Lab

Rana Hussein
rana@exascale.info

UNI
FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

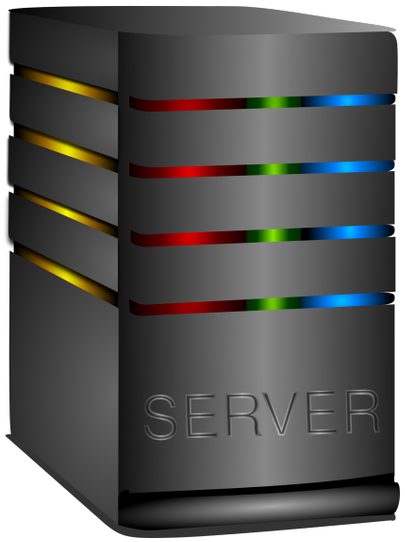
Lab Outline

- Cluster Connection recap
- Running Spark Shell
- Spark examples
- How to submit a job
- Demo

Using the Cluster

your laptop

SSH



diufrm202

Yarn

XI Cluster

ssh bdi18@diufrm202

hdfs dfs -ls …

spark-submit \
--master yarn …

Spark!

- Few examples…
- … and how to run them!
- It'll all be in Scala
- Spark is also available in Python …

Scala Recap

Lightweight Syntax

Anonymous functions
Higher - order functions

```
//Variables:
val x = 1 + 1
println(x) // 2

val x: Int = 1 + 1 // Types of values can be inferred, but you can also explicitly
state the type

//Functions:
(x: Int) => x + 1  //anonymous function (i.e. no name) that returns a given integer
plus one

val add = (x: Int, y: Int) => x + y  // multiple parameters
println(add(1, 2)) // 3

val getTheAnswer = () => 42 // no parameters

def add(x: Int, y: Int): Int = x + y // To define a method: def is followed by a name,
parameter lists, a return type, and a body
println(add(1, 2)) // 3

def getSquareString(input: Double): String = {  // Multiline method
  val square = input * input
  square.toString
}
```

Spark-Shell

- You can play with spark in an interactive fashion!

```
bin/spark-shell \
--master yarn \
--deploy-mode client \
--num-executors 3 \
--driver-memory 1g \
--executor-memory 1g
```

You can run a local spark-shell by using —master local

Let’s Do it Together!

./bin/spark-shell

```
var firstRDD = sc.parallelize(1 to 8)
firstRDD.collect()

firstRDD.saveAsTextFile("/user/<your_user>/firstRDD")

firstRDD.partitions.size
var secondRDD = sc.parallelize(1 to 8,2)

secondRDD.saveAsTextFile("/user/<your_user>/secondRDD")

var thirdRDD = sc.textFile("/user/<your_user>/file_example.txt")

val filteredRDD = thirdRDD.filter(line => line.contains(“testword")).count()

var tuple = thirdRDD.map(name => (name.charAt(0), name))
var values = tuple.groupByKey()
var counts = values.mapValues(name => name.toSet.size)
counts.collect()
```

Remember Word Count in MR!

```
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();
    Job job = Job.getInstance(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);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Word Count in Spark

```
def main(args: Array[String]): Unit = {
  val conf = new SparkConf()
    .setAppName("WordCount")

  val sc = new SparkContext(conf)

  val inputPath = args(0)
  val outputDataset = args(1)

  val tweets: RDD[String] = sc.textFile(tweetsPath)

  // what is the difference between map and flatMap?
  val wordCounts = tweets.flatMap(line => line.split("\\s+"))
    .groupBy(identity)
    .map { case (word, words) => s"$word\t${words.size}" }

  wordCounts.saveAsTextFile(outputDataset)
```

Each element of the RDD is a string

Reads a text file **or a dir**. *1line => 1RDD entry*

Type?

GroupBy returns a pair **(group, elements)**

Saves the RDD in a text file, one line per element

This function is applied to all elements of the RDD

Word Count++

```scala
val wordCounts: RDD[(String, Int)] = tweets
  .flatMap(line => line.split("\\s+"))
  .groupBy(identity)
  .map { case (word, words) => (word, words.size) }

val sortedCounts = wordCounts
  .sortBy { case (_, count) => -count }
  .map { case (word, count) => s"$word\t$count" }

sortedCounts.saveAsTextFile(outputDataset)
```

Key -> Value RDDs have lots of useful methods (**join**, **reduceByKey**, …)

Homework: rewrite the word counter by using **reduceByKey**

Hint: reduce on a collection of pairs (<word>, 1)

Building and Running

1. Create a fat-jar
   - if you use SBT: **`sbt assembly`** from the root of your project
   - libraryDependencies += **"org.apache.spark"** % **"spark-core_2.11"** % **"2.0.2"** % **"provided"**

2. Use **bin/spark-submit** (from ‘bundle/spark2’)

3. If you are running in local mode, add “file:////“ before the path of your files.

```
spark-submit \
--master local[*] \
--class xi.examples.HelloSpark \
<your jar file> \
path_in_your_local_machine \
path_in_your_local_machine
```

Test in local mode first…

```
spark-submit \
--master yarn \
--deploy-mode cluster \
--num-executors 5 \
--driver-memory 1g \
--executor-memory 1g \
--class xi.examples.HelloSpark \
<your jar file> \
path_in_hdfs \
path_in_hdfs
```

… and then run in the cluster!

Demo

- SBT project template on ILIAS
- build a fat-JAR file with SBT assembly
- submit the job!

How to create the Previous Dataset

Input: TSV text files (location, id, txt, date)

```
val tweetsRaw: RDD[String] = sc.textFile(tweetspath)


val split = tweetsRaw
  .map(line => line.trim.split("\t"))
  .filter(split => split.length == 4)
  .map { split =>
    val Array(location, tweetId, text, date) = split
    val sdf = new SimpleDateFormat("d.M.y k:m")
    (sdf.parse(date), tweetId.toLong, location, text)
  }


val FilterLocationText = split.filter { case (_, _, location, _)
=>  location == "San Francisco, CA”}
.map { case (_, _, _,text) => text}


FilterLocationText.saveAsTextFile(outputdataset)
```

What is the type of split RDD?

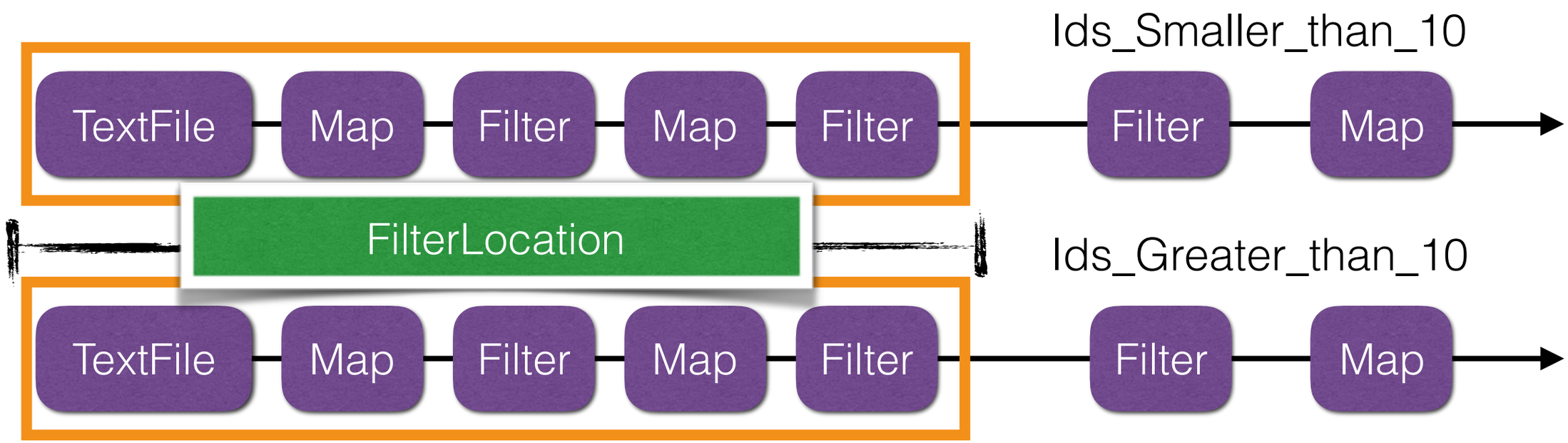
Caching

- What's wrong with this code?

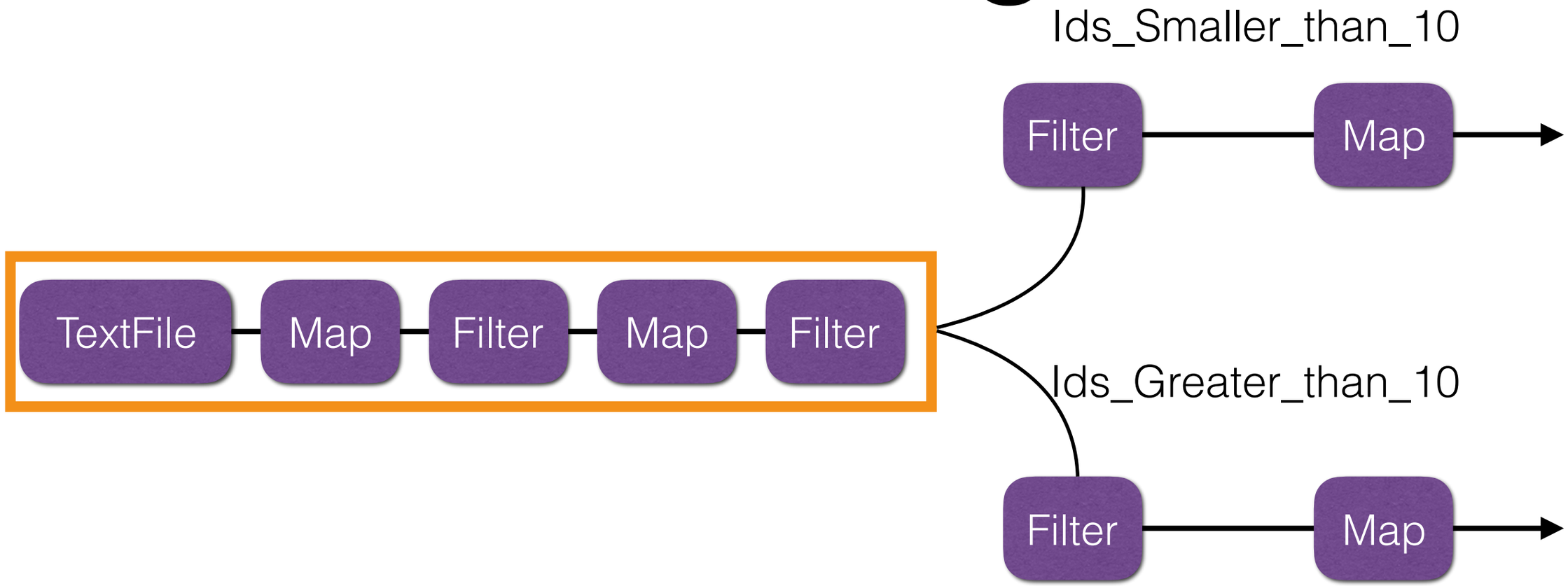
```
val FilterLocation: RDD [(Date, Long, String, String)]= ……….

FilterLocation.filter { case (_, tweetId, _,_) => tweetId < 10}
  .map(_._4)
  .saveAsTextFile(outputDataset + "Ids_Smaller_than_10")

FilterLocation.filter { case (_, tweetId, _,_) => tweetId > 10}
  .map(_._4)
  .saveAsTextFile(outputDataset + "Ids_Greater_than_10")
```



Caching




```
val FilterLocation = tweetsRaw
  .map(line => line.trim.split("\t"))
  .filter(split => split.length == 4)
  .map { split =>
    val Array(location, tweetId, text, date) = split
    val sdf = new SimpleDateFormat("d.M.y k:m")
    (sdf.parse(date), tweetId.toLong, location, text)
  }
  .filter { case (_, _, location, _) =>  location ==
   “San Francisco, CA"}
  .cache
```

1) What’s the difference between cache() and persist()?

2) What are the StorageLevels?

Pointers

- http://spark.apache.org/docs/latest/programming-guide.html
- Spark operators explained: http://nbviewer.jupyter.org/github/jkthompson/pyspark-pictures/blob/master/pyspark-pictures.ipynb
- SBT: http://www.scala-sbt.org/
- SBT assembly (builds fat jars): https://github.com/sbt/sbt-assembly
- Scala: http://scala-lang.org/
- Scala cheat sheet http://alvinalexander.com/scala/scala-cheat-sheet-reference-examples

Thanks!

Homework on ILIAS!

Send homework to rana@exascale.info until:

1st of February 2019