Spark in Action

Fast Big Data Analytics using Scala

Matei Zaharia

University of California, Berkeley

www.spark-project.org
My Background

Grad student in the AMP Lab at UC Berkeley
   » 50-person lab focusing on big data

Committer on Apache Hadoop

Started Spark in 2009 to provide a richer, Hadoop-compatible computing engine
Spark Goals

Extend the MapReduce model to support more types of applications efficiently
  » Spark can run 40x faster than Hadoop for iterative and interactive applications

Make jobs easier to program
  » Language-integrated API in Scala
  » Interactive use from Scala interpreter
Why go Beyond MapReduce?

MapReduce simplified big data analysis by giving a reliable programming model for large clusters.

But as soon as it got popular, users wanted more:

» More complex, multi-stage applications
» More interactive ad-hoc queries
Why go Beyond MapReduce?

Complex jobs and interactive queries both need one thing that MapReduce lacks:

Efficient primitives for data sharing

Iterative algorithm

Interactive data mining
Why go Beyond MapReduce?

Complex jobs and interactive queries both need one thing that MapReduce lacks:

Efficient primitives for data sharing

In MapReduce, the only way to share data across jobs is stable storage (e.g. HDFS) -> slow!
Examples

HDFS read → iter. 1 → HDFS read → iter. 2 → HDFS write → ... → HDFS write

Input

HDFS read → query 1 → result 1
HDFS read → query 2 → result 2
HDFS read → query 3 → result 3

I/O and serialization can take 90% of the time
Goal: In-Memory Data Sharing

Input

one-time processing

Distributed memory

iter. 1

iter. 2

... 

query 1

query 2

query 3

... 

10-100x faster than network and disk
Solution: Resilient Distributed Datasets (RDDs)

Distributed collections of objects that can be stored in memory for fast reuse

Automatically recover lost data on failure

Support a wide range of applications
Outline

Spark programming model

User applications

Implementation

Demo

What’s next
Programming Model

Resilient distributed datasets (RDDs)
  » Immutable, partitioned collections of objects
  » Can be cached in memory for efficient reuse

Transformations (e.g. map, filter, groupBy, join)
  » Build RDDs from other RDDs

Actions (e.g. count, collect, save)
  » Return a result or write it to storage
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count

... 
```

**Result:** scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
RDD Fault Tolerance

RDDs track the series of transformations used to build them (their lineage) to recompute lost data.

E.g: messages = textFile(...).filter(_.contains("error")).map(_.split(\"\t\")\(2\))
Example: Logistic Regression

Goal: find best line separating two sets of points
Example: Logistic Regression

```scala
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: "+ w)
```
Logistic Regression Performance

Running Time (s) vs. Number of Iterations for Hadoop and Spark.

- Hadoop: 110 s / iteration
- Spark: first iteration 80 s, further iterations 6 s
Spark Users

CONVIVA
foursquare
quantifind
KLOUT
Yahoo!
University of California Berkeley
Princeton University
UCSF
User Applications

In-memory analytics on Hive data (Conviva)
Interactive queries on data streams (Quantifind)
Exploratory log analysis (Foursquare)
Traffic estimation w/ GPS data (Mobile Millennium)
Algorithms for DNA sequence analysis (SNAP)

...
Conviva GeoReport

Group aggregations on many keys with the same WHERE clause

40× gain over Apache Hive comes from avoiding repeated reading, deserialization and filtering
Quantifind Stream Analysis

Load new documents every few minutes

Compute an in-memory table of time series

Let users query interactively via web app
Implementation

Runs on Apache Mesos cluster manager to coexist w/ Hadoop

Supports any Hadoop storage system (HDFS, HBase, ...)

Easy local mode and EC2 launch scripts

No changes to Scala
Task Scheduler

Runs general DAGs

Pipelines functions within a stage

Cache-aware data reuse & locality

Partitioning-aware to avoid shuffles

= cached data partition
Language Integration

Scala closures are Serializable Java objects
  » Serialize on master, load & run on workers

Not quite enough
  » Nested closures may reference entire outer scope, pulling in non-Serializable variables not used inside
  » Solution: bytecode analysis + reflection

Interpreter integration
  » Some magic tracks variables, defs, etc that each line depends on and automatically ships them to workers
Demo
What’s Next?
Hive on Spark (Shark)

Compatible port of the SQL-on-Hadoop engine that can run 40x faster on existing Hive data

Scala UDFs for statistics and machine learning

Alpha coming really soon
Streaming Spark

Extend Spark to perform streaming computations

Run as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

Alpha expected by June

tweetStream
.flatMap(_.toLowerCase.split)
.map(word => (word, 1))
.reduceByWindow(5, _ + _)
Conclusion

Spark offers a simple, efficient and powerful programming model for a wide range of apps

Shark and Spark Streaming coming soon

Download and docs: [www.spark-project.org](http://www.spark-project.org)

@matei_zaharia / matei@berkeley.edu
Related Work

DryadLINQ
  » Build queries through language-integrated SQL operations on lazy datasets
  » Cannot have a dataset persist across queries

Relational databases
  » Lineage/provenance, logical logging, materialized views

Piccolo
  » Parallel programs with shared distributed hash tables; similar to distributed shared memory

Iterative MapReduce (Twister and HaLoop)
  » Cannot define multiple distributed datasets, run different map/reduce pairs on them, or query data interactively
Related Work

Distributed shared memory (DSM)
  » Very general model allowing random reads/writes, but hard to implement efficiently (needs logging or checkpointing)

RAMCloud
  » In-memory storage system for web applications
  » Allows random reads/writes and uses logging like DSM

Nectar
  » Caching system for DryadLINQ programs that can reuse intermediate results across jobs
  » Does not provide caching in memory, explicit support over which data is cached, or control over partitioning

SMR (functional Scala API for Hadoop)
Behavior with Not Enough RAM

<table>
<thead>
<tr>
<th>% of working set in cache</th>
<th>Iteration time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache disabled</td>
<td>68.8</td>
</tr>
<tr>
<td>25%</td>
<td>58.1</td>
</tr>
<tr>
<td>50%</td>
<td>40.7</td>
</tr>
<tr>
<td>75%</td>
<td>29.7</td>
</tr>
<tr>
<td>Fully cached</td>
<td>11.5</td>
</tr>
</tbody>
</table>