Reactive Dashboards Using Apache Spark

Rahul Kumar
Software Developer
@rahul_kumar_aws
Agenda

- Dashboards
- Big Data Introduction
- Apache Spark
- Introduction to Reactive Applications
- Reactive Platform
- Live Demo
Dashboards

A dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance*. 

* Stephen Few’s definition of a dashboard
Key characteristics of a dashboard

• All component should fit in a single screen
• Interactivity such as filtering, drill down can be used.
• The displayed data automatically updated without any assistance from the user.
A typical database application
Sub second response

Multi Source Data Ingestion

Gb’s to Petabyte Data

Realtime update

Scalable
Three V’s of Big Data
Scale vertically (scale up)
Scale horizontally (scale out)
Apache Spark is a fast and general engine for large-scale data processing.
File Format supports

- **CSV**
- **TSV**
- **JSON**
- **ORC**
Apache Spark Stack

- Spark SQL
- Spark Streaming
- MLlib
- GraphX

Apache Spark
Spark Log Analysis
• Apache Spark Setup
• Interaction with Spark Shell
• Setup a Spark App
• RDD Introduction
• Deploy Spark app on Cluster
Prerequisite for cluster setup

Spark Cluster

Spark runs on Java 6+, Python 2.6+ and R 3.1+. For the Scala API, Spark 1.4.1 uses Scala 2.10.

Java 8
sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer

Scala 1.10.4
http://www.scala-lang.org/files/archive/scala-2.10.4.tgz
$tar -xvzf scala-2.10.4.tgz
vim ~/.bashrc
export SCALA_HOME=/home/ubuntu/scala-2.10.4
export PATH=$PATH:$SCALA_HOME/bin
Spark Setup

http://spark.apache.org/downloads.html
Download Spark

The latest release of Spark is Spark 1.4.1, released on July 15, 2015 (release notes) (git tag)

1. Choose a Spark release: 1.4.1 (Jul 15 2015)
2. Choose a package type: Source Code [can build several Hadoop versions]
   - Pre-build with user-provided Hadoop [can use with most Hadoop distributions]
   - Pre-built for Hadoop 2.6 and later
   - Pre-built for Hadoop 2.4 and later
   - Pre-built for Hadoop 2.3
   - Pre-built for Hadoop 1.X
   - Pre-built for CDH 4
3. Choose a download type:
4. Download Spark: spark-
5. Verify this release using:

Note: Scala 2.11 users should download the Spark source package and build with Scala 2.11 support.
Running Spark Example & Shell

$ cd spark-1.4.1-bin-hadoop2.6

$.bin/run-example SparkPi 10

Pi is roughly 3.144248
cd spark-1.4.1-bin-hadoop2.6

spark-1.4.1-bin-hadoop2.6 $ ./bin/spark-shell --master local[2]

The --master option specifies the master URL for a distributed cluster, or local to run locally with one thread, or local[N] to run locally with N threads.
```scala
val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

distData.reduce(_+_). Click to add notes
```

Spark Jobs

- Total Uptime: 3.2 min
- Scheduling Mode: FIFO
- Completed Jobs: 1
- Event Timeline
- Enable zooming

Completed Jobs (1)

<table>
<thead>
<tr>
<th>Job Id</th>
<th>Description</th>
<th>Submitted</th>
<th>Duration</th>
<th>Stages: Succeeded/Total</th>
<th>Tasks (for all stages): Succeeded/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reduce at &lt;console&gt;:26</td>
<td>2015/08/08 23:14:13</td>
<td>0.3 s</td>
<td>1/1</td>
<td>2/2</td>
</tr>
</tbody>
</table>
RDD Introduction

Resilient Distributed Data Set

Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

RDD shard the data over a cluster, like a virtualized, distributed collection.

Users create RDDs in two ways: by loading an external dataset, or by distributing a collection of objects such as List, Map etc.
RDD Operations

RDDs support two types of operations: **transformations** and **actions**.

Spark computes RDD only in a **lazy fashion**.

Only computation start when an **Action** call on RDD.
Simple SBT project setup [https://github.com/rahulkumar-aws/HelloWorld](https://github.com/rahulkumar-aws/HelloWorld)

```scala
$ mkdir HelloWorld
$ cd HelloWorld
$ mkdir -p src/main/scala
$ mkdir -p src/main/resources
$ mkdir -p src/test/scala
$ vim build.sbt
   name := "HelloWorld"

   version := "1.0"

   scalaVersion := "2.10.4"
$ mkdir project
$ cd project
$ vim build.properties
   sbt.version=0.13.8

$ vim scr/main/scala/HelloWorld.scala

   object HelloWorld { def main(args: Array[String]) = println("HelloWorld!") }  
$ sbt run
```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
object SparkWordCount {

    def main(args: Array[String]): Unit = {

        val sc = new SparkContext("local", "SparkWordCount")

        val wordsCounted = sc.textFile(args(0)).map(line=> line.toLowerCase)
            .flatMap(line => line.split("\W+"))
            .groupBy(word => word)
            .map{ case(word, group) => (word, group.size)}

        wordsCounted.saveAsTextFile(args(1))
        sc.stop()
    }
}

$sbt "run-main ScalaWordCount src/main/resources/sherlockholmes.txt out"
Launching Spark on Cluster
Cluster Manager Can be Spark's own Standalone Cluster Manager or Mesos or YARN
Spark supports pulling data sets into a cluster-wide in-memory cache.

scala> val textFile = sc.textFile("README.md")


scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))


scala> linesWithSpark.cache()


scala> linesWithSpark.count()

res12: Long = 19
## Storage

<table>
<thead>
<tr>
<th>RDD Name</th>
<th>Storage Level</th>
<th>Cached Partitions</th>
<th>Fraction Cached</th>
<th>Size in Memory</th>
<th>Size in ExternalBlockStore</th>
<th>Size on Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapPartitionsRDD</td>
<td>Memory Deserialized 1x Replicated</td>
<td>2</td>
<td>100%</td>
<td>3.3 KB</td>
<td>0.0 B</td>
<td>0.0 B</td>
</tr>
</tbody>
</table>
Spark SQL Introduction

Spark SQL is Spark's module for working with structured data.

- Mix SQL queries with Spark programs
- Uniform Data Access, Connect to any data source
- DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC.
- Hive Compatibility Run unmodified Hive queries on existing data.
- Connect through JDBC or ODBC.
Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
$ git clone https://github.com/rahulkumar-aws/WordCount.git
$ nc -lk 9999
sbt "run-main StreamingWordCount"
Reactive Application

- Responsive
- Resilient
- Elastic
- Event Driven

http://www.reactivemanifesto.org
Responsive

Scalable
Resilient

Event-Driven
Typesafe Reactive Platform
Play Framework

The High Velocity Web Framework For Java and Scala

- RESTful by default
- JSON is a first class citizen
- Web sockets, Comet, EventSource
- Extensive NoSQL & Big Data Support

https://www.playframework.com/download

https://downloads.typesafe.com/typesafe-activator/1.3.5/typesafe-activator-1.3.5-minimal.zip
Akka

Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM.

- Simple Concurrency & Distribution
- Resilient by Design
- High Performance
- Elastic & Decentralised
- Extensible

Akka uses **Actor Model** that raise the abstraction level and provide a better platform to build **scalable**, **resilient** and **responsive applications**.
References

http://spark.apache.org/docs/latest/quick-start.html

Learning Spark Lightning-Fast Big Data Analysis
By Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia

https://www.playframework.com/documentation/2.4.x/Home
http://doc.akka.io/docs/akka/2.3.12/scala.html
Thank You