Parallel Programming With Spark

Matei Zaharia

UC Berkeley

www.spark-project.org

What is Spark?

Fast and expressive cluster computing system compatible with Apache Hadoop

Improves efficiency through:

» General execution graphs

» In-memory storage

→ Up to 10× faster on disk, 100× in memory

Improves usability through:

» Rich APIs in Java, Scala, Python

» Interactive shell

→ 2-5× less code

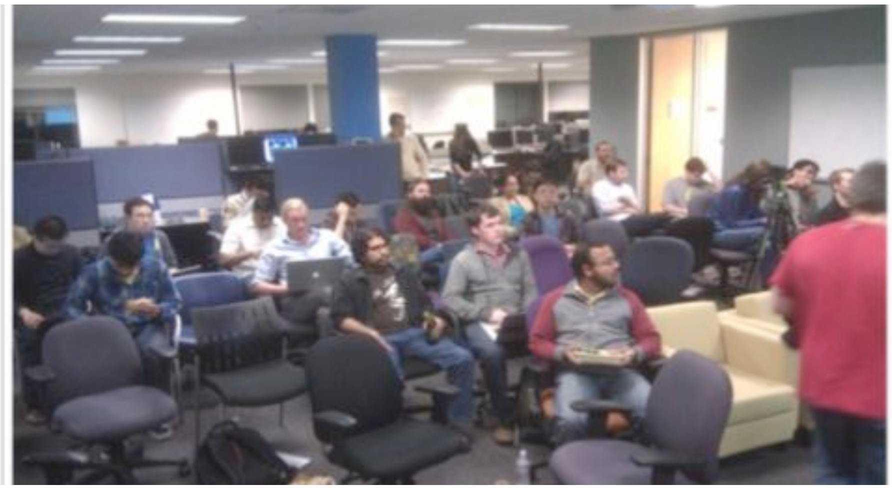
Project History

Spark started in 2009, open sourced 2010

In use at Intel, Yahoo!, Adobe, Quantifind, Conviva, Ooyala, Bizo and others

Entered Apache Incubator in June

Open Source Community



1000+ meetup members

70+ contributors

20 companies contributing

This Talk

Introduction to Spark

Tour of Spark operations

Job execution

Standalone apps

Key Idea

Write programs in terms of transformations on distributed datasets

Concept: resilient distributed datasets (RDDs)

» Collections of objects spread across a cluster
» Built through parallel transformations (map, filter, etc)
» Automatically rebuilt on failure
» Controllable persistence (e.g. caching in RAM)

Operations

Transformations (e.g. map, filter, groupBy)

» Lazy operations to build RDDs from other RDDs

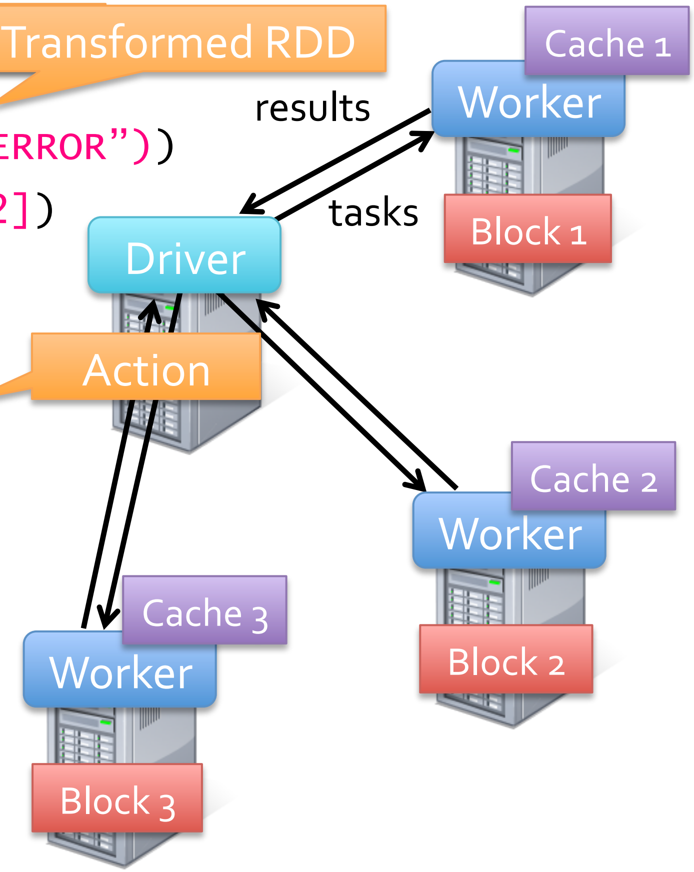
Actions (e.g. count, collect, save)

» 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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
. . .
```

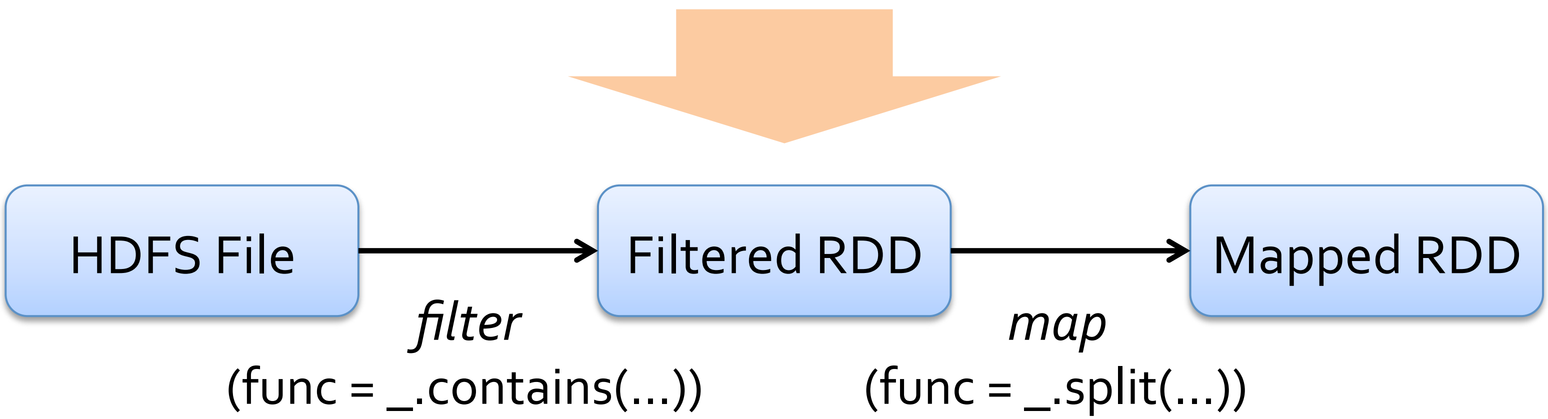


Result: scaled to 1 TB data in 5 sec
(vs 180 sec for on-disk data)

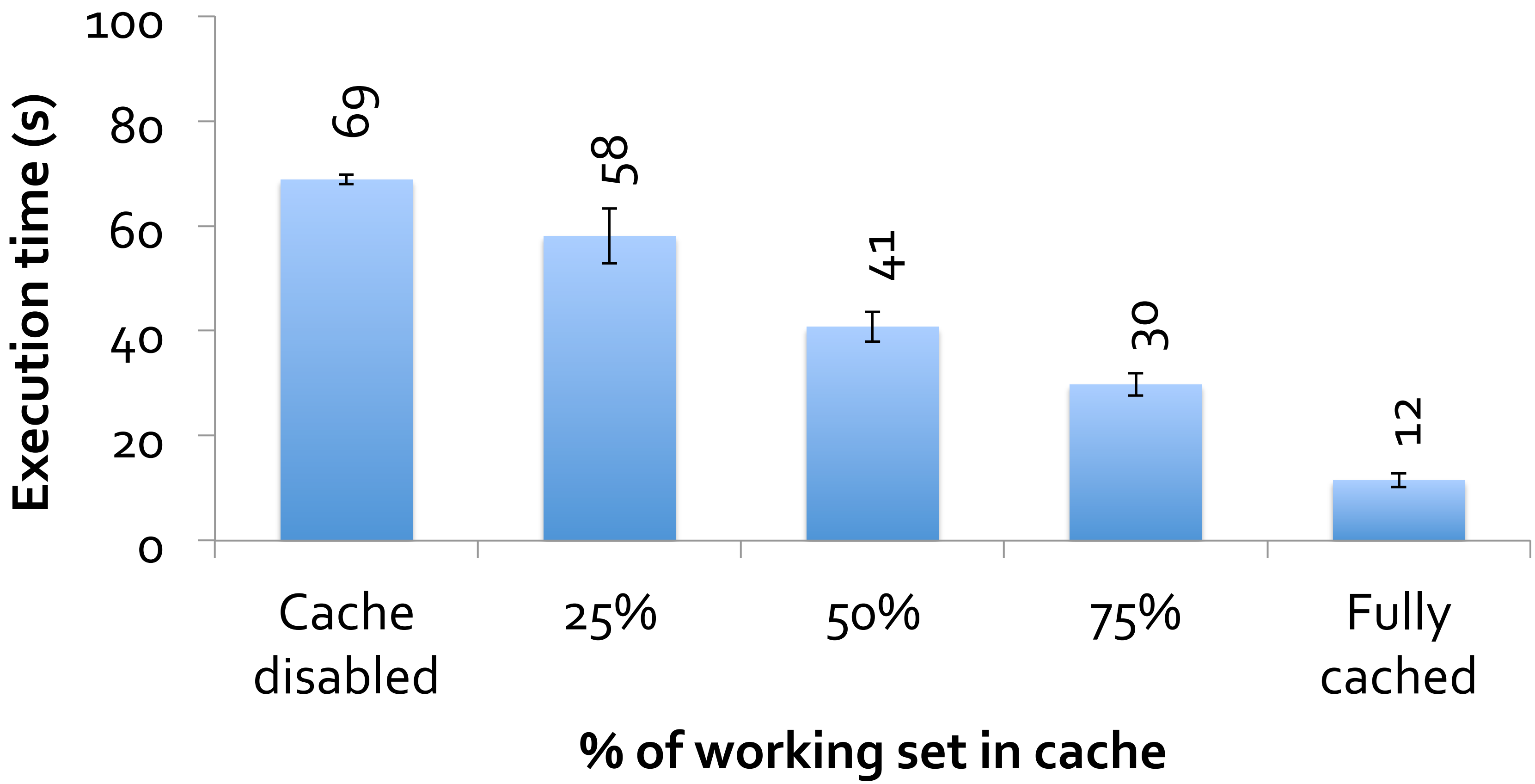
Fault Recovery

RDDs track *lineage* information that can be used to efficiently recompute lost data

Ex:
```
msgs = textFile.filter(lambda s: s.startsWith(“ERROR”))
               .map(lambda s: s.split(“\t”)[2])
```



Behavior with Less RAM



Spark in Scala and Java

```
// Scala:

val lines = sc.textFile(...)
lines.filter(x => x.contains(“ERROR”)).count()


// Java:

JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains(“error”);
  }
}).count();
```

Which Language Should I Use?

Standalone programs can be written in any, but interactive shell is only Python & Scala

Python users: can do Python for both

Java users: consider learning Scala for shell

Performance: Java & Scala are faster due to static typing, but Python is often fine

Scala Cheat Sheet

Variables:

```
var x: Int = 7
var x = 7       // type inferred

val y = "hi"    // read-only
```

Functions:

```
def square(x: Int): Int = x*x

def square(x: Int): Int = {
  x*x   // last line returned
}
```

Collections and closures:

```
val nums = Array(1, 2, 3)

nums.map((x: Int) => x + 2) // {3,4,5}
nums.map(x => x + 2)        // same
nums.map(_ + 2)             // same

nums.reduce((x, y) => x + y) // 6
nums.reduce(_ + _)           // same
```

Java interop:

```
import java.net.URL

new URL("http://cnn.com").openStream()
```

More details: scala-lang.org

This Talk

Introduction to Spark

Tour of Spark operations

Job execution

Standalone apps

Learning Spark

Easiest way: the shell (`spark-shell` or `pyspark`)

» Special Scala / Python interpreters for cluster use

Runs in local mode on 1 core by default, but can control with `MASTER` environment var:

```
MASTER=local    ./spark-shell   # local, 1 thread
MASTER=local[2] ./spark-shell   # local, 2 threads
MASTER=spark://host:port ./spark-shell  # cluster
```

First Stop: SparkContext

Main entry point to Spark functionality

Available in shell as variable `sc`

In standalone programs, you'd make your own (see later for details)

Creating RDDs

```
# Turn a Python collection into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
sc.textFile(“file.txt”)
sc.textFile(“directory/*.txt”)
sc.textFile(“hdfs://namenode:9000/path/file”)

# Use existing Hadoop InputFormat (Java/Scala only)
sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```

Basic Transformations

```
nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
even = squares.filter(lambda x: x % 2 == 0) // {4}

# Map each element to zero or more others
nums.flatMap(lambda x: => range(x))
   # => {0, 0, 1, 0, 1, 2}
```

Range object (sequence of numbers 0, 1, …, x-1)

Basic Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Return first K elements
nums.take(2)   # => [1, 2]

# Count number of elements
nums.count()   # => 3

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

Working with Key-Value Pairs

Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

Python:

```python
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

Scala:

```scala
val pair = (a, b)
pair._1 // => a
pair._2 // => b
```

Java:

```java
Tuple2 pair = new Tuple2(a, b);
pair._1 // => a
pair._2 // => b
```

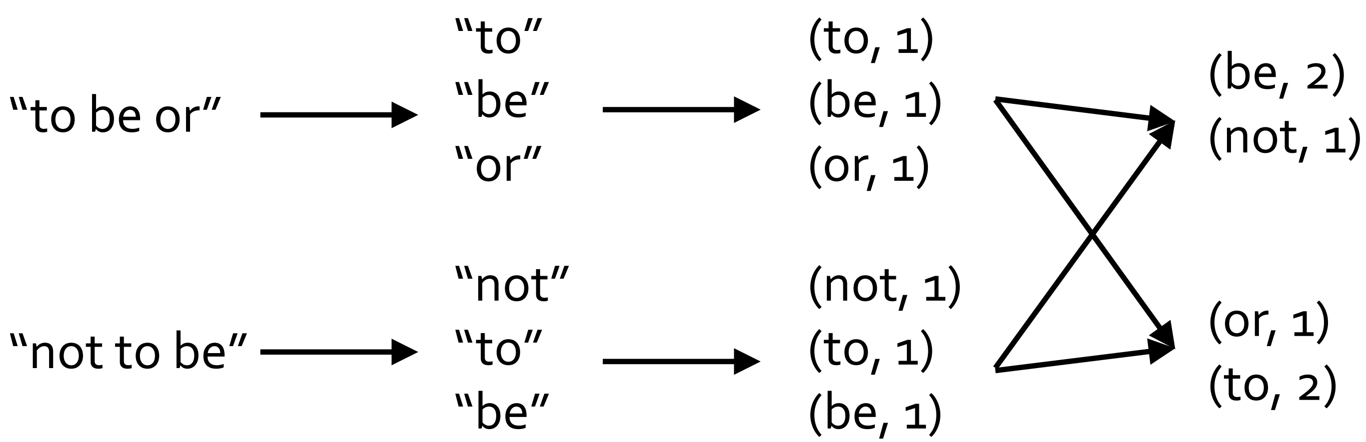
Some Key-Value Operations

```
pets = sc.parallelize(
  [(“cat”, 1), (“dog”, 1), (“cat”, 2)])

pets.reduceByKey(lambda x, y: x + y)
                  # => {(cat, 3), (dog, 1)}

pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}

pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog, 1)}
```

`reduceByKey` also automatically implements combiners on the map side

Example: Word Count

```
lines = sc.textFile(“hamlet.txt”)

counts = lines.flatMap(lambda line: line.split(“ ”))
              .map(lambda word => (word, 1))
              .reduceByKey(lambda x, y: x + y)
```



Other Key-Value Operations

```
visits = sc.parallelize([ (“index.html”, “1.2.3.4”),
                          (“about.html”, “3.4.5.6”),
                          (“index.html”, “1.3.3.1”) ])

pageNames = sc.parallelize([ (“index.html”, “Home”),
                             (“about.html”, “About”) ])

visits.join(pageNames)
# (“index.html”, (“1.2.3.4”, “Home”))
# (“index.html”, (“1.3.3.1”, “Home”))
# (“about.html”, (“3.4.5.6”, “About”))

visits.cogroup(pageNames)
# (“index.html”, ([“1.2.3.4”, “1.3.3.1”], [“Home”]))
# (“about.html”, ([“3.4.5.6”], [“About”]))
```

Setting the Level of Parallelism

All the pair RDD operations take an optional second parameter for number of tasks

```
words.reduceByKey(lambda x, y: x + y, 5)

words.groupByKey(5)

visits.join(pageViews, 5)
```

Using Local Variables

Any external variables you use in a closure will automatically be shipped to the cluster:

```
query = sys.stdin.readline()
pages.filter(lambda x: query in x).count()
```

Some caveats:

» Each task gets a new copy (updates aren't sent back)

» Variable must be Serializable / Pickle-able

» Don't use fields of an outer object (ships all of it!)

Closure Mishap Example

```
class MyCoolRddApp {
  val param = 3.14
  val log = new Log(...)
  ...

  def work(rdd: RDD[Int]) {
    rdd.map(x => x + param)
       .reduce(...)
  }
}
```

NotSerializableException: MyCoolRddApp (or Log)

How to get around it:

```
class MyCoolRddApp {
  ...

  def work(rdd: RDD[Int]) {
    val param_ = param
    rdd.map(x => x + param_)
       .reduce(...)
  }
}
```

References only local variable instead of `this.param`

Other RDD Operators

map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	cross	save
rightOuterJoin	zip	...

More details: spark-project.org/docs/latest/

This Talk

Introduction to Spark

Tour of Spark operations

Job execution

Standalone apps

Software Components

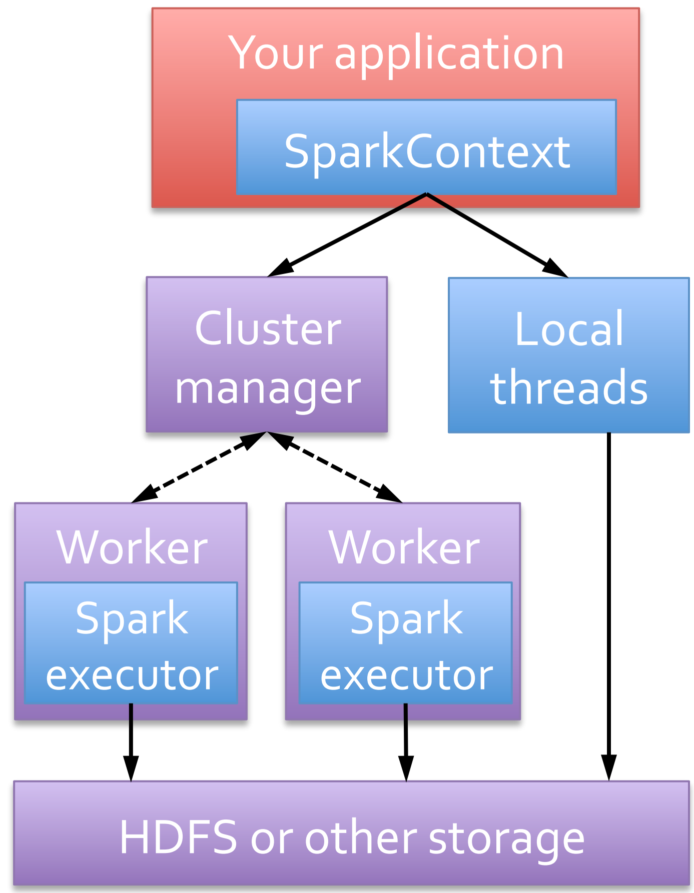
Spark runs as a library in your program (1 instance per app)

Runs tasks locally or on cluster

» Mesos, YARN or standalone mode

Accesses storage systems via Hadoop InputFormat API

» Can use HBase, HDFS, S3, …



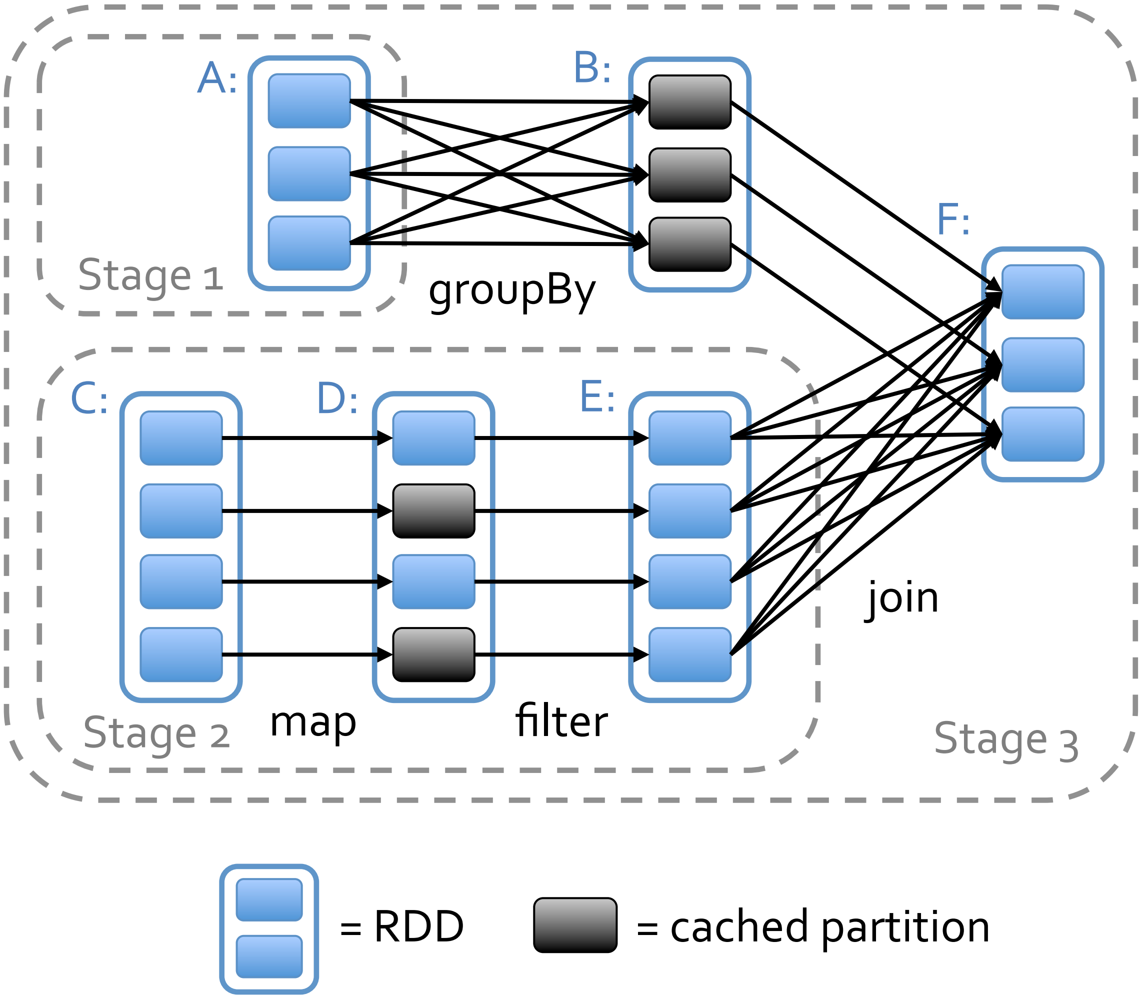
Task Scheduler

General task graphs

Automatically pipelines functions

Data locality aware

Partitioning aware to avoid shuffles



Advanced Features

Controllable partitioning

» Speed up joins against a dataset

Controllable storage formats

» Keep data serialized for efficiency, replicate to multiple nodes, cache on disk

Shared variables: broadcasts, accumulators

See online docs for details!

This Talk

Introduction to Spark

Tour of Spark operations

Job execution

Standalone apps

Add Spark to Your Project

Scala / Java: add a Maven dependency on

groupId: `org.spark-project`
artifactId: `spark-core_2.9.3`
version: `0.7.3`

Python: run program with our `pyspark` script

Create a SparkContext

Scala

```scala
import spark.SparkContext
import spark.SparkContext._

val sc = new SparkContext("url", "name", "sparkHome", Seq("app.jar"))
```

Cluster URL, or local / local[N]

App name

Spark install path on cluster

List of JARs with app code (to ship)

Java

```java
import spark.api.j            nt

JavaSparkContext sc = new JavaSparkContext(
    "masterUrl", "name", "sparkHome", new String[] {"app.jar"}));
```

Python

```python
from pyspark import SparkContext

sc = SparkContext("masterUrl", "name", "sparkHome", ["library.py"]))
```

Example: PageRank

Good example of a more complex algorithm

» Multiple stages of map & reduce

Benefits from Spark’s in-memory caching

» Multiple iterations over the same data

Basic Idea

Give pages ranks (scores) based on links to them

- » Links from many pages ➔ high rank
- » Link from a high-rank page ➔ high rank

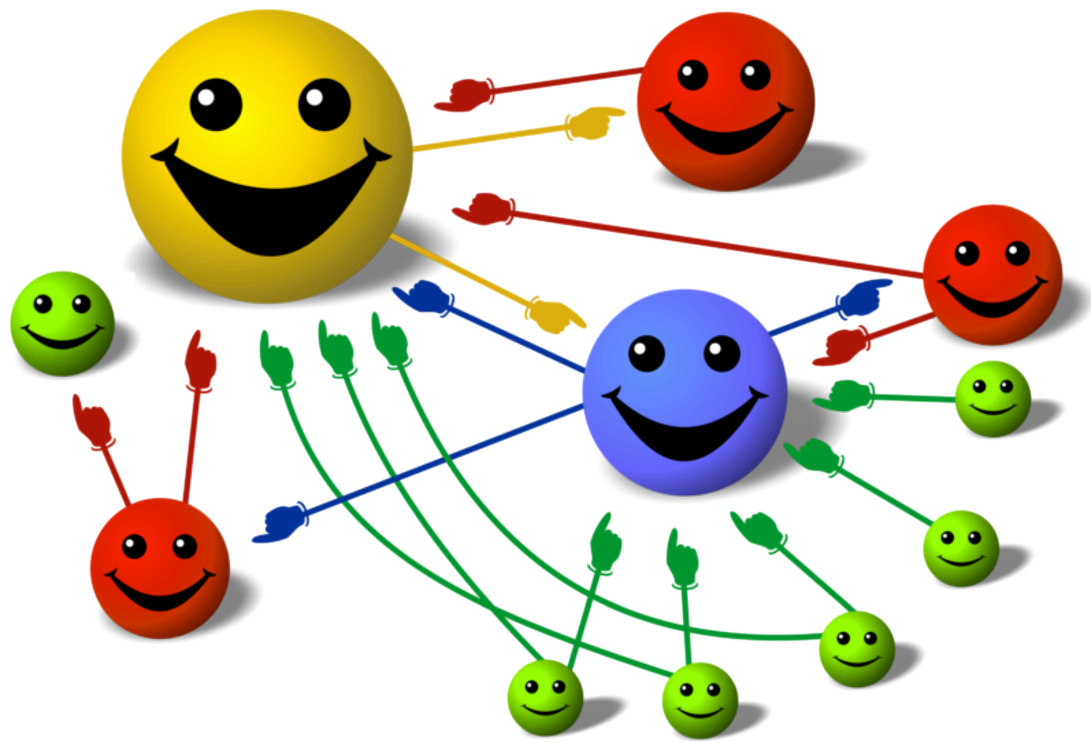
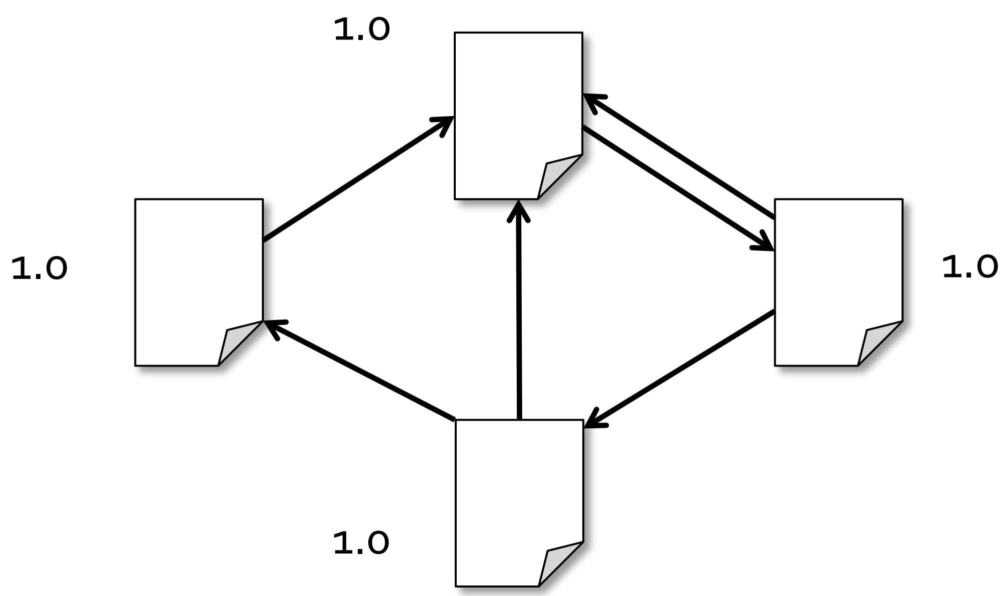


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

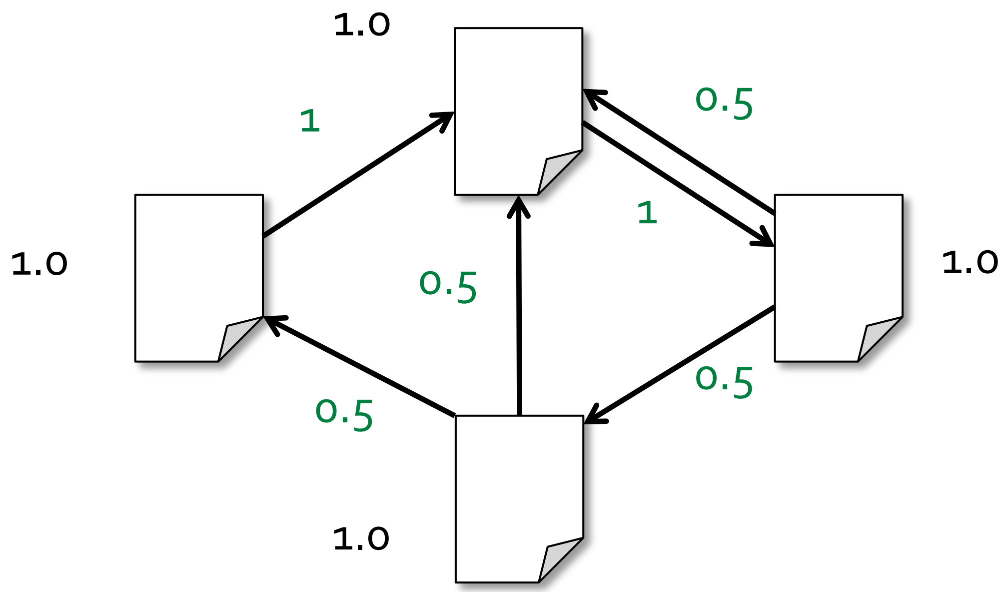
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times contribs$



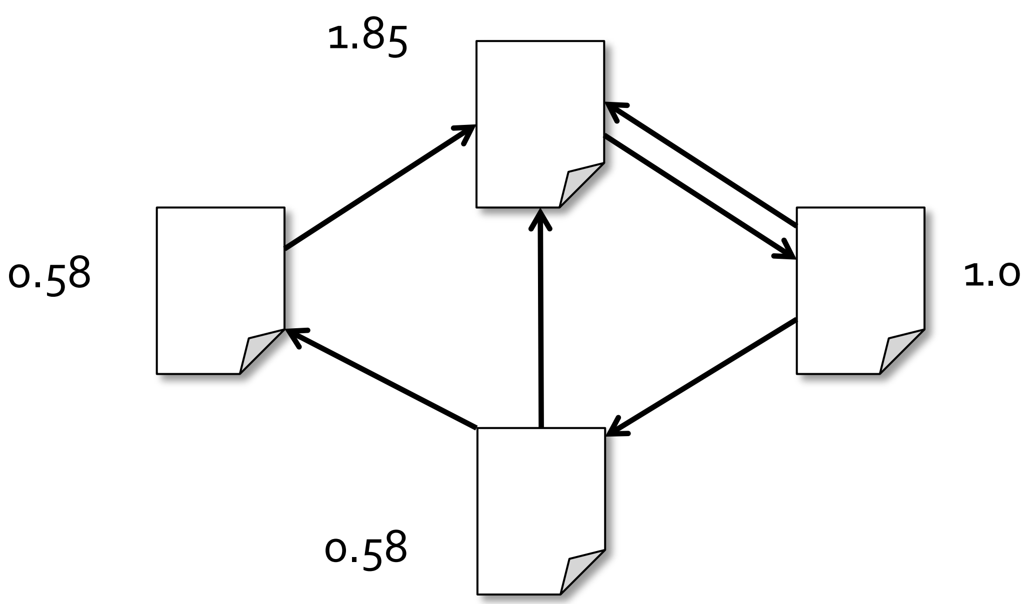
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p / |neighbors_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times contribs$



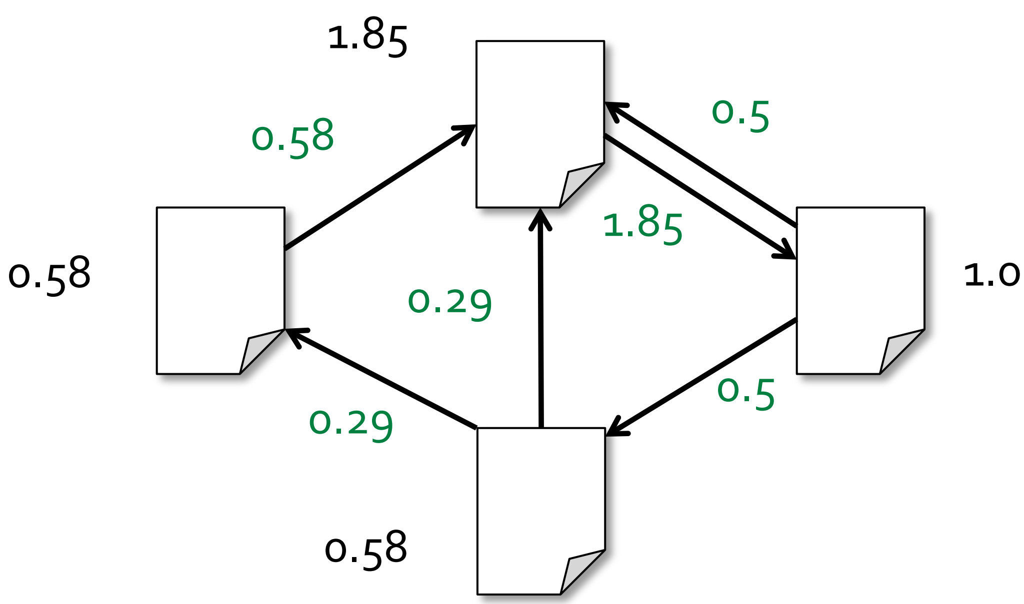
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



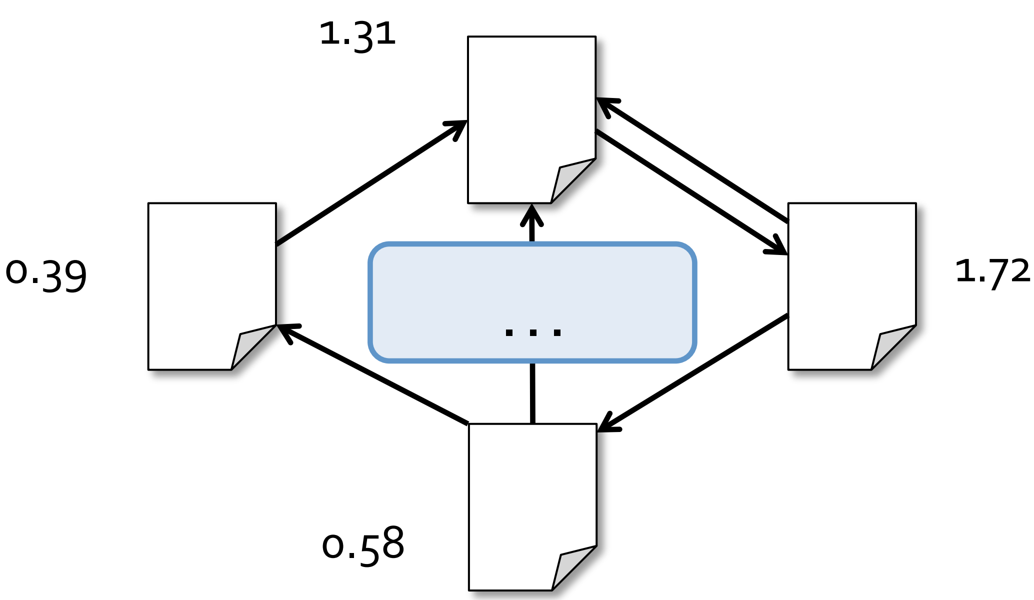
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



Algorithm

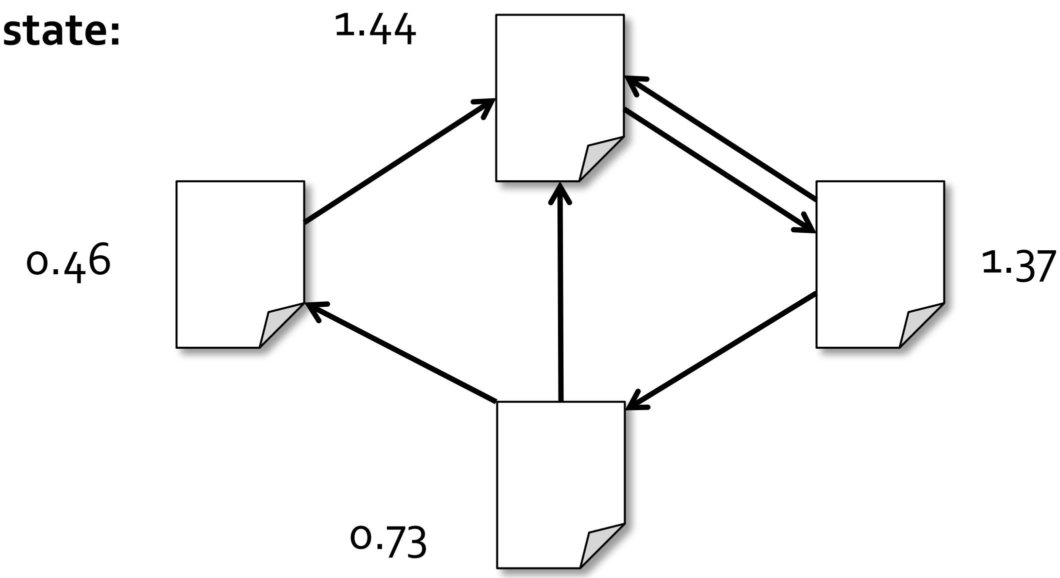
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times \text{contribs}$



Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

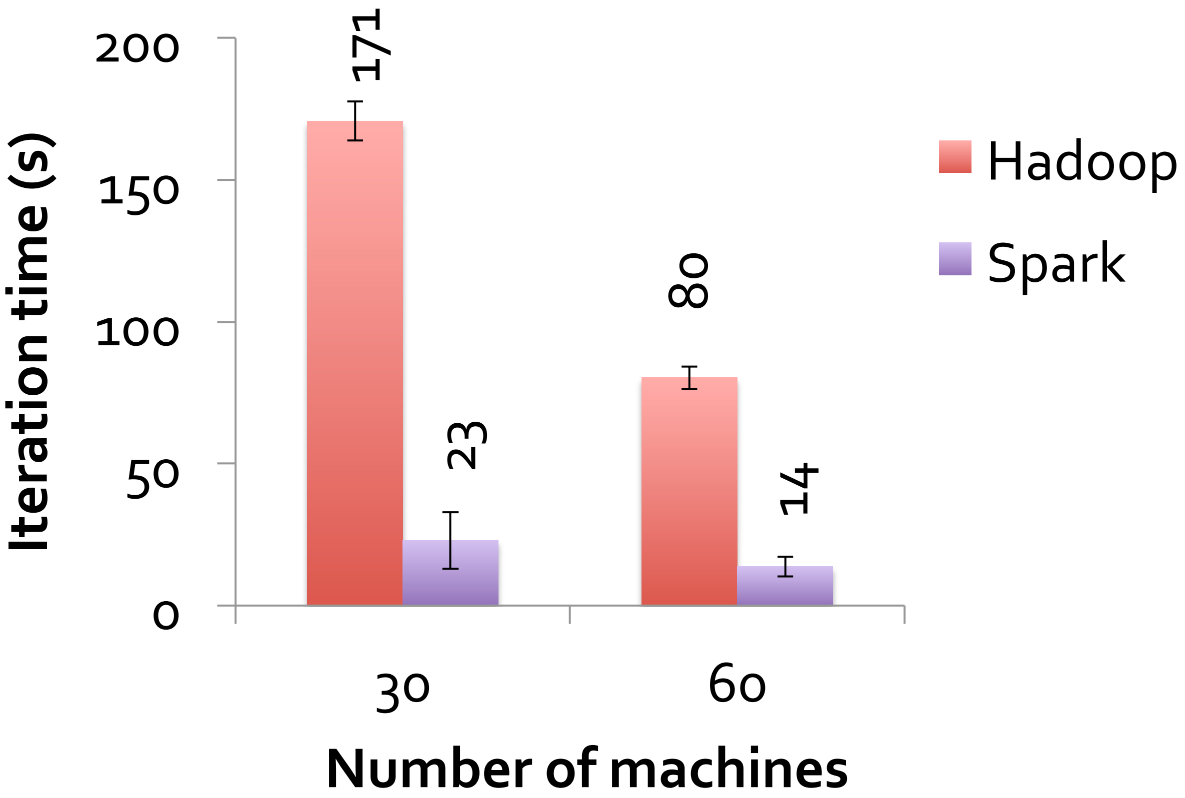
Final state:



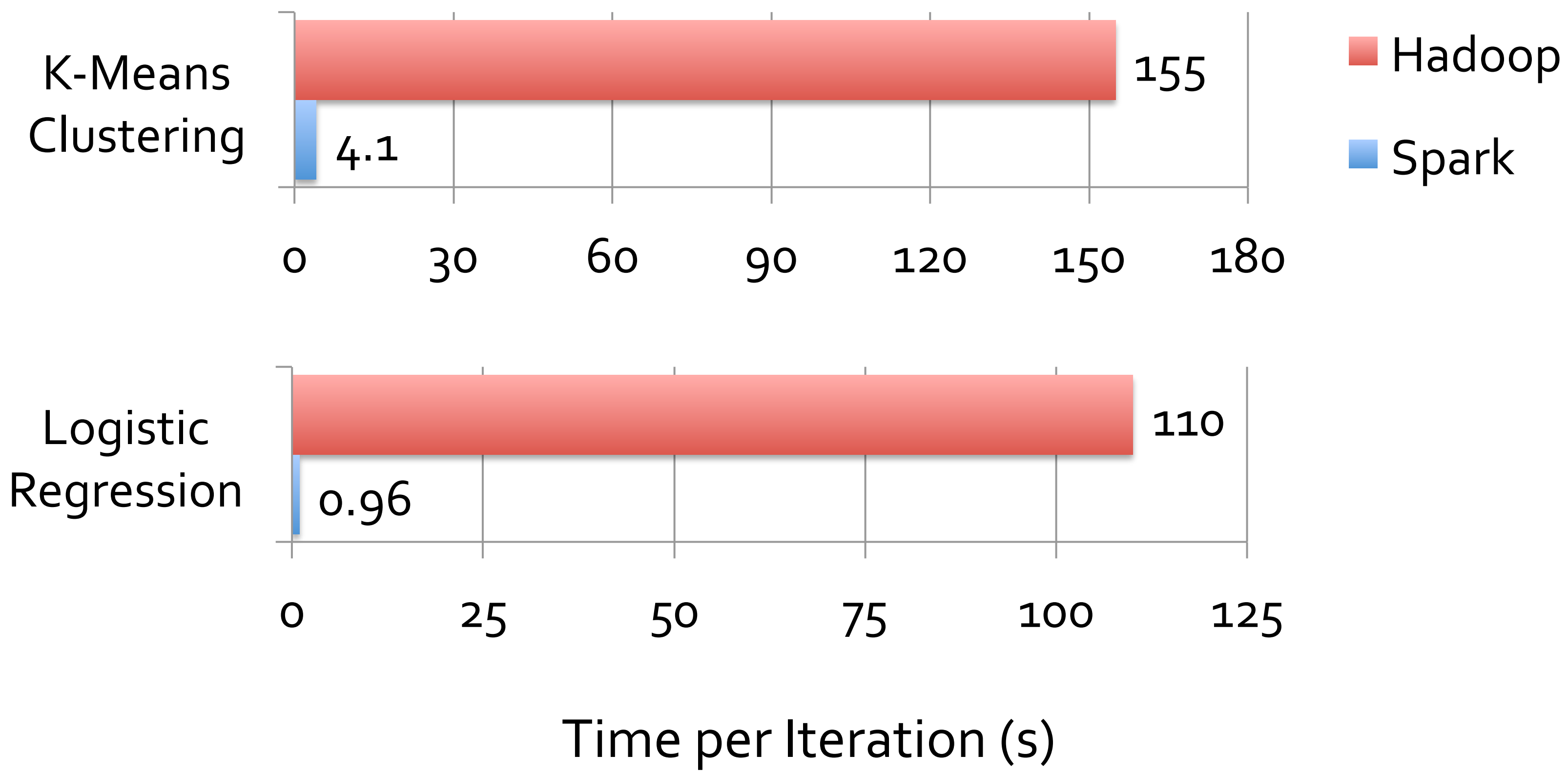
Scala Implementation

```
val sc = new SparkContext("local", "PageRank", sparkHome,
                          Seq("pagerank.jar"))

val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

PageRank Performance



Other Iterative Algorithms



Getting Started

Download Spark: spark-project.org/downloads

Documentation and video tutorials:
www.spark-project.org/documentation

Several ways to run:

» Local mode (just need Java), EC2, private clusters

Local Execution

Just pass `local` or `local[k]` as master URL

Debug using local debuggers

- » For Java / Scala, just run your program in a debugger
- » For Python, use an attachable debugger (e.g. PyDev)

Great for development & unit tests

Cluster Execution

Easiest way to launch is EC2:

```
./spark-ec2 -k keypair –i id_rsa.pem –s slaves \
  [launch|stop|start|destroy] clusterName
```

Several options for private clusters:

» Standalone mode (similar to Hadoop's deploy scripts)

» Mesos

» Hadoop YARN

Amazon EMR: tinyurl.com/spark-emr

Conclusion

Spark offers a rich API to make data analytics *fast*: both fast to write and fast to run

Achieves 100x speedups in real applications

Growing community with 20+ companies contributing

www.spark-project.org