MSDS 694
Distributed Computing

DIANE WOODBRIDGE, PH.D

Announcement

Programming Assignment : Nov 4th 10am

Group Project

- Task 1 (Nov 5) : Data Selection

Office Hour : Tuesday 9:15-10 am

Professionalism

- **Class Attendance**
  - No cellphones, social media, slack, texting during the class .
  - **Please close your laptop except for doing programming exercises.**

- **Assignment**
  - No late submissions are allowed.
  - Do not share any homework and exam files - All the codes including the last 6 years will be tested by Moss (Measure Of Software Similarity).
  - Make sure that your code runs in Python 3.10 and Pyspark 3.3.

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- Transformation
- Action (Week 3)

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- Transformation
- Action (Week 3)

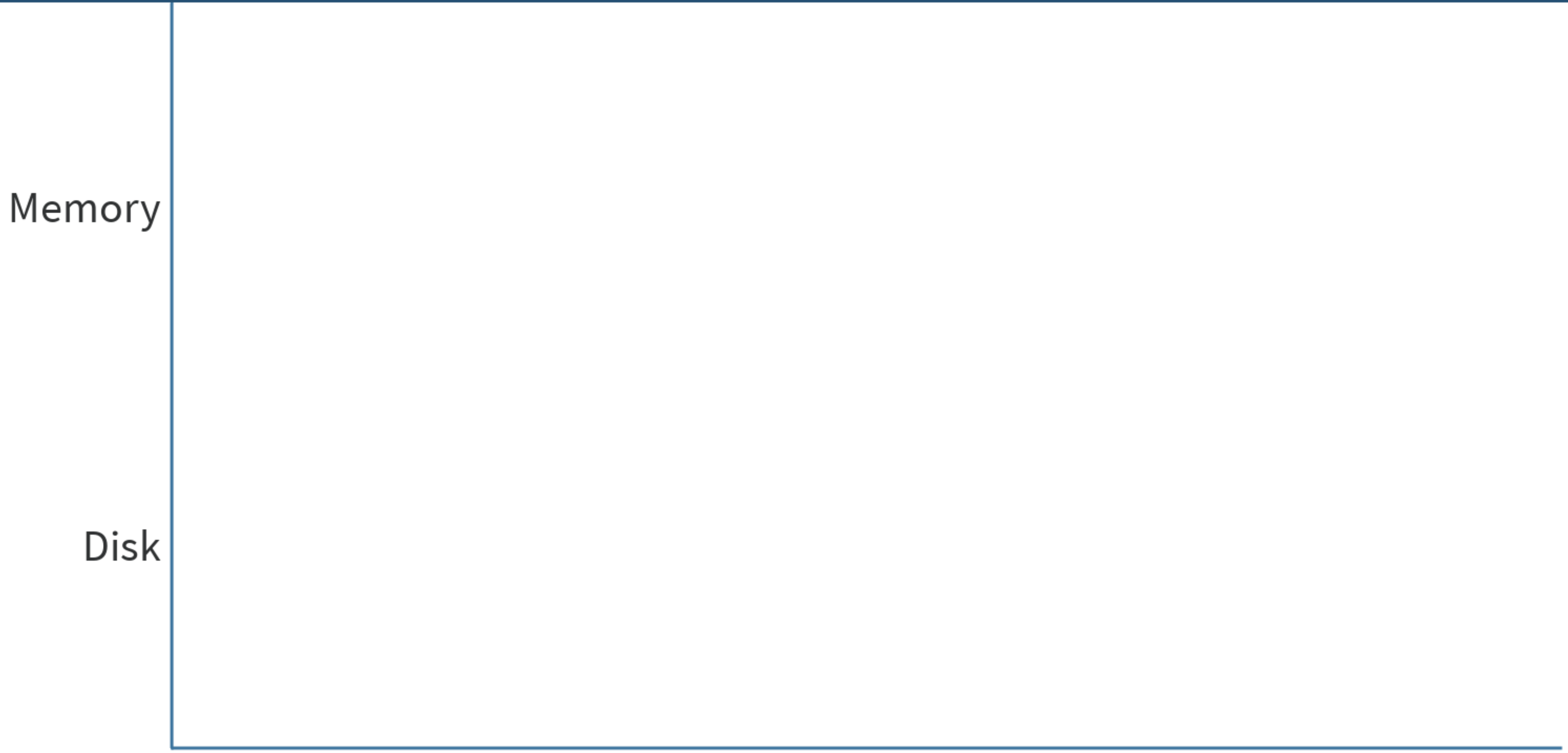
Hadoop MapReduce

First introduced the MapReduce concept.

Hadoop MapReduce solved issues of..

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

Which one is faster for I/O ?



Hadoop MapReduce

Limitations

- Hadoop MapReduce is powerful, but can be slow.
  - MapReduce job results need to be stored in Hadoop File System (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.

Spark outperforms Hadoop MapReduce

WHY ??

	Hadoop MapReuce	Spark
Speed	Decently fast	100 times faster than Hadoop
Ease of Use	No interactive modes and Hard to learn	Provides interactive modes and Easy to learn
Costs	Open source	Open source
Data Processing	Batch Processing	Batch Processing + Streaming
Fault Tolerance	Fault Tolerant	Fault Tolerant

Hadoop MapReduce vs. Spark

Spark

- You can write distributed programs in a manner similar to writing local programs.
  - Spark **abstracts 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.

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations

- Transformation
- Action (Week 3)

Lightning-fast cluster computing

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

Achieves

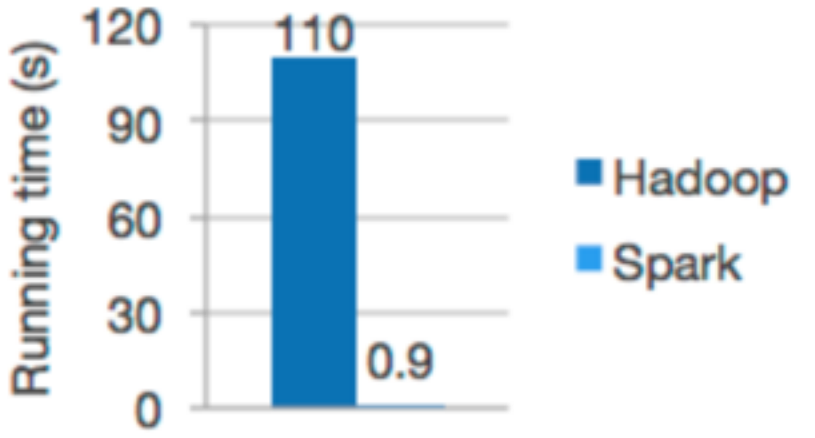
1. Speed.
2. Ease of Use.
3. Generality.
4. Runs everywhere.

http://spark.apache.org/

Lightning-fast cluster computing

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

- Apache Spark has an advanced DAG (Directed Acyclic Graph) execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

http://spark.apache.org/

APACHE Spark Lightning-fast cluster computing

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

1. Spark offers high-level operators that make it easy to build parallel apps. This allows developer to focus on logic rather than infrastructure.
2. Developers can use Spark interactively from the Scala, Python and R shells.

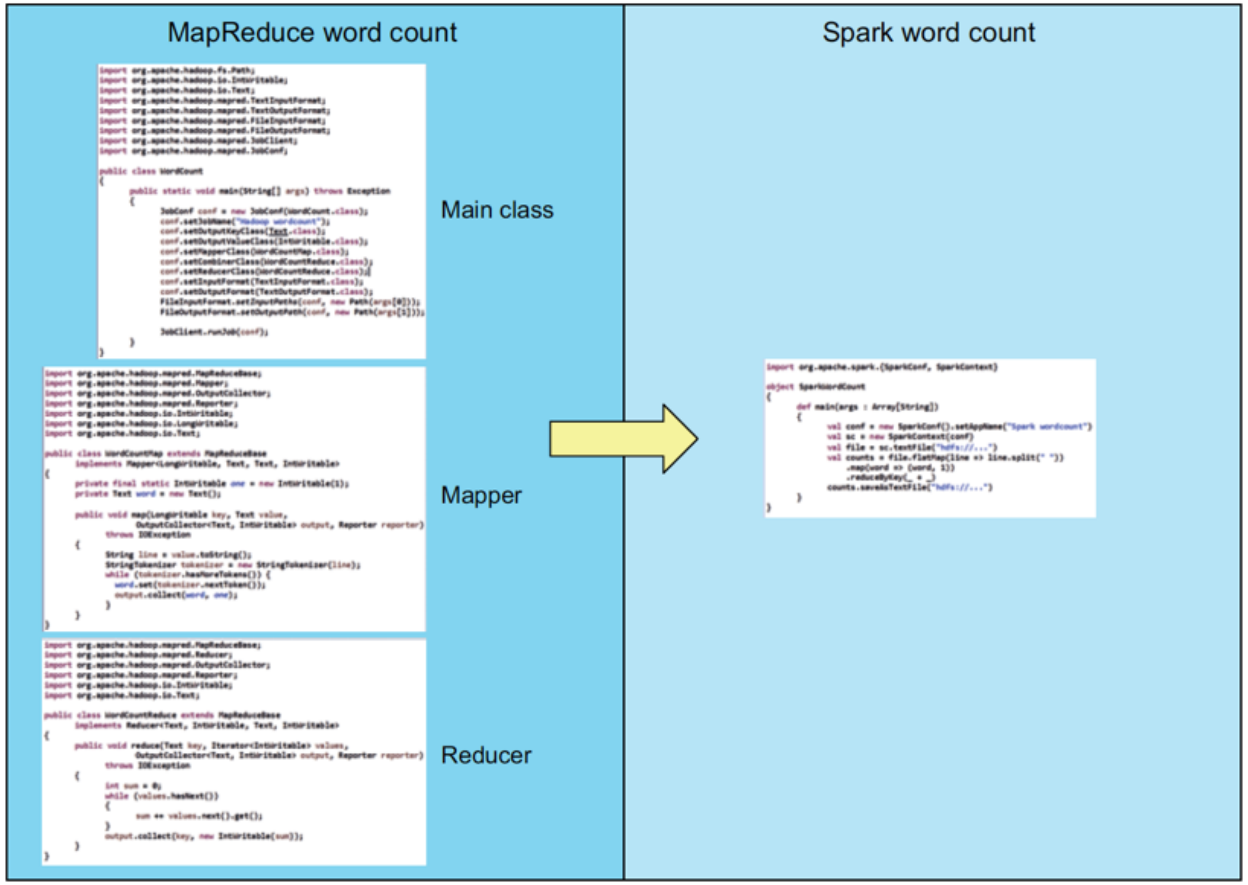
```
text_file = spark.textFile("hdfs://...")

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/

APACHE Spark

Lightning-fast cluster computing

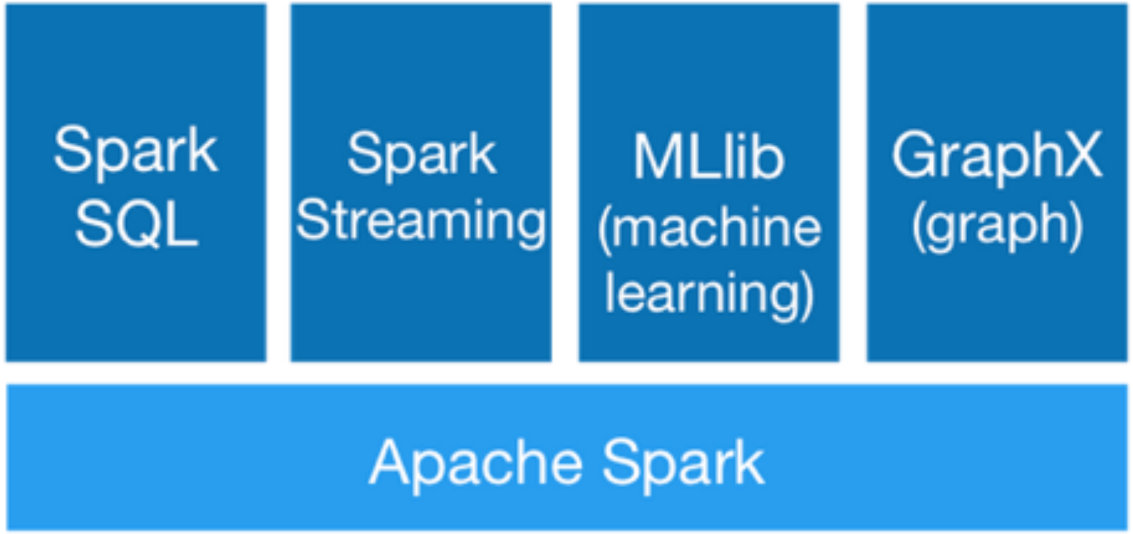


http://spark.apache.org/

Lightning-fast cluster computing

3. 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.



http://spark.apache.org/

Lightning-fast cluster computing

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

http://spark.apache.org/

APACHE Spark™ *Lightning-fast cluster computing*

Community

- Most active open source community for big data.
- Developed collaboratively by a community of hundreds of developers from hundreds of organizations.

http://spark.apache.org/powered-by.html

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- Transformation
- Action (Week 3)

Spark Stack

Cluster Managers

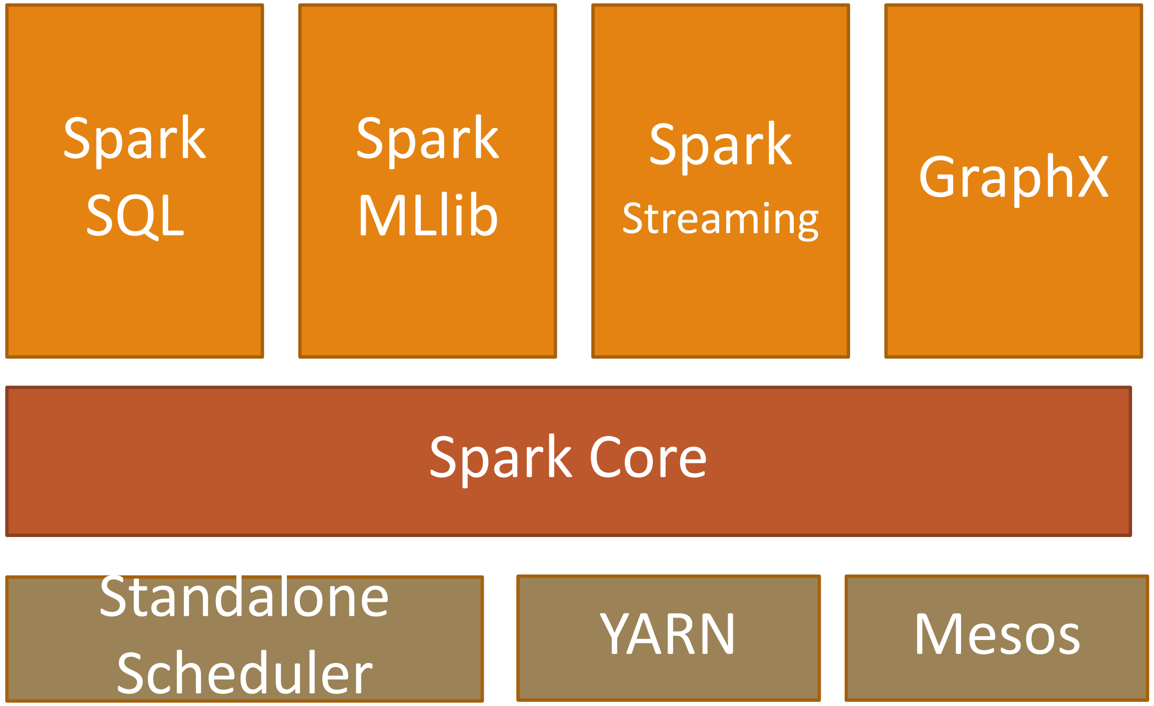
Spark Core

Spark SQL

Spark Mllib

Spark Streaming

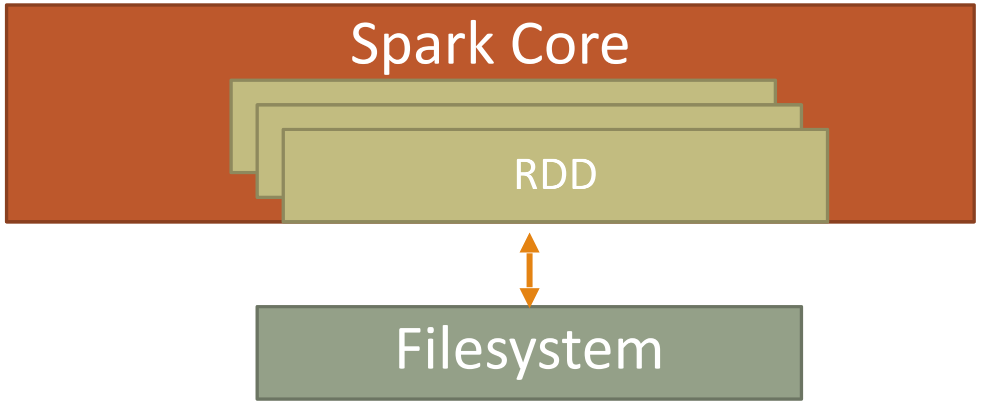
GraphX



Spark Components

Spark Core

- Contains Spark functionalities required for running jobs and needed by other components.
  - Resilient Distributed Dataset (RDD) : Abstraction of a distributed collection of items with operations and transformation applicable to the dataset.
  - Fundamental functions such as basic I/O functionalities, networking, security, scheduling and data shuffling.



Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- Transformation
- Action (Week 3)

Spark Examples

Extract-transformation-load (ETL) operations

Predictive analytics

Machine learning

Data access operation (SQL queries and visualizations)

Text mining and text processing

Real-time event processing

Graph applications

Pattern Recognition

Recommendation engines

And many more..

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

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

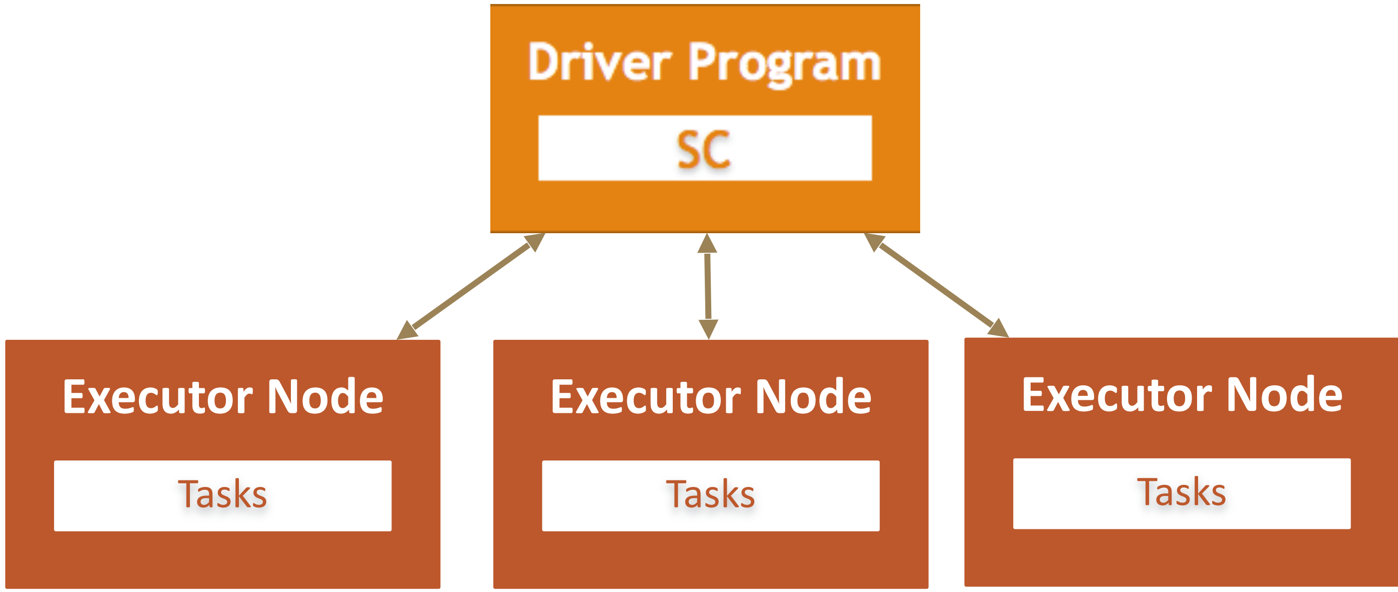
RDD Operations

- Transformation
- Action (Week 3)

Distributed Data Sets

Through a SparkContext (sc) object, a driver program can access Spark.

- SparkContext
  - Application instance representing the connection to the spark cluster.
  - Instantiated at the beginning of a Spark application and created by spark driver.
  - Once having a SparkContext, you can use it to build resilient distributed data sets (RDDs).



Resilient Distributed Dataset (RDD)

RDD

- Distributed datasets that consists of records.

Key Ideas

- Distributed
  - The data in RDDs is divided into one or many partitions and distributed as in-memory collections of objects across worker nodes.
- Immutable
  - Read-only : Once created, RDDs never change.
- Resilient
  - Automatically rebuilt on failure.
  - RDDs track lineage info to rebuild lost data. (Instead of replication.)

Today’s Example

Create an RDD from supervisor_sf.tsv that includes zip and supervisor_id separated by tab (‘\t’)

Return unique zip code associated with id, 9 or 10 in ascending order.

94102	8
94102	6
94102	3
94102	5
94103	8
94103	9
94103	10
94103	6
94103	3
94103	5
94104	6
94104	3
94105	6
94105	3
94107	10
94107	6
94108	6
94108	3
94109	2
94109	6
94109	3
94109	5
94110	8
94110	11
94110	9
94110	10
94111	6
94111	3
94112	7
94112	8
94112	11
94112	9
94112	10
94114	7
94114	8
94114	5
94115	2
94115	1
94115	5
94116	7
94116	4
94117	1
94117	7
94117	8
94117	5
94118	2

Creating and Distributing Data

Two ways of creating distributed data sets.

1. Loading an external data and return it as an RDD of Strings.

```
lines = sc.textFile("README.md", minPartitions=None)
```

   - `minPartitions`: **Suggested** minimum number of partitions for the resulting RDD.

2. 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!"],numSlices=None)
```

   - *`numSlices`* : the number of partitions the data would be parallelized to.

http://spark.apache.org/docs/latest/api/python/pyspark.htm
https://spark.apache.org/docs/latest/api/java/org/apache/spark/SparkContext.html
http://spark.apache.org/docs/latest/tuning.html#level-of-parallelism
https://spark.apache.org/docs/latest/api/python/_modules/pyspark/context.html#SparkContext.textFile

Checking Data Distribution

1. Check the number of partitions.

   `lines.getNumPartitions()`

2. Coalescing data within each partition and see what is in each partition.

   `lines.glom().collect()`

   - `glom()` – Return an RDD created by coalescing all elements within each partition into a list.
   - `collect()` - Return a list that contains all of the elements in this RDD.

http://spark.apache.org/docs/latest/api/python/pyspark.htm

Ex01

1. Create an rdd using data from supervisor_sf.tsv
2. Load data to 8 partitions.
3. See the data in RDD.
4. How many data are in RDD?
5. See how data is distributed on 8 partitions.

Ex01

1. Create an rdd using data from supervisor_sf.tsv
   - Load data to 8 partitions.

```
input_rdd = sc.textFile("../data/SF_business/supervisor_sf.tsv", 8)  # Load data from the file.
```

2. See the data in RDD.

```
input_rdd.collect()
```

```
['94102\t8',
 '94102\t6',
 '94102\t3',
 '94102\t5',
 '94103\t8',
 '94103\t9',
 '94103\t10',
 '94103\t6',
 '94103\t3',
 '94103\t5',
 '94104\t6',
 '94104\t3',
```

Ex01

3. How many data are in RDD?

```
input_rdd.count()
```

```
75
```

4. See how data is distributed on 8 partitions.

```
input_rdd.glom().collect()
```

```
[['94102\t8',
  '94102\t6',
  '94102\t3',
  '94102\t5',
  '94103\t8',
  '94103\t9',
  '94103\t10',
  '94103\t6',
  '94103\t3',
  '94103\t5'],
 ['94104\t6',
  '94104\t3',
  '94105\t6'
```

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations

- **Transformation**
- Action (Week 3)

Note :
Python Lambda Expression

Python Lambda Expression

- A shortened way to define functions inline.
  - Create an anonymous function using the “lambda” keyword at runtime.
  - Keyword : lambda

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

Ex02

```
In [1]: def add_2(x):
            return x+2

In [2]: add_2_lambda = lambda x : x + 2

In [3]: add_2(2)

Out[3]: 4

In [4]: add_2_lambda(2)

Out[4]: 4
```

RDD Operations

Two types

- Transformation
  - Perform functions against each element in an RDD and return a new RDD.
  - **Lazy evaluation** – operations are only evaluated when an action is requested.

- Action
  - Trigger a computation and return a value to the Spark driver.

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

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- **Transformation**
- Action (Week 3)

Appendix : Running a Spark application (.py) using the spark-submit

RDD Operations - Transformation

Transformation

- Construct a new RDD from an existing RDD.
  - Doesn’t change the original RDDs.

```
splitted_input_rdd = input_rdd.map(lambda x : x.split('\t'))
```

- Lazy Evaluation.
  - Computation doesn’t take place until an action is triggered.

RDD Operations - Transformation

Transformation Operation Types

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
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).
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
distinct()	Return a new dataset that contains the distinct elements of the source dataset.
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.
subtract(*otherDataset***)**	Return each value in self that is not contained in oterDataset.
cartesian(*otherDataset***)**	Return the Cartesian product of the RDD and otherDataset.
sortBy(*func*, *ascending=True*)	Sorts this RDD by the given *func*.

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

RDD Operations - Transformation

Narrow Transformation - Transform data without any shuffles.

- map(*func*)
  - Return a new RDD by applying a function to each element of this RDD.
- flatmap(*func*)
  - Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
- filter(*func*)
  - Return a new RDD containing only the elements that satisfy a predicate.

http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://databricks.com/glossary/what-are-transformations

ex01

5. Convert data in RDD to a tuple of integers.
    - Ex. [94102, 8], [94102, 6], etc.
    - What happens if you use flatMap() instead of map()?

ex01

5. Convert data in RDD to a tuple of integers.
   - Ex. [94102, 8], [94102, 6], etc.
   - What happens if you use flatMap() instead of map()?

```
splitted_rdd = input_rdd.map(lambda x: x.split('\t'))
```

```
zip_id = splitted_rdd.map(lambda x: [int(x[0]), int(x[1])])
```

```
zip_id.collect()
```

```
[[94102, 8],
 [94102, 6],
 [94102, 3],
 [94102, 5],
 [94103, 8],
 [94103, 9],
 [94103, 10],
 [94103, 6],
 [94103, 3],
 [94103, 5],
 [94104, 6],
 [94104, 3],
```

Variable Inspe

When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join

Which of the followings are action? (Multiple)

glom()

collect()

count()

map()

filter()

RDD Operations - Transformation

Wide Transformation - Transformation that may require data shuffling

- distinct()
  - Return only one of each element.
- sortBy(*func*, *ascending=True*)
  - Sorts this RDD by the given *func*
- Set Operation - Format : rdd1.operator(rdd2)
  - 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)

http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://databricks.com/glossary/what-are-transformations

ex01

6. Return unique zip code associated with id, 9 or 10.
7. Order zip in ascending order.

ex01

6. Return unique zip code associated with id, 9 or 10.

```
zip = zip_id.filter(lambda x: x[1] == 9 or x[1] == 10)\
            .map(lambda x: x[0])
```

```
distinct_zip = zip.distinct()
```

7. Order zip in ascending order.

```
ordered_distinct_zip = distinct_zip.sortBy(lambda x: x, ascending=False)
```

```
ordered_distinct_zip.collect()
```

```
[94158, 94134, 94124, 94112, 94110, 94107, 94103]
```

Stopping the Spark Context

Shut down the SparkContext.

```
sc.stop()
```

ex03

We are interested in the zipcodes including 94103, 94105 and 94107.

See what is the output of union, intersection, subtract and cartesian between the given 3 zip codes and output of ex01.

ex03

We are interested in the zipcodes including 94103, 94105 and 94107.

See what is the output of union, intersection, subtract and cartesian between the given 3 zip codes and output of ex01.

```
zip_id_2 = sc.parallelize([94103, 94105, 94107])
```

```
union_zip = zip_id_2.union(ordered_distinct_zip)
union_zip.collect()
```

```
[94103, 94105, 94107, 94158, 94134, 94124, 94112, 94110, 94107, 94103]
```

```
intersection_zip = zip_id_2.intersection(ordered_distinct_zip)
intersection_zip.collect()
```

```
[94107, 94103]
```

```
subtract_zip = zip_id_2.subtract(ordered_distinct_zip)
subtract_zip.collect()
```

```
[94105]
```

```
cartesian_zip = zip_id_2.cartesian(ordered_distinct_zip)
cartesian_zip.collect()
```

```
[(94103, 94158),
 (94103, 94134),
 (94103, 94124),
```

RDD Operations - Transformation

Transformation Operation Types

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
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).
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
distinct()	Return a new dataset that contains the distinct elements of the source dataset.
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.
subtract(*otherDataset***)**	Return each value in self that is not contained in oterDataset.
cartesian(*otherDataset***)**	Return the Cartesian product of the RDD and otherDataset.
sortBy(*func*, *ascending=True*)	Sorts this RDD by the given *func*.

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

Contents

Hadoop MapReduce vs. Spark

Spark Overview

Spark Components

Spark Examples

Resilient Distributed Dataset (RDD)

RDD Operations
- Transformation
- Action (Week 3)

Week 2 - Comments (What you liked/disliked so far? What should I do for you?)

“ Thanks for changing the format of the variable names in the homework pdf :) ”

Tips

How to work without internet

- When creating a Spark Context, add configuration to use localhost.

```
conf = pyspark.SparkConf().set("spark.driver.host", "localhost")
sc = pyspark.SparkContext(appName="app", conf=conf)
```

How to get rid of warning

- Change the log level.

```
sc.setLogLevel('OFF')
```

Reference

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

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