Distributed Computing

DIANE WOODBRIDGE, PH.D

Distributed Computing

Course Objectives

Understand needs and concepts of distributed computing.

Understand the Spark and its stack.

Being competent to work with Spark on a distributed computing environment.

- Programming with RDDs.
- Work with key/value pairs.
- Work with DataFrame.
- Work with SparkSQL, MLlib, ML, Spark Streaming, (and GarphX).
- Work on Amazon AWS.

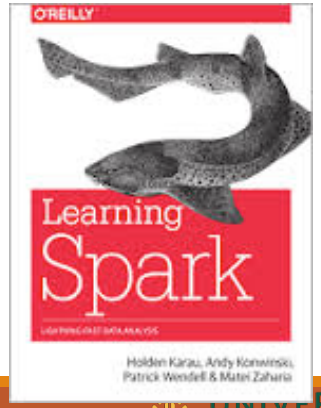
Reference Materials

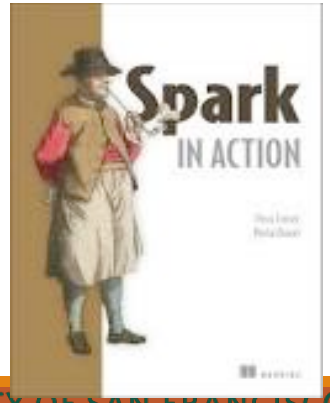
Spark Online Documentation, http://spark.apache.org/docs/latest/

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

Zecevic, Petar, et al. Spark in Action, Manning, 2016.

AWS Online Documentation, https://aws.amazon.com/documentation/





Course Evaluation

Attendance (10%)
- Randomly and multiple times a day.
- No cellphones!

Homework (25 % - 5% of each)
- No late submission/resubmission allowed!! (We are going to run the grading code only once.)

Quiz (Programming, multiple choices/short answers) (40% - 5% to 7% of each)

Group Project (25%)
- Data Selection/Loading on AWS.
- Data Stats (using RDD or DataFrame)
- Data Analytics (Spark MLlib, Spark ML)
- Final Presentation and Code Submission.

Course Schedule

10 am – 10:30 am : Quiz

10:30 am – 12:00 am : Morning Lecture

12 pm – 1pm : Lunch Break

1 pm – 3:30 pm : Afternoon Lecture

3:30 pm – 4pm : Homework and Office Hour

Jab 9	**Jab 10**	**Jab 11**	**Jab 12**	**Jab 13**
		No Morning Class – Office Hour 1-4pm AWS Hands on by AWS*		
Jab 16	**Jab 17**	**Jab 18**	**Jab 19**	**Jab 20**
MLK (No Class)		2-4pm Click-Through Rate (CTR) with Yannet		Final Group Presentation

Others…

Example Data

- https://github.com/dianewoodbridge/msan694-example.git

Poll

- https://pollev.com/dianewoodbri311

Spark Interview Questions

What is Apache Spark?

Explain the key features of Spark.

What is RDD?

How to create RDD.

What is ”partitions”?

Types or RDD operations?

What is “transformation”?

What is “action”?

Functions of “spark core”?

What is “spark context”?

What is an “RDD lineage”?

Which file systems does Spark support?

List the various types of “Cluster Managers” in Spark.

What is “YARN”?

What is “Mesos”?

What is a “worker node”?

What is an “accumulator”?

What is “Spark SQL” (Shark)?

What is “SparkStreaming”?

What is “GraphX”?

What is “MLlib”?

https://www.edureka.co/blog/interview-questions/top-apache-spark-interview-questions-2016/

Spark Interview Questions

What are the advantages of using Apache Spark over Hadoop MapReduce for big data processing?

What are the languages supported by Apache Spark for developing big data applications?

Can you use Spark to access and analyse data stored in Cassandra databases?

Is it possible to run Apache Spark on Apache Mesos?

How can you minimize data transfers when working with Spark?

Why is there a need for broadcast variables?

Name a few companies that use Apache Spark in production.

What are the various data sources available in SparkSQL?

What is the advantage of a Parquet file?

What do you understand by Pair RDD?

Is Apache Spark a good fit for Reinforcement learning?

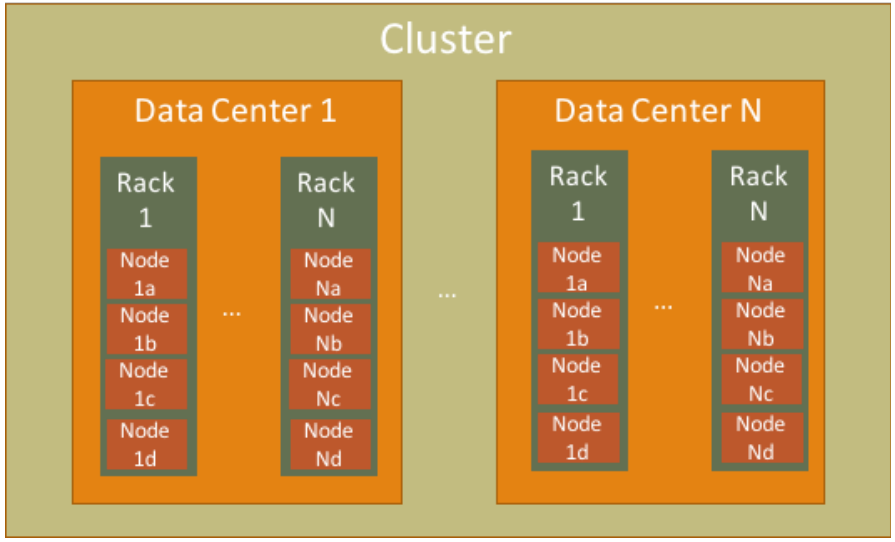
https://www.dezyre.com/article/top-50-spark-interview-questions-and-answers-for-2016/208

Why Distributed Computing?

Large volumes of data

- **Scaling out** instead of scale up.
  - Cheaper : Run large data on clusters of many smaller and cheaper machines.
  - Reliable : If one node fails, data is still available on other replicas.
  - Faster : System picks how to split each operator into tasks and where to run each task.

What is Distributed Computing?



Node : A node is the storage layer within a server.

Rack (Server Rack) : A logical set of nodes in close proximity .

Data Center : A logical set of racks.

Cluster : A set of data centers.

MapReduce - Concept

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
  - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
  - Map : Return key-value pairs from a single aggregate.
  - Reduce : Return one key-value pair from multiple key-value pairs.

Walmart Order

OrderID: 1
Customer : Diane
Timestamp : 2016-08-15 05:04:32 PST
Items:
{ProductName : San Francisco Giants Hat M,
{Qty: 2, UnitPrice : 10, Price : 20}},
{ProductName : San Francisco Giants Hat S,
{ Qty: 3, UnitPrice : 8, Price : 24}}
Shipping: Corte Madera

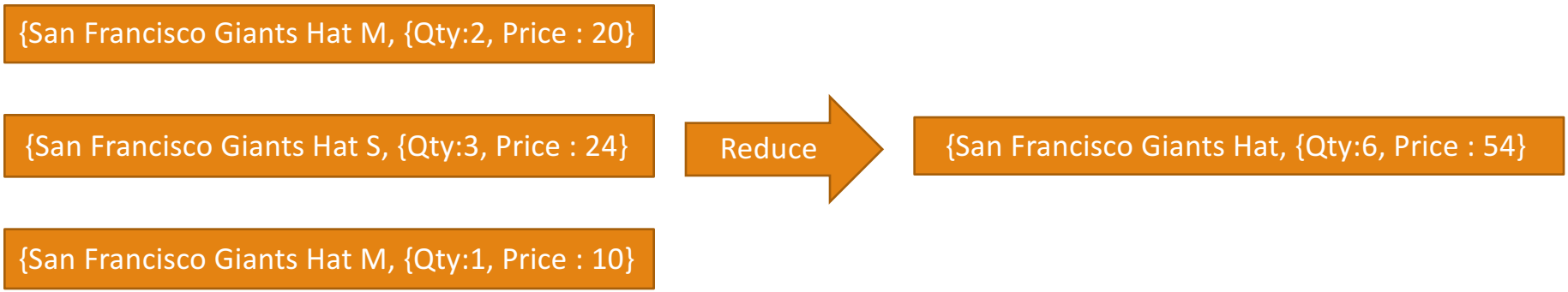
{San Francisco Giants Hat M, {Qty:2, Price : 20}

{San Francisco Giants Hat S, {Qty:3, Price : 24}

MapReduce - Concept

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
  - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
  - Map : Return key-value pairs from a single aggregate.
  - Reduce : Return one key-value pair from multiple key-value pairs.



Hadoop MapReduce

Open source, distributed, Java computation framework consisting of Hadoop distributed File System (HDFS) and MapReduce.

Hadoop solved issues of..

- Parallelism : Perform subsets of the computation simultaneously.
- Distribution : Distribute the data.
- Fault Tolerance : Handle component failure.

Hadoop MapReduce - Limitations

Hadoop is powerful, but can be slow.

- MapReduce job results need to be stored in HDFS (disk) before they can be used by another job. ➔ Slow with iterative algorithms.

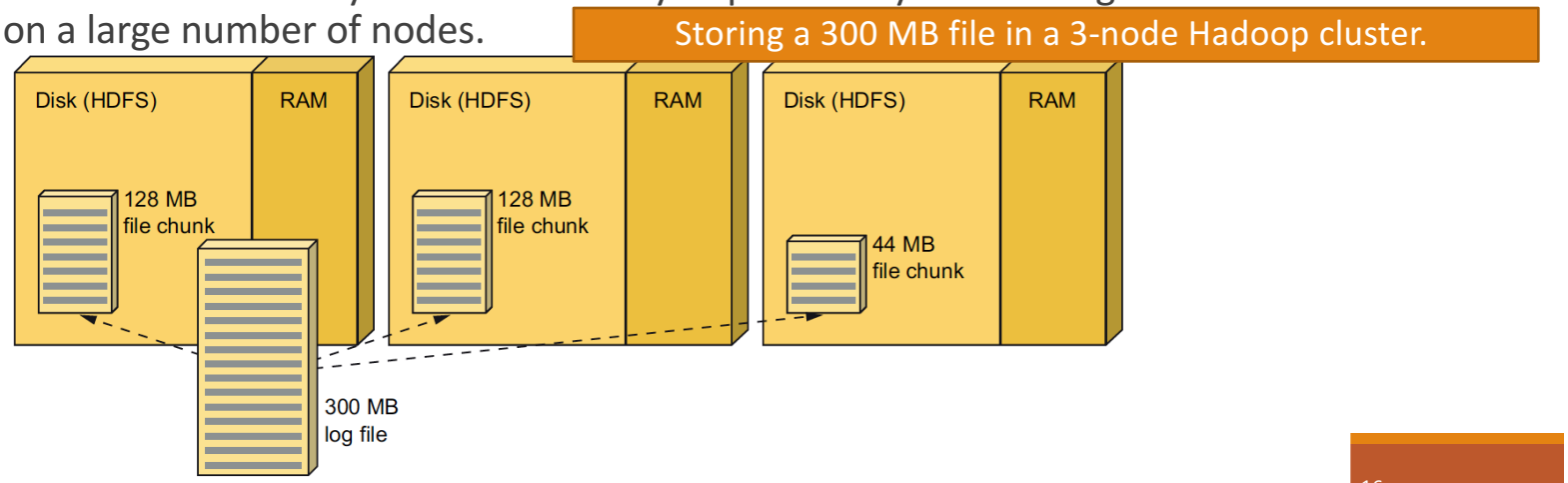
Many kinds of problems don't easily fit MapReduce's two-step paradigm.

Hadoop is a low-level framework, so myriad tools have sprung up around it and brings additional complexity and requirements.

Hadoop vs. Spark

Spark

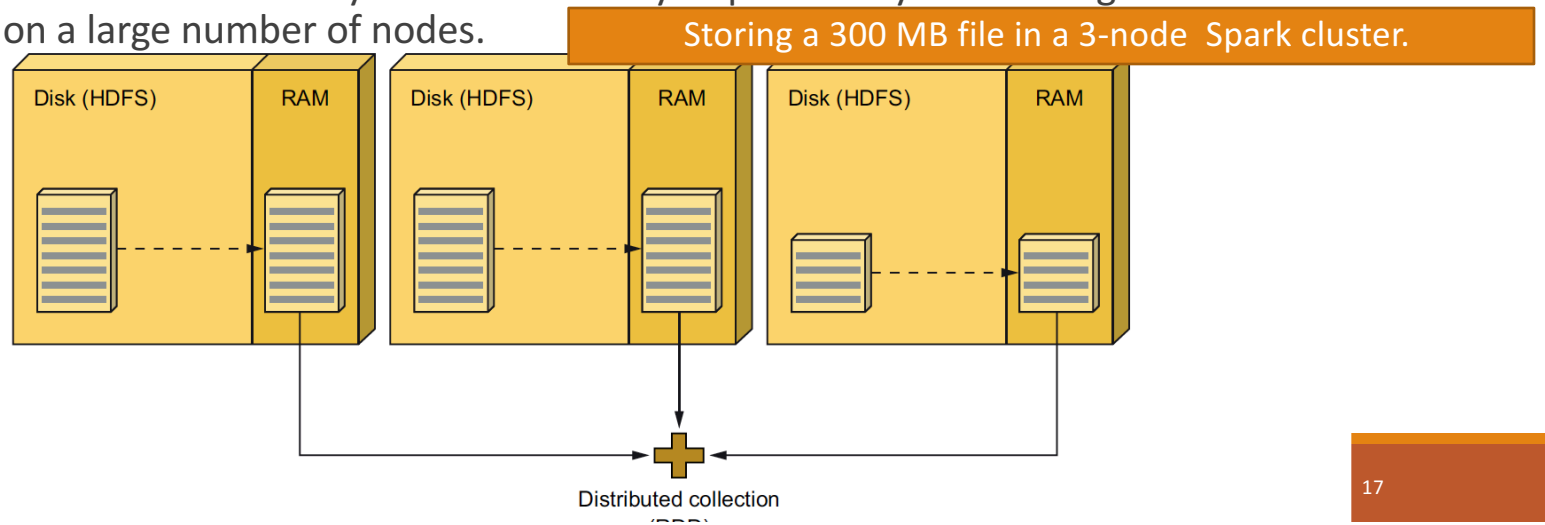
- It keeps large amounts of data in memory which offers tremendous performance improvements. (x100 faster than Hadoop MapReduce.) ➔ Good for iterative algorithms such as machine learning, graph algorithms and others that need to reuse data.
- You can write distributed programs in a manner similar to writing local programs.
  - Spark's collections abstract away the fact that they're potentially referencing data distributed on a large number of nodes.



Hadoop vs. Spark

Spark

- It keeps large amounts of data in memory which offers tremendous performance improvements. (x100 faster than Hadoop MapReduce.) ➔ Good for iterative algorithms such as machine learning, graph algorithms and others that need to reuse data.
- You can write distributed programs in a manner similar to writing local programs.
  - Spark's collections abstract away the fact that they're potentially referencing data distributed on a large number of nodes.



Hadoop vs. Spark

Spark

- It keeps large amounts of data in memory which offers tremendous performance improvements. (x100 faster than MapReduce.) ➔ Good for iterative algorithms such as machine learning, graph algorithms and others that need to reuse data.
- You can write distributed programs in a manner similar to writing local programs.
  - Spark's collections abstract away the fact that they're potentially referencing data distributed on a large number of nodes.
- Spark combines MapReduce like capabilities for batch programing, real time data processing, SQL-like handling of structured data, graph algorithms and machine learning all in a single frame work.

Extends the MapReduce model with primitives for efficient data sharing (Using Resilient Distributed Datasets (RDDs)).

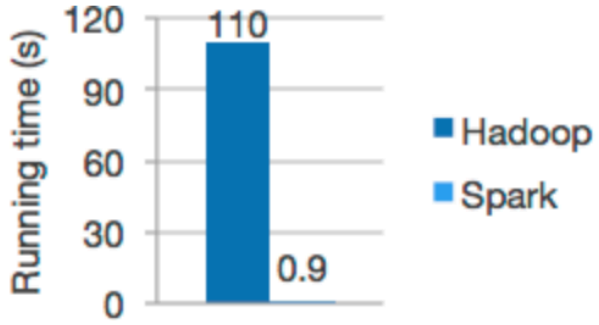
Achieves

- Speed.
- Ease of Use.
- Generality.
- Runs everywhere.

Lightning-fast cluster computing

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

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



Logistic regression in Hadoop and Spark

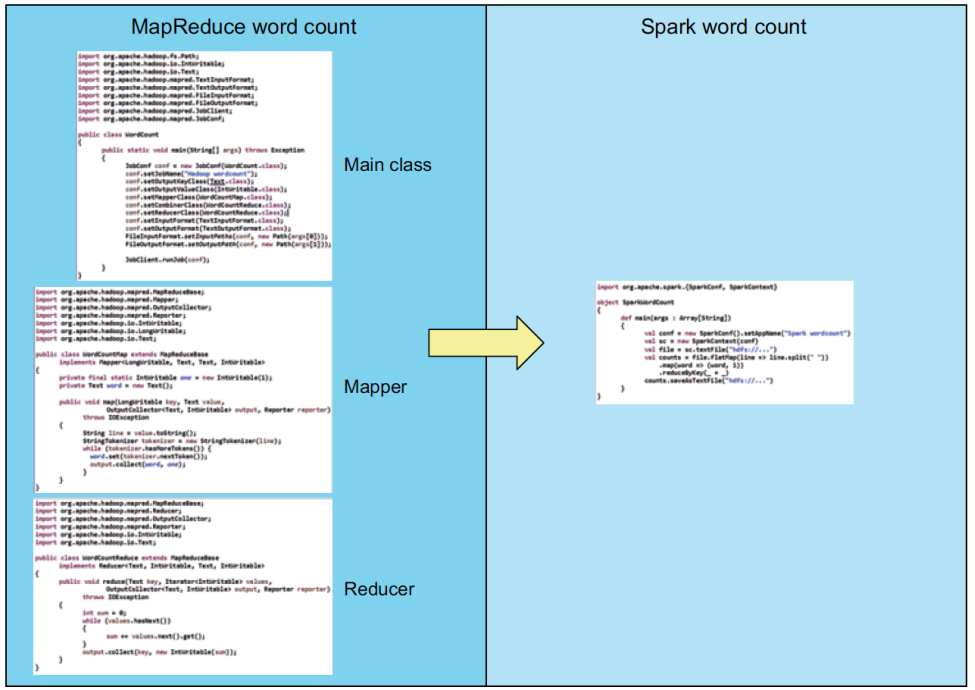
Ease of Use : Write applications quickly in Java, Scala, Python, R.

- 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.

```
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

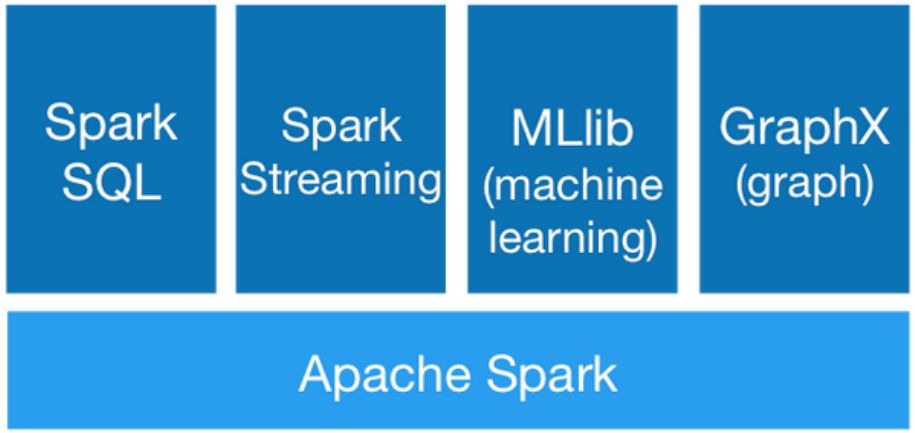


http://spark.apache.org/

Lightning-fast cluster computing

Generality : Combine SQL, streaming, and complex analytics.

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



http://spark.apache.org/

Lightning-fast cluster computing

Runs everywhere : Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

- You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos. Access data in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source.

Community

- Most active open source community for big data.
- 200+ developers, 50+ companies contributing.

Spark Stack

Cluster Managers

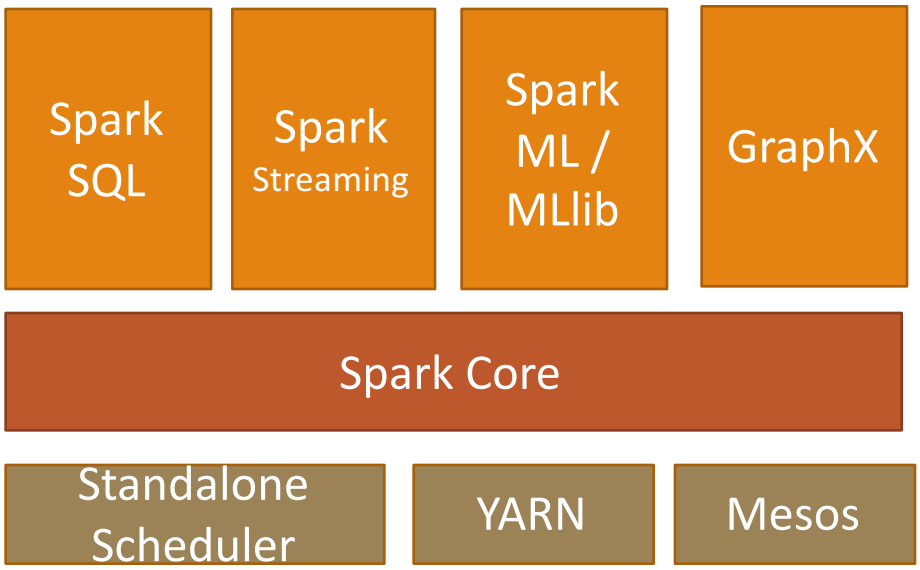
Spark Core

Spark SQL

Spark Streaming

Spark MLlib and ML

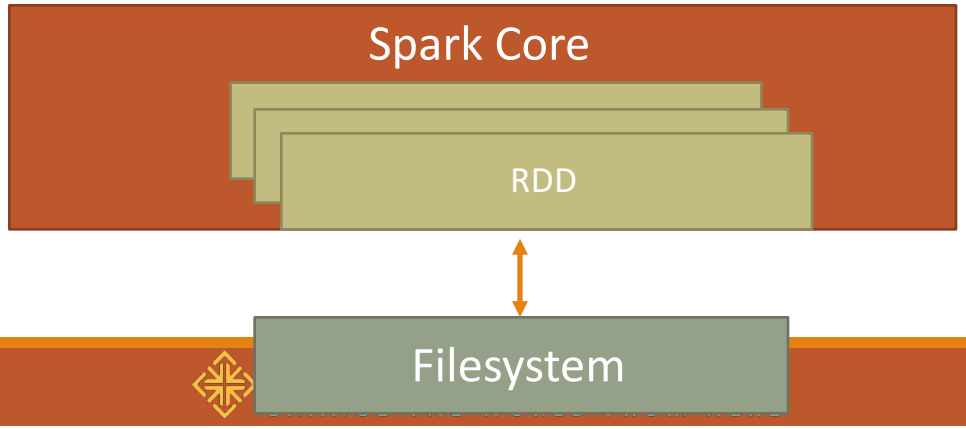
GraphX



Spark Components

Spark Core

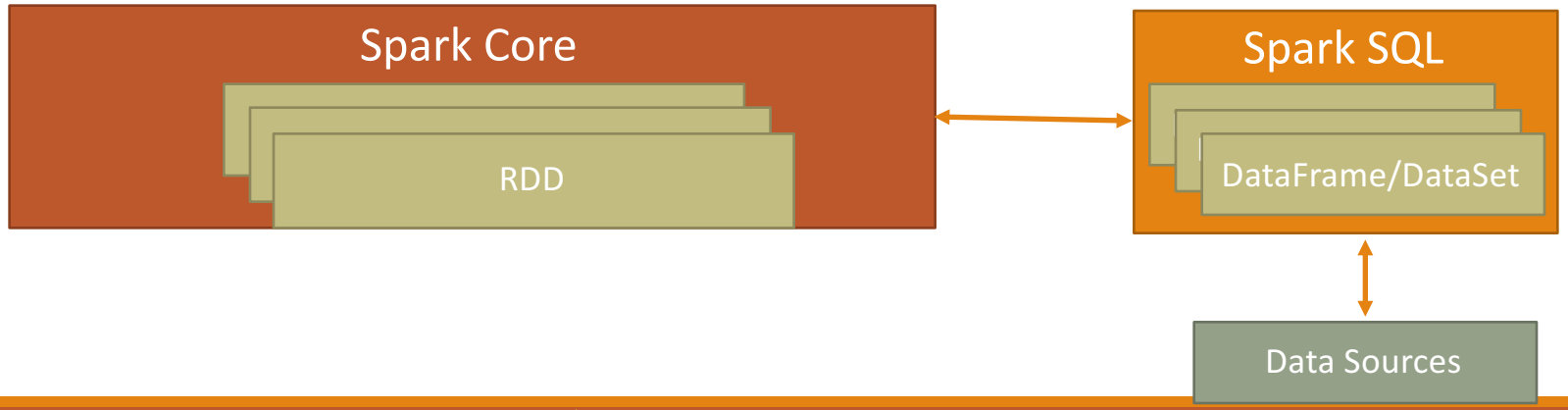
- Contains basic Spark functionalities (including **Resilient Distributed Datasets (RDDs)**) required for running jobs and needed by other components.
  - RDD : Abstraction of a distributed collection of items with operations and transformation applicable to the dataset.
  - Logic for accessing various filesystems including HDFS, GlusterFS, Amazon S3.
  - Fundamental functions such as networking, security, scheduling and data shuffling.



Spark Components

Spark SQL

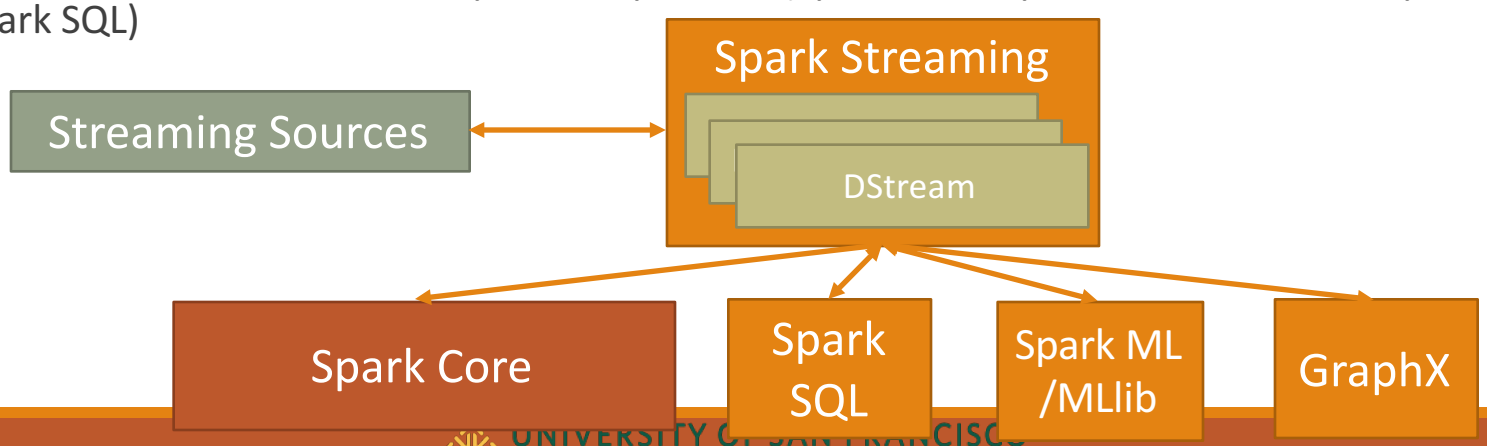
- Provides functions for manipulating large sets of distributed, structured data using an SQL subset.
- Uses `DataFrames` and `DataSets`
  - Spark SQL transforms operations on DataFrames/DataSets to operations on RDDs.
- Data Sources include Hive, JSON, relational databases, NoSQL databases and Parquet files.



Spark Components

Spark Streaming

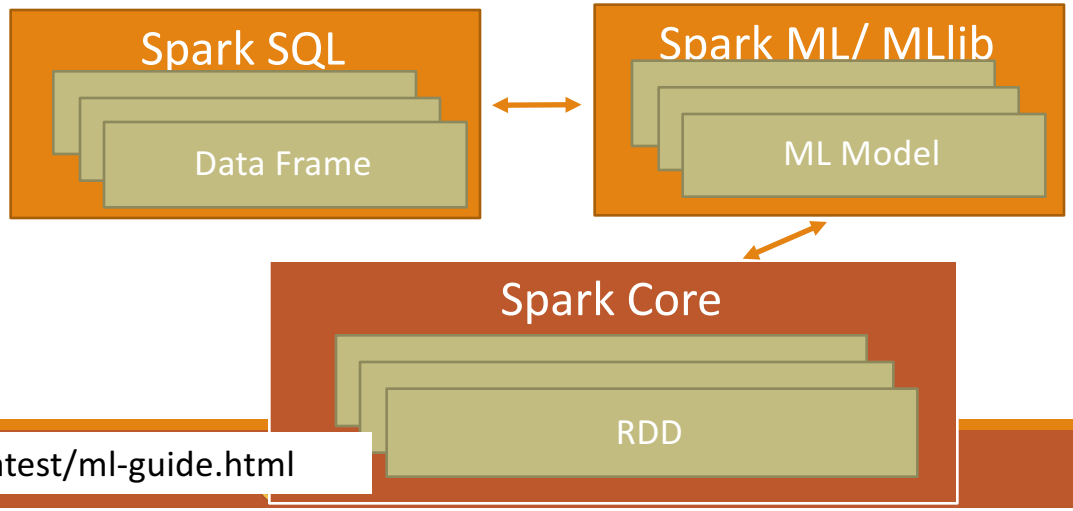
- Ingest real-time streaming data from various sources including HDFS, Kafka, Flume, Twitter, ZeroMQ and custom ones.
- Recover from failure automatically.
- Represent streaming data using discretized streams (Dstreams), which periodically create RDDs containing the data that came in during the last time window.
- Can be combined with other Spark components (Spark Core, SparkML and Mllib, GraphX, Spark SQL)



Spark Components

Spark MLlib and ML

- Library of machine learning algorithms
- Include logistic regression, naïve Bayes, support vector machine, decision trees, random forests, linear regression and k-mean clustering.
- MLlib :RDD-based APIs.
- Spark ML : DataFrame-based APIs.

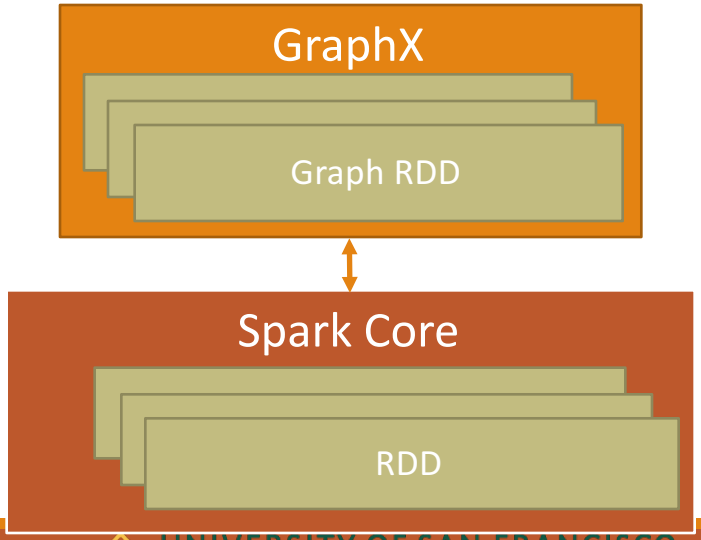


http://spark.apache.org/docs/latest/ml-guide.html

Spark Components

Spark GraphX

- Provide functions for building graphs, represented as graph RDDs : EdgeRDD and `VertexRDD`.
- Contain important algorithms of graph theory such as page rank, connected components, shortest paths, SVD++ .



Spark Examples

Word Count

Text Search

Prediction with Logistic Regression

Page Rank

And many more..

http://spark.apache.org/examples.html

Spark Installation

Prerequisite:

- Use python version 2.7.12.
- Install Anaconda
  - This includes Ipython, jupyter
- Download JDK

```
brew update
brew cask install java
```

✓Spark runs on Java 7+, Python 2.6+/3.4+ and R 3.1+. For the Scala API, Spark 2.0.0 uses Scala 2.11. You will need to use a compatible Scala version (2.11.x).

Install Spark

```
brew install apache-spark
```

https://gist.github.com/ololobus/4c221a0891775eaa86b0

Spark Installation

Install Spark

```
brew install apache-spark
```

```
ML-ITS-603436:bin dwoodbridge$ brew info apache-spark
apache-spark: stable 2.0.2, HEAD
Engine for large-scale data processing
https://spark.apache.org/
/usr/local/Cellar/apache-spark/2.0.2 (1,206 files, 204.5M) *
  Built from source on 2016-12-13 at 09:42:19
From: https://github.com/Homebrew/homebrew-core/blob/master/Formula/apache-spark
.rb
```

https://gist.github.com/ololobus/4c221a0891775eaa86b0

Spark Installation

Spark-shell – uses scala or python.

- A program written in the shell is discarded after you exit the shell.

➔ Testing and developing applications.

`spark-shell`

`Pyspark`

https://gist.github.com/ololobus/4c221a0891775eaa86b0

Spark Installation

Spark-shell – uses scala or python.

spark-shell

```
ML-ITS-603436:bin dwoodbridge$ spark-shell
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).
16/12/13 09:58:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using buil
tin-java classes where applicable
16/12/13 09:58:47 WARN SparkContext: Use an existing SparkContext, some configuration may not take effect.
Spark context Web UI available at http://192.168.1.9:4040
Spark context available as 'sc' (master = local[*], app id = local-1481651927379).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_112)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

https://gist.github.com/ololobus/4c221a0891775eaa86b0

Spark Installation

Spark-shell – uses scala or python.

pyspark

```
ML-ITS-603436:~ dwoodbridge$ pyspark
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul  2 2016, 17:43:17)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
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).
16/12/13 10:04:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your p
e applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Python version 2.7.12 (default, Jul  2 2016 17:43:17)
SparkSession available as 'spark'.
>>>
```

https://gist.github.com/ololobus/4c221a0891775eaa86b0

Spark Installation

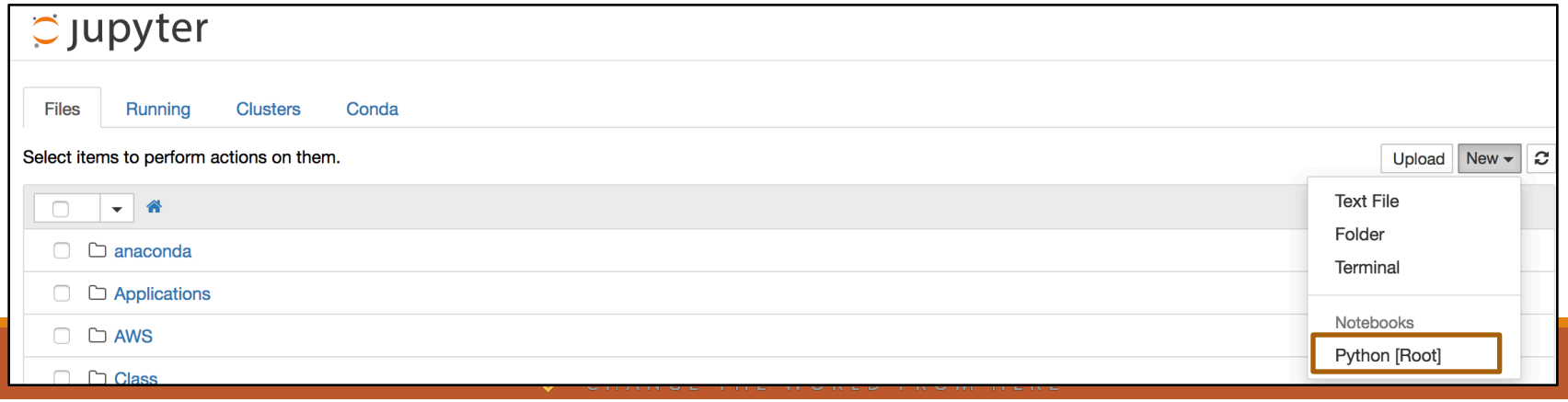
Jupyter set up (include in your ~/.profile or .bash_profile)

```
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

Start pyspark (Python version spark shell)

```
pyspark
```

When jupyter notebook starts, choose “new python”



Spark Context and Executor

Driver Program

- Launch parallel operations on a cluster.
- Contain an application's main function.
- Define distributed datasets on the cluster.
- Apply operations on the distributed datasets on the cluster.
- Access Spark through a SparkContext (sc) object.
  - SparkContext : A connection to a cluster and you can use it to build RDDs.


```
In [13]: sc

Out[13]: <pyspark.context.SparkContext at 0x10b9409d0>
```

Spark Context and Executor

Driver Program

- Launch parallel operations on a cluster.
- Contain an application's main function.
- Define distributed datasets on the cluster.
- Apply operations on the distributed datasets on the cluster.
- Access Spark through a SparkContext (sc) object.
  - SparkContext : A connection to a cluster and you can use it to build RDDs.

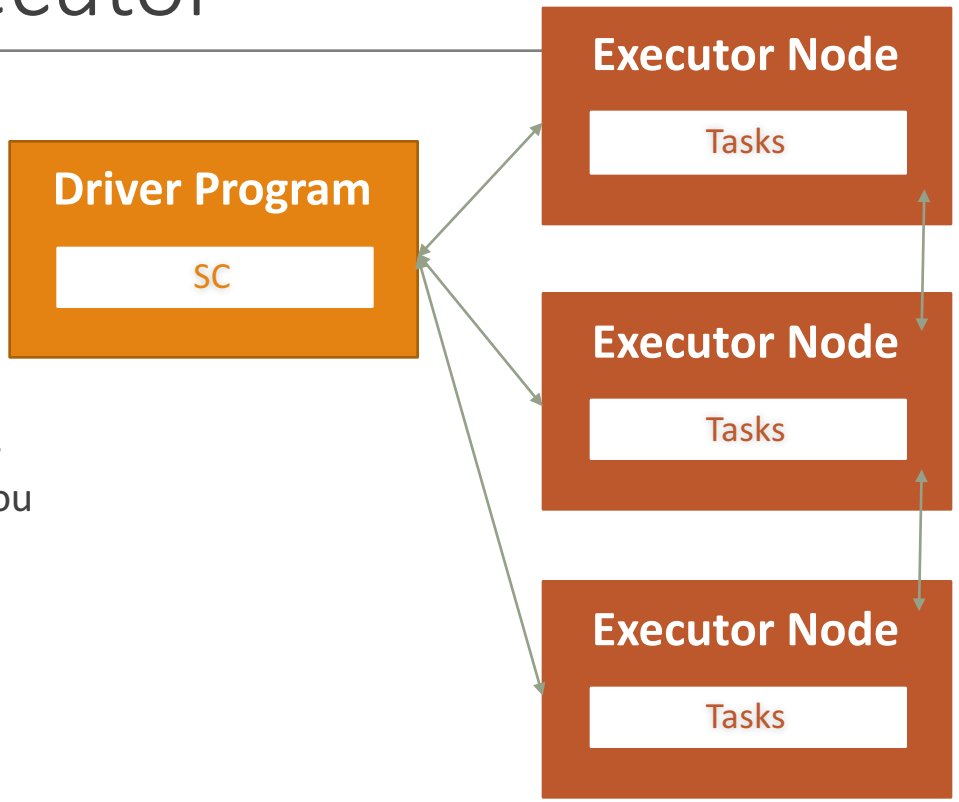

```
In [13]: sc

Out[13]: <pyspark.context.SparkContext at 0x10b9409d0>
```

Spark Context and Executor

Driver Program

- Launch parallel operations on a cluster.
- Contain an application's main function.
- Define distributed datasets on the cluster.
- Apply operations on the distributed datasets on the cluster.
- Access Spark through a SparkContext (sc) object.
  - SparkContext : A connection to a cluster and you can use it to build RDDs.
  - Once having a SparkContet, you can use it to build RDDs.



What is RDD?

Resilient Distributed Dataset (RDD)

- RDD is a distributed collection of items (Spark's primary abstraction).
- RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs.
- Key Ideas
  - Distributed
    - Built via parallel transformations (map, filter, ...).
    - User-controlled partitioning & storage (memory, disk, ...). : RDDs know their partitioning functions.
  - Immutable
    - Read-only. : Once created, RDDs never change.
  - Resilient
    - Automatically rebuilt on failure.
    - RDDs track lineage info to rebuild lost data. (Instead of replication.)

RDD Creation

Two ways of creating RDDs.

- Loading an external data.

```
lines = sc.textFile(“README.md”)
```

- Takes a collection such as `Seq` (`Array` or `List`) and creates RDD from its element and distribute to Spark executors in the process.

```
lines = sc.parallelize(["spark", "spark is fun!"])
```

- Check the number of partitions.

```
lines.getNumPartitions()
```

https://spark.apache.org/docs/1.2.0/configuration.html

Python Lambda Expression

Python Lambda Expression

- Help create an anonymous function (not bound to a name) at runtime.
- Keyword : lambda

```
def f(x):
        return x+2
g = lambda x : x+2
g(2)
```

➔ Can be used as a function parameter for RDD operations.

https://docs.python.org/3/tutorial/controlflow.html
http://www.secnetix.de/olli/Python/lambda_functions.hawk

RDD Operations

Two types

- Transformation
  - Produce a new RDD by performing data manipulation on another RDD.
  - Ex. Map, filter, flatmap, mapPartitions, sample, union, intersection, distinct, groupByKey, reduceByKey, aggregateByKey, sortByKey, join, cogroup, cartesian, pipe, coalesce, repartition, repartitionAndSortWithinPartitions.
- Actions
  - Trigger a computation to return the result to the calling program or to perform some actions on an RDD's elements.
  - Ex. Reduce, collect, count, first, take, takeSample, takeOrdered, saveAsTextFile, saveAsSequenceFile, saveAsObjectFile, countByValue, foreach.
- .persist if the result will be reused.

https://spark.apache.org/docs/1.2.0/programming-guide.html

RDD Operations - Transformation

Construct a new RDD from an existing RDD.

```
line_with_spark = lines.filter(lambda lines : "spark" in lines)
```

Lazy Evaluation. (cf. Hadoop MapReduce)

- Computation doesn't take place until an action is triggered.

Return RDDs.

RDD Operation - Transformation

Transformation Operation Types

http://spark.apache.org/docs/latest/programming-guide.html

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.

RDD Operation - Transformation

Element-wise Transformation

- map()
  - Apply a function to each element in the RDD.
- filter()
  - Return an RDD that passes the filtering requirement.
- flatmap()
  - Call each element in RDD individually.
  - Concatenates multiple arrays into a collections that has one level structure.

RDD Operation - Transformation

Set Operation

- Format : rdd1.operator(rdd2)
- distinct()
  - Return only one of each element.
- union()
  - If there are duplicated elements, it returns all duplicates.
- intersection()
  - Return common elements.
- subtract()
  - Return elements that are in rdd1 only.
- cartesian()
  - Return cartesian product (all pairs between rdd1 and rdd2)

RDD Operation - Action

Compute a result based on an RDD.

Return the result to the driver program or save it to external storage system.

```
line_with_spark.count()
```

```
Return non-RDDs.
```

RDD Operation - Action

Action Operation Types

http://spark.apache.org/docs/latest/programming-guide.html

aggregate(zero)(SeqOp, combOp)	Similar to reduce() but used to return a different type.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
countByValue()	Return the number of times each elemnt occurs in the RDD.
first()	Return the first element of the dataset (similar to take(1)).
fold(zero)(func)	Same as reduce(), but with the provided zero value.
take(n)	Return an array with the first n elements of the dataset.

RDD Operation - Action

Action Operation Types

http://spark.apache.org/docs/latest/programming-guide.html

takeSample(withReplacement, num, [seed])	Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(n, [ordering])	Return the first n elements of the RDD using either their natural order or a custom comparator.
top(num)	Return the top num elements of the RDD.
reduce(func)	Combine the elements of the RDD together in parallel. (eg. Sum)
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems. Note: modifying variables other than Accumulators outside of the foreach() may result in undefined behavior. See Understanding closures for more details.

RDD Operation - Action

- reduce(func)
  - Take a function that operates on two elements of the type in your RDD
  - Returns a new element of the same type.

    Ex. sum = rdd.reduce(lambda x, y = x+y)
- fold(zeroValue)(func)
  - Take a function with the same signature as needed for reduce().
  - Take a “zero value” to be used for the initial call on each partition.
  - Returns a new element of the same type.
- aggregate(zeroValue)(seqOp, combOp)
  - Supply an initial zero value of the type we want to return.
  - seqOp : Function to combine the elements from the RDD with the accumulator. Runs once in a partition.
  - combOp : Function to merge two accumulatros, given that each noe acculates its own results locally.

RDD Operation - Action

- collect()
  - Return the entire RDD's contents.
- count()
  - Return the count of elements.
- take(n)
  - Return n elements.
- first()
  - Return the first element of the data.
- takeSample(withReplacement, num, seed)
  - Return a fixed-size sample subset of an RDD.
- foreach()
  - Used for performing computations on each element in the RDD.

RDD Operation - Action

sample() vs. takeSample()

- sample(withReplacement, fraction, seed) : Transformation
  - Creates a new RDD with random elements from the calling RDD.
    - withReplacement : allow sample multiple times.
    - fraction :
      - expected number of times each element is going to be sampled (positive double), when replacement is used.
      - Expected probability that each element is going to be sampled (between 0 and 1), when replacement is not used.
    - Seed : Random number generation. (Same seeds generates the same quasi-random numbers.)

RDD Operation - Action

sample() vs. takeSample()

- takeSample(withReplacement, num, seed) : Action
  - num : The number of sampled element. (Integer)
  - Return a fixed-size sample subset of an RDD as an array (not RDD).

Example 1

Count number of lines in the “REAME.md”

- lines=sc.textFile("README.md")
- lines.count()

Get the first line in the “REAME.md”

- lines.first()

Example 1

Count number of lines in the “REAME.md”

- `file_name = "README.md"`
- `lines=sc.textFile(file_name)`
- `lines.count()`

Get the first line in the “REAME.md”

- `lines.first()`

Create am RDD representing the lines of text in a file.

Example 1

Count number of lines in the “REAME.md”

- lines=sc.textFile(file_name)
- lines.count()

Get the first line in the “REAME.md”

- lines.first()

Executor nodes’ jobs (action)

Example 1 – Results

```
In [1]: file_name = "/Users/dwoodbridge/Class/MSAN694/Example_Data/README.md"

In [2]: lines = sc.textFile(file_name)

In [3]: lines.count()
Out[3]: 99

In [4]: lines.first()
Out[4]: u'# Apache Spark'
```

Example 2

Find lines including “spark”

- `lines=sc.textFile(file_name)`
- `line_with_spark = lines.filter(lambda lines : "spark" in lines)`

Example 2 - Result

```
In [45]: #find line with "spark".

In [46]: file_name = "README.md"

In [47]: lines=sc.textFile(file_name)

In [48]: line_with_spark = lines.filter(lambda lines : "spark" in lines)
```

Example 2

How many lines include ”spark”?

What is the first line?

What are the first two elements?

Try takeSample(), with allowing replacement to sample 7 elements. (Try multiple times)

Example 4 : Map vs Flat Map

Create RDD with ["spark rdd example", "sample example"]

Using split() method, place words in the map and flatmap.

Example 4

- `x = sc.parallelize(["spark rdd example", "sample example"])`
- `x.collect()`
- `x_map = x.map(lambda a : a.split())`
- `x_map.collect()`
- `x_flatmap = x.flatMap(lambda a : a.split())`
- `x_flatmap.collect()`

Example 4

```
In [1]: x = sc.parallelize(["spark rdd example", "sample example"])

In [2]: x.collect()
Out[2]: ['spark rdd example', 'sample example']

In [3]: x_map = x.map(lambda a : a.split())

In [4]: x_map.collect()
Out[4]: [['spark', 'rdd', 'example'], ['sample', 'example']]

In [5]: x_flatmap = x.flatMap(lambda a : a.split())

In [6]: x_flatmap.collect()
Out[6]: ['spark', 'rdd', 'example', 'sample', 'example']
```

Example 13

For the numbers between 1 and 10, calculate sum of the odd numbers using filter() and reduce().

How's fold(), aggregate() working?

Example 13

```
nums = sc.parallelize(range(1,10),2)
```

```
odd_num = nums.filter(lambda a : a%2 == 1)
```

```
odd_num.collect()
```

```
[1, 3, 5, 7, 9]
```

```
odd_num.reduce(lambda a,b : a+b)
```

```
25
```

Example 13

```
odd_num.fold(0, lambda a,b : a+b)
```

```
25
```

```
odd_num.fold(1, lambda a,b : a+b)
```

```
28
```

```
odd_num.fold(2, lambda a,b : a+b)
```

```
31
```

Example 13

```
odd_num.fold(0, lambda a,b : a+b)
```

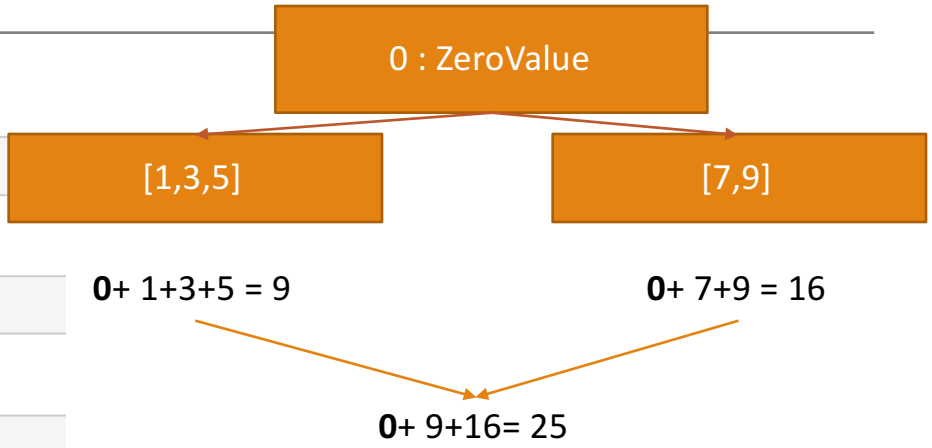
```
25
```

```
odd_num.fold(1, lambda a,b : a+b)
```

```
28
```

```
odd_num.fold(2, lambda a,b : a+b)
```

```
31
```



Example 13

```
odd_num.fold(0, lambda a,b : a+b)
```

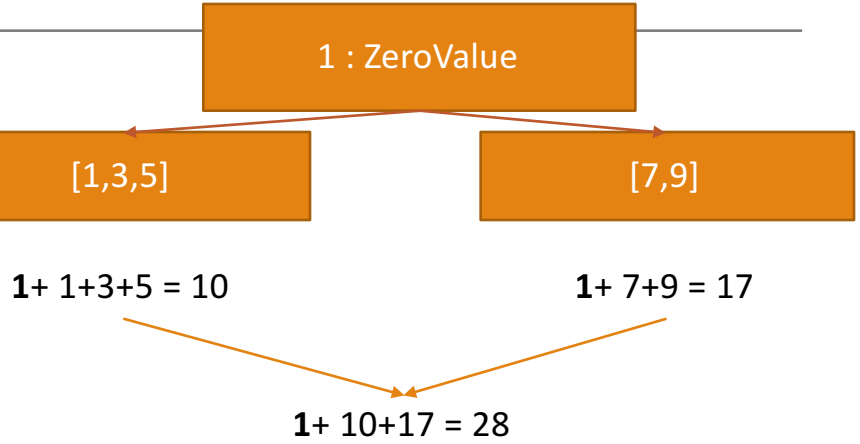
```
25
```

```
odd_num.fold(1, lambda a,b : a+b)
```

```
28
```

```
odd_num.fold(2, lambda a,b : a+b)
```

```
31
```



RESULTS CHANGES DEPENDING ON THE PARTITION SIZE:
nums = sc.parallelize(range(1,10),CHANGE AND TRY)

Example 13

```
seqOp = (lambda x, y: (x[0] + y, x[1] + 1))
```

```
combOp = (lambda x, y: (x[0] + y[0], x[1] + y[1]))
```

```
odd_num.aggregate((0,0), seqOp, combOp)
```

```
(25, 5)
```

Example 13

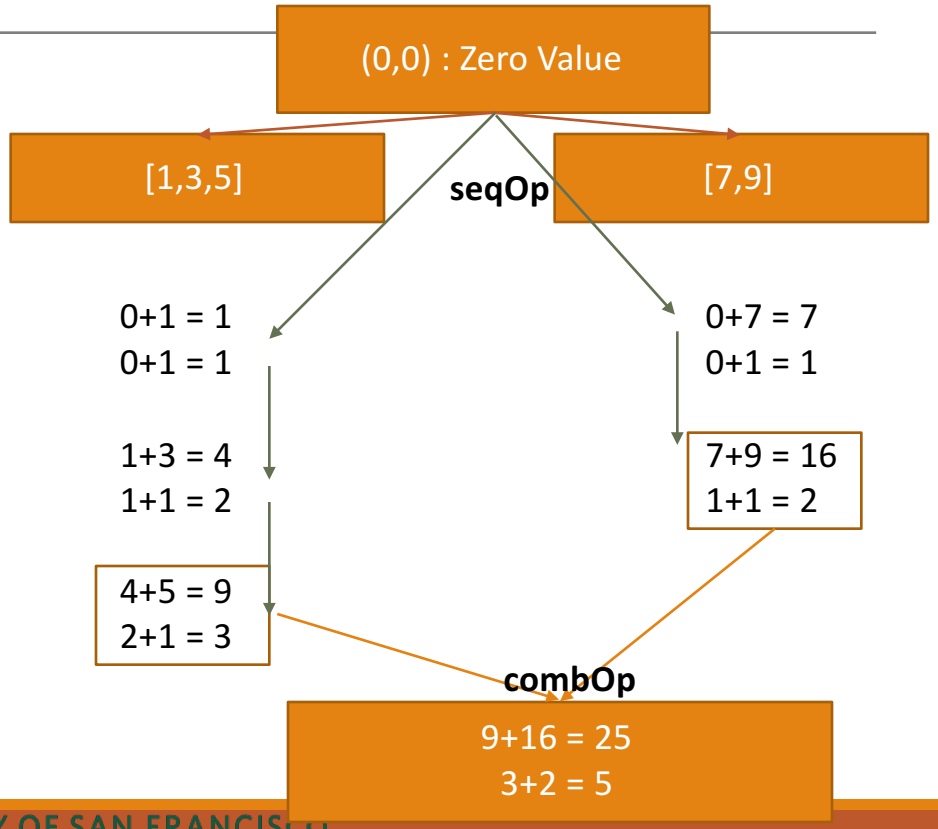
Sum of the elements and number of the elements?

```
seqOp = (lambda x, y: (x[0] + y, x[1] + 1))

combOp = (lambda x, y: (x[0] + y[0], x[1] + y[1]))

odd_num.aggregate((0,0), seqOp, combOp)
```

```
(25, 5)
```



Example 5

Generate an RDD called x with numbers between 1 and 4.

Generate an RDD called y with numbers between 2 and 5.

Do operations

- union()
- Distinct() elements in union
- Subtract()
- Interection()
- Cartesian()

Example 5

- `x = sc.parallelize(range(1,5))`
- `y = sc.parallelize(range (2,6))`
- `x.union(y)`
- `x.union(y).distinct()`
- `x.subtract(y)`
- `x.intersection(y)`
- `x.cartesian(y)`

Example 5

```
In [26]: x = sc.parallelize(range(1,5))
         print(x.collect())
         y = sc.parallelize(range (2,6))
         print(y.collect())

         [1, 2, 3, 4]
         [2, 3, 4, 5]

In [31]: x.union(y).collect()

Out[31]: [1, 2, 3, 4, 2, 3, 4, 5]

In [32]: x.union(y).distinct().collect()

Out[32]: [1, 2, 3, 4, 5]

In [35]: x.subtract(y).collect()

Out[35]: [1]

In [36]: x.intersection(y).collect()

Out[36]: [2, 3, 4]

In [38]: x.cartesian(y).collect()

Out[38]: [(1, 2),
          (1, 3),
          (1, 4),
          (1, 5),
          (2, 2),
          (2, 3),
          (2, 4),
```

Example 6

- `x = sc.parallelize([3,4,1,2])`
- `y = sc.parallelize(range(2,6))`
- `z = x.union(y)`

Try `collect()`, `count()`, `countByValue()`, `top(n)`, `take(n)`, `first()`, `takeSample()` operations on `z`.

Example 6

- `x = sc.parallelize([3,4,1,2])`
- `y = sc.parallelize(range(2,6))`
- `z = x.union(y)`
- `z.collect()`
- `z.count()`
- `z.countByValue()`
- `z.top(2)`
- `z.take(2)`
- `z.first()`

Example 6

```
In [46]: x = sc.parallelize([3,4,1,2])
         y = sc.parallelize(range(2,6))
         z = x.union(y)

Out[46]: UnionRDD[51] at union at NativeMethodAccessorImpl.java:-2

In [37]: z.collect()

Out[37]: [3, 4, 1, 2, 2, 3, 4, 5]

In [38]: z.count()

Out[38]: 8

In [39]: z.countByValue()

Out[39]: defaultdict(int, {1: 1, 2: 2, 3: 2, 4: 2, 5: 1})

In [40]: z.top(2)

Out[40]: [5, 4]

In [41]: z.take(2)

Out[41]: [3, 4]

In [42]: z.first()

Out[42]: 3
```

Example 6

- `z.takeSample(False,20)`
- `z.takeSample(True, 20)`
- `z.takeSample(True, 20)`
- `z.takeSample(True, 20, 1)`
- `z.takeSample(True, 20, 1)`
- `z.takeSample(True, 20, 2)`
- `z.takeSample(True, 20, 2)`

Example 6

```
In [60]: z.takeSample(False,20)
Out[60]: [4, 4, 1, 5, 3, 2, 3, 2]

In [61]: z.takeSample(True, 20)
Out[61]: [3, 1, 2, 2, 3, 2, 2, 3, 4, 3, 1, 2, 3, 5, 2, 2, 1, 4, 5, 2]

In [69]: z.takeSample(True, 20)
Out[69]: [1, 4, 3, 2, 4, 2, 2, 5, 4, 3, 2, 2, 4, 3, 4, 5, 4, 4, 4, 3]

In [65]: z.takeSample(True, 20, 1)
Out[65]: [3, 2, 4, 5, 4, 4, 3, 3, 4, 4, 1, 2, 3, 3, 3, 4, 5, 3, 2, 2]

In [66]: z.takeSample(True, 20, 1)
Out[66]: [3, 2, 4, 5, 4, 4, 3, 3, 4, 4, 1, 2, 3, 3, 3, 4, 5, 3, 2, 2]

In [67]: z.takeSample(True, 20, 2)
Out[67]: [5, 3, 4, 4, 1, 5, 2, 2, 2, 4, 3, 3, 3, 3, 3, 3, 5, 2, 4, 2]

In [68]: z.takeSample(True, 20, 2)
Out[68]: [5, 3, 4, 4, 1, 5, 2, 2, 2, 4, 3, 3, 3, 3, 3, 3, 5, 2, 4, 2]
```

Example 7

In “USF_Mission.txt”, find and count the number of lines with “USF”.

In “USF_Mission.txt”, find and count the number of lines with “University of San Francisco”.

Find the number of lines with either “USF” or “University of San Francisco”.

Example 7

```
file_name = "USF_Mission.txt”
input_RDD = sc.textFile(file_name)
USF_RDD = input_RDD.filter(lambda x: "USF" in x)
USF_RDD.count()
Univ_SF_RDD = input_RDD.filter(lambda x: "University of San Francisco" in x)
Univ_SF_RDD.count()
Univ_RDD = USF_RDD.union(Univ_SF_RDD)
Univ_RDD.count()
type(USF_RDD)
type(Univ_SF_RDD)
type(Univ_RDD)
```

Example 7

```
In [3]: file_name = "USF_Mission.txt"

In [4]: input_RDD = sc.textFile(file_name)

In [5]: USF_RDD = input_RDD.filter(lambda x: "USF" in x)

In [6]: USF_RDD.count()
Out[6]: 8

In [7]: Univ_SF_RDD = input_RDD.filter(lambda x: "University of San Francisco" in x)

In [8]: Univ_SF_RDD.count()
Out[8]: 1

In [9]: Univ_RDD = USF_RDD.union(Univ_SF_RDD)

In [10]: Univ_RDD.count()
Out[10]: 9

In [11]: type(USF_RDD)
Out[11]: pyspark.rdd.PipelinedRDD

In [12]: type(Univ_SF_RDD)
Out[12]: pyspark.rdd.PipelinedRDD

In [13]: type(Univ_RDD)
Out[13]: pyspark.rdd.RDD

In [14]: type(Univ_RDD.count())
Out[14]: int
```

Example 8

Print lines with either “USF” or “University of San Francisco”.

Example 8

- file_name = "USF_Mission.txt”
- input_RDD = sc.textFile(file_name)
- USF_RDD = input_RDD.filter(lambda x: "USF" in x or "University of San Francisco" in x)
- USF_RDD.count()
- for line in USF_RDD.take(USF_RDD.count()):
- print line
- OR
- for lines in USF_RDD.collect():
- print lines

Example 8

```
In [1]: file_name = "USF_Mission.txt"

In [2]: input_RDD = sc.textFile(file_name)

In [3]: USF_RDD = input_RDD.filter(lambda x: "USF" in x or "University of San Francisco" in x)

In [4]: USF_RDD.count()
Out[4]: 9

In [5]: for line in USF_RDD.take(USF_RDD.count()):
            print line

University of San Francisco - Change the World from Here
    Give to USF
    myUSF
    About USF Academics Admission
About USF / Who We Are / Our Values
At USF, excellence is the standard for teaching, scholarship, creative expression, and service. Individuals from all
faiths or with no religious affiliations contribute to the diversity of perspectives and experiences that are essenti
al to our truly global education.
USF students are inspired by this Jesuit tradition to confront the things that degrade human dignity and to expose th
e voices of the underserved, disadvantaged, and poor.
Get a Taste of What's Happening at #USFCA
    Careers at USF
```

RDD Operation – Persist in Memory

RDDs are by default recomputed each time.

However, if you want to reuse an RDD for multiple actions, you can ask Spark to be persist to store the content in memory and query repeatedly.

- `line_with_spark.persist()`

cache() is the same as calling persist() with the default storage level.

Unpersist() lets you manually remove them from the cache.

RDD Operation – Persist in Memory

Persistency Level

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. **This is the default level.**
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

Example 9

- `file_name = "README.md”`
- `lines=sc.textFile(file_name)`
- `line_with_spark = lines.filter(lambda lines : "spark" in lines)`
- `line_with_spark.persist`(StorageLevel.MEMORY_ONLY)

RDD Operation - Passing functions

Passing functions to Spark.

- Most of Spark's transformations, and some of its actions, depend on passing in functions that are used by Spark to compute data.
  1. Pass in lambda functions : for shorter functions.
  2. Pass in top-level functions or locally defined functions.

Example 12

In “USF_Mission.txt”, find lines with “USF” using lambda and a user defined function.

Example 12

```
In [2]: file_name = "USF_Mission.txt"

In [3]: input_RDD = sc.textFile(file_name)

In [4]: USF_RDD = input_RDD.filter(lambda x: "USF" in x)

In [5]: def containsUSF(x):
            return "USF" in x

In [6]: USF_RDD_2 = input_RDD.filter(containsUSF)

In [9]: USF_RDD.collect() == USF_RDD_2.collect()
Out[9]: True
```

Run python script on Spark

```
from pyspark import SparkConf, SparkContext
```

Create SparkConf object to configure the application.

```
conf = SparkConf().setMaster("local").setAppName("AppName")
```

Initializing a SparkContext (SC).

```
sc = SparkContext(conf = conf)
```

For closing Spark, call `sc.stop()`

Run your standalone program.

```
spark-submit yourscript
```

Run python script on Spark

```
from pyspark import SparkConf, SparkContext
```

Create SparkConf object to configure the application.

```
conf = SparkConf().setMaster("local").setAppName("AppName")
```

Initializing a SparkContext (SC).

```
sc = SparkContext(conf = conf)
```

For closing Spark, call `sc.stop()`.

Run your standalone program.

```
Bin/spark-sumbit yourscript
```

Cluster URL
"local" : indicate one thread on the local machine.

Run python script on Spark

```
from pyspark import SparkConf, SparkContext
```

Create SparkConf object to configure the application.

```
conf = SparkConf().setMaster("local").setAppName("AppName")
```

Initializing a SparkContext (SC).

```
sc = SparkContext(conf = conf)
```

For closing Spark, call `sc.stop()`.

Run your standalone program.

```
spark-submit yourscript
```

Identify the application if you're connecting to a cluster.

Example 3

Write a python script (.py) for printing the number of lines in ”README.md” and the first line and run on spark.

Example 3

```
from pyspark import SparkConf, SparkContext

#Create SparkContext
conf = SparkConf().setMaster("local").setAppName("read_lines")
sc = SparkContext(conf = conf)

#Load Data.
lines=sc.textFile("README.md")
print(lines.count())
print(lines.first())

sc.stop()
```

spark-submit your_ex3.py > output_file.txt

Double RDD Functions

RDD containing only Double elements can use several extra functions.

- Statistics functions

sum	Add up the elements in this RDD.
mean	Compute the mean of this RDD's elements.
stats	Return a StatCounter object that captures the mean, variance and count of the RDD's elements in one operation.
variance	Compute the variance of this RDD's elements.
stdev	Compute the standard deviation of this RDD's elements.
sampleStdev	Compute the sample standard deviation of this RDD's elements.
histogram	Compute a histogram of the data using bucketCount number of buckets evenly spaced between the minimum and maximum of the RDD.
sumApprox	Approximate operation to return the mean within a timeout.
meanApprox	Approximate operation to return the mean within a timeout.

https://spark.apache.org/docs/1.0.2/api/java/org/apache/spark/rdd/DoubleRDDFunctions.html

Double RDD Functions

RDD containing only Double elements can use several extra functions.

- Statistics functions
  - `sumApprox and meanApprox`

    `sumApprox/meanApprox(timeout long, confidence double = 0.95)`
    - Calculate the approximate sum and mean in a specified timeframe (milliseconds).
    - If it doesn't return by the timeout, the result computed until that point is returned.
    - Returns a `PartialResult` object.

Double RDD Functions

RDD containing only Double elements can use several extra functions.

- Statistics functions
  - `histogram (2 versions)`
    1. Takes an array of Double values that represent interval limits and returns an array with counts of elements belonging to each interval.
    2. Takes a number of intervals, which is used to split the input data range into intervals of equal size and returns a tuple whose second element contains the counts.

https://spark.apache.org/docs/1.0.2/api/java/org/apache/spark/rdd/DoubleRDDFunctions.html

Example 10

Read “grade” and calculate sumApproximate() and meanApproximate() within 1 milisecond.

Example 10

```
file_name = "grade"
```

```
lines = sc.textFile(file_name)
```

```
grades = lines.map(lambda a : int(a))
```

```
grades.sumApprox(1)
```

```
500002731.0
```

```
grades.sum()
```

```
500002731
```

```
grades.meanApprox(1)
```

```
50.000273099999646
```

```
grades.mean()
```

```
50.000273099999646
```

Example 11

Read “grade” and generate histogram using

1. intervals of [0,10,20,30,40,50,60,70,80,90,100]
2. and a 10 interval size

Example 11

```
file_name = "grade"
```

```
lines = sc.textFile(file_name)
```

```
grades = lines.map(lambda a : int(a))
```

```
grades.histogram([0,10,20,30,40,50,60,70,80,90,100])
```

```
([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
 [989846,
  991921,
  990442,
  988840,
  989541,
  990181,
  988575,
  989670,
  991612,
  1089372])
```

```
grades.histogram(10)
```

```
([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
 [989846,
  991921,
  990442,
```

Reference

Distributed Computing with Spark, Reza Zadeh, http://stanford.edu/~rezab/slides/bayacm_spark.pdf.

Spark Online Documentation : http://spark.apache.org/docs/latest/

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

Zecevic, Petar, et al. Spark in Action, Manning, 2016.