Apache Spark - A Scala Killer App?

Markus Dale

2015

Slides And Code

- Slides: https://github.com/medale/spark-mail/blob/master/presentation/Spark-ScalaKillerApp.pdf
- Spark Code Examples: https://github.com/medale/spark-mail/

What's Apache Spark?

- Large-scale data processing framework written in Scala
- Replacement for Hadoop MapReduce?
  - In-memory caching
  - Advanced directed acyclic graph of computations - optimized
  - Rich high-level Scala, Java, Python, R and SQL APIs
    - 2-5x less code than Hadoop M/R
- Unified batch, SQL, streaming, graph and machine learning
- Interactive data exploration via spark-shell

Spark - Fast and Efficient: GraySort Record

	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Figure 1: Spark GraySort Results Xin (2014)

Apache Spark Buzz

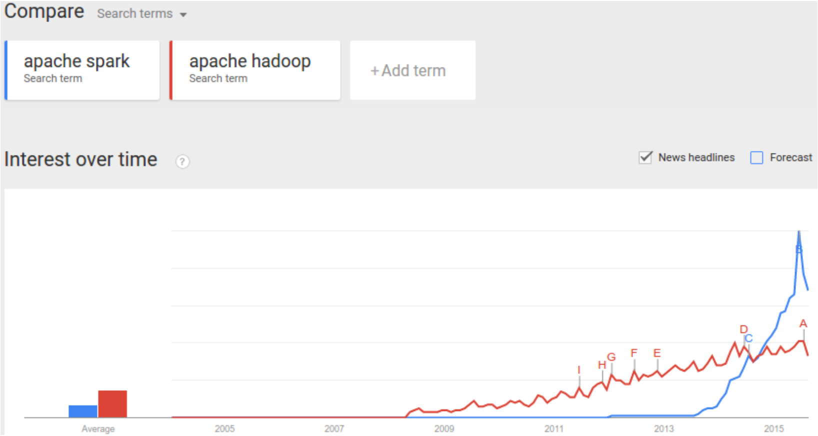

Figure 2: Google Trends Apache Spark/Apache Hadoop August 2015

Spark Ecosystem

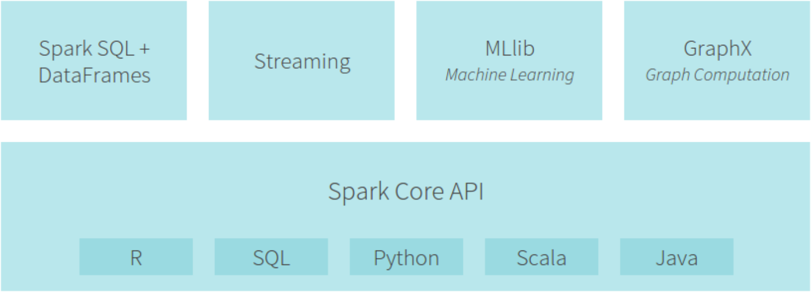

Figure 3: Databricks Spark 1.4.1 Ecosystem (2015)

Spark Lines of Code

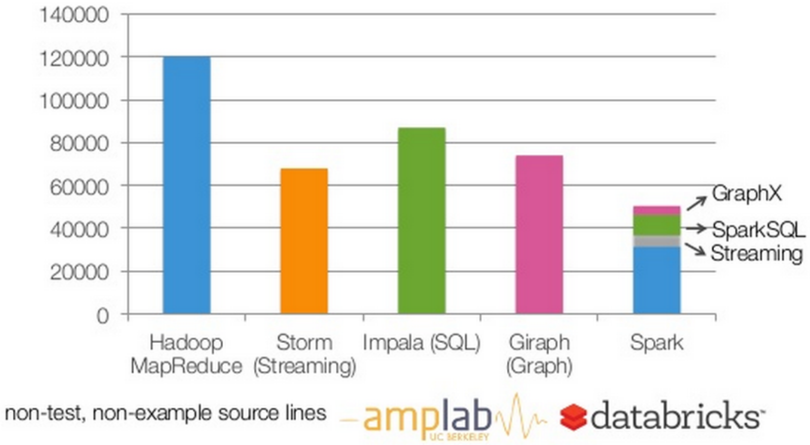

Figure 4: Spark LOC Armbrust (2014)

Spark Academic Papers

- Spark: Cluster computing with working sets (Zaharia et al. 2010)
- Resilient Distributed Datasets: A fault-tolerant abstraction for in-memory cluster computing (Zaharia et al. 2012)
- GraphX: A Resilient Distributed Graph System on Spark (Xin et al. 2013)
- Spark SQL: Relational data processing in Spark (Armbrust et al. 2015)
- MLlib: Machine Learning in Apache Spark (Meng et al. 2015)

Spark Clusters

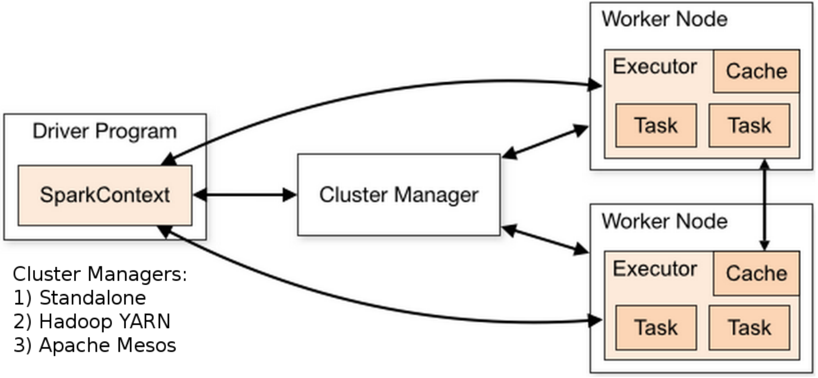

Figure 5: Spark Cluster Managers SparkWebsite (2015)

Getting Spark

- http://spark.apache.org/downloads.html
  - Source
  - Pre-built binaries for multiple versions of Hadoop
- Set JAVA_HOME to root of JDK installation
- Local mode:
  - Untar spark-xxx-.tgz
  - cd spark-xxx/bin
  - ./spark-shell

Spark with external cluster manager

- Spark Standalone cluster
- Hadoop YARN - install on cluster edge node
  - Set HADOOP_CONF_DIR (NameNode, ResourceManager)
  - Hortonworks Data Platform - HDP includes Spark
  - Cloudera...
- Apache Mesos
- Note: Driver must be able to communicate with executors (ports open!)

Spark in the Cloud

- Amazon EC2 deploy script - standalone cluster/S3
- Amazon Elastic MapReduce (EMR) - Spark install option
- Google Compute Engine - Hadoop/Spark
- Databricks Spark Clusters - Notebooks, Jobs, Dashboard

Running Spark

- Interactive (spark-shell)
- Batch mode (spark-submit)

Interactive shell on Hadoop YARN

```
spark-shell --master yarn-client \
  --num-executors 3 \
  --driver-memory 4g \
  --executor-memory 4g \
  --executor-cores 4 \
  --jars project.jar
```

spark-shell --help for all options

Inside Spark Shell - Spark Context

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM)...)
Type in expressions to have them evaluated.
Type :help for more information.
15/09/09 19:18:29 INFO SparkUI: Started SparkUI at http...
Spark context available as sc.
SQL context available as sqlContext.
scala>
```

Spark Context

- Holds connection info to cluster, configuration
- Read in data, for example:
  - parallelize(seq, numPartitions)
  - textFile(path)
  - newAPIHadoopFile(path, inputFormatClass, keyClass, valueClass, hadoopConf)

Spark Context in Scala

```scala
package com.spark

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
...

object MySparkJob {

  def main(args: Array[String]): Unit = {
    val sparkConf = new SparkConf().
       setAppName("My Spark Job")
    val sc = new SparkContext(sparkConf)
    ...
  }
}
```

Batch Mode - Spark Submit

```
spark-submit --class com.spark.MySparkJob \
--master yarn-cluster [options] \
<app jar> [app options]
```

Resilient Distributed Dataset (RDD)

- Treat distributed, **immutable** data set as a collection
  - Lineage - remember origin and transformations
- Resilient: recompute failed partitions using lineage
- Two forms of RDD operations:
  - Transformations (applied lazily - optimized evaluation)
  - Actions (cause transformations to be executed)
- Rich functions on RDD abstraction (Zaharia et al. 2012)

RDD from Hadoop Distributed File System (HDFS)

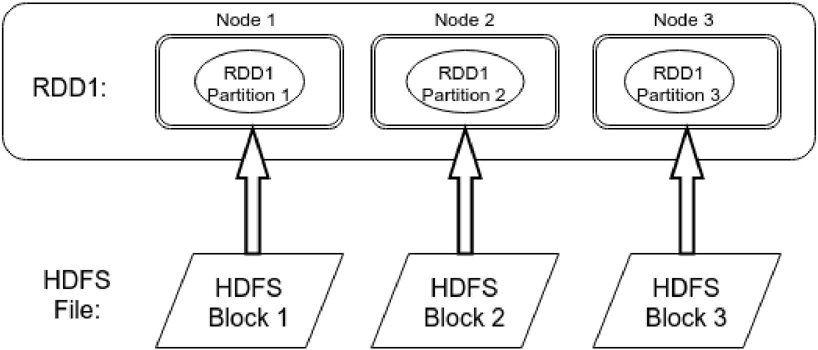

Figure 6: RDD partitions

Background: Scala List Combinators

- map
- flatMap
- filter
- reduce...

=> Methods that take function(s) as their argument(s)

map

- Method signature for List[A]
  - map(f: (A) => B): List[B]
- create a new List by applying function to each element of original collection
- one input element - one output element (can be of different type)

map - Scala

```
def computeLength(w: String): Int = w.length

val words = List("when", "shall", "we", "three",
  "meet", "again")

val lengths = words.map(computeLength)
> lengths  : List[Int] = List(4, 5, 2, 5, 4, 5)
```

map - Scala syntactic sugar

```scala
//anonymous function (specifying input arg type)
val list2 = words.map((w: String) => w.length)

//let compiler infer arguments type
val list3 = words.map(w => w.length)

//use positionally matched argument
val list4 = words.map(_.length)
```

flatMap

- Method signature for List[A]
  - flatMap(f: (A) => GenTraversableOnce[B]): List[B]
- create a new List by applying function to each element
- Output of applying function to each element is "iterable"
  - Could be empty
  - Could have 1 to many output elements
- flatten - take each element in output "iterable" and copy it to overall output List
  - remove one level of nesting (flatten)

flatMap Example

```scala
val macbeth = """When shall we three meet again?
|In thunder, lightning, or in rain?""".stripMargin
val macLines = macbeth.split("\n")

//Non-word character split
val macWordsNested: Array[Array[String]] =
      macLines.map{line => line.split("""\W+""")}
//Array(Array(When, shall, we, three, meet, again),
//      Array(In, thunder, lightning, or, in, rain))

val macWords: Array[String] =
     macLines.flatMap{line => line.split("""\W+""")}
//Array(When, shall, we, three, meet, again, In,
//      thunder, lightning, or, in, rain)
```

filter

- Method signature for List[A]
  - filter(p: (A) => Boolean): List[A]
- selects all elements of this list which satisfy a predicate.
- returns - a new list consisting of all elements of this list that satisfy the given predicate p. The order of the elements is preserved.

filter Example

```
val macWordsLower = macWords.map{_.toLowerCase}
//Array(when, shall, we, three, meet, again, in, thunder,
//      lightning, or, in, rain)

val stopWords = List("in","it","let","no","or","the")
val withoutStopWords =
  macWordsLower.filter(word => !stopWords.contains(word))
// Array(when, shall, we, three, meet, again, thunder,
//       lightning, rain)
```

So what does this have to do with Apache Spark?

- Resilient Distributed Dataset (RDD)
- From API docs: "immutable, partitioned collection of elements that can be operated on in parallel"
- map, flatMap, filter, reduce, fold, aggregate...

RDD Transformations vs. Actions

- Transformations are evaluated lazily
  - Build up lineage graph until action is invoked
  - Optimize execution of lineage graph
- Actions
  - Cause any previously applied transformations to be executed at once

Some RDD Transformations

- map, flatMap, filter
- sample(withReplacement, fraction, [seed]): RDD[T]
- distinct(): RDD[T]
- union(otherDataset): RDD[T]
- zip(other: RDD[U]): RDD[(T, U)]
  - must have same number of partitions/elements per partition
- coalesce(numPartitions)/repartition(numPartitions)

Some RDD Actions

- reduce(f: (T, T)   T): T
  - function must be commutative and associative
- collect(): Array[T]
  - materialize all RDD elements on driver (danger!)
- count()
- first()
- take(n)
- takeSample(withReplacement, num, [seed]): Array[T]

RDD Save Actions

- saveAsTextFile(path)
- saveAsSequenceFile(path)
  - elements must implement Hadoop Writable
- saveAsObjectFile(path)
  - Uses Java Serialization (elements implement Java Serializable)

Reading Avro MailRecord objects

```scala
val hadoopConf = sc.hadoopConfiguration

val mailRecordsAvroRdd =
   sc.newAPIHadoopFile("enron.avro",
      classOf[AvroKeyInputFormat[MailRecord]],
      classOf[AvroKey[MailRecord]],
      classOf[NullWritable], hadoopConf)

val recordsRdd = mailRecordsAvroRdd.map {
  case(avroKey, _) => avroKey.datum()
}
```

com.uebercomputing.analytics.basic.BasicRddFunctions

```scala
//compiler can infer bodiesRdd type - reader clarity
val bodiesRdd: RDD[String] =
  recordsRdd.map { record => record.getBody }

val bodyLinesRdd: RDD[String] =
  bodiesRdd.flatMap { body => body.split("\n") }

val bodyWordsRdd: RDD[String] =
  bodyLinesRdd.flatMap { line => line.split("""\W+""") }

val stopWords = List("in", "it", "let", "no", "or", "the")
val wordsRdd = bodyWordsRdd.filter(!stopWords.contains(_))

//Lazy eval all transforms so far - now action!
println(s"There were ${wordsRdd.count()} words.")
```

Spark Scala API

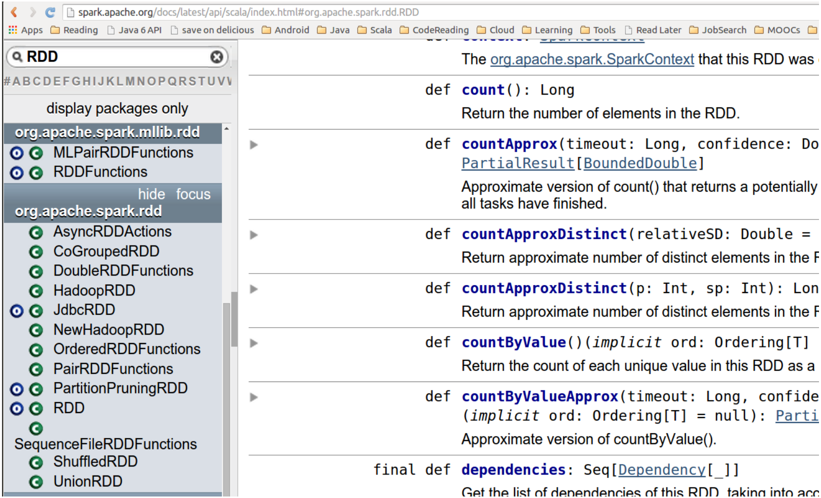

Figure 7: Spark Scala API RDD

Spark - From RDD to PairRDDFunctions

- If an RDD contains tuples (K,V)
  - can apply PairRDDFunctions
- Mechanism: implicit conversion from RDD to PairRDDFunctions

RDD to PairRDDFunctions Example - Ye Olde Word Count

```
> val words = List("to","be","or","not","to","be")
> val wordsRdd = sc.parallelize(words)

> val wordCountRdd = wordsRdd.map(w => (w, 1))
wordCountRdd: org.apache.spark.rdd.RDD[(String, Int)]

> val wordSumRdd =
    wordCountRdd.reduceByKey( (a,b) => a + b )

> wordSumRdd.collect()
res4: Array[(String, Int)] =
    Array((not,1), (or,1), (be,2), (to,2))
```

PairRDDFunctions

- keys/values - return RDD of keys/values
- mapValues - transform each value with a given function
- flatMapValues - flatMap each value (0, 1 or more output per value)
- groupByKey - RDD[(K, Iterable[V])]
  - Note: expensive for aggregation/sum - use reduce/aggregateByKey!
- reduceByKey - return same type as value type
- foldByKey - zero/neutral starting value
- aggregateByKey - can return different type
- lookup - retrieve all values for a given key
- join (left/rightOuterJoin), cogroup ...

From RDD to DoubleRDDFunctions

- From API docs: "Extra functions available on RDDs of Doubles through an implicit conversion."
- mean, stddev, stats (count, mean, stddev, min, max)
- sum
- histogram ...

DoubleRDDFunctions example

```
> val heights = List(76, 54, 62, 65, 78, 48, 55, 60)
> val heightsRdd = sc.parallelize(heights)
org.apache.spark.rdd.RDD[Int]

> heightsRdd.stats
StatCounter = (count: 8, mean: 62.250000, stdev: 9.832980,
  max: 78.000000, min: 48.000000)

> heightsRdd.histogram(4)
(Array(48.0, 55.5, 63.0, 70.5, 78.0),Array(3, 2, 1, 2))
```

RDD Persistence

- cache() == persist(StorageLevel.MEMORY_ONLY)
- persist(storageLevel) - trade-off memory/CPU
  - MEMORY_ONLY (recompute partitions that don't fit)
  - MEMORY_ONLY_2 (also for all other options)
  - MEMORY_ONLY_SER (much smaller memory footprint)
  - MEMORY_AND_DISK (spill to local disk)
  - MEMORY_AND_DISK_SER

Task Serialization

- Serialize tasks from Driver to Executor
  - Closure (function can reference vars outside of its declaration)
  - vars must be objects or serializable classes
  - Can call object methods ($\sim$ static method in Java)
  - Can make copy of instance variables of a class
  - Task not serializable: java.io.NotSerializableException

RDD Content Serialization

- Move content of partition to another executor or driver
  - e.g. shuffle, collect(), take(2)
- Must serialize each object in RDD
- Default: Java Serialization (slow)
- Production: Use Kryo serialization (http://spark.apache.org/docs/latest/tuning.html#data-serialization)

Spark Web UI - Job with Cache

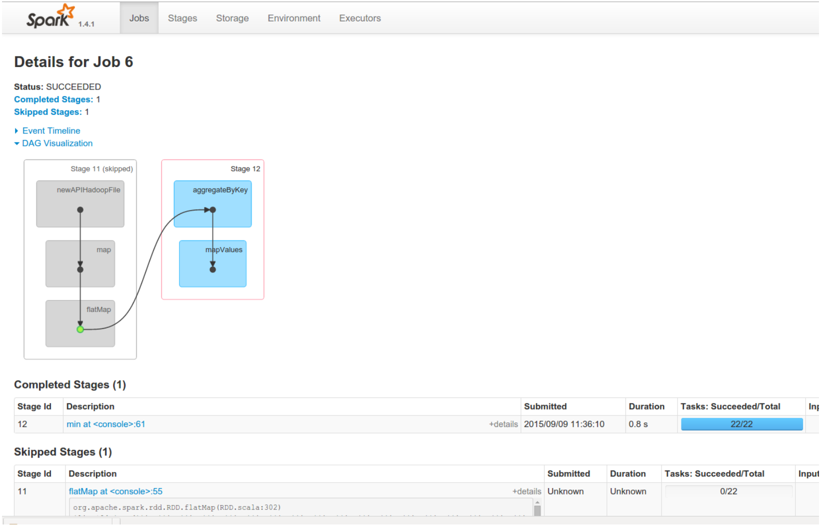

Figure 8: Spark Web UI - Job/DAG

Spark Web UI - Storage

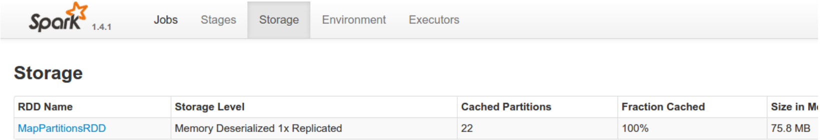

Figure 9: Spark Web UI - Storage

Spark SQL

```scala
import org.apache.spark.sql._
import com.databricks.spark.avro._

val recordsDf = sqlContext.avroFile("enron.avro")

val uniqueFroms =
  recordsDf.select("from").distinct.count()
```

- http://spark-packages.org/ - also MongoDB, Cassandra, HBase...

Spark Streaming - DStreams

```
val conf = new SparkConf().setMaster("local[2]").
   setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))

ssc.start()
ssc.awaitTermination()
```

Example from http://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark GraphX

- Property graph
  - directed multigraph
  - user-defined objects attached to each vertex and edge
- Logical representation:

```
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
}
```

Spark GraphX Operations

- mapVertices, mapEdges, reverse
- subgraph (vertex/edge conditions)
- groupEdges, joinVertices
- collectNeighborIds
- Graph Algorithms
  - Page Rank
  - Connected Components
  - Triangle count

Spark MLLib

- Classification and regression
  - Support Vector Machines, logistic/linear regression
  - Decision trees, random forests...
- Clustering
  - k-means
  - latent Dirichlet allocation (LDA)
- Dimensionality reduction
  - Singular Value decomposition (SVD)
  - Principal Component Analysis (PCA)
- ...
- See http://spark.apache.org/docs/latest/mllib-guide.html and http://spark.apache.org/docs/latest/ml-guide.html

Challenges

- Task serialization
- Parallelism - partitions
  - coalese(numPartitions)
  - repartition(numPartitions)
- Parameter tuning (http://spark.apache.org/docs/latest/tuning.html)
  - Broadcast ($\sim$ Hadoop Distributed Cache)
  - Garbage Collection - Project Tungsten

Learning Resources

- https://github.com/medale/spark-mail
- https://github.com/medale/spark-mail-docker
- O'Reilly: Learning Spark, Advanced Analytics with Spark
- EdX:
  - Introduction to Big Data with Apache Spark
  - Scalable Machine Learning
- Coursera: 2 Scala MOOCs by Martin Odersky
- Databricks: https://databricks.com/spark/developer-resources

References I

References II

Armbrust, Michael. 2014. "Intro to Spark and Spark SQL." Berkeley, CA, USA. http://www.slideshare.net/jeykottalam/spark-sqlamp-camp2014.

Armbrust, Michael, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, et al. 2015. "Spark SQL - Relational Data Processing in Spark." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–94. SIGMOD '15. New York, NY, USA: ACM. doi:10.1145/2723372.2742797.

Ecosystem. 2015. "Databricks Spark Ecosystem." https://databricks.com/spark/about.

Meng, Xiangrui, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, et al. 2015. "MLlib - Machine Learning in Apache Spark." *arXiv Preprint arXiv:1505.06807*.

SparkWebsite. 2015. "Spark - Cluster Overview."