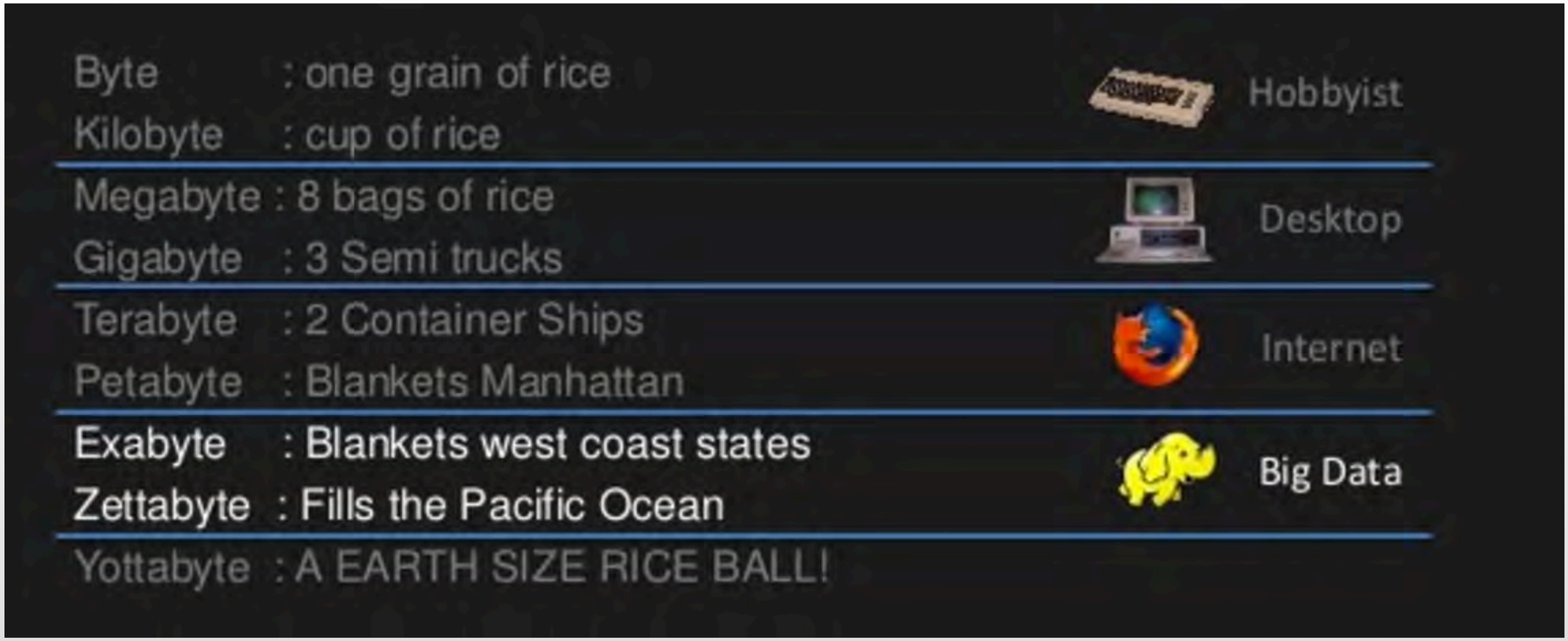
Big Data Analytics with Hadoop and Spark at OSC

04/13/2017
OSC workshop

Shameema Oottikkal
Data Application Engineer
Ohio SuperComputer Center
email:soottikkal@osc.edu

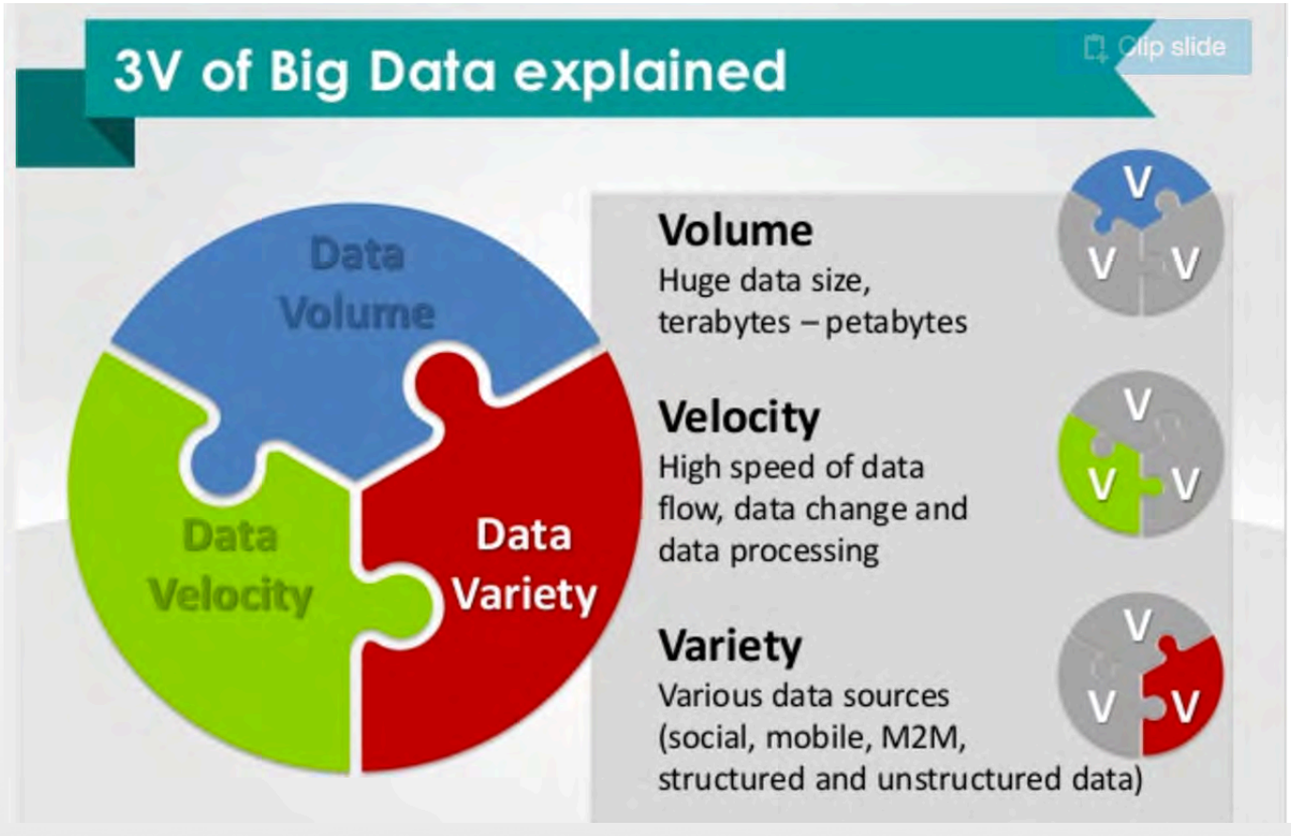
What is Big Data

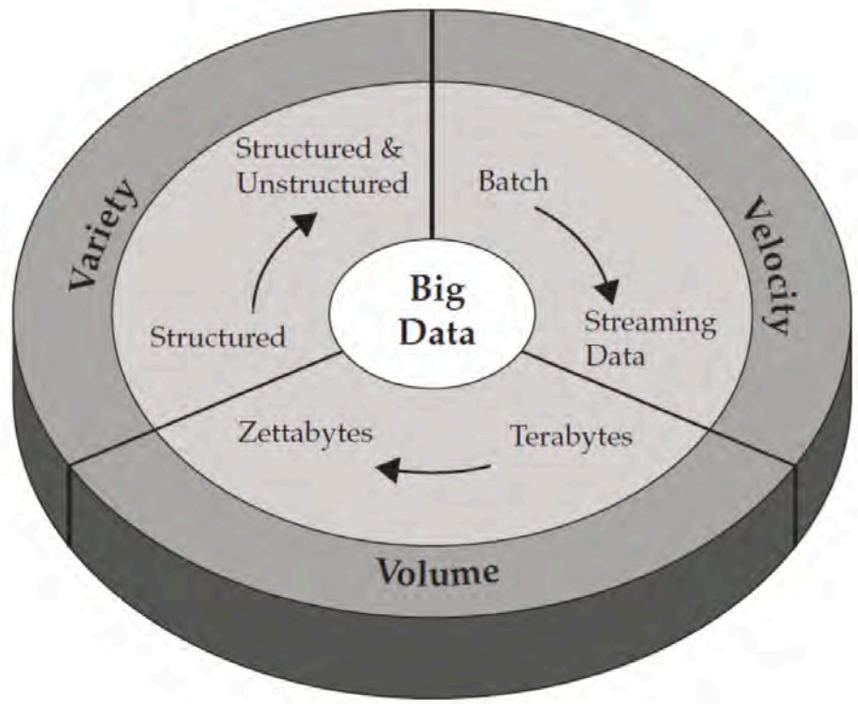
Big data is an evolving term that describes any voluminous amount of structured and unstructured data that has the potential to be mined for information.



Ref: http://www.slideshare.net/dwellman/what-is-big-data-24401517/3

The 3V of Big Data





- Key enablers for the growth of “Big Data” are:
  - Increase of storage capacities
  - Increase of processing power
  - Availability of data

Data Analytical Tools

	Examples	Characteristics	Typical tools	Analytical methods
Small Data (megabytes)	Sales records, Customers database (small and medium companies)	Hundreds – thousands of records	Personal computer, Excel, R, other basic statistics software	Simple statistics
Large Data (gigabytes-terabytes)	Customer databases (big companies)	Millions of records, mostly structured data	Server workstation computer, Relational database systems, data warehouses	Advanced statistics, business intelligence, data mining,
Big Data (terabytes – petabytes)	Customer interactions (social media, mobile), multimedia (video, images, free text), location-based data, RFIM	Over millions of records, distributed, unstructured	Cloud, data centers, Distributed databases, NoSQL, Hadoop	MapReduce, Distributed File Systems

Supercomputers at OSC

	Owens (2016)	Ruby (2014)	Oakley (2012)
Theoretical Performance	~750 TF	~144 TF	~154 TF
# Nodes	~820	240	692
# CPU Cores	~23,500	4800	8304
Total Memory	~120 TB	~15.3 TB	~33.4 TB
Memory per Core	>5 GB	3.2 GB	4 GB
Interconnect	EDR IB	FDR/EN IB	QDR IB

Storage

Home Directory Space

900 TB usable (Disk) (Allocated to each user, 500 GB quota limit)

Scratch – DDN GPFS

1 PB with 40-50 GB/s peak performance

Project – DDN GPFS

3.4 PB

Data Analytics@OSC

Python: A popular general-purpose, high-level programming language with numerous mathematical and scientific packages available for data analytics.

R: A programming language for statistical and machine learning applications with very strong graphical capabilities.

MATLAB:A full featured data analysis toolkit with many advanced algorithms readily available.

Spark and Hadoop: Frameworks for running map reduce algorithms

Intel Compilers: Compilers for generating optimized code for Intel CPUs.

Intel MKL: The Math Kernel Library provides optimized subroutines for common computation tasks such as matrix-matrix calculations.

Statistical software: Octave, Stata, FFTW, ScaLAPACK, MINPACK, sprng2

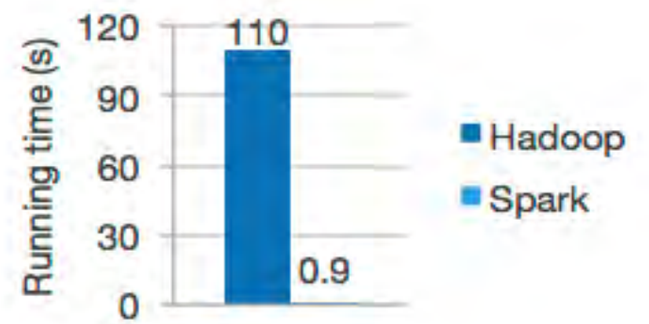
Apache Spark

Apache Spark is an open source cluster computing framework originally developed in the AMPLab at University of California, Berkeley but was later donated to the Apache Software Foundation where it remains today. In contrast to Hadoop's disk-based analytics paradigm, Spark has multi-stage in-memory analytics.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

Ease of Use

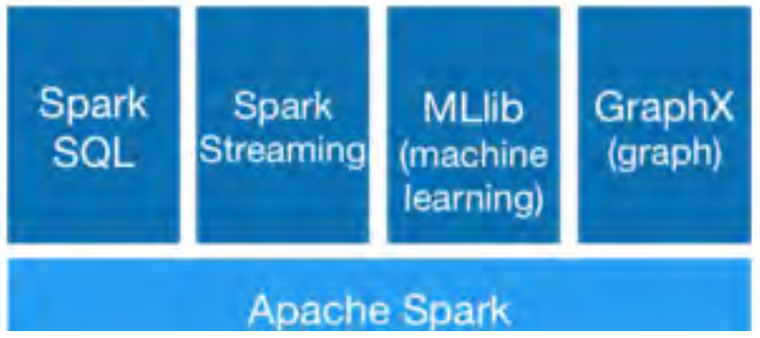
Write applications quickly in Java, Scala, Python, R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python and R shells.

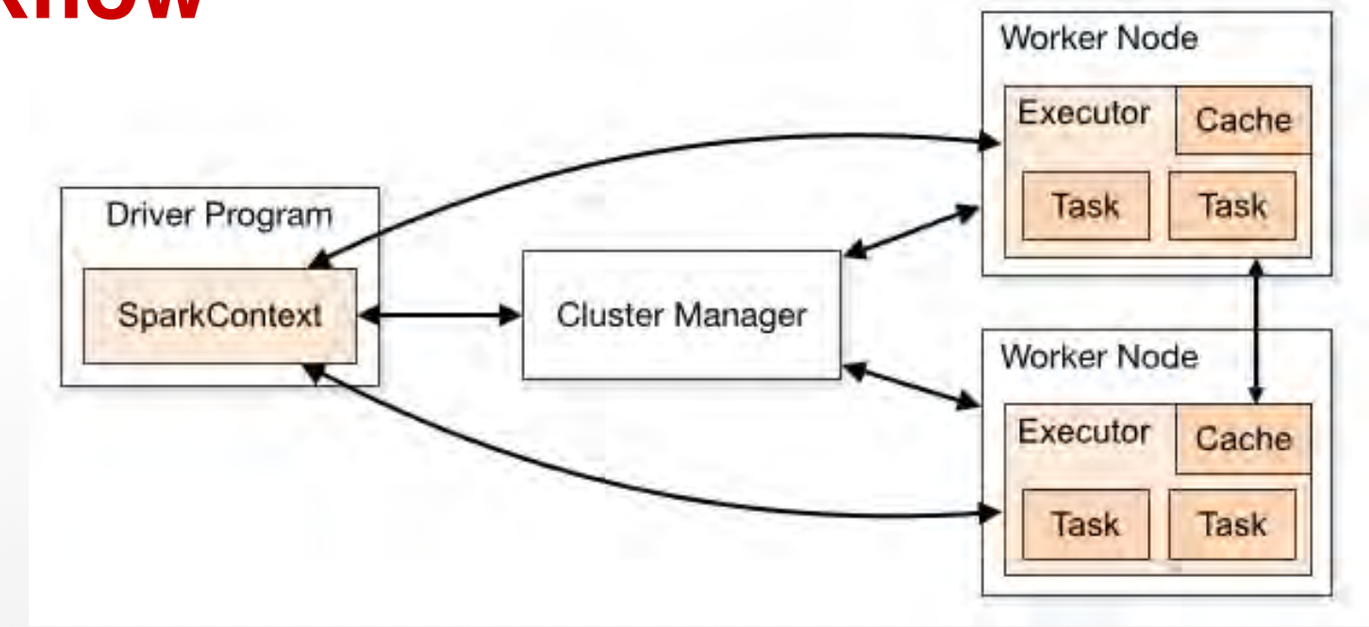
Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



Spark workflow



Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program).

Requires cluster managers which allocate resources across applications.

Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.

Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.

RDD- Resilient Distributed Datasets

RDD (Resilient Distributed Dataset) is the main logical data unit in Spark. They are

- Distributed and partitioned
- Stored in memory
- Immutable
- Partitions recomputed on failure

RDD- Transformations and Actions

Transformations are executed on demand. That means they are computed lazily. Eg: filter, join, sort

Actions return final results of RDD computations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system. Eg: collect(), count(), take()

RDD Operations

Transformations	Actions
map(func)	take(N)
flatMap(func)	count()
filter(func)	collect()
groupByKey()	reduce(func)
reduceByKey(func)	takeOrdered(N)
mapValues(func)	top(N)
...	...

Interactive Analysis with the Spark Shell

./bin/pyspark # Opens SparkContext

1. Create a RDD

```
>>> data = sc.textFile("README.md")
```

2. Transformation of RDD

```
>>>linesWithSpark = data.filter(lambda line: "Spark" in line)
```

3. Action on RDD

```
>>> linesWithSpark.count()  # Number of items in this RDD
126
```

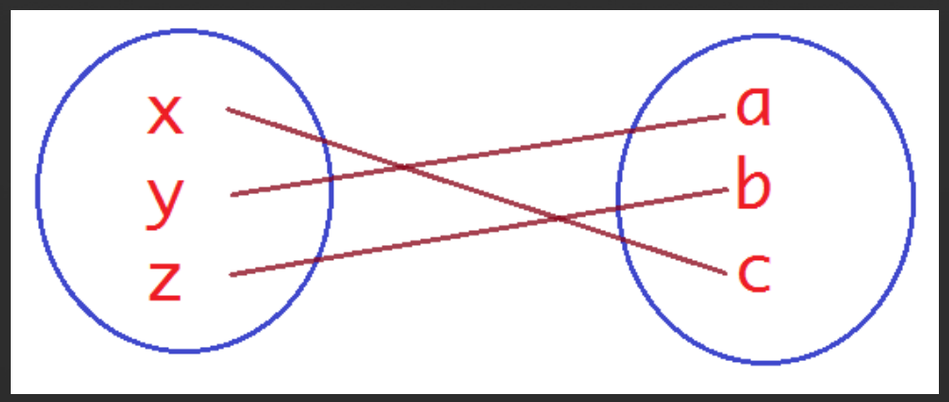
4. Combining Transformation and Actions

```
>>> data.filter(lambda line: "Spark" in line).count() # How many lines contain "Spark"?
15
```

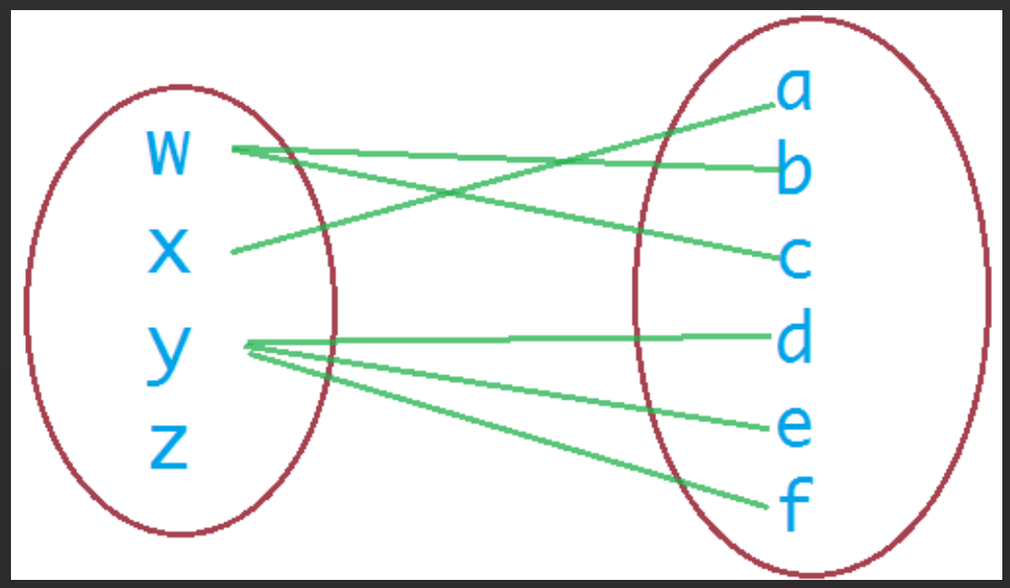
Word count Example

Map: One element in input gets mapped to only one element in output.
Flatmap: One element in input maps to zero or more elements in the output.

Map



Flatmap




```
>>>wordCounts = data.flatMap(lambda line: line.split()).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a+b)

>>> wordCounts.collect()
[(u'and', 9), (u'A', 1), (u'webpage', 1), (u'README', 1), (u'Note', 1), (u'"local"', 1), (u'variable', 1), ...]
```

Spark documentation at OSC

https://www.osc.edu/resources/available_software/software_list/spark_documentation

Availability & Restrictions

Spark is available to all OSC users without restriction.

The following versions of Spark are available on OSC systems:

VERSION	OAKLEY	OWENS
1.5.2	X	
1.6.1	X	
2.0.0*	X	X

NOTE: * means it is the default version.

Set-up

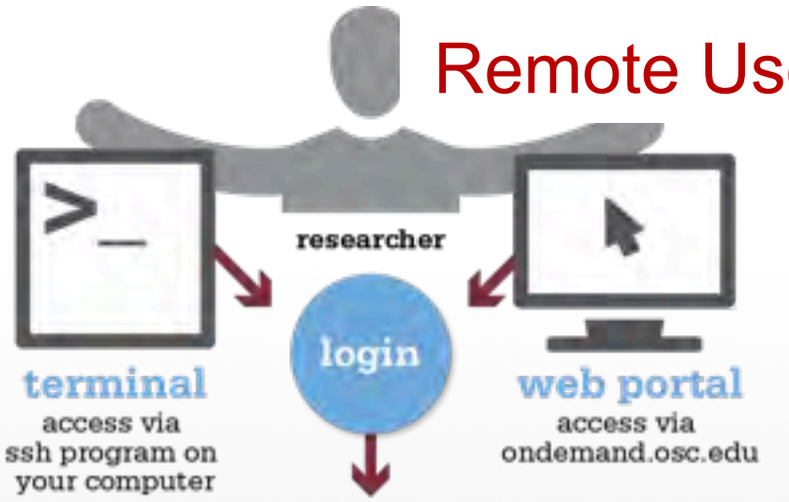
In order to configure your environment for the usage of Spark, run the following command:

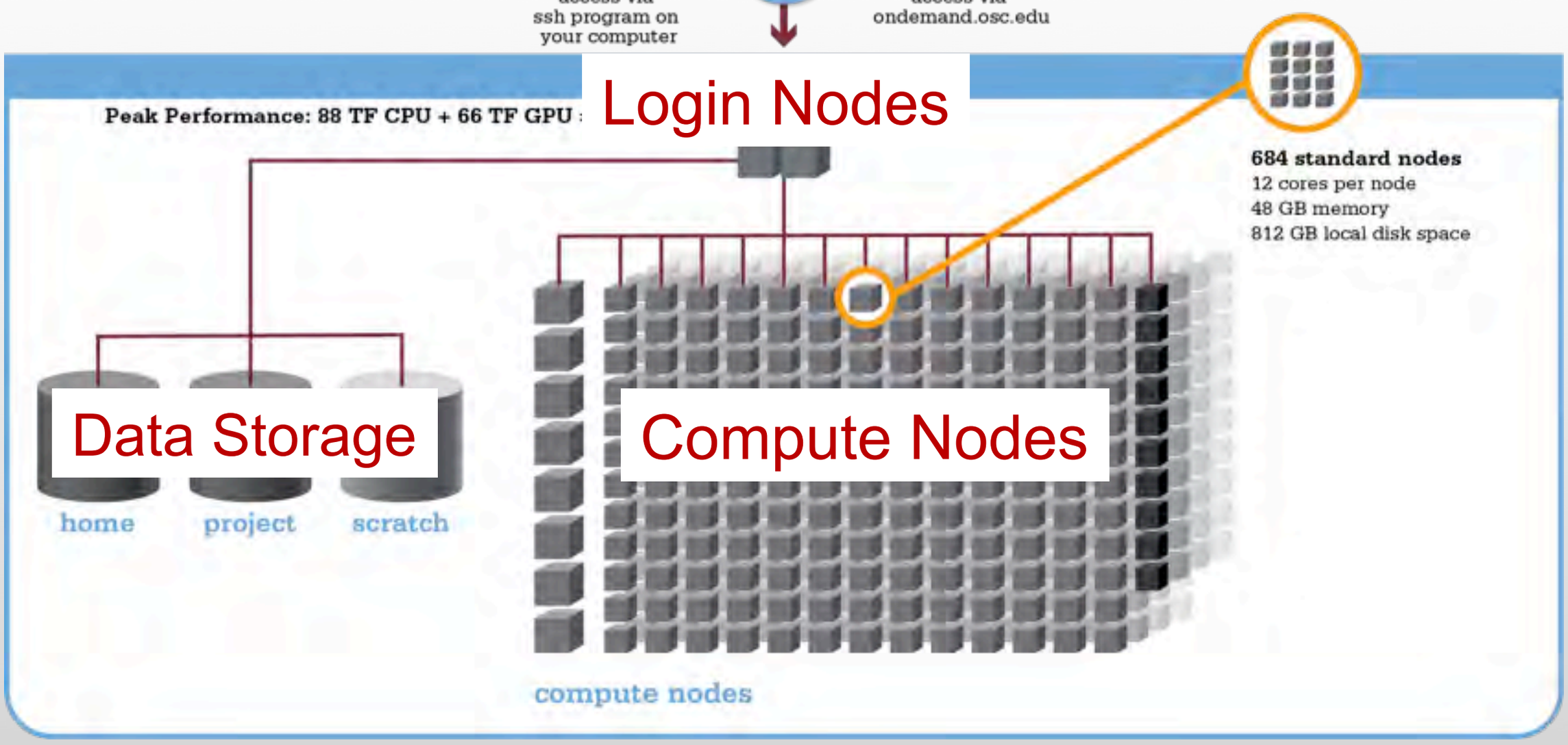
```
module load spark
```

In order to access a particular version of Spark, run the following command

```
module load spark/2.0.0
```

Structure of a Supercomputer





Using Spark

In order to run Spark in batch, reference the example batch script below. This script requests 6 node on the Oakley cluster for 1 hour of walltime. The script will submit the pyspark script called test.py using pbs-spark-submit command into the PBS queue.

```
#PBS -N Spark-example

#PBS -l nodes=6:ppn=12

#PBS -l walltime=01:00:00

module load spark

cd $PBS_O_WORKDIR

cp test.py $TMPDIR

cd $TMPDIR

pbs-spark-submit test.py  > test.log

cp * $PBS_O_WORKDIR
```

Running Spark interactively in batch

To run Spark interactively, but in batch on Oakley please run the following command,

```
qsub -I -l nodes=2:ppn=12 -l walltime=01:00:00
```

When your interactive shell is ready, please launch spark by running commands like

```
pyspark or sparkR
```

```
[soottikkal@owens-login01 IPOIB]$ pyspark
Python 2.7.5 (default, Oct 11 2015, 17:47:16)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
17/02/23 10:16:30 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Python version 2.7.5 (default, Oct 11 2015 17:47:16)
SparkSession available as 'spark'.
>>>
```

Running Spark using PBS script

1. Create an App in python: stati.py

```
from pyspark import SparkContext
import urllib
f = urllib.urlretrieve ("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data.gz","kddcup.data.gz")

data_file = "./kddcup.data.gz"
sc = SparkContext(appName="Stati")
raw_data = sc.textFile(data_file)

import numpy as np

def parse_interaction(line):
    line_split = line.split(",")
    symbolic_indexes = [1,2,3,41]
    clean_line_split=[item for i, item in enumerate(line_split) if i not in symbolic_indexes]
    return np.array([float(x) for x in clean_line_split])

vector_data=raw_data.map(parse_interaction)


from pyspark.mllib.stat import Statistics
from math import sqrt

summary = Statistics.colStats(vector_data)


print ("Duration Statistics:")
print (" Mean %f" % (round(summary.mean()[0],3)))
print ("St. deviation : %f"%(round(sqrt(summary.variance()[0]),3)))
print (" Max value: %f"%(round(summary.max()[0],3)))
print (" Min value: %f"%(round(summary.min()[0],3)))
```

2. Create a PBS script: stati.pbs

```
#PBS -N spark-statistics
#PBS -l nodes=18:ppn=28
#PBS -l walltime=00:10:00
module load spark/2.0.0
cp stati.py $TMPDIR
cd $TMPDIR
pbs-spark-submit stati.py > stati.log
cp *  $PBS_O_WORKDIR
```

3. Run Spark job

```
qsub stati.pbs
```

4. Output: stati.log

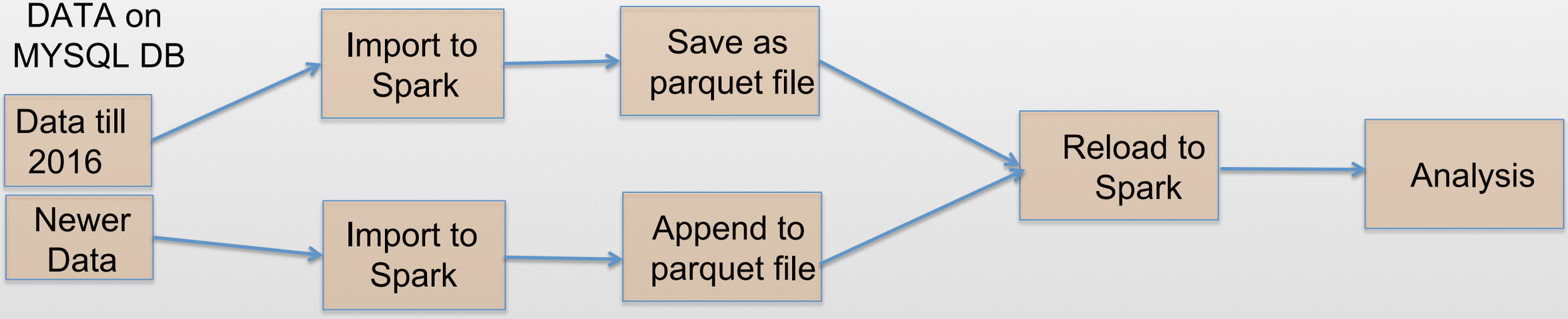
```
sync from spark://n0381.ten.osc.edu:7077
starting org.apache.spark.deploy.master.Master, logging to /nfs/15/soottikkal/spark/kdd/
spark-soottikkal-org.apache.spark.deploy.master.Master-1-n0381.ten.osc.edu.out
failed to launch org.apache.spark.deploy.master.Master:
full log in /nfs/15/soottikkal/spark/kdd/spark-soottikkal-
org.apache.spark.deploy.master.Master-1-n0381.ten.osc.edu.out

Duration Statistics:
Mean 48.342000
St. deviation : 723.330000
Max value: 58329.000000
Min value: 0.000000
Total value count: 4898431.000000
Number of non-zero values: 118939.000000


SPARK_MASTER=spark://n0381.ten.osc.edu:7077
```

CASE STUDY

Data mining of historical jobs records of OSC's clusters

Aim: To understand client utilizations of OSC recourses.

Data: Historical records of every Job that ran on any OSC clusters that includes information's such as number of nodes, software, CPU time and timestamp.



Pyspark code for data analysis

#importing data

```
df=sqlContext.read.parquet("/fs/scratch/pbsacct/Jobs.parquet")
df.show(5)
```

jobid	username	system	nproc	submit_date	end_date	jobname	sw_app	queue
13780.owens-batch...	osu0833	owens	280	2016-09-28	2016-10-08	MMPCDH24EC1-3-2eq...	namd	parallel
13786.owens-batch...	com0644	owens	96	2016-09-28	2016-10-05	FR181-011DS	foam	parallel
13798.owens-batch...	com0480	owens	252	2016-09-28	2016-10-03	TSRD-5-3-012DS	foam	parallel
13800.owens-batch...	com0480	owens	252	2016-09-28	2016-10-02	TSRD-5-3-013MSE	foam	parallel
13804.owens-batch...	com0480	owens	252	2016-09-28	2016-10-02	TSRD-5-3-014MSE	foam	parallel

#Which types of queue is mostly used

```
df.select("jobid","queue").groupBy("queue").count().show()
```

queue	count
debug	157
serial	288174
montecarlo	12
parallel	41214
hugemem	102
largeparallel	60
longserial	66
dedicated	8

#Which software is used most?

```
df.select("jobid","sw_app").groupBy
("sw_app").count().sort(col("count").desc()) .show()
```

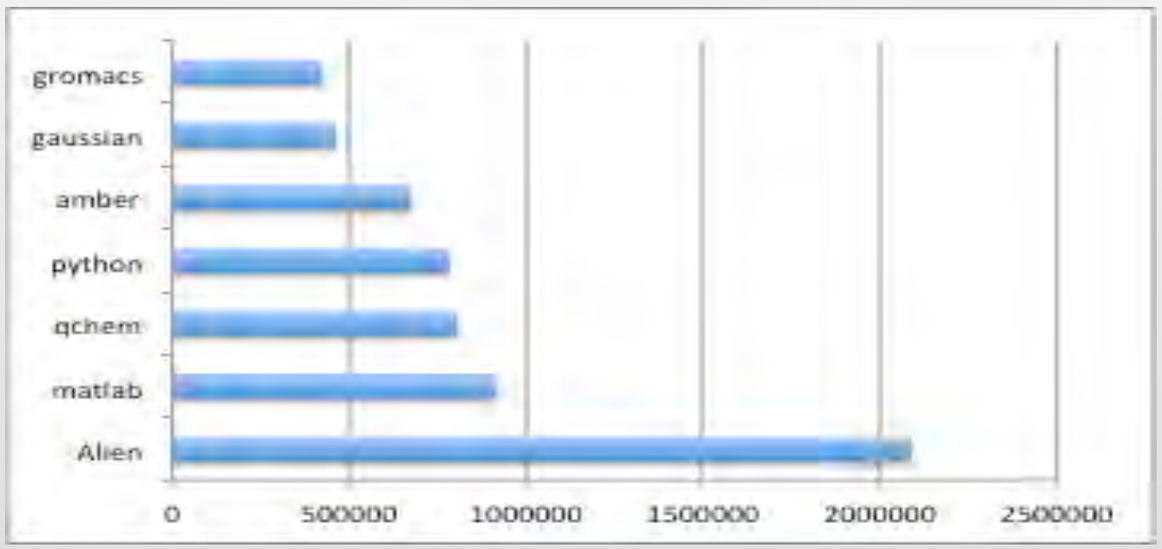
sw_app	count
condor	40199
fastsimcoal	39535
null	36914
amber	35304
real_exe	31076
molcas	23695
vasp	18164
gadget	13880
bam	13189
hpl	9820

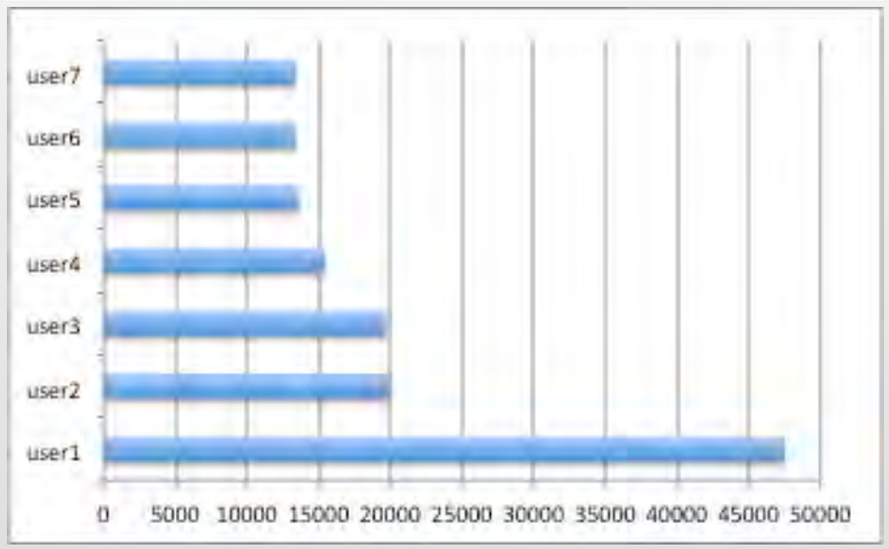
#who uses gaussian software most?

```
df.registerTempTable("Jobs")
sqlContext.sql(" SELECT username FROM
Jobs WHERE sw_app='gaussian' " ).show()
```

Results

Statistics	MYSQL	SPARK
Job vs CPU	1 hour	5 sec
CPU vs Account	1.25 hour	5 sec
Walltime vs user	1.40 hour	5 sec





Running Hadoop at OSC

A Hadoop cluster can be launched within the HPC environment, but managed by the PBS job scheduler using Myhadoop framework developed by San Diego Supercomputer Center. (Please see http://www.sdsc.edu/~allans/MyHadoop.pdf)

Availability & Restrictions

Hadoop is available to all OSC users without restriction.

The following versions of Hadoop are available on OSC systems:

VERSION	OAKLEY	OWENS
3.0.0*		X

NOTE: * means it is the default version.

Set-up

In order to configure your environment for the usage of Hadoop, run the following command:

```
module load hadoop
```

In order to access a particular version of Hadoop, run the following command

```
module load hadoop/3.0.0-alpha1
```

Using Hadoop: Sample PBS Script

```
#PBS -N hadoop-example

#PBS -l nodes=6:ppn=12

#PBS -l walltime=01:00:00

setenv WORK $PBS_O_WORKDIR

module load hadoop/3.0.0-alpha1

module load myhadoop/v0.40

setenv HADOOP_CONF_DIR $TMPDIR/mycluster-conf-$PBS_JOBID

cd $TMPDIR

myhadoop-configure.sh -c $HADOOP_CONF_DIR -s $TMPDIR

$HADOOP_HOME/sbin/start-dfs.sh

hadoop dfsadmin -report

hadoop  dfs -mkdir data

hadoop  dfs -put $HADOOP_HOME/README.txt  data/

hadoop  dfs -ls data

hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.0-alpha1.jar
wordcount data/README.txt wordcount-out

hadoop  dfs -ls wordcount-out

hadoop  dfs  -copyToLocal -f  wordcount-out  $WORK

$HADOOP_HOME/sbin/stop-dfs.sh

myhadoop-cleanup.sh
```

Using Hadoop: Sample PBS Script

```
#PBS -N hadoop-example

#PBS -l nodes=6:ppn=12

#PBS -l walltime=01:00:00

setenv WORK $PBS_O_WORKDIR

module load hadoop/3.0.0-alpha1

module load myhadoop/v0.40

setenv HADOOP_CONF_DIR $TMPDIR/mycluster-conf-$PBS_JOBID

cd $TMPDIR

myhadoop-configure.sh -c $HADOOP_CONF_DIR -s $TMPDIR

$HADOOP_HOME/sbin/start-dfs.sh

hadoop dfsadmin -report

hadoop  dfs -mkdir data

hadoop  dfs -put $HADOOP_HOME/README.txt  data/

hadoop  dfs -ls data

hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.0-alpha1.jar
wordcount data/README.txt wordcount-out

hadoop  dfs -ls wordcount-out

hadoop  dfs  -copyToLocal -f  wordcount-out  $WORK

$HADOOP_HOME/sbin/stop-dfs.sh

myhadoop-cleanup.sh
```

Spark Exercise

Connect to Owens cluster through putty terminal:
ssh username@owens.osc.edu
Enter password

```
#Copy necessary files
cp -r ~soottikkal/workshop/April17-Bigdata ./

#check files
cd April17-Bigdata
ls
cat instructions

#open another terminal
# request 1 interactive node

qsub -I -l nodes=1:ppn=28 -l walltime=04:00:00 -A PZS0687

#check files
cd April17-Bigdata
ls
cd spark

#launch spark
module load spark/2.0.0
pyspark --executor-memory 10G --driver-memory 10G
```

```
#Example 1: Unstructured Data

#create a RDD
>>> data = sc.textFile("README.md")

#count number of lines
>>> data.count()
99

#see the content of the RDD
>>> data.take(3)
[u'# Apache Spark', u'', u'Spark is a fast and general cluster computing system for Big Data. It provides']
>>> data.collect()

#check data type
>>> type(data)
<class 'pyspark.rdd.RDD'>

#transformation of RDD
>>> linesWithSpark = data.filter(lambda line: "Spark" in line)
#action on RDD
>>> linesWithSpark.count()
19

##combining transformation and actions
>>> data.filter(lambda line: "Spark" in line).count()
19
```

#Example 2: Structured Data

#About the data: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99

#load data and run basic operations

```
>>> raw_data=sc.textFile("data.gz")
>>> raw_data.count()
494021

>>> raw_data.take(1)
[u'0,tcp,http,SF,
181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,
0.00,0.00,0.00,0.00,normal.']
```

```
>>> raw_data.take(3)
[u'0,tcp,http,SF,
181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,
0.00,0.00,0.00,0.00,normal.', u'0,tcp,http,SF,
239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00
,0.00,0.00,0.00,0.00,normal.', u'0,tcp,http,SF,
235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.0
0,0.00,0.00,0.00,0.00,normal.']
```

```
#SparkSQL
>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>> from pyspark.sql import Row

#transform to csv
>>> csv_data=raw_data.map(lambda l: l.split(","))
>>> selected_data=csv_data.map(lambda p: Row(
        duration=int(p[0]),
        protocal_type=p[1],
        service=p[2],
        flag=p[3],
        src_bytes=int(p[4]),
        dst_bytes=int(p[5])
        )
    )

>>> interactions_df = sqlContext.createDataFrame(selected_data)
>>> interactions_df.registerTempTable("interactions")

>>> interactions_df.printSchema()
root
 |-- dst_bytes: long (nullable = true)
 |-- duration: long (nullable = true)
 |-- flag: string (nullable = true)
 |-- protocal_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- src_bytes: long (nullable = true)
```

```
>>> interactions_df.show(5)
```

```
+---------+--------+----+-------------+-------+---------+
|dst_bytes|duration|flag|protocal_type|service|src_bytes|
+---------+--------+----+-------------+-------+---------+
|    29200|       0|  S1|          tcp|   http|      228|
|     9156|       0|  S1|          tcp|   http|      212|
|        0|       0| REJ|          tcp|  other|        0|
|        0|       0| REJ|          tcp|  other|        0|
|        0|       0| REJ|          tcp|  other|        0|
+---------+--------+----+-------------+-------+---------+
only showing top 5 rows
```

```
>>> interactions_df.select("dst_bytes","flag").show(5)
```

```
+---------+----+
|dst_bytes|flag|
+---------+----+
|     5450|  SF|
|      486|  SF|
|     1337|  SF|
|     1337|  SF|
|     2032|  SF|
+---------+----+
```

```
>>> interactions_df.filter(interactions_df.flag!="SF").show(5)
```

```
+---------+--------+----+-------------+-------+---------+
|dst_bytes|duration|flag|protocal_type|service|src_bytes|
+---------+--------+----+-------------+-------+---------+
|    29200|       0|  S1|          tcp|   http|      228|
|     9156|       0|  S1|          tcp|   http|      212|
|        0|       0| REJ|          tcp|  other|        0|
|        0|       0| REJ|          tcp|  other|        0|
|        0|       0| REJ|          tcp|  other|        0|
+---------+--------+----+-------------+-------+---------+
only showing top 5 rows
```

```
# Select tcp network interactions with more than 1 second duration and no transfer from destination
>>> tcp_interactions = sqlContext.sql(“””
                    SELECT duration, dst_bytes FROM interactions WHERE protocal_type = 'tcp' AND duration >
1000 AND dst_bytes = 0
“””")
tcp_interactions.show(5)


>>> interactions_df.select("protocal_type", "duration", "dst_bytes").groupBy("protocal_type").count().show()
```

```
+-------------+------+
|protocal_type| count|
+-------------+------+
|          tcp|190065|
|          udp| 20354|
|         icmp|283602|
+-------------+------+
```

```
>>> interactions_df.select("protocal_type", "duration",
"dst_bytes").filter(interactions_df.duration>1000).filter(interactions_df.dst_bytes==0).groupBy("protocal_type").
count().show()
```

```
+-------------+-----+
|protocal_type|count|
+-------------+-----+
|          tcp|  139|
+-------------+-----+
```

```
#exit from the interactive pyspark shell
>>> exit()

#exit from the compute node
exit
```

Submitting Spark and Hadoop job non-interactively

```
cd spark
ls
qsub stati.pbs
qstat
qstat  | grep `whoami`
ls
qsub sql.pbs


cd hadoop
qsub sub-wordcount.pbs
qsub sub-grep.pbs
```

References

1. Spark Programming Guide

https://spark.apache.org/docs/2.0.0/programming-guide.html

-Programming with Scala, Java and Python

2. Data Exploration with Spark

http://www.cs.berkeley.edu/~rxin/ampcamp-ecnu/data-exploration-using-spark.html

3. Hadoop

http://hadoop.apache.org/

4. OSC Documentation

https://www.osc.edu/documentation/software_list/spark_documentation
https://www.osc.edu/resources/available_software/software_list/hadoop

Thank you!

- Questions or comments: soottikkal@osc.edu
- General questions about OSC service: oschelp@osc.edu