Spark Tutorial 2

Let’s make some practice!

Download and launch the shell

We'll be using spark 2.3.2

see http://spark.apache.org/downloads.html

1. Go to your browser and download the latest version
2. Unzip the archive
3. Open your terminal and go to the extracted directory
4. Go in the folder ./**bin/**
5. Launch the script: **./pyspark**

PySpark

```
nadai@sibiu:/data/nadai/spark-2.3.0-bin-hadoop2.7$ ./bin/pyspark
Python 3.6.5 (default, May 26 2018, 18:07:52)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
18/10/30 17:04:27 WARN Utils: Your hostname, sibiu resolves to a loopback address: 127.0.1.1; using 192.168.159.39 instead (on interface em1)
18/10/30 17:04:27 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
18/10/30 17:04:28 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/10/30 17:04:33 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Python version 3.6.5 (default, May 26 2018 18:07:52)
SparkSession available as 'spark'.
>>>
```

Initializing Spark

Create a Spark Context
(already provided in the notebook)

```
conf = SparkConf().setAppName(appName).setMaster(master)
sc = SparkContext(conf=conf)
```

appName = Name of the application to show on the cluster UI

master = Spark URL ('spark://ip-address:7077') or 'local'

Resilient Distributed Datasets [RDD]

Parallelized Collections

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
                vs.
           sc.parallelize(data, 10)
```

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

Resilient Distributed Datasets [RDD]

External Datasets

- HDFS
- Cassandra
- Hbase
- Text File
- Amazon S3
- Sequence File
- NO Database (SparkSQL does it!)

```
distFile = sc.textFile("data.txt")
                        (with optional number of partitions)

sc.textFile          -> reads a file and returns one record per line
sc.wholeTextFiles    -> reads a folder as a pair <filename, content>
```

Resilient Distributed Datasets [RDD]

RDD Operations

- TRANSFORMATIONS create a new dataset from an existing one (e.g. map)
- ACTIONS return a value to the driver program after running a computation on the dataset (e.g. reduce)

```
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Remember: LAZY COMPUTATION!

Solution: persist or cache RDD in memory (or disk)

If you want to use lineLengths again:

```
lineLengths.cache()
```

Resilient Distributed Datasets [RDD]

Passing Functions to Spark

```
lines = sc.textFile("data.txt")

def computeLength(s):
    return len(s)

lineLengths = lines.map(computeLength)

def sumLengths(a, b):
    return a + b

totalLength = lineLengths.reduce(sumLenghts)
```

Resilient Distributed Datasets [RDD]

Understanding Closures

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
def increment_counter(x):
    global counter
    counter += x

rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Remember: do NOT mutate the state in a loop or locally defined method

If you need to change the state -> use an ACCUMULATOR

In the same way, do not try to print an RDD with rdd.foreach(println) in a cluster!

Resilient Distributed Datasets [RDD]

Working with Key-Value Pairs

Key-Value pair is built-in in python -> (key, value)
Simply call the desired operation

```
lines = sc.textFile("data.txt")

pairs = lines.map(lambda s: (s, 1))

counts = pairs.reduceByKey(lambda a, b: a + b)
```

Resilient Distributed Datasets [RDD]

Transformations

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- sample
- union
- intersection
- distinct
- groupByKey
- reduceByKey
- aggregateByKey
- sortByKey

- join
- cogroup
- cartesian
- pipe
- coalesce
- repartition
- repartitionAndSortWithinPartitions

Resilient Distributed Datasets [RDD]

Actions

- reduce
- collect
- count
- first
- takeSample
- takeOrdered
- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- countByKey
- foreach

Shared Variables

Broadcast Variables

- Keep a read-only variable cached on each machine rather than shipping a copy of it with tasks (e.g. give to every node a copy of a large dataset in an efficient manner)

```
broadcastVar = sc.broadcast([1, 2, 3])
```

```
broadcastVar.value()
```

Shared Variables

Accumulators

- Variables that are only “added” to through an associative and commutative operation (e.g. use for implementing counters)

```
accum = sc.accumulator(0)

sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))

accum.value
```

Shared Variables

Custom Accumulator

```
class VectorAccumulatorParam(AccumulatorParam):
    def zero(self, initialValue):
        return Vector.zeros(initialValue.size)

    def addInPlace(self, v1, v2):
        v1 += v2
        return v1

# Then, create an Accumulator of this type:
vecAccum = sc.accumulator(Vector(...), VectorAccumulatorParam())
```

Deploying to a cluster

Use the CommandLine Tool

```
$SPARK_HOME/bin/spark-shell    [options] \
                               --master spark://ip-address:7077 \
                               /path/to/script.py \
                               [extra-options for the script]
```

Using Pyspark

```
$SPARK_HOME/bin/pyspark
```

and then write your own code!

Using a Notebook

Code Repository

https://github.com/forons/BigDataExamples/

Today's exercise:

create an account on

https://databricks.com/try-databricks

My first application

Python

```
1. Data = [1, 2, 3, 4, 5]
2. dataRDD = sc.parallelize(data)
3. def filterOperation(number):
       return number < 4
4. dataRDD.filter(filterOperation).collect()
```

Simple WordCount application

Python

```
1. from operator import add
2. f = sc.textFile("README.md”)
3. wc = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x,1))
                                          .reduceByKey(add)
4. wc.saveAsTextFile("wc_out")
```

Simple WordCount application in MP (java)

```java
public class WordCount {
  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...] <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

```scala
val f = sc.textFile(inputPath)
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).saveAsText(outputPath)
```

WordCount in 3 lines of Spark

dbTrento

We can start

- **Find all words that are palindrome**

  { rotator } { gig } { pop } { level } { civic }

- **Find the words that occur exactly 5 times**

- **Group word by their number of occurrences**

- **Group anagrams**

  { mean, name } { flow, wolf }

Contacts

For any problem, send a mail to:

daniele.foroni@unitn.it