SPARK TUTORIALS WITH SCALA

THE BEGINNER'S GUIDE

Copyright © 2016 Supergloo, inc 3
Before We Begin.....4
Objectives and Expectations.....4
Assumptions.....4
Formatting.....5
Beyond this Book.....5
What, Why, How.....6
What is Apache Spark?.....6
Why Spark?.....6
Fundamentals of Apache Spark.....6
How to Be Productive with Spark?.....7
Apache Spark Ecosystem Components.....7
Conclusion What about Hadoop?.....7
Spark RDDs A Two Minute Guide For Beginners.....8
What is a Spark RDD?.....8
How are Spark RDDs created?.....8
Why Spark RDDs?.....8
When to use Spark RDDs?.....9
Apache Spark The Building Blocks.....10
Overview.....10
Requirements.....10
Spark with Scala First Tutorial.....10
Spark Context and Resilient Distributed Datasets.....12
Actions and Transformations.....13
Looking Ahead.....14

partner@supergloo.com
http://www.supergloo.com

Before We Begin

Thank you for purchasing version 0.8 of the Spark with Scala Tutorial cookbook. By purchasing this book, you are entitled to free upgrades for each new version. I hope you find this version and all future versions a valuable tool to advance your understanding and skill set using Spark and Scala.

Now, in order to make the most appropriate and desired upgrades in next versions of this book, I need to hear from you. I'd rather not guess as to what is important to you. So, please let me know your ideas and next versions.

You can send your ideas to partner@supergloo.com or visit the https://www.supergloo.com/contact page and submit a form.

If you believe this book deserves a good review, please do so. Positive reviews will encourage more readers. And if I have more readers, I may be able to save and purchase a new snap button shirt or take my kids to a hockey game.

Objectives and Expectations

Upon completion of the tutorials, readers will have a solid understanding of how to be productive in Spark with Scala. And when I say "Spark", I mean productive from Spark Core and major components such Spark SQL, Spark Streaming, and Machine Learning (MLlib).

As an added bonus, this book covers a variety of use cases which may be interesting for you including: Spark with Amazon S3 and Spark with Cassandra.

Consider this book similar to a recipe book for a cook. We will focus on cooking a variety of recipes in order to build up your confidence. You should be able to apply this new confidence With confidence and new skills to your use cases.

If there are any "recipes" you would like to see, again, please let me know.

Assumptions

I assume you have little or no experience with Spark, but do have a basic understanding of Scala. By no means do you need to be a Scala expert. There are many excellent books and courses on learning Scala available, but if you want something free and get started now, I recommend Scala school from Twitter engineering folks:

https://twitter.github.io/scala_school/

If you ever need suggestions on other resources, send me an email at partner@supergloo.com

Formatting

To help distinguish between text and code content, this book uses different fonts. It should be obvious between the fonts, but if not, again, I'm interested in hearing from you. So, let me know if font differences are not obvious.

Where I've thought it may be helpful, I've included links to screencasts and reference sites. There's nothing to free when going to these sites. They are free and will not try to sell you anything.

Beyond this Book

For more resources and free tutorials on data engineering, data science and advanced analytics, make sure to visit my website: https://www.supergloo.com

What, Why, How

What is Apache Spark?

Apache Spark is an open-source big data processing framework built in Scala and Java. Spark is known for its speed, ease of use, and sophisticated analytics. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project.

Apache Spark provides an interface to data structures called the Resilient Distributed Dataset (RDD). RDDs provide an abstraction to a diverse set of possible data sources including structured, semi-structured and unstructured data. Examples of possible datasets include any Hadoop compliant input sources, text files, graph data, relational databases, JSON, CSV, NoSQL databases as well as real-time streaming data from providers such as Kafka and Amazon Kinesis.

Providing a consistent interface to a multiple of input sources is one of the features which makes Spark attractive. It's especially beneficially to organizations attempting to find value from large and inconsistent data sets. Additional features and benefits will be covered later in this tutorial.

Why Spark?

Spark is an evolutionary step in how we address the challenges of Big Data. The first step in big data processing was Hadoop. Hadoop's primary processing abstraction is MapReduce with Java. MapReduce was an advance in big data processing. MapReduce programs read input data from disk, map a function across the input data, and then reduce the results of the map, and finally, store reduction results on disk.

Spark was developed in response to the limitations of the MapReduce paradigm.

Fundamentals of Apache Spark

A core construct of Spark is the data abstraction layer called Resilient Distributed Datasets (RDD). RDDs are utilized by developers, engineers, data scientists, and RDD compatible tool vendors through two categories of Spark API functions: Transformations and Actions.

RDDs, Spark transformations, and actions should be understood first. Then, the learning journey may continue with hands-on learning and/or examine the architectural questions and possible solutions.

Tutorials focus on these fundamentals

Chapter 2: Spark RDD Overview
Chapter 3: Spark Transformations
Chapter 4: Spark Actions

Transformations and Actions functions are accessible through Java, Scala, and Python APIs. The ability to use multiple languages with Spark is another attractive feature of the framework because there are language choice options.

How to Be Productive with Spark?

Spark APIs are available for Java, Scala or Python. The language to choose is highly dependent on the skills of your engineering teams and possibly corporate standards or guidelines. Many data engineering teams choose Scala for its type safety, performance and functional capabilities and is the focus of this book.

Apache Spark Ecosystem Components

Another reason Spark is gaining popularity is because of a vibrant ecosystem of component development. These components augment Spark Core, but provide a similar consistent interface.

The following components will be covered in this book's Scala tutorials:

Spark SQL
Spark Streaming
Machine Learning MLLib

Conclusion What about Hadoop?

A common misconception is that Spark and Hadoop are competitors. If the conversation is around whether Spark and MapReduce are competing approaches towards solving the processing of big data, then, yeah, the answer could easily be yes. The way MapReduce and Spark approach the problem of processing large amounts of data differs. So, in one sense, they compete.

But, it's just as important to know the Spark Hadoop or Hadoop Spark relationship is symbiotic. Spark is able to leverage an existing Hadoop-based infrastructure. First and foremost, Spark can utilize YARN and HDFS. This is the heart of most Hadoop environments. In addition, Spark is able to integrate with Hive and HBase without jumping through a million hoops.

As you learn more about Apache Spark, Hadoop related questions will eventually arise. While Spark presents an alternative to MapReduce, Hadoop constructs such as YARN and HDFS are still valuable in Spark based solutions today and foreseeable future.

Spark RDDs A Two Minute Guide For Beginners

What is a Spark RDD?

Spark RDD is short for Apache Spark Resilient Distributed Dataset. A Spark Resilient Distributed Dataset is often shortened to simply RDD. RDDs are a foundational component of the Apache Spark large scale data processing framework.

Spark RDDs are an immutable, fault-tolerant, and possibly distributed collection of data elements. RDDs may be operated on in parallel across a cluster of computer nodes. To operate in parallel, RDDs are divided into logical partitions. Partitions are computed on different nodes of the cluster through Spark Transformation APIs. RDDs may contain a type of Python, Java, or Scala objects, including user-defined classes.

RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value by performing a computation on the RDD.

How are Spark RDDs created?

Spark RDDs are created through the use of Spark Transformation functions. Transformation functions create new RDDs from a variety of sources; e.g. textFile function from a local filesystem, Amazon S3 or Hadoop's HDFS. Transformation functions may also be used to create new RDDs from previously created RDDs. For example, an RDD of all the customers from only North America could be constructed from an RDD of all customers throughout the world.

In addition to loading text files from file systems, RDDs may be created from external storage systems such as JDBC databases such as mySQL, HBase, Hive, Casandra or any data source compatible with Hadoop Input Format.

RDDs are also created an manipulated when using Spark modules such as Spark Streaming and Spark MLlib.

Why Spark RDDs?

Spark makes use of data abstraction through RDDs to achieve faster and more efficient performance than Hadoop's MapReduce.

RDDs support in-memory processing. Accessing data from memory is 10 to 100 times faster than accessing data from a network or disk. Data access from disk often occurs in Hadoop's MapReduce-based processing.

In addition to performance gains, working through an abstraction layer provides a convenient and consistent way for developers and engineers to work with a variety of data sets.

When to use Spark RDDs?

RDDs are utilized to perform computations on an RDD dataset through Spark Actions such as a count or reduce when answering questions such as "how many times did xyz happen?" or "how many times did xyz happen by location?"

Often, RDDs are transformed into new RDDs in order to better prepare datasets for future processing downstream in the processing pipeline. To reuse a previous example, let's say you want to examine North America customer data and you have an RDD of all worldwide customers in memory. It could be beneficial from a performance perspective to create a new RDD for North America only customers instead of using the much larger RDD of all worldwide customers.

Depending on the Spark operating environment and RDD size, RDDs should be cached (via cache function) or persisted to disk when there is an expectation for the RDD to be utilized more than once.

Apache Spark The Building Blocks

Becoming productive with Apache Spark requires an understanding of just a few fundamental elements. In this chapter, the building blocks of Apache Spark will be covered quickly and backed with real-world examples.

The intention is for readers to understand basic Spark concepts, so tutorials later in the book are easier to complete. It assumes you are familiar with installing software and unzipping files. Later posts will deeper dive into Apache Spark and example use cases.

Overview

Next, we're going to go over much code and working examples. Then, we'll describe Spark concepts and return back to these examples.

PLEASE NOTE: *I do not not expect you to follow all the examples yet. While you are welcome to type these commands, the intention is to show an overview of what we will be running later in this book. At the end of this section, there is a link to a screencast which may be helpful to view.*

Requirements

If you would like to run these examples yourself, the following is required.

- Java 6 or newer installed
- Download NY State Baby Names in CSV format from: http://www.healthdata.gov/dataset/baby-names-beginning-2007. (Rename the file to "`baby_names.csv`")

The CSV file we will use has following structure:

```
Year,First Name,County,Sex,Count
2012,DOMINIC,CAYUGA,M,6
2012,ADDISON,ONONDAGA,F,14
2012,JULIA,ONONDAGA,F,15
```

Spark with Scala First Tutorial

Let's ensure you have a working environment. If you already have a Spark environment you can skip this section. This section is designed for Linux or Mac users. For Windows users, I suggest you consult the following link: http://stackoverflow.com/questions/25481325/how-to-set-up-spark-on-windows

Spark Installation Steps

1. Download from http://spark.apache.org/downloads.html. Select the "Pre-built package for Hadoop 2.4"
2. Unpack it. (untar, unzip, etc.)
3. From terminal, go to the root directory of where Spark was unzipped and run the spark-shell via: bin/spark-shell

For example, if you unzipped Spark download to `/Users/yourname/Development/spark-1.6.1-bin-hadoop2.4/`, open a terminal window and: `cd /Users/toddmcgrath/Development/spark-1.6.1-bin-hadoop2.4/` Then, you will be able to run `bin/spark-shell`

Windows users: update these commands appropriately. For example, you will likely run `bin/spark-shell.cmd` instead

The spark-shell is also known as the Spark REPL which is stands for: read-eval-print-loop and is construct found in many frameworks and languages. The REPL or "shell" provides a convenient environment to run and evaluate statements and programs.

Let's run some Scala code

If you would like me run these steps below, here is a link to a screencast of me running through these examples:

(Links not available from Preview)

If this is your first experience with the Spark Shell with Scala, stop here and watch this very short screencast. It's under a minute and will help.

Scala Code Examples from the Spark Shell

After you open the Spark shell with `bin/spark-shell` as described above, you will see a prompt which looks like `scala>` Again, please see the screencast at end of these examples if it helps clarify the following examples.

Open up spark shell:

```
scala> val babyNames = sc.textFile("baby_names.csv") // baby_names.csv
must be in same directory
babyNames: org.apache.spark.rdd.RDD[String] = baby_names.csv
MappedRDD[1] at textFile at <console>:12

scala> babyNames.count
res0: Long = 35218

scala> babyNames.first()
```

```
res177: String = Year,First Name,County,Sex,Count
```

So, we know there are 35218 rows in the CSV

Next, let's convert the CSV to an Array of Strings. Then, determine there are 62 unique NY State counties over the years of data collect in the CSV.

```
scala> val rows = babyNames.map(line => line.split(","))
rows: org.apache.spark.rdd.RDD[Array[String]] = MappedRDD[17] at map
at <console>:14

scala> rows.map(row => row(2)).distinct.count
res22: Long = 62
```

There are 136 rows containing the name "DAVID". How do we know?

```
scala> val davidRows = rows.filter(row => row(1).contains("DAVID"))
davidRows: org.apache.spark.rdd.RDD[Array[String]] = FilteredRDD[24]
at filter at <console>:16

scala> davidRows.count
res32: Long = 136
```

Number of rows where "NAME" DAVID has a "Count" greater than 10

```
scala> davidRows.filter(row => row(4).toInt > 10).count()
res41: Long = 89
```

17 unique counties which have had the name DAVID over 10 times in a given year

```
scala> davidRows.filter(row => row(4).toInt > 10).map( r =>
r(2) ).distinct.count
res57: Long = 17
```

If you would like me run these steps above, here is a link to a screencast of me running through these examples:

(Screencast link not available from Preview)

Spark Context and Resilient Distributed Datasets

The way to interact with Spark is via a SparkContext. The example used the Spark Console which provides a SparkContext automatically. Did you notice the last line in the spark-shell:

```
06:32:25 INFO SparkILoop: Created spark context..
Spark context available as sc.
```

That's how we're able to use `sc` from within the examples above and in particular, `sc.textFile` function.

After obtaining a `SparkContext`, developers interact with Spark via Resilient Distributed Datasets.

Resilient Distributed Datasets (RDDs) are a immutable, distributed collection of elements. These collections may be parallelized across a cluster. RDDs are loaded from an external data set or created via a SparkContext. We'll cover both of these scenarios.

In the previous example, we create a RDD via:

```
scala> val babyNames = sc.textFile("baby_names.csv")
```

We also created RDDs other ways as well, which we'll cover a bit later.

When utilizing Spark, you will be doing one of two primary interactions: creating new RDDs or transforming existing RDDs to compute a result. The next section describes these two Spark interactions.

Actions and Transformations

When working with a Spark RDDs, there are two available operations: actions or transformations. An action is an execution which produces a result. Examples of actions in previous are count, first.

Spark Actions Examples

We saw a few examples of Spark Actions in the examples above including:

```
babyNames.count() // number of lines in the CSV file

babyNames.first() // first line of CSV
```

Spark Transformation Examples

Transformations create new RDDs using existing RDDs. We created a variety of RDDs in our example:

```
val rows = babyNames.map(line => line.split(","))

val davidRows = rows.filter(row => row(1).contains("DAVID"))
```

Looking Ahead

In this chapter, we covered the fundamentals for being productive with Apache Spark from a big picture perspective.

As you now know, there are just a few key Spark concepts of RDDs, Transformations and Actions to know before being able to be productive.

In the next two chapters, we're going to do the equivalent of swinging the baseball bat 100 times with 100 different pitches OR trying to play the most popular notes on a guitar OR trying 50 different dance moves OR saying the 75 most popular words in a foreign language. You understand the analogy? We're going to exercise the Spark API with Scala in many different ways in the next two chapters.

The intention is NOT to memorize these 50, 75, 100 data points. Rather, it's intended for you to gain exposure in small steps and start building momentum.

Momentum matters for building confidence and persistence to continue.

I know these examples in next two chapters might seem a bit easy. But, please try to not get frustrated. Do the work and stay the course. It will pay off later in your journey.

(This is end of the preview.)