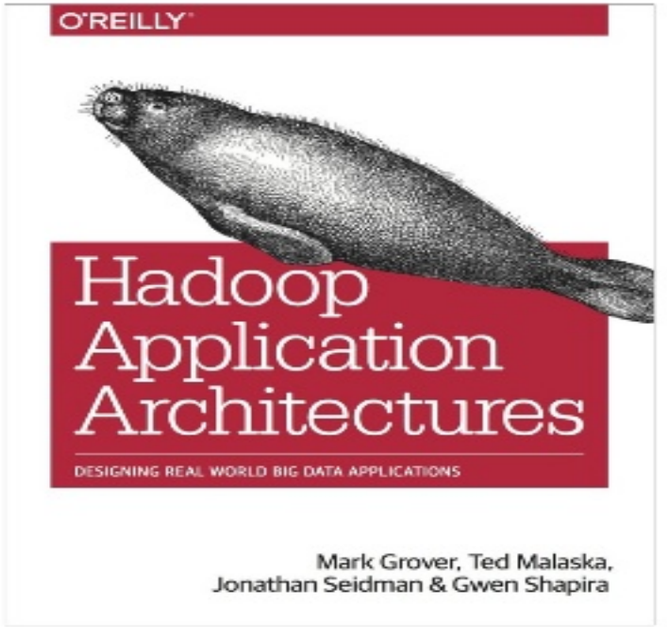
Mastering Spark Unit Testing

Theodore Malaska
Blizzard Entertainment, Group Technical Architect

About Me

- Ted Malaska
  - Architect at Blizzard Ent
    - Working on Battle.net
  - PSA at Cloudera
    - Worked with over 100 companies
    - Committed to over a dozen OS Projects
    - Co-Wrote a Book
  - Architect at FINRA
    - Worked on OATS



About Blizzard

- Maker of great games
  - World of Warcraft
  - StarCraft
  - Hearthstone
  - Overwatch
  - Heroes of the Storm
  - Diablo
- Tech Giant
  - Massive clusters world wide
  - > 100 billions of record of data a day
  - Driven to improve gamer experience through technology

Overview

- Why to run Spark outside of a cluster
- What to test
- Running Local
- Running as a Unit Test
- Data Structures
- Supplying data to Unit Tests

Why to Run Spark Outside of a Cluster?

- Time
- Trusted Deployment
- Logic
- Money

What to test

- Experiments
- Complex logic
- Samples
- Business generated scenarios

What about Notebooks

- UIs like Zeppelin
- For experimentations
- Quick feedback
- Not for productization
- Lacks IDE Features

Running Local

- A test doesn't always need to be a unit test
- Running local in your IDE is priceless

Example

```
//Get spark context
val sc:SparkContext = if (runLocal) {
  val sparkConfig = new SparkConf()
  sparkConfig.set("spark.broadcast.compress", "false")
  sparkConfig.set("spark.shuffle.compress", "false")
  sparkConfig.set("spark.shuffle.spill.compress", "false")
  new SparkContext("local[2]", "DataGenerator", sparkConfig)
} else {
  val sparkConf = new SparkConf().setAppName("DataGenerator")
  new SparkContext(sparkConf)
}
val filterWordSetBc = sc.broadcast(scala.io.Source.fromFile(filterWordFile).getLines.toSet)
val inputRDD = sc.textFile(inputFolder)
val filteredWordCount: RDD[(String, Int)] = filterAndCount(filterWordSetBc, inputRDD)
filteredWordCount.collect().foreach{case(name: String, count: Int) => println(" - " + name + ":" + count)}
```

Live Demo

Things to Note

- Needed a Spark Context
- Parallelize function
- Separate out testable work from driver code
- Everything is fully debuggable

Now on to Unit Testing

- Just like running local be easier

Example Code

```
class FilterWordCountSuite  extends FunSuite with SharedSparkContext {
 test("testFilterWordCount") {
   val filterWordSetBc = sc.broadcast(Set[String]("the", "a", "is"))
   val inputRDD = sc.parallelize(Array("the cat and the hat", "the car is blue",
    "the cat is in a car", "the cat lost his hat"))

   val filteredWordCount: RDD[(String, Int)] =
    FilterWordCount.filterAndCount(filterWordSetBc, inputRDD)

   assert(filteredWordCount.count() == 8)
   val map = filteredWordCount.collect().toMap
   assert(map.get("cat").getOrElse(0) == 3)
   assert(map.get("the").getOrElse(0) == 0)
 }
}
```

Shared Spark Context

```
trait SharedSparkContext extends BeforeAndAfterAll { self: Suite =>

  @transient private var _sc: SparkContext = _

  def sc: SparkContext = _sc

  var conf = new SparkConf(false)

  override def beforeAll() {
    _sc = new SparkContext("local[4]", "test", conf)
    super.beforeAll()
  }

  override def afterAll() {
    LocalSparkContext.stop(_sc)
    _sc = null
    super.afterAll()
  }
}
```

Maven Dependency

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.binary.version}</artifactId>
    <version>${spark.version}</version>
    <type>test-jar</type>
    <scope>test</scope>
</dependency>

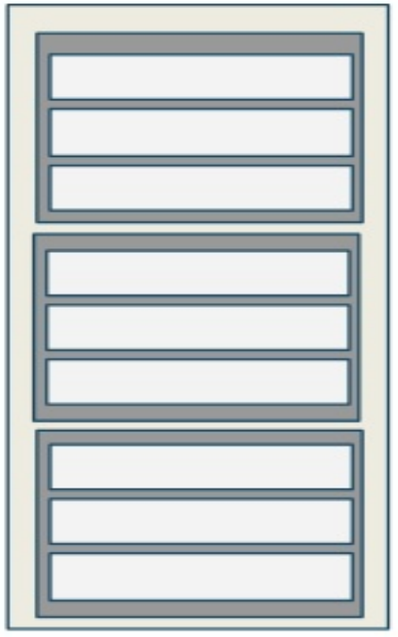
<dependency>
    <groupId>org.scalatest</groupId>
    <artifactId>scalatest_${scala.binary.version}</artifactId>
    <version>2.2.1</version>
    <scope>test</scope>
</dependency>
```

Live Demo

Data Structures

Collection

RDD



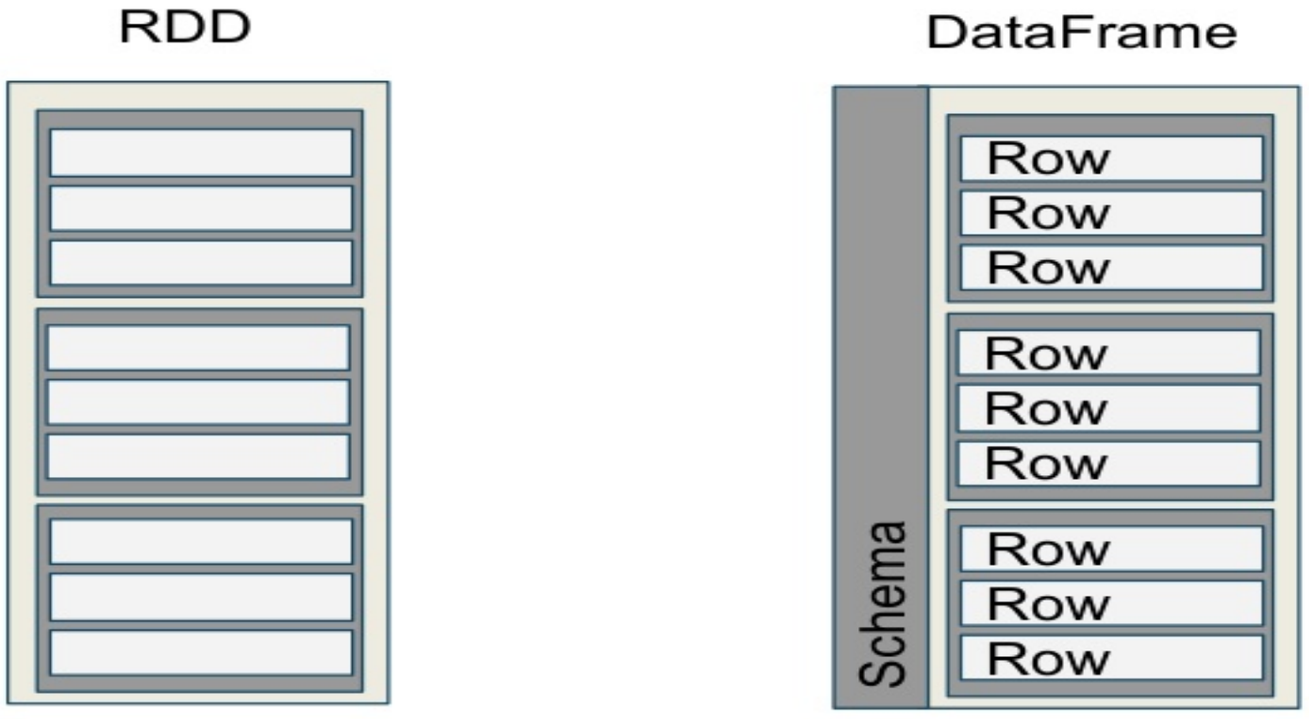
Creating an RDD

```
val inputRDD = sc.parallelize(Array("the cat and the hat",
  "the car is blue",
  "the cat is in a car",
  "the cat lost his hat"))
```

Method to Selecting Data

- Developer defined data
- Selected sampling of data from production system
- Generated data from a data generator
  - ssCheck
  - SparkTestingBase
- Requirement/Business driven selection and generation

What about DataFrames and SQL



How to create a DataFrame

- Code and make a structType
- Take the schema from an existing Table

Creating a Stuct Type

```
hc.sql("create external table trans (user string, time string, amount string) " +
  "location '" + outputTableLocation  + "'")

val rowRdd = sc.textFile(inputCsvFolder).map(r => Row(r.split(",")))

val userField = new StructField("user", StringType, nullable = true)
val timeField = new StructField("time", StringType, nullable = true)
val amountField = new StructField("amount", StringType, nullable = true)

val schema = StructType(Array(userField, timeField, amountField))

hc.createDataFrame(rowRdd, schema).registerTempTable("temp")

hc.sql("insert into trans select * from temp")
```

Taking a Schema from a Table

```
hc.sql("create external table trans (user string, time string, amount string) " +
  "location '" + outputTableLocation  + "'")

val rowRdd = sc.textFile(inputCsvFolder).map(r => Row(r.split(",")))

val emptyDf = hc.sql("select * from trans limit 0")

hc.createDataFrame(rowRdd,  emptyDf.schema).registerTempTable("temp")

hc.sql("insert into trans select * from temp")
```

What about Nested DataFrames

- ```
  hc.sql("create external table trans (" +
    " user string," +
    " time string," +
    " items array<struct<" +
    "   amount: string " +
    ">>) " +
    "location '" + outputTableLocation  + "'")

  val rowRdd = sc.parallelize(Array(
    Row("bob", 10, Seq(
      Row(1, 2, 3, 4)
    ))
  ))
  ```


  ```

Important NOTE

- Spark should be used to bring the big to the small
- In that case more of the logic may be in the small

Dataframe Testing Example

```
test("RunCountingSQL") {
  val rdd = sqlContext.sparkContext.parallelize(Array(
    Row("bob", "1", "1"), Row("bob", "1", "1"),
    Row("bob", "1", "1"), Row("cat", "1", "1"),
    Row("bob", "1", "1"), Row("cat", "1", "1"), Row("bob", "1", "1")
))

  val userField = new StructField("user", StringType, nullable = true)
  val timeField = new StructField("time", StringType, nullable = true)
  val amountField = new StructField("amount", StringType, nullable = true)
  val schema = StructType(Array(userField, timeField, amountField))
  sqlContext.createDataFrame(rdd, schema).registerTempTable("trans")

  val results = RunCountingSQL.runCountingSQL(sqlContext)

  assert(results.length == 2)
}
```

What about Hive

- If you are testing Hive.
  - Spark will create a Local HMS
  - Allows all the creation, deleting, writing
  - Make sure you can write to the table location
  - Also feel free to delete the metastore folder in-between jobs

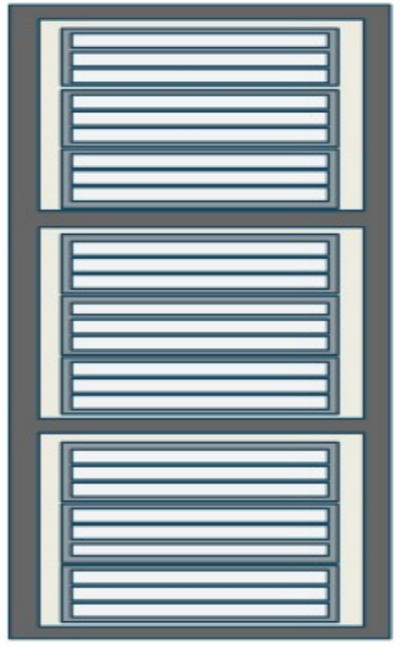
Now for Streaming

- Lets go one step more
- What is a DStream
- How do you test a DStream

Now for Streaming

RDD

DStream



TestOperation Example

```
class StreamFilterWordCount extends TestSuiteBase{
 test("mapPartitions") {
   assert(numInputPartitions === 2, "Number of input partitions has been changed from 2")

   val input = Seq(
    Seq("the cat and the hat", "the car is blue"),
    Seq("the cat is in a car", "the cat lost his hat"))

   val output = Seq(Seq(9), Seq(11))

   val operation = (r: DStream[String]) => StreamingWordCount.wordCount(r)

   testOperation(input, operation, output)
 }
}
```

What about other things

- MlLib
- GraphX

What about Integration Testing

- MiniClusters
  - HBase
  - Kafka
  - HDFS
  - Cassandra
  - Many more

Testing True Distributed

- Running on a Dev environment
- Using Docker to spin up environment
- Mini yarn or mesos

THANK YOU.

SPARK SUMMIT
EUROPE 2016