, to support different kind of cluster computing system
- Open sourced in 2010
- Project was donated in 2013, to the Apache Software Foundation and switched license to Apache 2.0
- Databricks set a new world record in large scale sorting using Spark in 2014.
- One of the most active projects in the Apache Software Foundation (~19,000 commits)





SMU®

http://spark.apache.org/
https://github.com/apache/spark

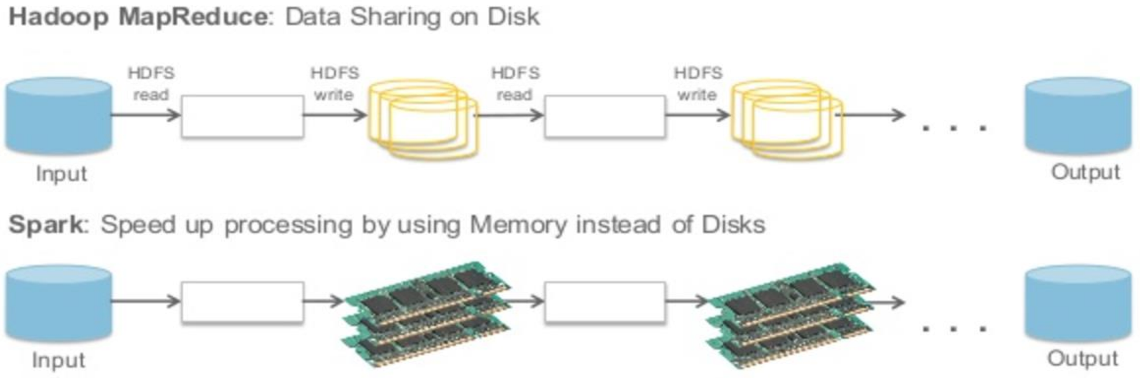
Spark Vs. hadoop

Hadoop

- Distributed data infrastructure -- allocates massive data collections across multiple nodes within a cluster of servers
- Designed to scale up from single servers to thousands of distributed machines
- Consist of two main components:
  - Hadoop File System (HDFS) – storage
  - MapReduce – programming framework (i.e. data sharing on disks)

Spark

- Framework for Big Data analytics
- Data-processing tool that operates on distributed data collections
- Does not include it's own file management system – need to integrate with HDFS or other cloud based platforms
- Developed on the basis of parallel data structure known as Resilient Distributed Datasets (stores data in memory)

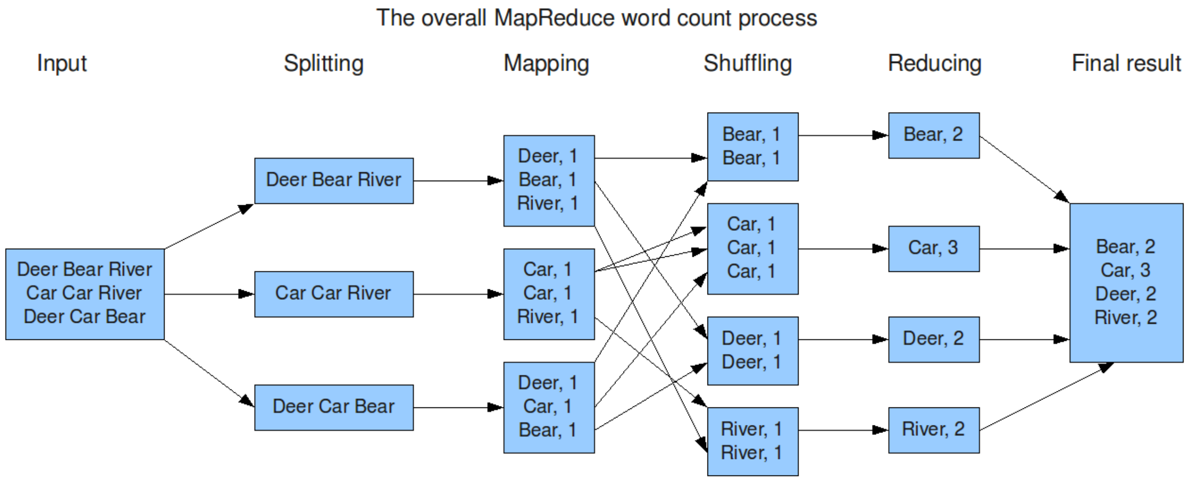


http://spark.apache.org/

MapReduce

- MapReduce is the heart of Hadoop®. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions.*

- The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples.*



http://datascienceguide.github.io/map-reduce

*https://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/

MapReduce vs. Spark

- **Spark tries to keep things in memory, whereas MapReduce keeps shuffling things in and out.**
  - MapReduce can be slow and laborious (i.e. repetition and disk storage)
    - Can be more cumbersome to program in MapReduce
    - MapReduce inserts barriers, and it takes a long time to read/write from disk.
    - Spark eliminates this restriction which makes Spark orders of magnitude faster.

- **It's easier to develop for Spark**.
  - More powerful and expressive in terms of how you give it instructions to crunch data.
  - Spark has a Map and a Reduce function like MapReduce.
    - It also adds others like Filter, Join, and Group-by.
    - This makes it easier to develop for Spark.
    - Spark provides for lots of instructions at a higher level of abstraction than MapReduce
    - Can consist of more than one single map and reduce (i.e. batch processing)

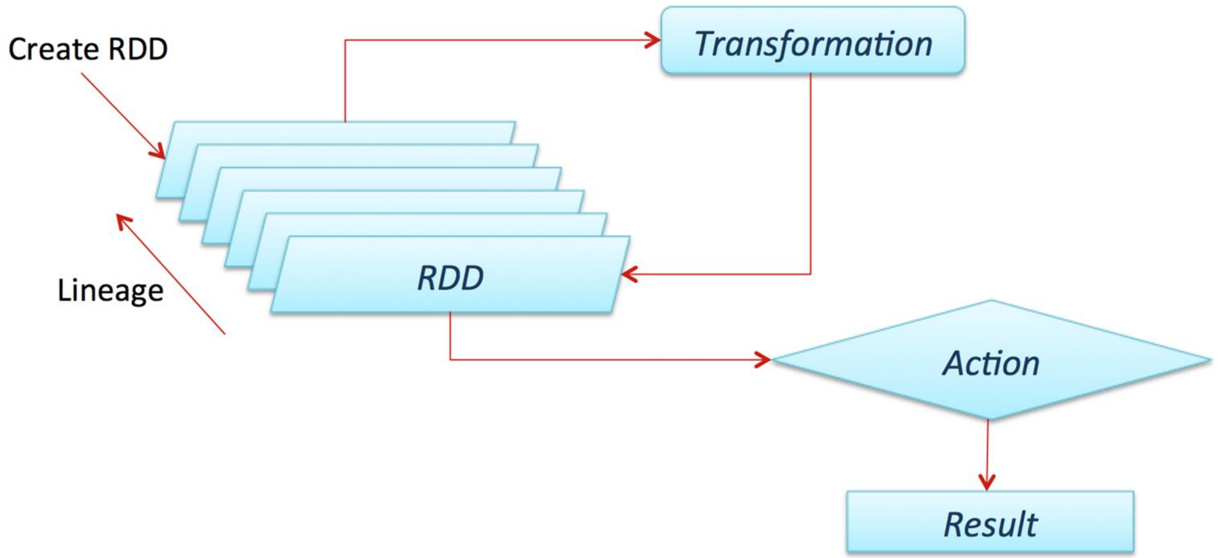
Resilient Distributed Dataset

- **Resilient Distributed Dataset** (**RDD**) is the primary data abstraction in Apache Spark and the core of Spark.
  - RDDs are a collection of immutable objects that can be operated on in parallel
  - Each RDD is split into multiple partitions and distributed over a series of nodes in the cluster

- **Resilient**, i.e. fault-tolerant with the help of [RDD lineage graph] which enables it to recompute missing or damaged partitions due to node failures.

- **Distributed** with data residing on multiple nodes in a [cluster].

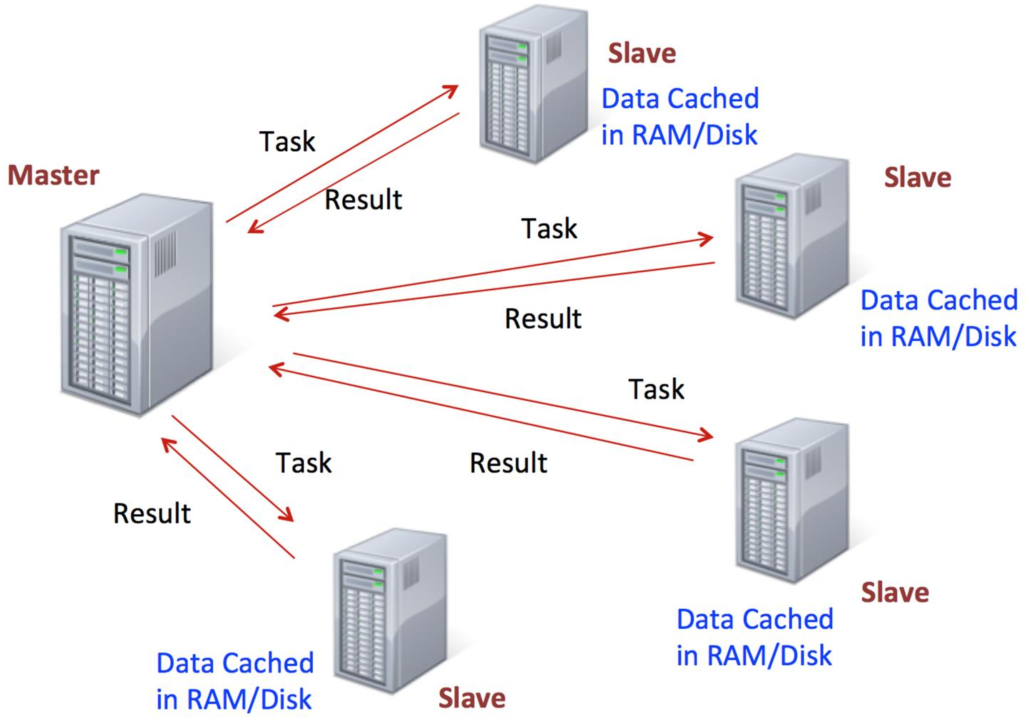
- **Dataset** is a collection of [partitioned data] with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

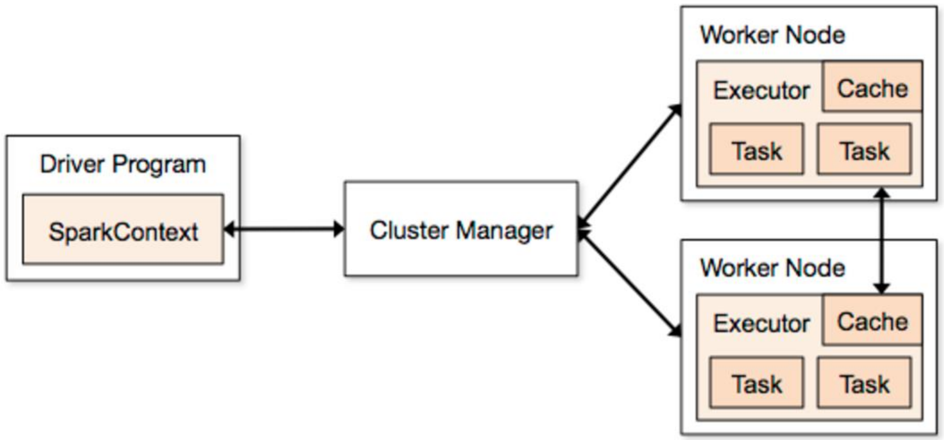


https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.htm
https://dzone.com/storage/rc-covers/6617-thumb.pngl

Spark Architecture

- Similar structure to Hadoop, Spark uses a master/worker architecture
- Cluster Manager allocate resources across applications
  - Standalone, which is a cluster manager included with Spark that makes it easy to set up a cluster
  - Run on YARN (Hadoop 2.0), a distributed container manager
- Master controls the workflow
- Worker launches executors responsible for executing part of the job submitted to the Spark master
- Single application can spawn multiple jobs and the jobs run in parallel





SMU

Getting Started with Spark

Spark python
Fast Data Applications

- Pyspark -- Python API interface to with Apache Spark
- Install Spark here: http://spark.apache.org/downloads
- Unpack .tgz file to root directory
- Modify .bash_profile and set-up the following alias:
    - *export SPARK_PATH=~/spark-2.1.0-bin-hadoop2.7*
    - *export PYSPARK_DRIVER_PYTHON="jupyter" export*
    - *PYSPARK_DRIVER_PYTHON_OPTS="notebook" alias*
    - *snotebook='$SPARK_PATH/bin/pyspark --master local[2]'*
- Launch Pyspark from shell: *bin/pyspark*

Download Apache Spark™

1. Choose a Spark release: 2.1.0 (Dec 28 2016)
2. Choose a package type: Pre-built for Hadoop 2.7 and later
3. Choose a download type: Direct Download
4. Download Spark: spark-2.1.0-bin-hadoop2.7.tgz

Launching IPython Notebook

```
tracesmith spark-2.0.0-bin-hadoop2.6 $ bin/pyspark
[I 20:46:57.753 NotebookApp] [nb_conda_kernels] enabled, 5 kernels found
[I 20:46:58.005 NotebookApp] The port 8888 is already in use, trying another port.
[I 20:46:58.006 NotebookApp] The port 8889 is already in use, trying another port.
[I 20:46:58.007 NotebookApp] The port 8890 is already in use, trying another port.
[I 20:46:58.057 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 20:46:58.057 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named 'nbbrowserpdf'
[I 20:46:58.105 NotebookApp] [nb_anacondacloud] enabled
[I 20:46:58.108 NotebookApp] [nb_conda] enabled
[I 20:46:58.112 NotebookApp] Serving notebooks from local directory: /Users/tracesmith/spark-2.0.0-bin-hadoop2.6
[I 20:46:58.112 NotebookApp] 0 active kernels
[I 20:46:58.112 NotebookApp] The Jupyter Notebook is running at: http://localhost:8891/
[I 20:46:58.112 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

SMU

PySpark Example

- Examine historical flight data from 2008 recorded by the U.S. Bureau of Transportation Statistics
  - 11 Various Attributes and roughly 7 million rows of data!
  - File size 650MB
  - *Source: http://stat-computing.org/dataexpo/2009/the-data.html*

- RDD is a collection of elements that are segmented across multiple nodes in a cluster in which can be processed in parallel
  - Two operations can be performed on an RDD:
    - Transformation: operation applied on a RDD which will creates new RDDs (i.e. map)
    - Action: aggregates output of the transformation by applying some computation (i.e. reduce)

```
try:
    data = sc.textFile("2008.csv")
    count_data = data.count()
    print "Total Number of Rows: {}".format(count_data)
except Exception as e:
    print str(e)
```

```
Total Number of Rows: 7009729
```

Example of performing a transformation and action

```
#transformation
split_data = data.map(lambda line: line.split(','))
#action
header = split_data.first()
print header
```

```
[u'Year', u'Month', u'DayofMonth', u'DayOfWeek', u'DepTime', u'CRSDepTime', u'ArrTime', u'CRSArrTime', u'UniqueCarrie
r', u'FlightNum', u'TailNum', u'ActualElapsedTime', u'CRSElapsedTime', u'AirTime', u'ArrDelay', u'DepDelay', u'Origi
n', u'Dest', u'Distance', u'TaxiIn', u'TaxiOut', u'Cancelled', u'CancellationCode', u'Diverted', u'CarrierDelay', u'W
eatherDelay', u'NASDelay', u'SecurityDelay', u'LateAircraftDelay']
```

SMU

PySpark Example

- After loading in csv file to memory, parse the file by splitting on " , "
- Read data into Spark DataFrame
- In Spark, a DataFrame is essentially equivalent to a table in a relational database or a data frame similar to Pandas (i.e. Python)
- DataFrames can be constructed from other sources such as structured data files, tables in Hive, external databases, or existing RDDs.

Transformation

Action

```
def parse(r):
    try:
        x = Row(Year=int(r[0]), Month=int(r[1]),DayofMonth=int(r[2]),\
            DayOfWeek=int(r[3]),DepTime=int(float(r[4])),CRSDepTime=int(r[5]),\
            ArrTime=int(float(r[6])),CRSArrTime=int(r[7]),UniqueCarrier=r[8],\
            DepDelay=int(float(r[15])),Origin=r[16],Dest=r[17],Distance=int(float(r[18])))
    except:
        x=None
    return x

textRDD = split_data.filter(lambda r: r != header)
rowRDD = textRDD.map(lambda r: parse(r)).filter(lambda r:r != None)
dataframe = sqlContext.createDataFrame(rowRDD)
dataframe.show(5)
```

ArrTime	CRSArrTime	CRSDepTime	DayOfWeek	DayofMonth	DepDelay	DepTime	Dest	Distance	Month	Origin	UniqueCarrier	Year
2211	2225	1955	4	3	8	2003	TPA	810	1	IAD	WN	2008
1002	1000	735	4	3	19	754	TPA	810	1	IAD	WN	2008
804	750	620	4	3	8	628	BWI	515	1	IND	WN	2008
1054	1100	930	4	3	-4	926	BWI	515	1	IND	WN	2008
1959	1925	1755	4	3	34	1829	BWI	515	1	IND	WN	2008

only showing top 5 rows

PySpark Example

- Load in Airport Location Data

```
airport_loc_df = pd.read_csv('airport_location.csv',index_col=0,
        names = ['name', 'city', 'country','faa_code','ICAO','lat',
        'lng','alt','TZone','DST','Tz'], header=0)
airport_loc_df.head()
```

	name	city	country	faa_code	ICAO	lat	lng	alt	TZone	DST	Tz
2	Madang	Madang	Papua New Guinea	MAG	AYMD	-5.207083	145.788700	20	10.0	U	Pacific/Port_Moresby
3	Mount Hagen	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.826789	144.295861	5388	10.0	U	Pacific/Port_Moresby
4	Nadzab	Nadzab	Papua New Guinea	LAE	AYNZ	-6.569828	146.726242	239	10.0	U	Pacific/Port_Moresby
5	Port Moresby Jacksons Intl	Port Moresby	Papua New Guinea	POM	AYPY	-9.443383	147.220050	146	10.0	U	Pacific/Port_Moresby
6	Wewak Intl	Wewak	Papua New Guinea	WWK	AYWK	-3.583828	143.669186	19	10.0	U	Pacific/Port_Moresby

- Spark SQL: module for working with structured data (i.e. DataFrame)

- Can Connect Pyspark directly to a NoSQL database like MongoDB

    - *sqlContext.read.format("com.mongodb.spark.sql").load()*
    - Convert RDD to DataFrame and Dataset
    - MongoRDD class provides helpers to create DataFrames

SQL: Which airport has the most delays?

```
delays = sqlContext.sql("SELECT Origin, count(*) Num_Flights,avg(DepDelay)\
                         Delay FROM dataframe GROUP BY Origin")
```

	Origin	Num_Flights	Delay
0	BGM	699	5.915594
1	PSE	742	0.057951
2	DLG	111	16.495495
3	INL	71	-4.802817
4	MSY	38510	8.891587

PySpark SQL Class:
from pyspark.sql import SQLContext

SMU®

PySpark Example

- Setup visualization of delayed flights across the U.S. in 2008
- First Load in matplotlib toolkit “Basemap” (i.e. plot contours lines for U.S.)

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color = 'white',alpha=0.3)
map.drawstates()
map.shadedrelief()
%matplotlib inline
```

- Join the two DataFrames on “Origin” and “FAA_Code” – Joining Flight Information and Airport Location DataFrames

```
#Join origin_df with airports
df_airports = pd.merge(origin_df,airport_loc_df,left_on = 'Origin',right_on = 'faa_code')
df_airports.head()
```

- Standardize the delay time and store the coordinates of the airports in x,y variables
- Adjust the magnitude of flights (i.e. for visualization rendering purposes)

```
#standardize delay times
def zscore(x):
    return (x-np.average(x))/np.std(x)

#Plot Airport Delay
countrange=max(df_airports['Num_Flights'])-min(df_airports['Num_Flights'])
standarize = (zscore(df_airports['Delay']))
x,y = map(np.asarray(df_airports['lng']),np.asarray(df_airports['lat']))
volume=df_airports['Num_Flights']*4000.0/countrange
```

SMU®

PySpark Example

- Set up coordinates system in Basemap for U.S.

```
fig = plt.figure(figsize=(20,20))
map = Basemap(projection = 'merc',area_thresh = 4000,resolution ='i',
                rsphere=6371200.,llcrnrlon=-130,llcrnrlat=21,urcrnrlon=-62, urcrnrlat=52)
```

- Create scatter plot of latitude and longitude locations of airport
- Size of the data point is based on the magnitude of delayed flight times

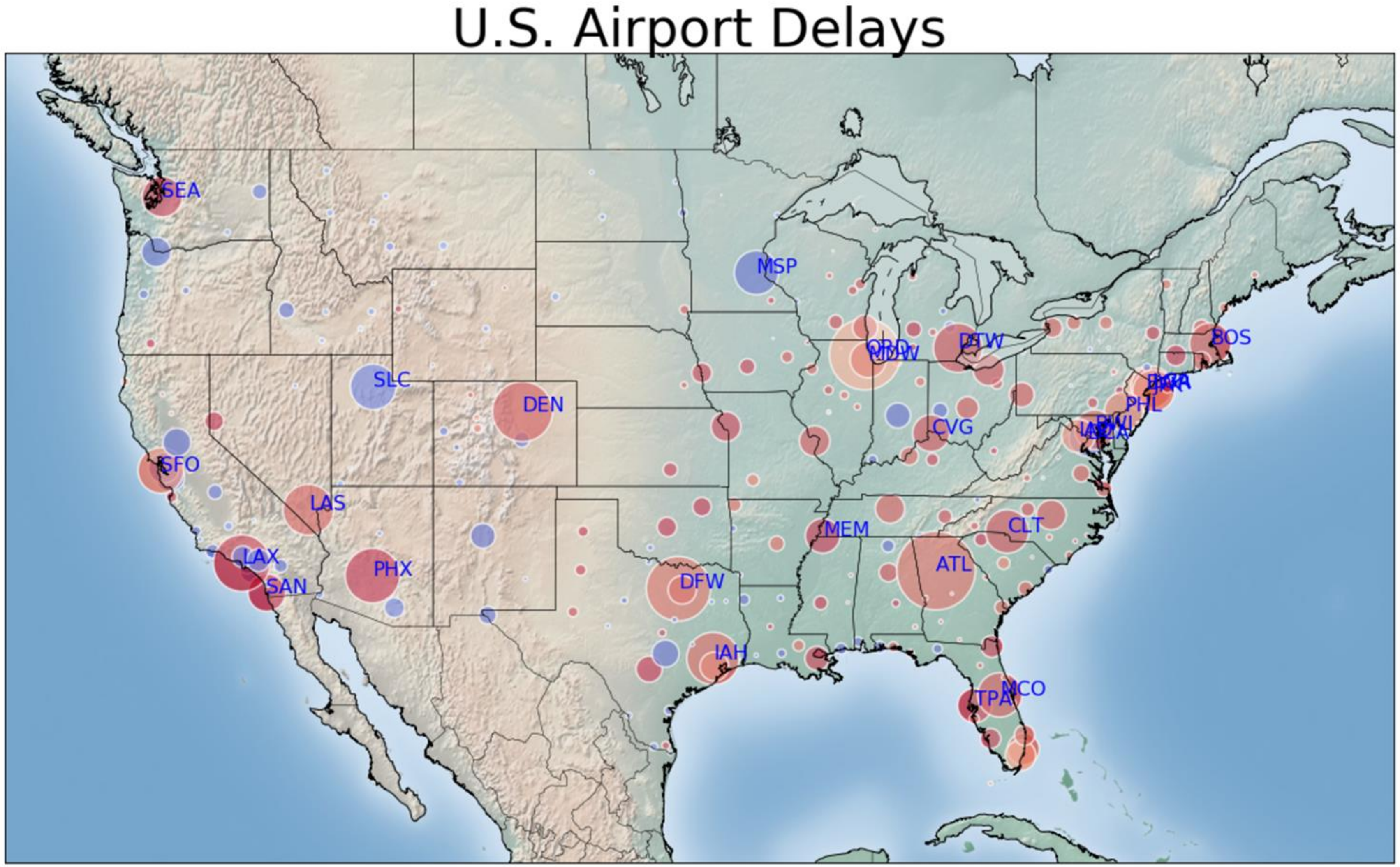
```
#Add color to map:
color = pl.get_cmap('coolwarm')(np.linspace(0.0,1.0,70))
color = np.flipud(color)
map.scatter(x, y,  marker='o', s= volume, linewidths=1.5,
    edgecolors='white', alpha = .7, color=color[(standarize*10)])
```

- Add labels to airports and exclude Hawaii and only add labels if the number of flights for the corresponding airport exceeds 70,000

```
#Add Labels to Map and ignoring Hawaii
df_text=df_airports[(df_airports['Num_Flights']>70000) &
                    (df_airports['faa_code'] != 'HNL')]
xtext,ytext = map(np.asarray(df_text['lng']),
                    np.asarray(df_text['lat']))
txt=np.asarray(df_text['faa_code'])
zp=zip(xtext,ytext,txt)
for row in zp:
    plt.text(row[0],row[1],row[2], fontsize=16, color='blue',)
plt.title("U.S. Airport Delays", fontsize = 42)
plt.show()
```

SMU

PySpark Example



References

http://datascienceguide.github.io/map-reduce

https://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.html

Demo: IBM Data Science Workbench -- Spark Tutorial

https://datascientistworkbench.com/

SMU®