

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 and R APIs
 - ▶ 2-5x less code than Hadoop M/R
- ▶ Unified batch, SQL, streaming, graph and machine learning

Apache Spark Buzz

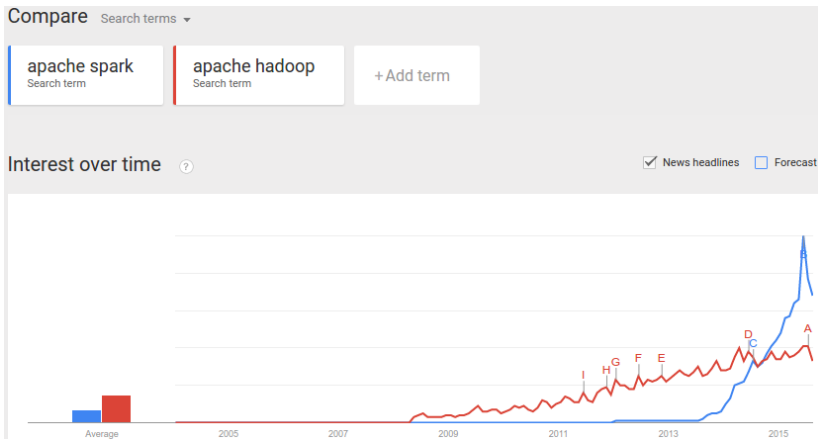


Figure : Google Trends Apache Spark/Apache Hadoop August 2015

Spark Ecosystem

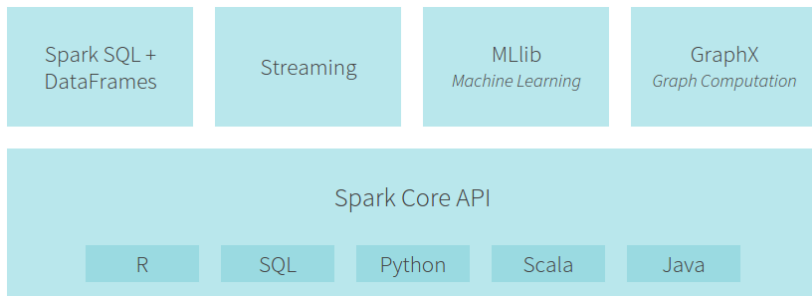
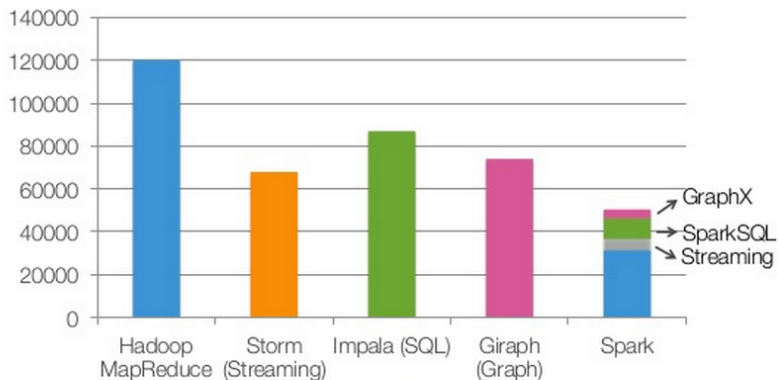


Figure : Databricks Spark 1.4.1 Ecosystem (2015)

Spark Lines of Code



non-test, non-example source lines



Figure : 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

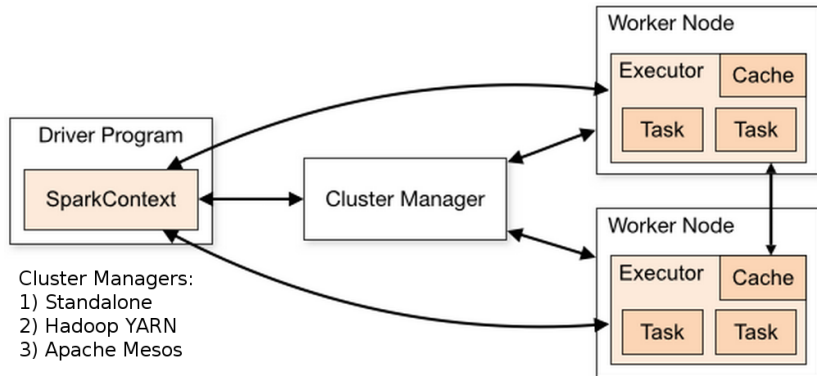


Figure : Spark Cluster Managers SparkWebsite (2015)

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

- ▶ 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

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)

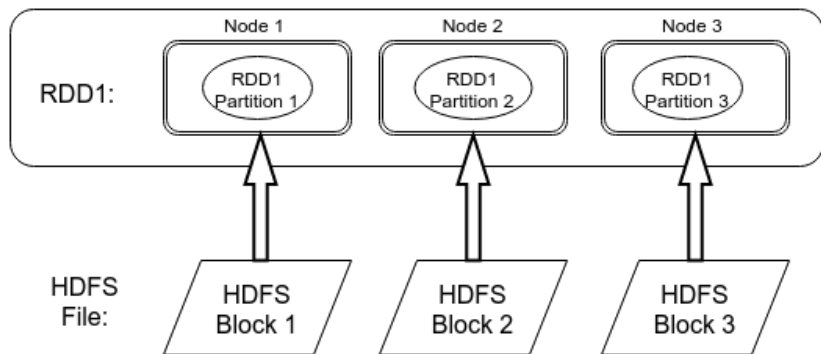


Figure : RDD partitions

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

```
//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 a "collection"
 - ▶ Could be empty
 - ▶ Could have 1 to many output elements
- ▶ flatten - take each element in output "collection" and copy it to overall output List
 - ▶ remove one level of nesting (flatten)

flatMap Example

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

com.uebercomputing.analytics.basic.BasicRddFunctions

```
//compiler can infer bodiesRdd type - reader clarity
val bodiesRdd: RDD[String] =
    analyticInput.mailRecordRdd.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

The screenshot shows the Spark Scala API documentation for RDD. The left sidebar has a search bar with 'RDD' entered. Below the search bar, there's a list of packages: 'org.apache.spark.mllib.rdd' and 'org.apache.spark.rdd'. The 'org.apache.spark.rdd' package is expanded, showing a list of classes: 'AsyncRDDActions', 'CoGroupedRDD', 'DoubleRDDFunctions', 'HadoopRDD', 'JdbcRDD', 'NewHadoopRDD', 'OrderedRDDFunctions', 'PairRDDFunctions', 'PartitionPruningRDD', 'RDD', 'SequenceFileRDDFunctions', 'ShuffledRDD', and 'UnionRDD'. The 'RDD' class is selected. The main content area shows the Scala API for RDD, including the following methods:

- `count(): Long`: Return the number of elements in the RDD.
- `countApprox(timeout: Long, confidence: Double): PartialResult[BoundedDouble]`: Approximate version of count() that returns a potentially all tasks have finished.
- `countApproxDistinct(relativeSD: Double = 0.05): Long`: Return approximate number of distinct elements in the RDD.
- `countApproxDistinct(p: Int, sp: Int): Long`: Return approximate number of distinct elements in the RDD.
- `countByValue()(implicit ord: Ordering[T]): Map[T, Long]`: Return the count of each unique value in this RDD as a Map.
- `countByValueApprox(timeout: Long, confidence: Double, implicit ord: Ordering[T] = null): Map[T, Long]`: Approximate version of countByValue().
- `dependencies: Seq[Dependency[_]]`: Get the list of dependencies of this RDD taking into account all transformations.

Figure : 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))
```

Spark Web UI - Tour

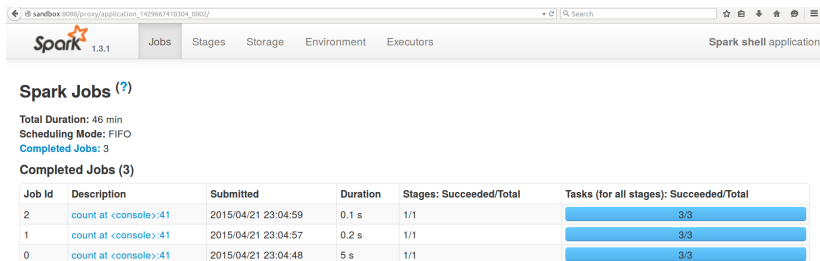


Figure : Spark Web UI

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

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*, 1383–1394. SIGMOD '15. New York, NY, USA: ACM.
doi:10.1145/2723372.2742797.
<http://doi.acm.org/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.

References II

“MLlib - Machine Learning in Apache Spark.” *ArXiv Preprint ArXiv:1505.06807*.

SparkWebsite. 2015. “Spark - Cluster Overview.” <http://spark.apache.org/docs/latest/cluster-overview.html>.

Xin, Reynold S., Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. 2013. “GraphX - A Resilient Distributed Graph System on Spark.” In *First International Workshop on Graph Data Management Experiences and Systems*, 2:1–2:6. GRADES '13. New York, NY, USA: ACM. doi:10.1145/2484425.2484427. <http://doi.acm.org/10.1145/2484425.2484427>.

Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.” In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2–2. NSDI'12. Berkeley, CA, USA:

References III

USENIX Association.

<http://dl.acm.org/citation.cfm?id=2228298.2228301>.

Zaharia, Matei, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. "Spark - Cluster Computing with Working Sets." In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 10:10.