Apache Spark - The 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?

- Next generation large-scale data processing framework written in Scala
- Replacement for Hadoop MapReduce?
  - In-memory caching
  - Advanced directed acyclic graph of computations - optimized
  - Rich Scala, Java, Python and R APIs - 2-5x less code than Hadoop M/R

Apache Spark Buzz

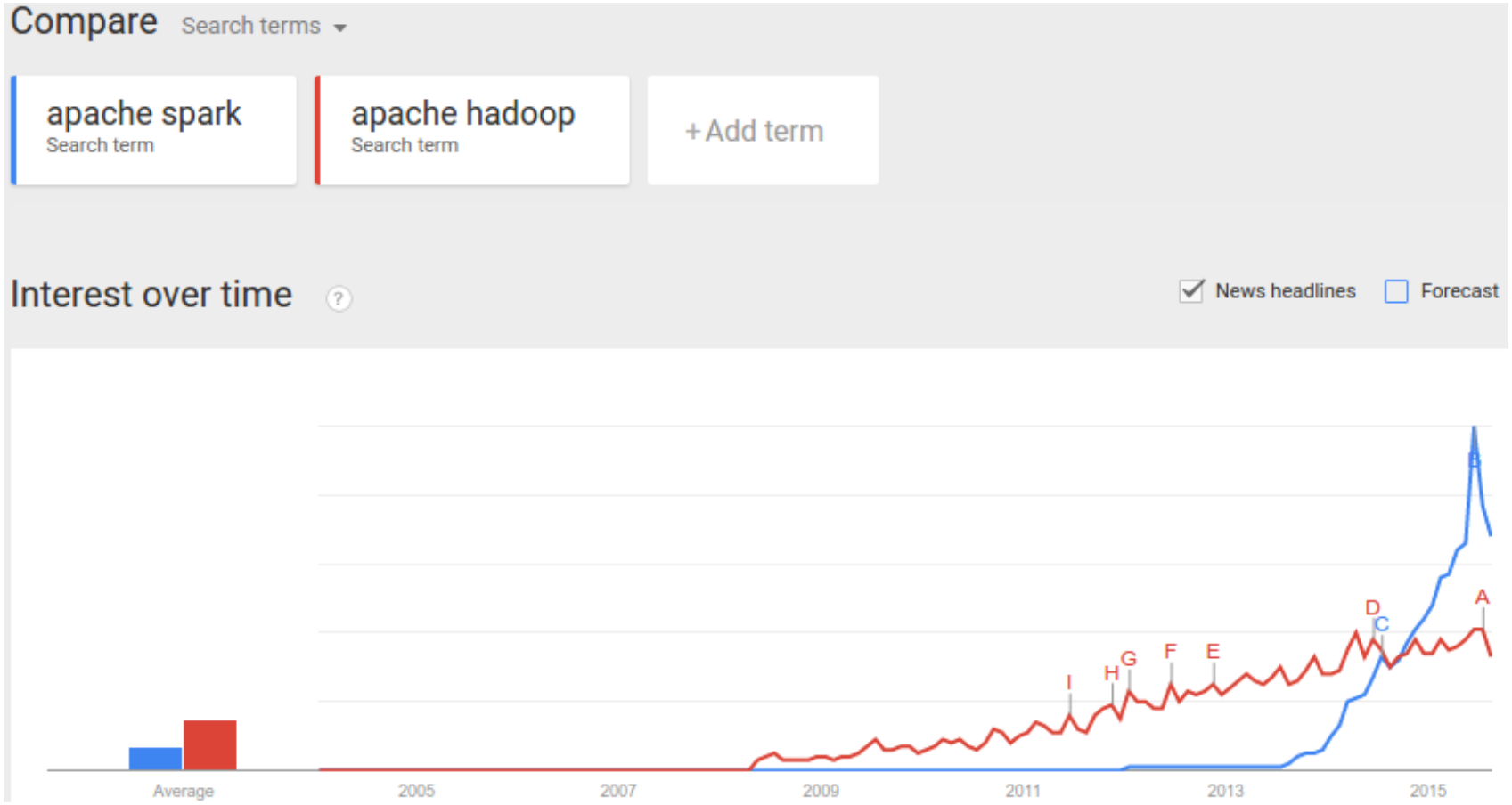

Figure : Google Trends Apache Spark/Apache Hadoop August 2015

Spark Ecosystem

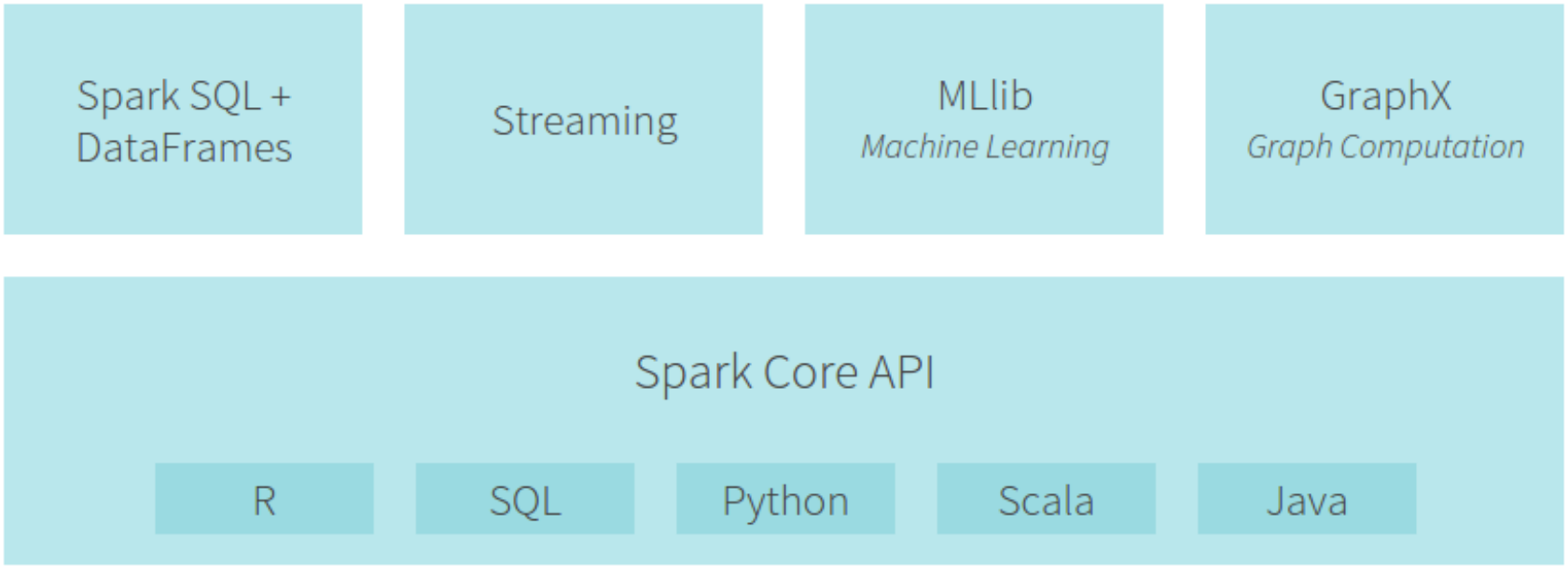

Figure : Databricks Spark 1.4.1 Ecosystem (2015)

Spark Lines of Code

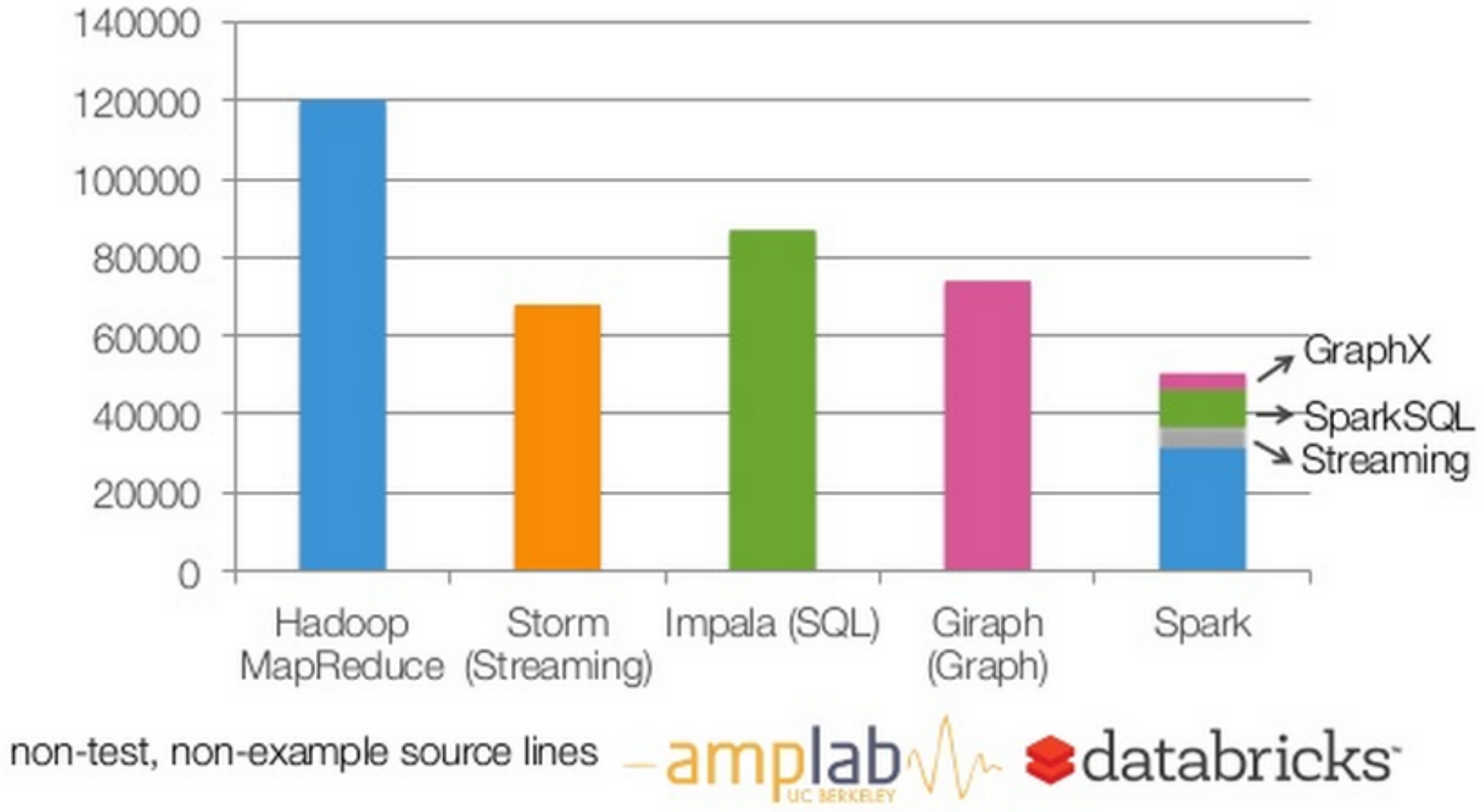

Figure : Spark LOC Armbrust (2014)

Spark Academic Papers

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

Resilient Distributed Dataset (RDD)

- Treat distributed, immutable data set as a collection
- Resilient: Use RDD lineage to recompute failed partitions
- 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)

Combinator functions on Scala collections

- Examples: map, flatMap, filter, reduce, fold, aggregate...

map

- applies a given function to every element of a collection
- returns collection of output of that function (one per original element)
- input argument - same type as collection type
- return type - can be any 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

- ScalaDoc:

```
List[A]
...
def flatMap[B](f: (A) =>
            GenTraversableOnce[B]): List[B]
```

- GenTraversableOnce - List, Array, Option...
- can be empty collection or None
- flatMap takes each element in the GenTraversableOnce and puts it in order to output List[B]
- removes inner nesting - flattens
- output list can be smaller or empty (if intermediates were empty)

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

```
List[A]
...
def 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)
```

reduce

```
List[A]
...
def reduce[A](op: (A, A) => A): A
```

- Creates one cumulative value using the specified associative binary operator.
- op - A binary operator that must be associative.
- returns - The result of applying op between all the elements if the list is nonempty. Result is same type as list type.
- UnsupportedOperationException if this list is empty.

reduce Example

```
//beware of overflow if using default Int!
val numberOfAttachments: List[Long] =
  List(0, 3, 4, 1, 5)
val totalAttachments =
  numberOfAttachments.reduce((x, y) => x + y)
//Order unspecified/non-deterministic, but one
//execution could be:
//0 + 3 = 3, 3 + 4 = 7,
//7 + 1 = 8, 8 + 5 = 13

val emptyList: List[Long] = Nil
//UnsupportedOperationException
emptyList.reduce((x, y) => x + y)
```

fold

```
List[A]
...
def fold[A](z: A)(op: (A, A) => A): A
```

- Very similar to reduce but takes start value z (a neutral value, e.g. 0 for addition, 1 for multiplication, Nil for list concatenation)
- returns start value z for empty list
- Note: See also foldLeft/Right (return completely different type)

```
foldLeft[B](z: B)(f: (B, A)   B): B
```

fold Example

```
val numbers = List(1, 4, 5, 7, 8, 11)
val evenCount = numbers.fold(0) { (count, currVal) =>
  println(s"Count: $count, value: $currVal")
  if (currVal % 2 == 0) {
    count + 1
  } else {
    count
  }
}
Count: 0, value: 1
Count: 0, value: 4
Count: 1, value: 5
Count: 1, value: 7
Count: 1, value: 8
Count: 2, value: 11
evenCount: Int = 2
```

aggregate

```
List[A]
...
def aggregate[B](z: B)(seqop: (B, A) => B,
                       combop: (B, B) => B): B
```

- More general than fold or reduce. Can return different result type.
- Apply seqop function to each partition of data.
- Then apply combop function to combine all the results of seqop.
- On a normal immutable list this is just a foldLeft with seqop (but on a parallelized list both operations are called).

aggregate Example

```
val wordsAll = List("when", "shall", "we", "three",
  "meet", "again", "in", "thunder", "lightning",
  "or", "in", "rain")
//Map(5 letter words ->3, 9->1, 2->4, 7->1, 4->3)
val lengthDistro = wordsAll.aggregate(Map[Int, Int]())(
  seqop = (distMap, currWord) =>
  {
    val length = currWord.length()
    val newCount = distMap.getOrElse(length, 0) + 1
    val newKv = (length, newCount)
    distMap + newKv
  },
  combop = (distMap1, distMap2) => {
    distMap1 ++ distMap2.map {
      case (k, v) =>
      (k, v + distMap1.getOrElse(k, 0))
    }
  })
```

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...

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 - RDD API

- RDD API
- Transforms - map, flatMap, filter, reduce, fold, aggregate...
  - Lazy evaluation (not evaluated until action! Optimizations)
- Actions - count, collect, first, take, saveAsTextFile...

Spark - From RDD to PairRDDFunctions

- If an RDD contains tuples (K,V) - can apply PairRDDFunctions
- Uses implicit conversion of RDD to PairRDDFunctions
- In 1.3 conversion is defined in RDD singleton object
- In 1.2 and previous versions available by importing org.apache.spark.SparkContext._

From 1.3.0 org.apache.spark.rdd.RDD (object):

```
implicit def rddToPairRDDFunctions[K, V](rdd: RDD[(K, V)])
(implicit kt: ClassTag[K], vt: ClassTag[V],
  ord: Ordering[K] = null): PairRDDFunctions[K, V] = {
  new PairRDDFunctions(rdd)
}
```

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 ...

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*.

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: USENIX Association. `http://dl.acm.org/citation.cfm?id=2228298.2228301`.

References III

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.