



# Big Data - Analytics with Spark

Dr. Qing “Matt” Zhang  
ITU



# Outline

- Why Spark
- The Spark programming model
- Language and deployment choices
- Example algorithm (PageRank)



# Introduction

# What is Spark?

Fast and Expressive Cluster Computing  
Engine Compatible with Apache Hadoop

Up to **10x** faster on disk,  
**100x** in memory

## Efficient

- General execution graphs
- In-memory storage

**2-5x** less code

## Usable

- Rich APIs in Java, Scala, Python
- Interactive shell



# Background

- Hadoop introduced a radical new approach based on two key concepts
  - Distribute data when the data is stored
  - Move computation to data
- Spark takes this new approach to the next level
  - Data is distributed in memory
- Apache Spark is a fast, general engine for large scale data processing on a cluster
  - Originally developed at AMPLab at UC Berkeley
  - Started as a research project in 2009
  - The creators founded Databricks to commercialize Spark

# The Spark Community

March 27th 2010 - November 30th 2013

Commits to master, excluding merge commits

Contribution Type: **Commits** ▼



# Get Started

## Up and Running in a Few Steps

- Download
- Unzip
- Shell

## Project Resources

- Examples on the Project Site
- Examples in the Distribution
- Documentation

<http://spark.apache.org>



The screenshot shows the Apache Spark website in a web browser. The browser's address bar displays `spark.incubator.apache.org`. The website features the Spark logo (a stylized orange star) and the tagline "Lightning-Fast Cluster Computing". A navigation bar includes links for Home, Downloads, Documentation, Examples, Mailing Lists, Research, and FAQ. The main content area is titled "What is Apache Spark?" and describes it as an open source cluster computing system. It highlights its speed and ease of use, comparing it to Hadoop MapReduce. A sidebar on the right lists "LATEST NEWS" with links to announcements about the first Spark Summit, the release of Spark 0.8.0, a user survey, and a new screencast. Below the news, there is a "Word Count implemented in Spark" section with a bar chart comparing Hadoop and Spark performance. The chart shows Spark is significantly faster than Hadoop for word counting tasks. The bottom of the page mentions "Who uses it?" and lists various companies and organizations that use Spark.

Apache Spark - Lightning - X  
spark.incubator.apache.org

**Spark**  
Lightning-Fast Cluster Computing

Sign up now!  
Dec 2-3, 2013  
San Francisco

Home Downloads Documentation Examples Mailing Lists Research FAQ

### What is Apache Spark?

Apache Spark is an open source cluster computing system that aims to make data analytics *fast* — both fast to run and fast to write.

To run programs faster, Spark offers a general execution model that can optimize arbitrary operator graphs, and supports in-memory computing, which lets it query data faster than disk-based engines like Hadoop.

To make programming faster, Spark provides clean, concise APIs in [Scala](#), [Java](#) and [Python](#). You can also use Spark interactively from the Scala and Python shells to rapidly query big datasets.

### What can it do?

Spark was initially developed for two applications where placing data in memory helps: *iterative* algorithms, which are common in machine learning, and *interactive* data mining. In both cases, Spark can run up to **100x** faster than Hadoop MapReduce. However, you can use Spark for general data processing too. Check out our [example jobs](#).

Spark is also the engine behind [Shark](#), a fully [Apache Hive](#)-compatible data warehousing system that can run 100x faster than Hive.

While Spark is a new engine, it can access any data source supported by Hadoop, making it easy to run over existing data.

### Who uses it?

Spark was initially created in the [UC Berkeley AMPLab](#), but is now being used and developed at a wide array of companies. See our [powered by page](#) for a list of users, and our [list of committers](#). In total, over 25

#### LATEST NEWS

- [Announcing the first Spark Summit: December 2, 2013](#) (October 08, 2013)
- [Spark 0.8.0 released](#) (September 25, 2013)
- [Spark user survey and "Powered By" page](#) (September 05, 2013)
- [Fourth Spark screencast released](#) (August 27, 2013)

[News Archive](#)

```
file = spark.textFile("hdfs://...")  
  
file.flatMap(line => line.split(" "))  
    .map(word => (word, 1))  
    .reduceByKey(_ + _)
```

#### Word Count implemented in Spark

■ Hadoop ■ Spark

Timing Time (s)

| Task | Hadoop (s) | Spark (s) |
|------|------------|-----------|
| 1    | ~1000      | ~100      |
| 2    | ~1500      | ~200      |
| 3    | ~2500      | ~300      |
| 4    | ~3500      | ~400      |



# Spark Programming Model



# Key Concept: RDD's

Write programs in terms of operations on distributed datasets

## Resilient Distributed Datasets

- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

## Operations

- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)



# RDD

- RDD (Resilient Distributed Dataset)
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – stored in memory across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- RDDs are the fundamental unit of data in Spark
- Most Spark programming consists of performing operations on RDDs

# Example: Log Mining

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

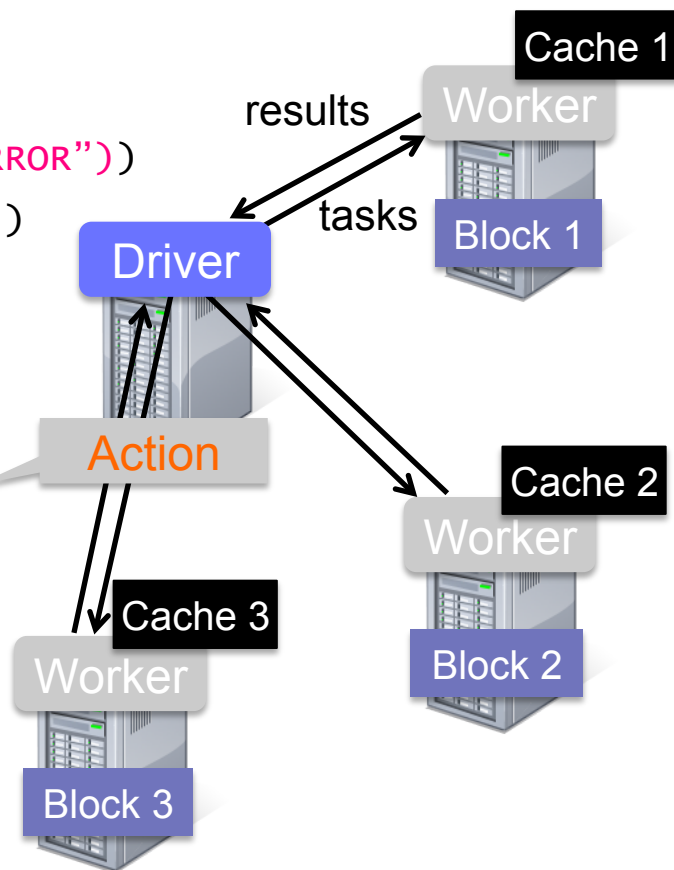
Transformed RDD

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

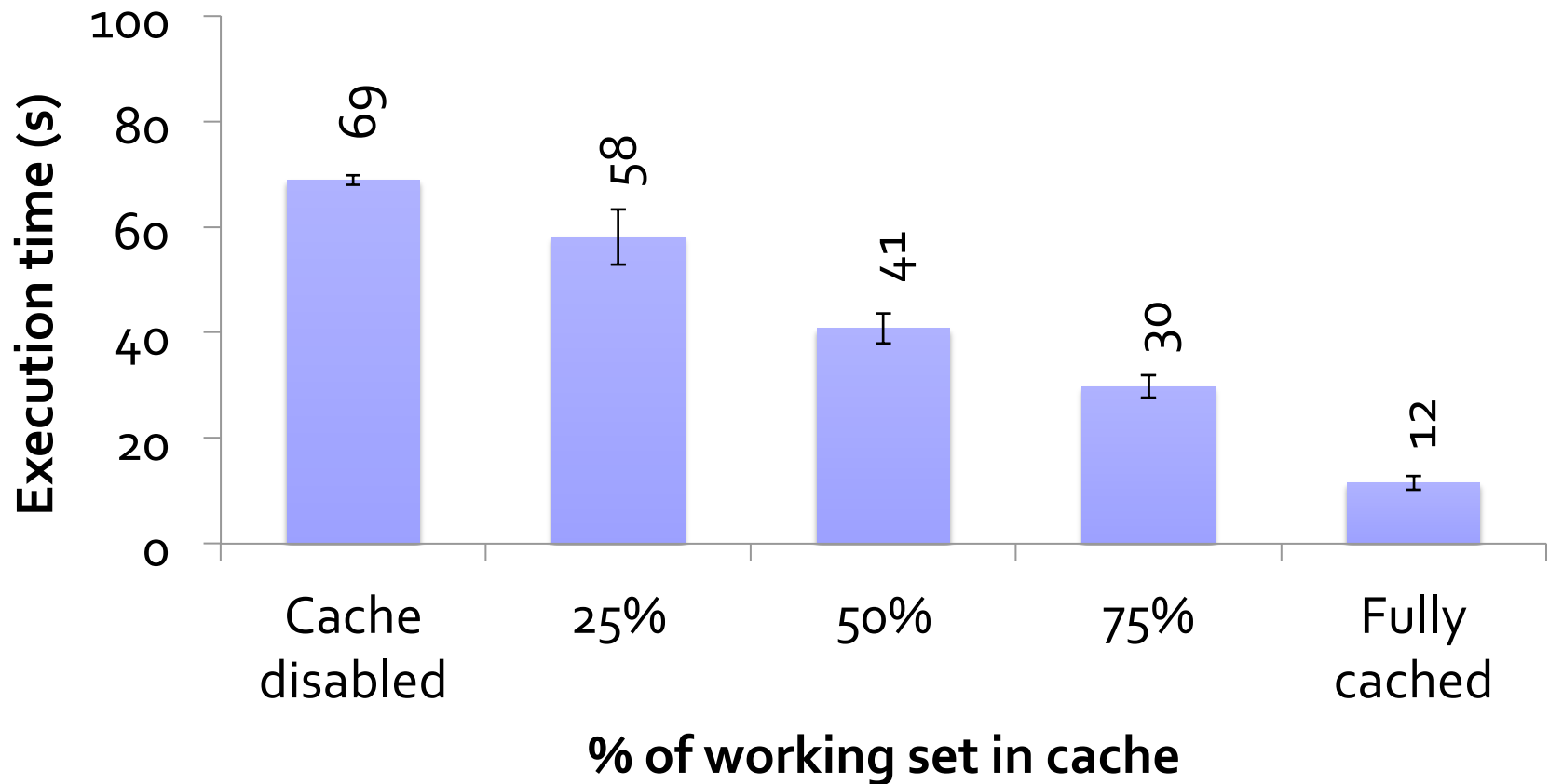
```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
. . .
```

## Full-text search of Wikipedia

- 60GB on 20 EC2 machine
- 0.5 sec vs. 20s for on-disk



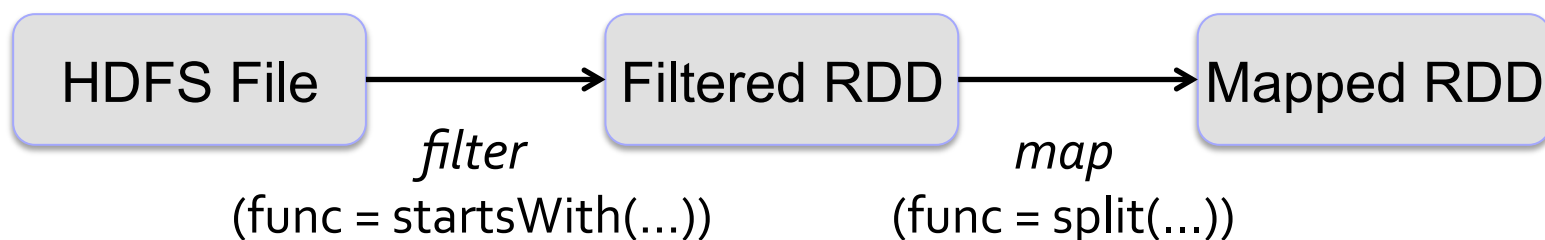
# Scaling Down



# Fault Recovery

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

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





# Programming with RDD's



# 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

- Three ways to create an RDD
  - From a file or set of files
  - From data in memory
  - From another RDD



# Creating RDDs

# Turn a Python collection into an RDD

- > `sc.parallelize([1, 2, 3])`
- > The elements of the collection are copied to form a distributed dataset that can be operated on in parallel.

# 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)`



# RDD Operations

- Two types of RDD operations
  - Actions: return values
  - Transformations: define a new RDD based on the current one

# 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")
```



# Transformations

- Transformations create a new RDD from an existing one
- RDDs are immutable
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
- Some common transformations
  - *map(func)* – creates a new RDD by performing a function on each record in the base RDD
  - *filter(function)* – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

# Basic Transformations

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

```
# Pass each element through a function
```

```
> squares = nums.map(lambda x: x*x) // {1, 4, 9}
```

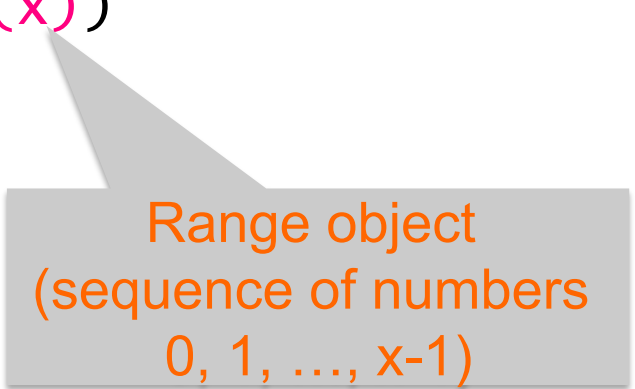
```
# Keep elements passing a predicate
```

```
> 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)

# Working with Key-Value Pairs

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

**Python:**

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

**Scala:**

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

**Java:**

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

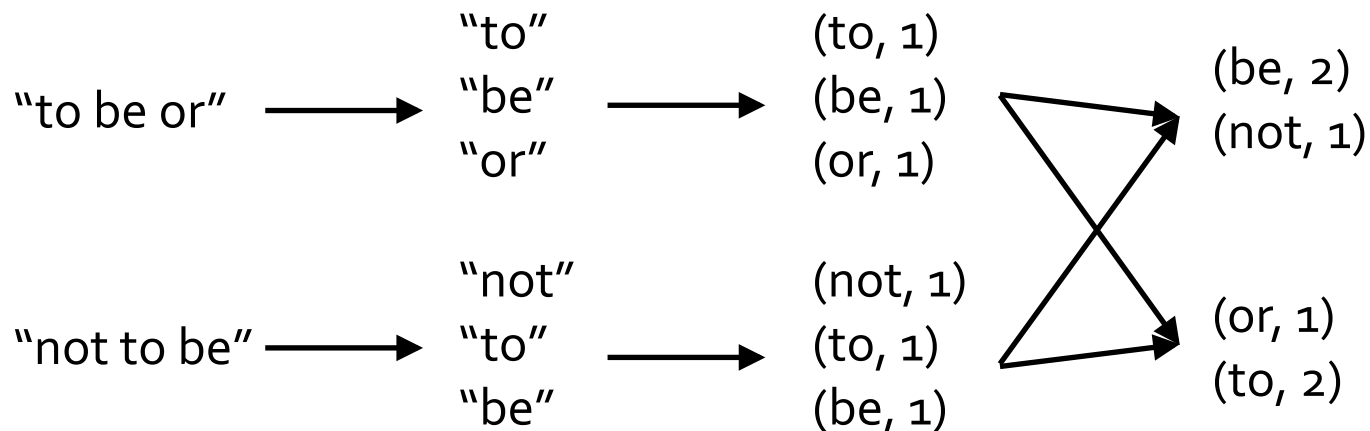
# Some Key-Value Operations

```
> 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:

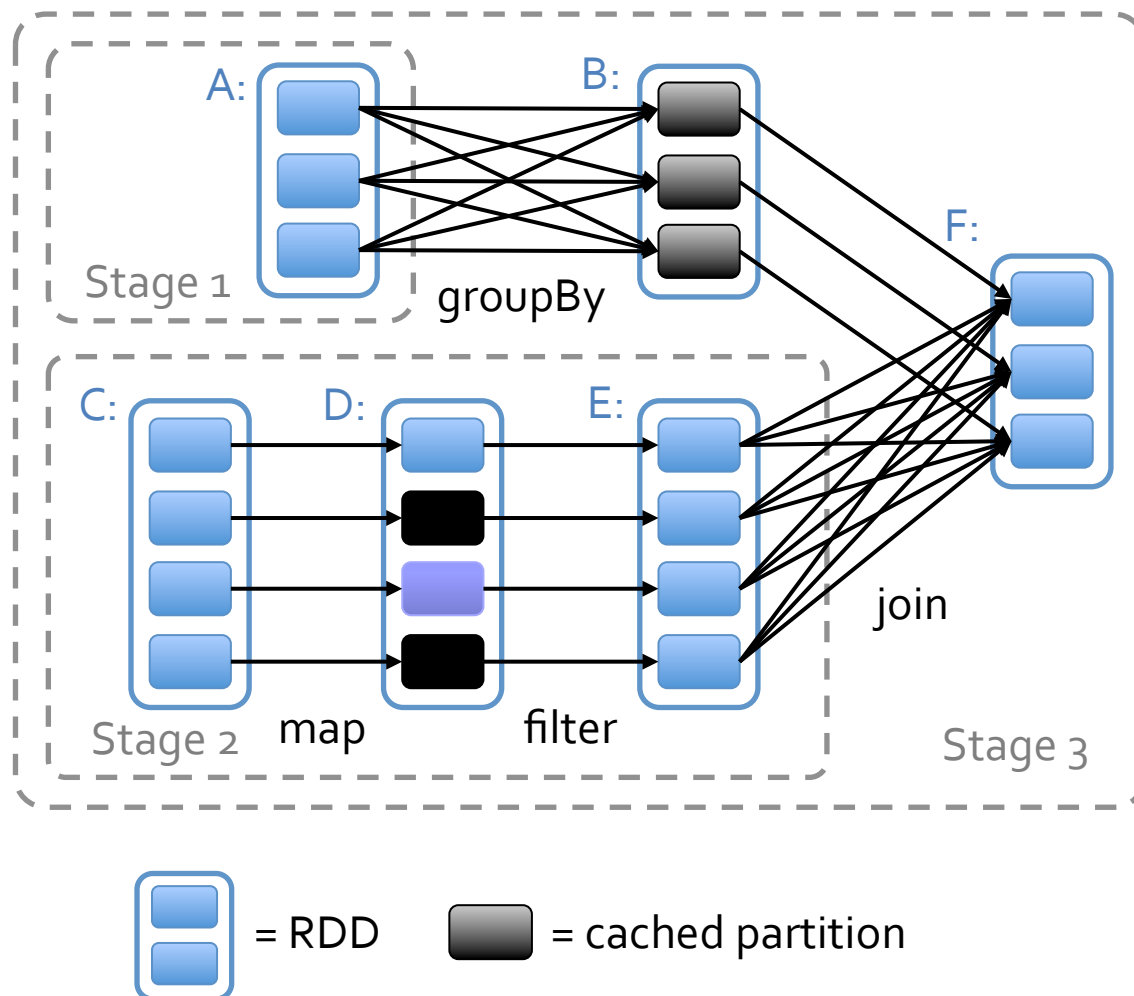
```
> query = sys.stdin.readline()
> pages.filter(lambda x: query in x).count()
```

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!)

# Under The Hood: DAG Scheduler

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



# More 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         |             |



# How to Run Spark

# Language Support

## Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

## 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();
```

## Standalone Programs

- Python, Scala, & Java

## Interactive Shells

- Python & Scala

## Performance

- Java & Scala are faster due to static typing
- ...but Python is often fine

- The Fastest Way to Learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark Cluster...
- OR Can run locally

```
cloudera-5-testing — root@ip-172-31-11-254:~ — ssh — 85x22  
root@ip-172-31-11-254:~  
[root@ip-172-31-11-254 ~]# /opt/cloudera/parcels/SPARK/pyspark  
...  
Welcome to  
  
      _/_/_/_/_/  
     /_/_/_/_/_/\ version 0.8.0  
    /_/_/_/_/_/\  
   /_/_/_/_/_/\
```

Using Python version 2.6.6 (r266:84292, Sep 11 2012 08:34:23)  
Spark context available as sc.

```
>>> file = sc.textFile("hdfs://ip-172-31-11-254.us-west-2.compute.internal:8020/user/hdfs/ec2-data/pageviews/2007/2007-12/pagecounts-20071209-180000.gz")  
>>> file.count()  
856769  
>>> file.filter(lambda line: "Holiday" in line).count()
```



# ... or a Standalone Application

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( "local", "WordCount",
                      sys.argv[0], None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(" ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```

# Create a SparkContext

Scala

```
import org.apache.spark.SparkContext
import org.apache.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

```
import org.apache.spark.api.java.JavaSparkContext;
```

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

Python

```
from pyspark import SparkContext
```

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



# Add Spark to Your Project

- Scala / Java: add a Maven dependency on

|                    |                   |
|--------------------|-------------------|
| <b>groupId:</b>    | org.spark-project |
| <b>artifactId:</b> | spark-core_2.10   |
| <b>version:</b>    | 0.9.0             |

- Python: run program with our pyspark script

# Administrative GUIs

<http://<Standalone Master>:8080> (by default)

**Spark Master at spark://mbp-2.local:7077**

URL: spark://mbp-2.local:7077  
Workers: 3  
Cores: 24 Total, 24 Used  
Memory: 45.0 GB Total, 1536.0 MB Used  
Applications: Running, 0 Completed

**Workers**

| Id  |
|---|
| worker-20131202231645-192.168.1.106-56789 |
| worker-20131202231657-192.168.1.106-56801 |
| worker-20131202231705-192.168.1.106-56806 |

**Running Applications**

| ID                      | Name        |
|-------------------------|-------------|
| app-20131202231712-0000 | Spark shell |

**Spark Stages**

Total Duration: 3.8 m  
Scheduling Mode: FIFO  
Active Stages: 0  
Completed Stages: 2  
Failed Stages: 0

**Active Stages (0)**

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Shuffle Read |
|----------|-------------|-----------|----------|------------------------|--------------|
|----------|-------------|-----------|----------|------------------------|--------------|

**Completed Stages (2)**

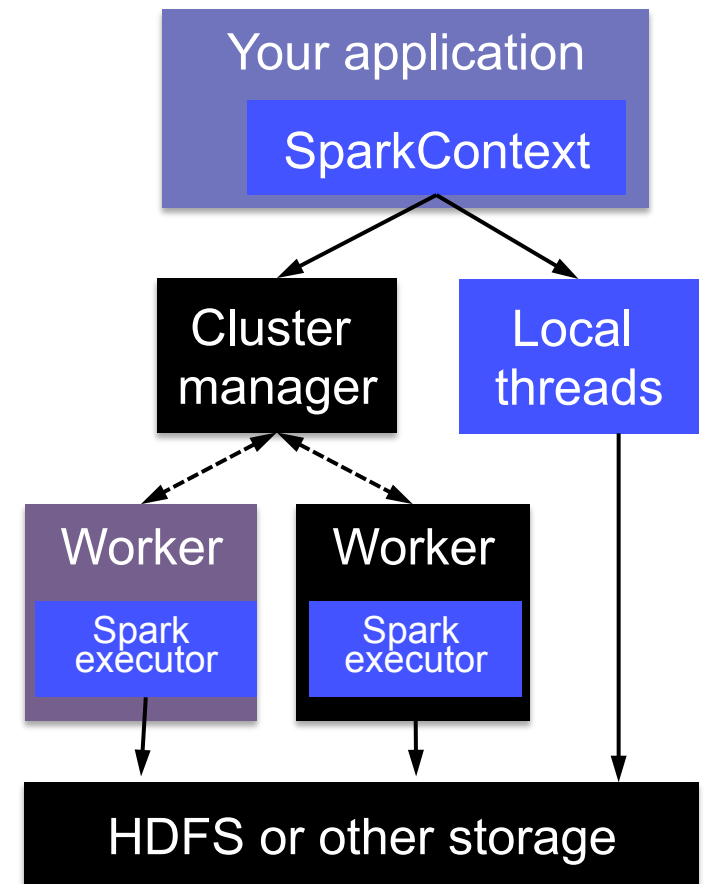
| Stage Id | Description                 | Submitted           | Duration | Tasks: Succeeded/Total | Shuffle |
|----------|-----------------------------|---------------------|----------|------------------------|---------|
| 0        | count at <console>:13       | 2013/12/02 21:07:55 | 83 ms    | 2/2                    | 754.0 B |
| 1        | reduceByKey at <console>:13 | 2013/12/02 21:07:55 | 345 ms   | 2/2                    |         |

**Failed Stages (0)**

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Shuffle Read |
|----------|-------------|-----------|----------|------------------------|--------------|
|----------|-------------|-----------|----------|------------------------|--------------|

# Software Components

- Spark runs as a library in your program (1 instance per app)
  - Mesos, YARN or standalone mode
- Runs tasks locally or on cluster
  - Can use HBase, HDFS, S3, ...





# 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](http://tinyurl.com/spark-emr)



# Example Application: PageRank





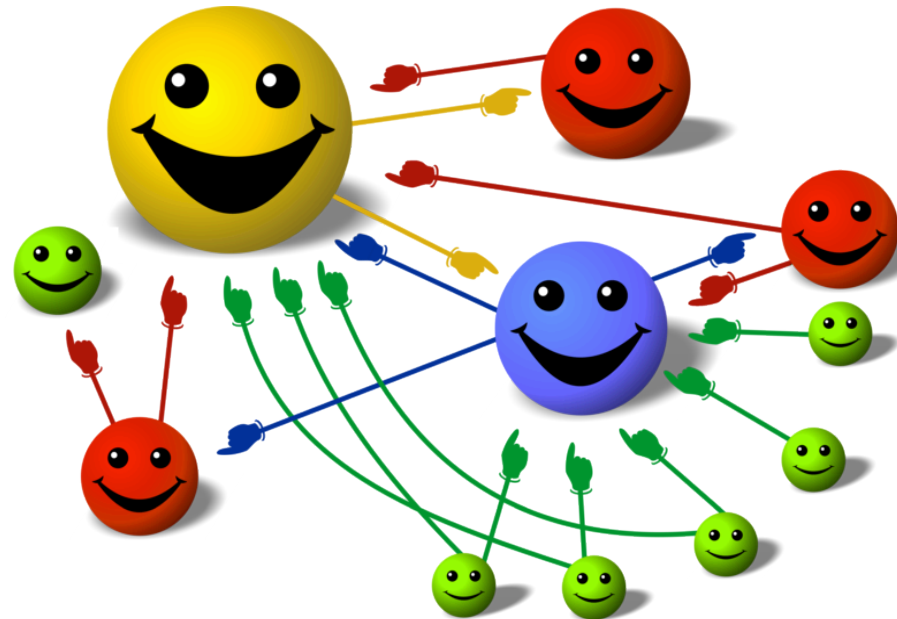
# Example: PageRank

- 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

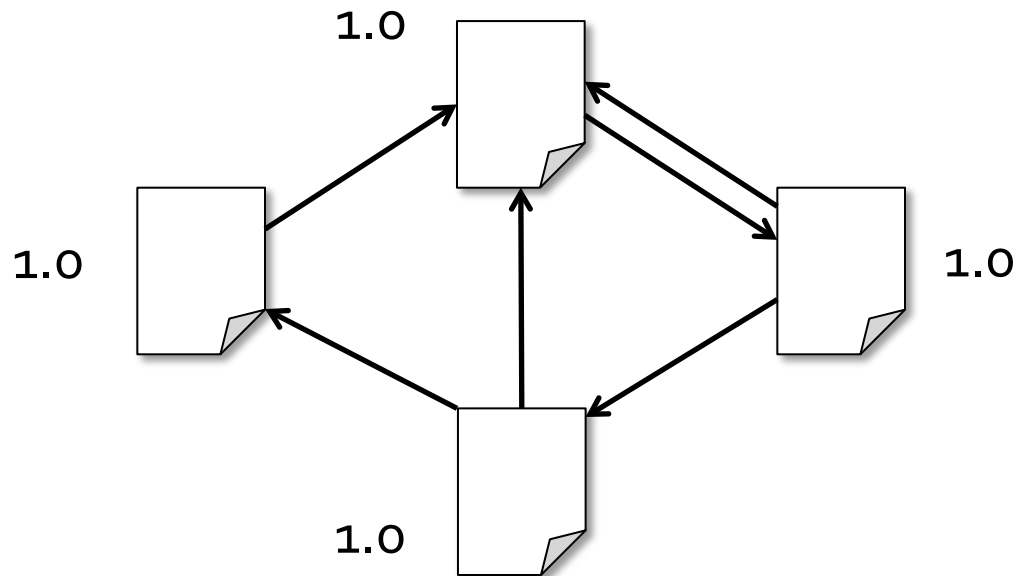
Give pages ranks (scores) based on links to them

- Links from many pages → high rank
- Link from a high-rank page → high rank



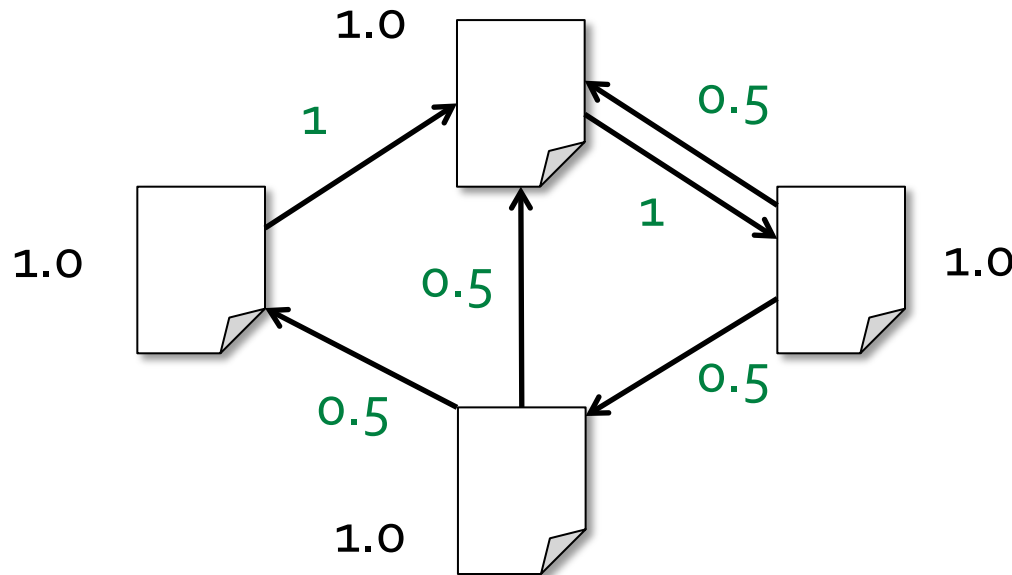
# Algorithm

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  $0.15 + 0.85 \times \text{contribs}$



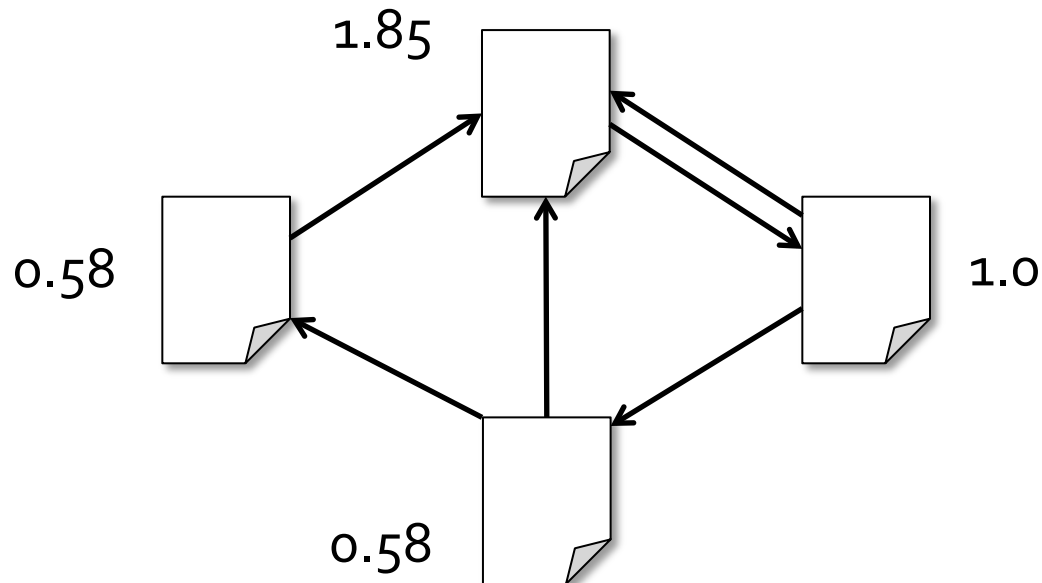
# Algorithm

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  $0.15 + 0.85 \times \text{contribs}$



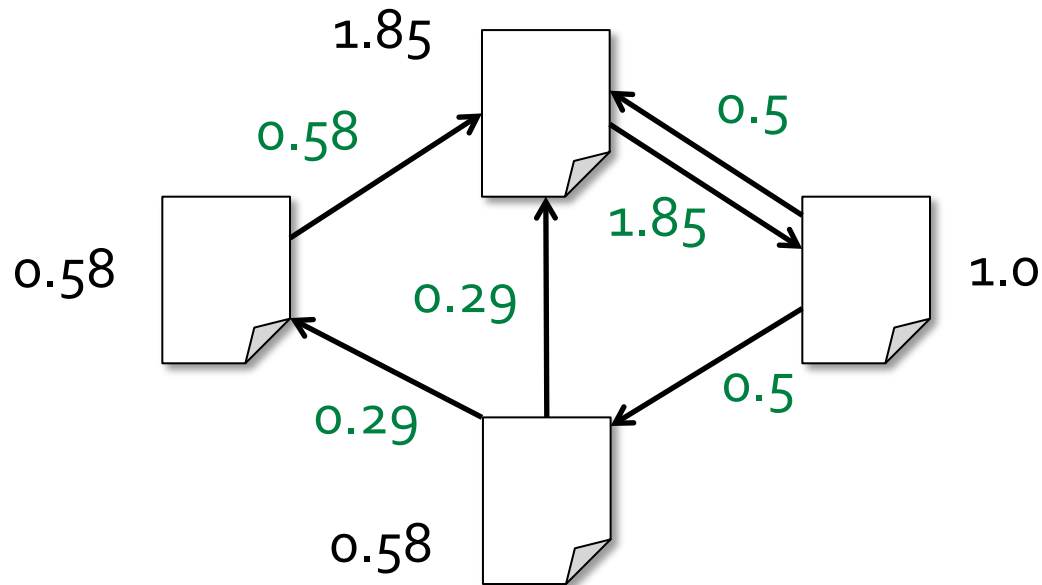
# Algorithm

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



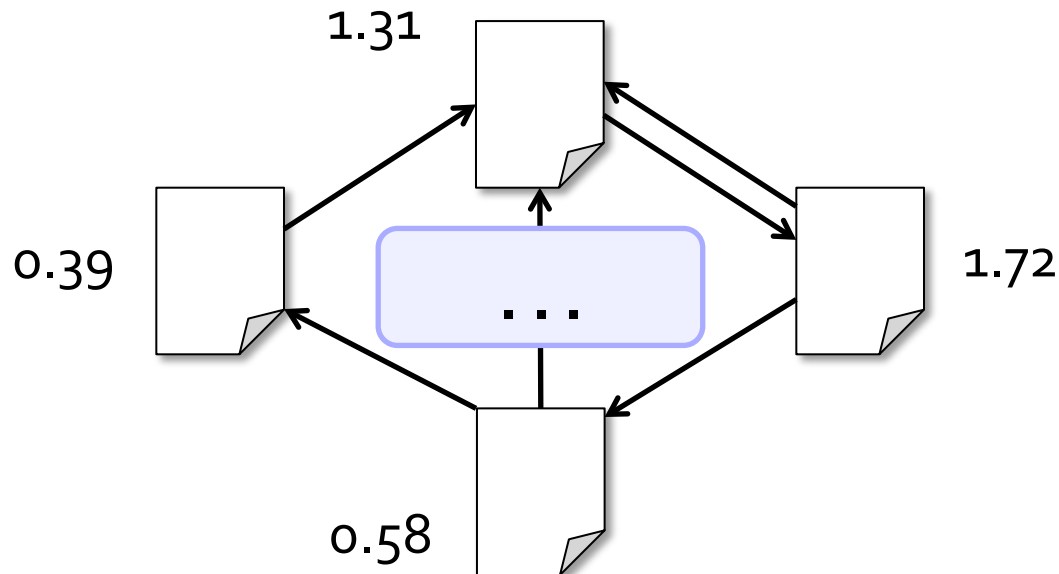
# Algorithm

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

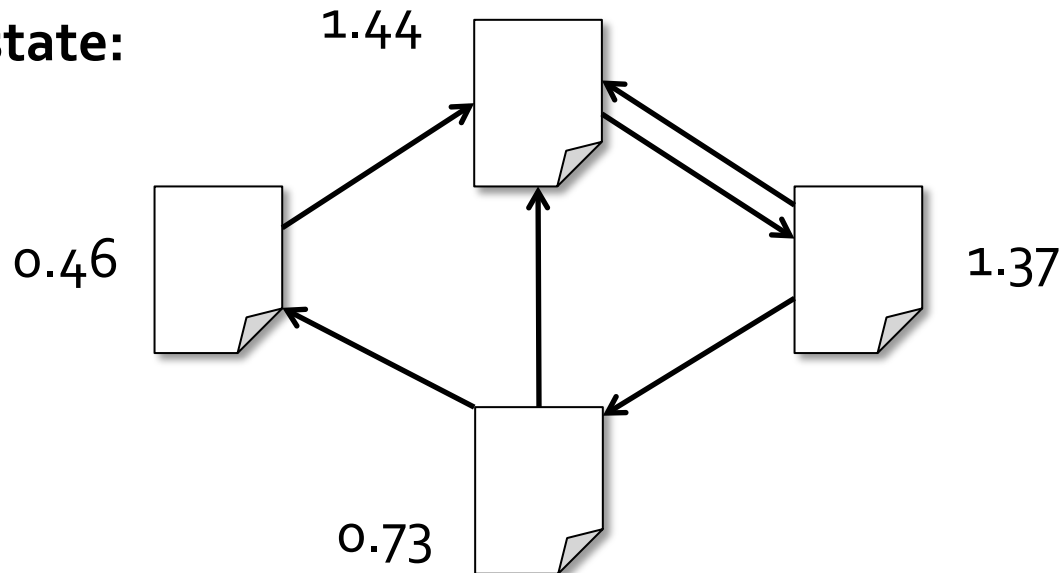
1. Start each page at a rank of 1
2. On each iteration, have page  $p$  contribute  $\text{rank}_p / |\text{neighbors}_p|$  to its neighbors
3. Set each page's rank to  $0.15 + 0.85 \times \text{contribs}$



# 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  $0.15 + 0.85 \times \text{contribs}$

Final state:



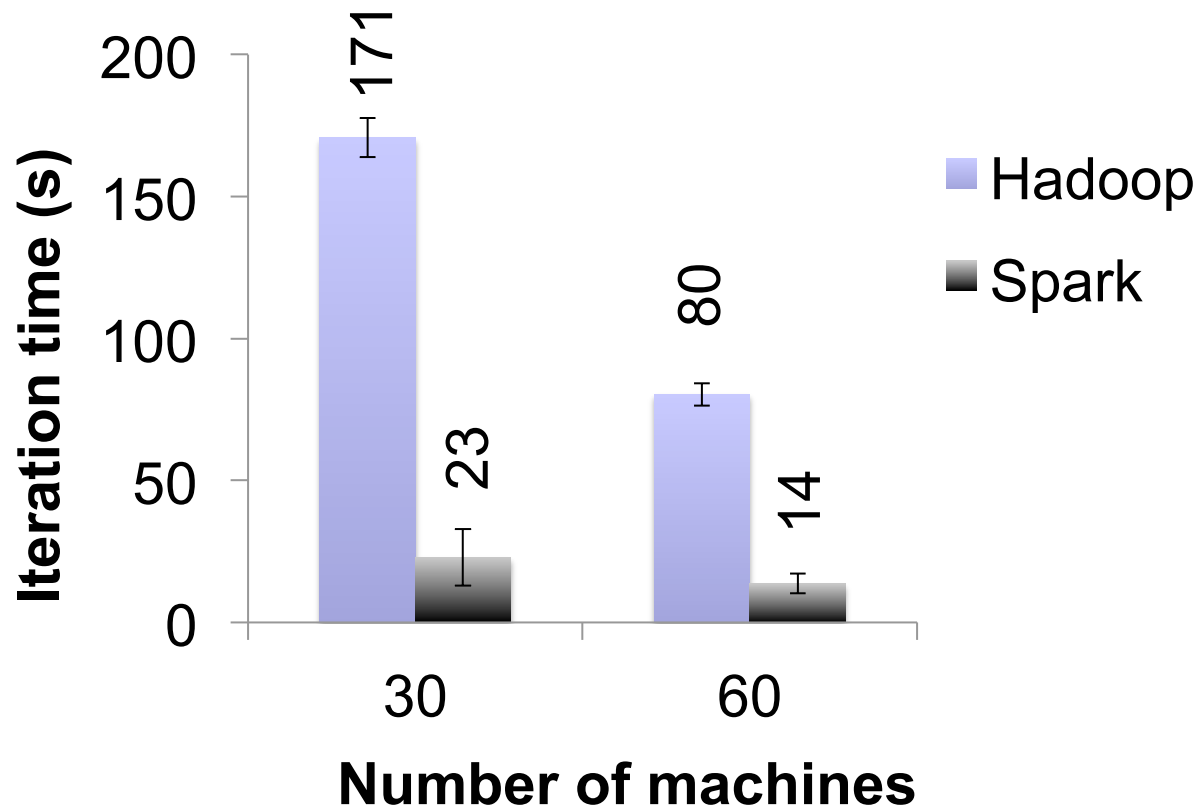


# Scala Implementation

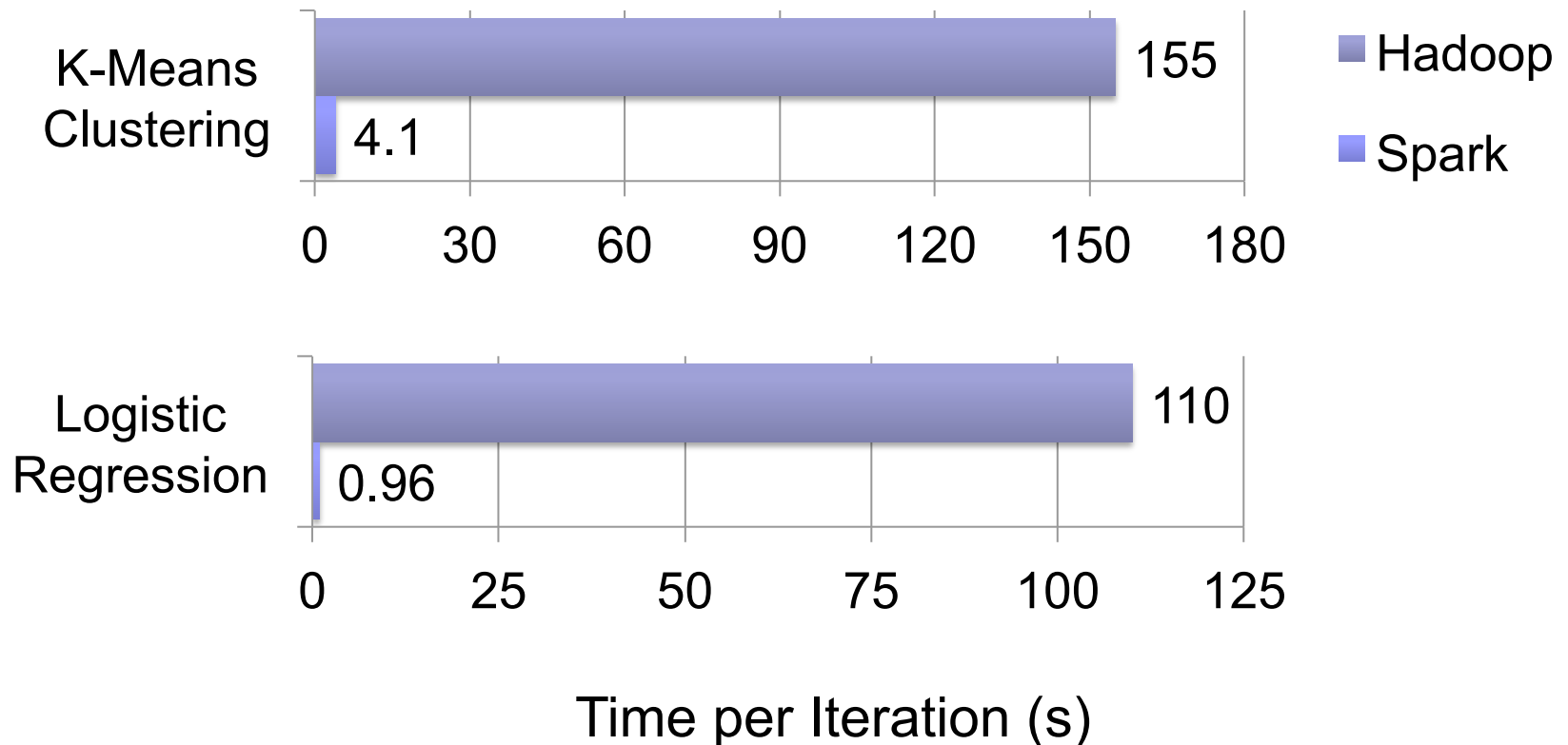
```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load RDD of (url, rank) pairs

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



A decorative graphic on the left side of the slide. It consists of a vertical stack of squares in various shades of blue, from light to dark. To the right of these squares is a solid dark blue horizontal bar that spans the width of the slide.

# Summary



# Summary

- 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 25+ companies contributing



END