

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
- ▶ Interactive data exploration via spark-shell

Spark 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 : Spark GraySort Results Xin (2014)

Apache Spark Buzz

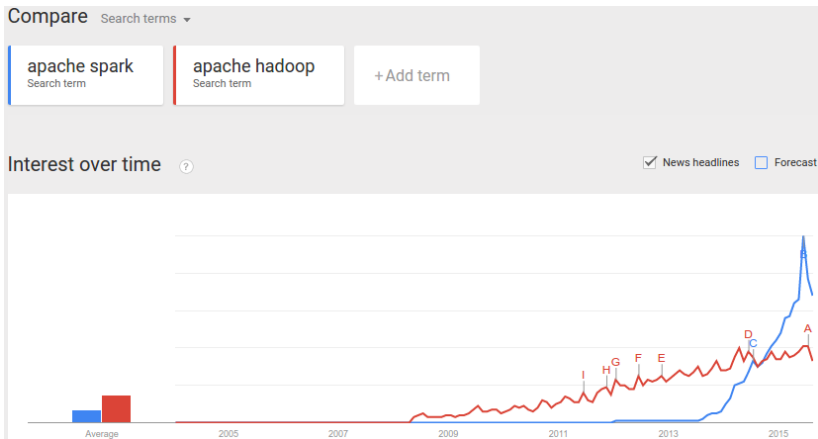


Figure : Google Trends Apache Spark/Apache Hadoop August 2015

Spark Ecosystem

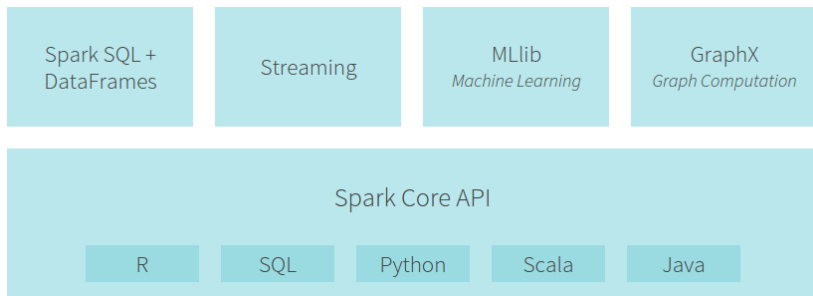
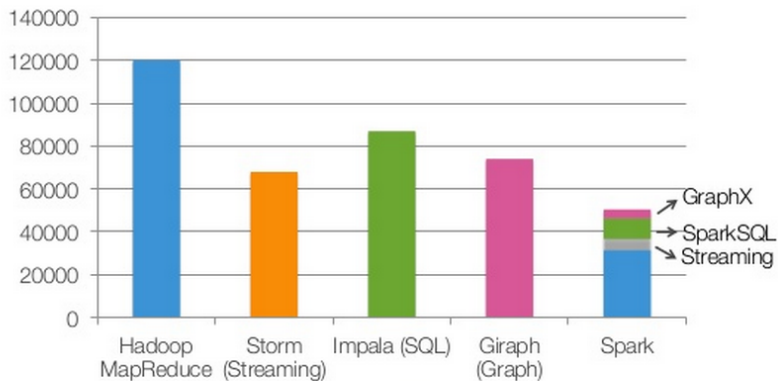


Figure : Databricks Spark 1.4.1 Ecosystem (2015)

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

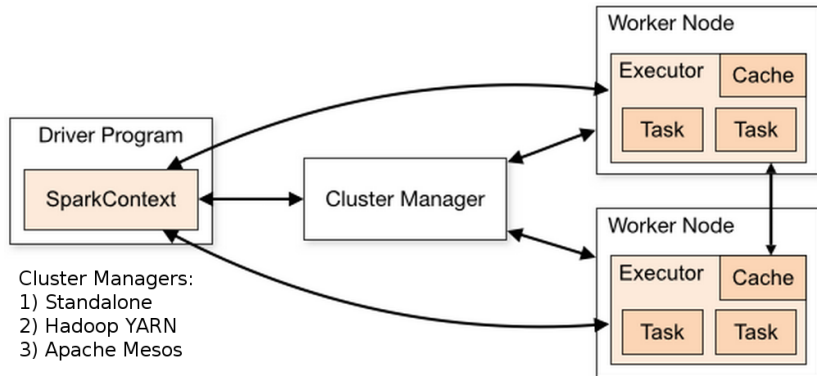


Figure : Spark Cluster Managers SparkWebsite (2015)

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

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

```
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)

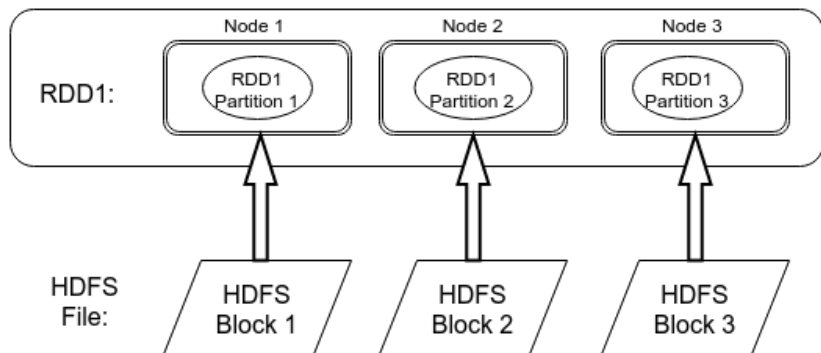


Figure : RDD partitions

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] =
    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 the `RDD` trait. The left sidebar contains a search bar with the text "RDD" and a list of packages. The main content area displays the Scala code for the `RDD` trait, including the `count()`, `countApprox()`, `countApproxDistinct()`, `countByValue()`, `countByValueApprox()`, and `dependencies` methods.

Search: **RDD**

display packages only

org.apache.spark.mllib.rdd

- MLPairRDDFunctions
- RDDFunctions

hide focus

org.apache.spark.rdd

- AsyncRDDActions
- CoGroupedRDD
- DoubleRDDFunctions
- HadoopRDD
- JdbcRDD
- NewHadoopRDD
- OrderedRDDFunctions
- PairRDDFunctions
- PartitionPruningRDD
- RDD
- SequenceFileRDDFunctions
- ShuffledRDD
- UnionRDD

The `org.apache.spark.SparkContext` that this RDD was created in.

def **count**(): Long

Return the number of elements in the RDD.

def **countApprox**(timeout: Long, confidence: Double): [PartialResult\[BoundedDouble\]](#)

Approximate version of count() that returns a potentially approximate result once all tasks have finished.

def **countApproxDistinct**(relativeSD: Double = 0.05): Long

Return approximate number of distinct elements in the RDD.

def **countApproxDistinct**(p: Int, sp: Int): Long

Return approximate number of distinct elements in the RDD.

def **countByValue**()(implicit ord: Ordering[T]): Map[T, Long]

Return the count of each unique value in this RDD as a Map.

def **countByValueApprox**(timeout: Long, confidence: Double)(implicit ord: Ordering[T] = null): [PartialResult\[BoundedDouble\]](#)

Approximate version of countByValue().

final def **dependencies**: Seq[[Dependency](#)[_]]

Get the list of dependencies of this RDD taking into account all partitions.

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))
```


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 (~ 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

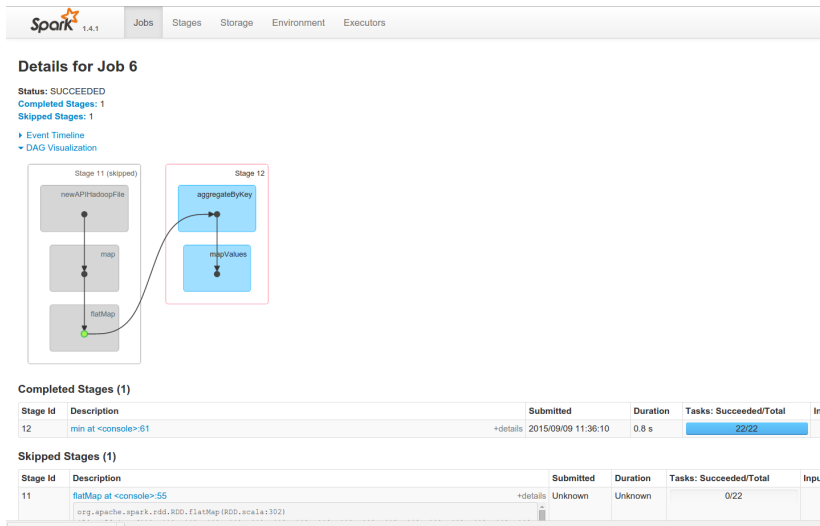


Figure : Spark Web UI - Job/DAG

Spark Web UI - Storage



Figure : Spark Web UI - Storage

Spark SQL

```
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

Spark GraphX

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
 - ▶ `coalesce(numPartitions)`
 - ▶ `repartition(numPartitions)`
- ▶ Parameter tuning
(<http://spark.apache.org/docs/latest/tuning.html>)
 - ▶ Broadcast (~ 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

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. 2014. “Spark Officially Sets a New Record in Large-Scale Sorting.”

<https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.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>.

References III

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: 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.