

Parallel Data with MapReduce and Spark (Part 2, Spark)

Daniel Lawson — University of Bristol

Lecture 11.1.2 (v1.0.2)

Signposting

- ▶ Block 11 on **parallel infrastructure** is paired with Block 10 on **parallel algorithms**.
- ▶ This week we cover **parallel data frameworks**, which use parallel algorithms on distributed datasets.
- ▶ Lecture 11.1 covers all the abstract content, whilst the workshop 11.2 focusses on practicalities.
- ▶ This is 11.1.2, which covers:
 - ▶ Spark overview
 - ▶ Resilient Distributed Datasets
 - ▶ Spark

Spark

- ▶ Like Hadoop, Spark accesses data stored on HDFS via YARN. It offers many additional features, including:
 - ▶ **Data abstractions**, both data table and graph-based;
 - ▶ Interactive, **stateful** data representations;
 - ▶ Interfaces for multiple programming languages (Scala, Python, Java);
 - ▶ MLlib, a distributed machine learning toolkit.
- ▶ We'll focus on **pyspark**.
 - ▶ This means that we access the features of Spark through python code.
 - ▶ It is still necessary to learn the **concepts** of Spark.
 - ▶ The code that we write will be python, though the setup of a Spark session involves very specific commands.

Resilient Distributed Dataset (RDD)

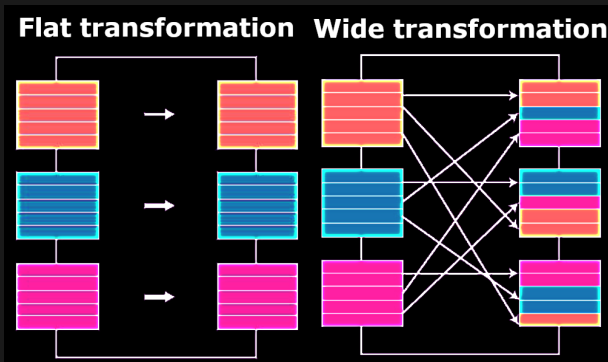
- ▶ The core concept of Spark is the RDD.
- ▶ RDDs are **immutable**, **distributed** collection of elements of your data that can be **stored in memory or disk**.
- ▶ They should be thought of as a new type of data frame, e.g. numpy, pandas, RDD.
- ▶ Interacting with them is mostly just learning new notation. . .
- ▶ With the exception that it operates through:
 - ▶ **transformations**, which create a new dataset from an existing one,
 - ▶ **actions**, which return a value.
- ▶ As in Hadoop, Spark also makes strong use of key/value pairs.

Transformations

- ▶ Transformations ¹ are **lazy**, i.e. they are not evaluated until the answer is required. This means that they can be efficiently compiled into complex batch operations.
- ▶ Transformations can **persist**, i.e. be retained in the memory of each worker node.
- ▶ Naive use of transformations can be inefficient, due to data duplication. This is why they are batched together.
- ▶ Behind the scenes, **computational graphs** are being exploited to ensure **parallelisation** and lazy, i.e. efficient **evaluation**.
- ▶ Chaining multiple transformations allow only the RDD at the start and end of the operation is (explicitly) stored.

¹Spark RDD Transformations

Transformation types



- ▶ Transformations can be thought of in two key types.
 - ▶ **Narrow transformations:** which operate locally on data (embarrassingly parallel),
 - ▶ **Wide transformations:** which operate on the whole of the data.

Actions

- ▶ Actions are simpler concepts than transformations: they return a value.
- ▶ They return a “value”, i.e. a not an RDD, either to the interface or to disk.
- ▶ They trigger the evaluation of transformations.

RDD Examples

- ▶ Which of these are narrow transformations? Which are wide? Which are actions?
 - ▶ Collect: Collects data to the interface.
 - ▶ Map: Map as in MapReduce.
 - ▶ Intersection: Compute the intersection data in multiple RDDs.
 - ▶ Distinct: Obtain only distinct elements, discarding duplicates.
 - ▶ Filter: Remove elements satisfying some criterion.
 - ▶ First: Get the first few elements.
 - ▶ Sample: Get a sample of elements.
 - ▶ Union: Combine two RDDs.
 - ▶ ReduceByKey: Reduce an RDD by key.
 - ▶ Take: Get specific elements.
 - ▶ Join: Merge two RDDs.

RDD Examples - Answers

- ▶ Narrow Transformations:

- ▶ Map
- ▶ Filter
- ▶ Sample
- ▶ Union

- ▶ Wide Transformations:

- ▶ Intersection
- ▶ Distinct
- ▶ ReduceByKey
- ▶ Join

- ▶ Actions:

- ▶ Collect
- ▶ First
- ▶ Take

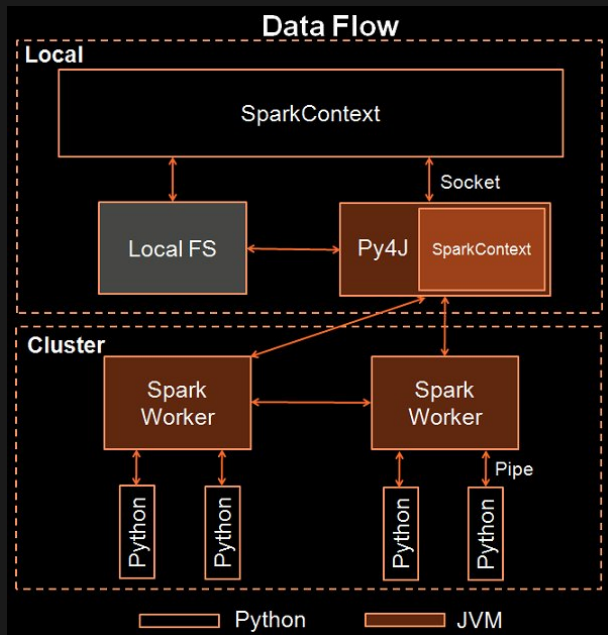
Pyspark

- ▶ Interacting with RDDs requires learning the new schema associated with them.
- ▶ Apart from interacting with RDDs, **pyspark can use standard python functions** to perform calculations.
- ▶ This means that you can use standard “boiler plate” RDD manipulation (copied from the internet),
- ▶ And write your own dedicated analysis in a familiar language.

The Spark Context

- ▶ Spark is really a **framework** running in Java, by which compute processes communicate.
- ▶ On the **head** machine (“Local”) you create a **SparkContext** instance, which sets up **Spark Worker** instances on (typically remote) compute nodes.
- ▶ These will operate on RDDs seamlessly for you as the user.
- ▶ Users can:
 - ▶ interact with the local file system,
 - ▶ distribute data via RDDs,
 - ▶ distribute variables via direct communication,
- ▶ All seamlessly, **as if the data were stored on their local instance.**

The Spark Context



Passing functions to Spark

```
def myFunc(s):  
    words = s.split(" ")  
    return len(words)  
sc = SparkContext(...)  
sc.textFile("file.txt").map(myFunc)
```

Sharing data across nodes

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

- ▶ Any communication that can occur via RDDs should do so, as this is computationally efficient.
- ▶ However, Spark supports **communication between nodes** in a number of ways.
 - ▶ One is the **broadcast**, which shares results with all other nodes.
- ▶ This is a way to share common information.

Important transformations

- ▶ See the Spark RDD guide for many more transformations:
- ▶ Map/Reduce:
 - ▶ **map**: as we know from map/reduce.
 - ▶ **reduceByKey**: as we know from map/reduce, but with flexible key specification.
- ▶ Database:
 - ▶ **join**: merge datasets by a key.
 - ▶ **filter**: selection of items by feature.
 - ▶ **sortByKey**: sorting by key, as from map/sort/reduce.
 - ▶ **aggregateByKey**: aggregate/combine the data into a new type.
- ▶ Data management:
 - ▶ **sample**: random selection of items (as an RDD).
 - ▶ **repartition**: reshuffle the data across the nodes.

Accumulate example

```
accum = sc.accumulator(0)
sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
accum.value
## 10
```

- ▶ As in Python Map/Reduce, **Reducing** is called many things.
- ▶ Just like Python, each does a slightly different thing. One key distinction is whether the reduce is a transformation, or an action.
- ▶ An **Accumulator** is the main **Action** for reducing.
- ▶ You can of course run a **reduceByKey** followed by a **collect** to achieve a similar thing.

Summary

- ▶ Parallel computing with Spark provides a transparent way to scale to **big data**, too large to fit on one machine.
- ▶ It requires a paradigm shift to its concept of RDDs, and their associated **transformations** and **actions**.
- ▶ There are some simple (enough) commands to create the required infrastructure.
- ▶ Beyond this, everything is vanilla python (with **pyspark**) or indeed vanilla R (with **SparkR**).

Reflection

- ▶ What are the key properties of an RDD?
- ▶ Why do we use transformations and actions?
- ▶ What advantages does Spark offer over Hadoop? How is it related?
- ▶ How are RDDs related to the Map/Reduce concepts?
- ▶ By the end of the course you should:
 - ▶ Be able to work with RDDs,
 - ▶ Understand Spark at a high level,
 - ▶ Be able to work with Big Data by coding in a Distributed Dataset programming environment.

Signposting

- ▶ In the workshop we will explore Hadoop and pyspark in a toy context.
- ▶ Note that the University doesn't support genuine HDFS deployment.
- ▶ We'll apply the ideas above into code.
- ▶ The final Block 12 is about being a "well rounded data scientist" - covering ethics and privacy.
- ▶ References:
 - ▶ Spark RDD guide
 - ▶ pyspark
 - ▶ Spark RDD Transformations