GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 21: DISTRIBUTED SYSTEMS - SPARK

Using Apache Spark

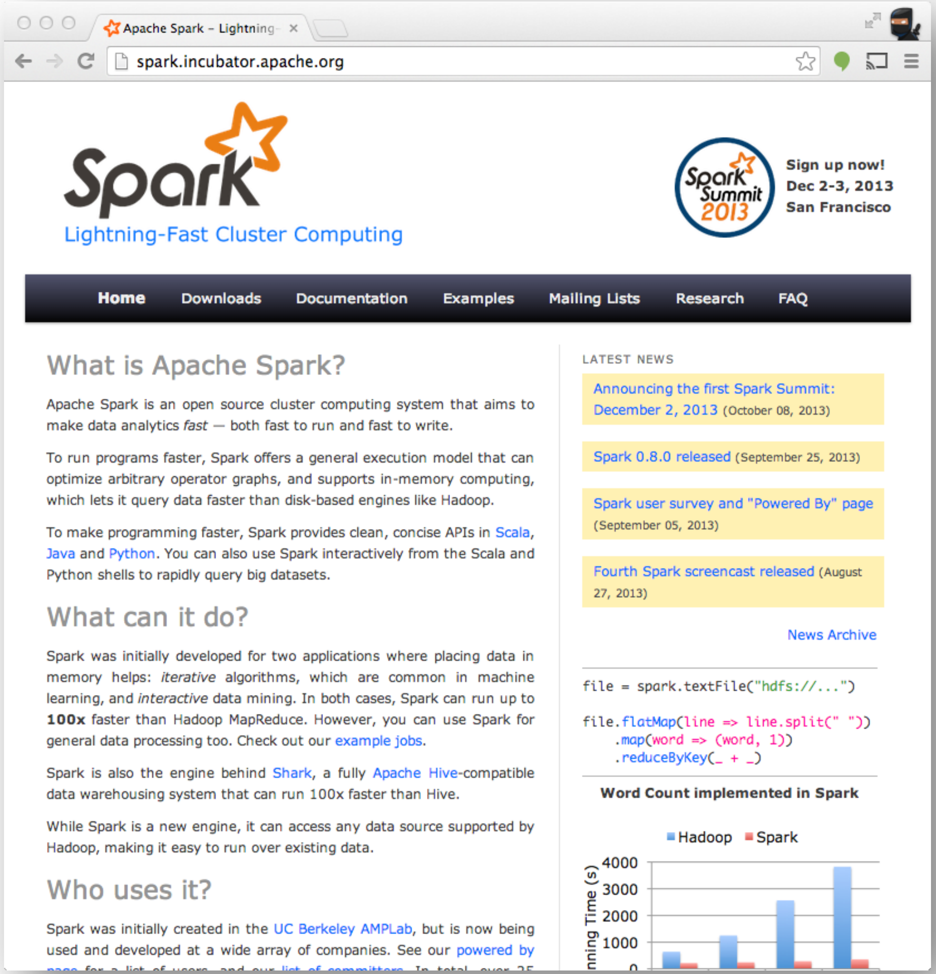
Pat McDonough - Databricks

Apache Spark

spark.apache.org

github.com/apache/spark

user@spark.apache.org



The Spark Community

amplab UC BERKELEY
databricks™
YAHOO!
amazon web services™
wanDISCO
intel
cloudera
阿里巴巴 Alibaba.com
Hortonworks
CLOUD PHYSICS
guavus Go Decisively
CONVIVA
Adobe
AdMobius
ClearStory DATA Now You See It™
IBM
TAGGED
43
stratio
bizo
webtrends
tuplejump
MAPR TECHNOLOGIES
ADATAO BIG DATA INTELLIGENCE
sharethrough
HHMI janelia farm research campus
OOYALA
quantiFind

INTRODUCTION TO APACHE SPARK

What is Spark?

Fast and Expressive Cluster Computing System
Compatible with Apache Hadoop

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

2-5× less code

Efficient

- General execution graphs
- In-memory storage

Usable

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

Key Concepts

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets

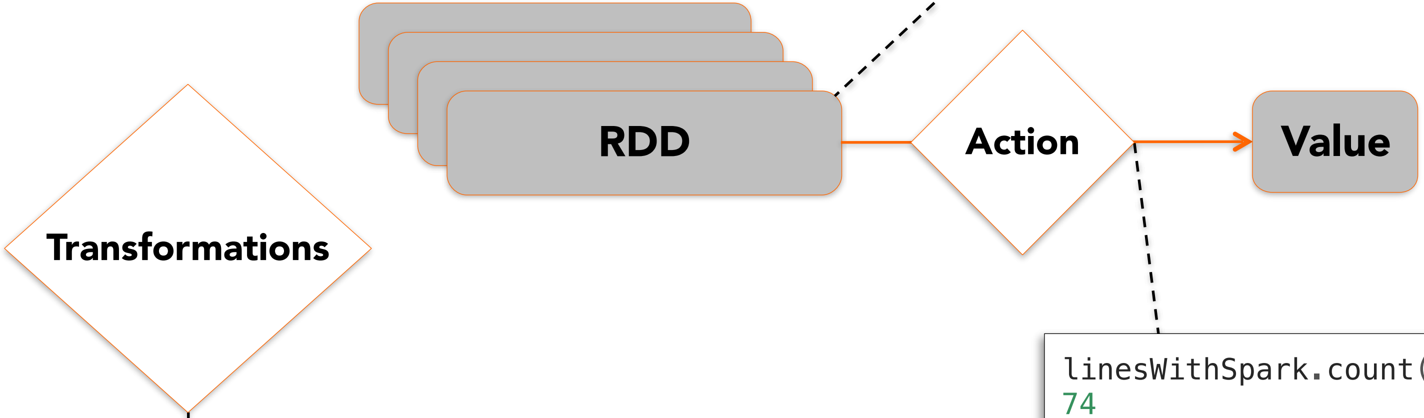
- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

Operations

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

Working With RDDs

```python
textFile = sc.textFile("SomeFile.txt")
```




```python
linesWithSpark.count()
74

linesWithSpark.first()
# Apache Spark
```

```python
linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

Example: Log Mining

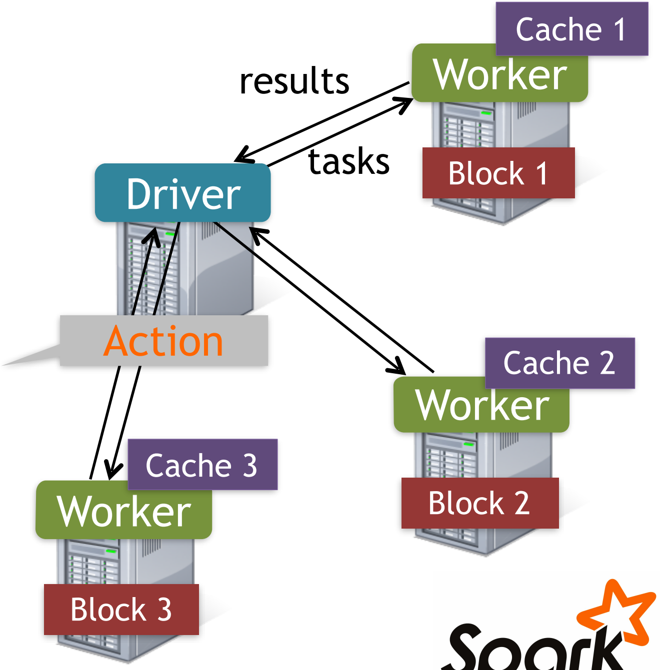
Load error messages from a log into memory, then interactively search for various patterns

B Transformed RDD

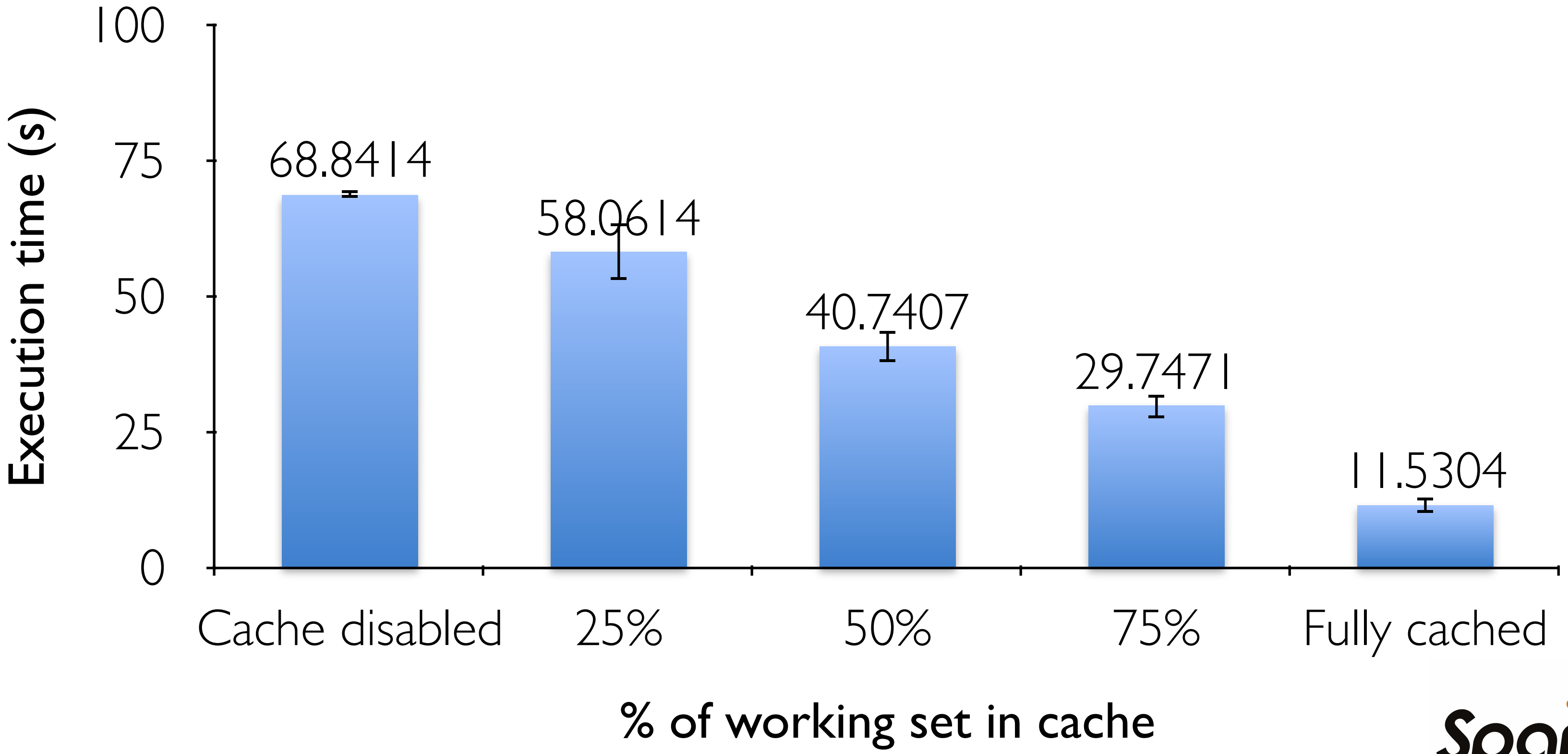
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


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



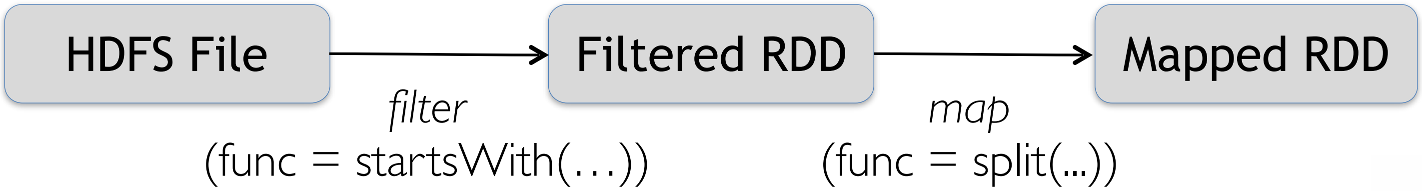
Scaling Down



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])
```



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

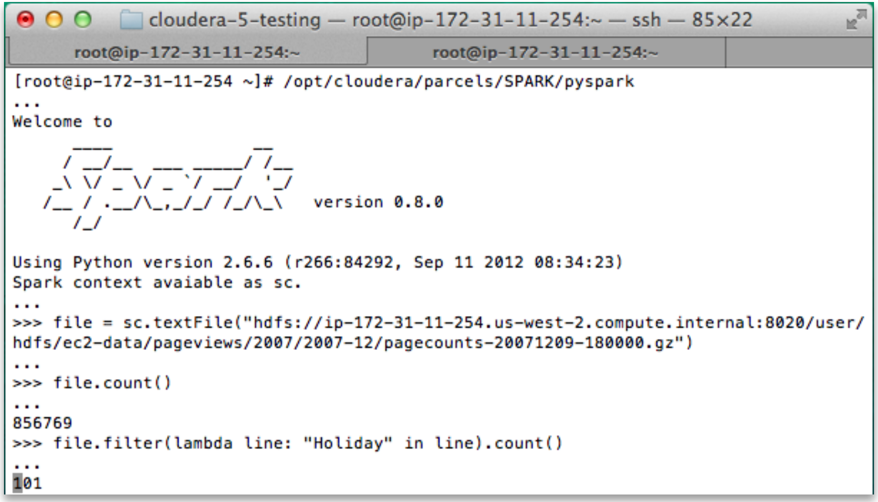
- Python & Scala

Performance

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

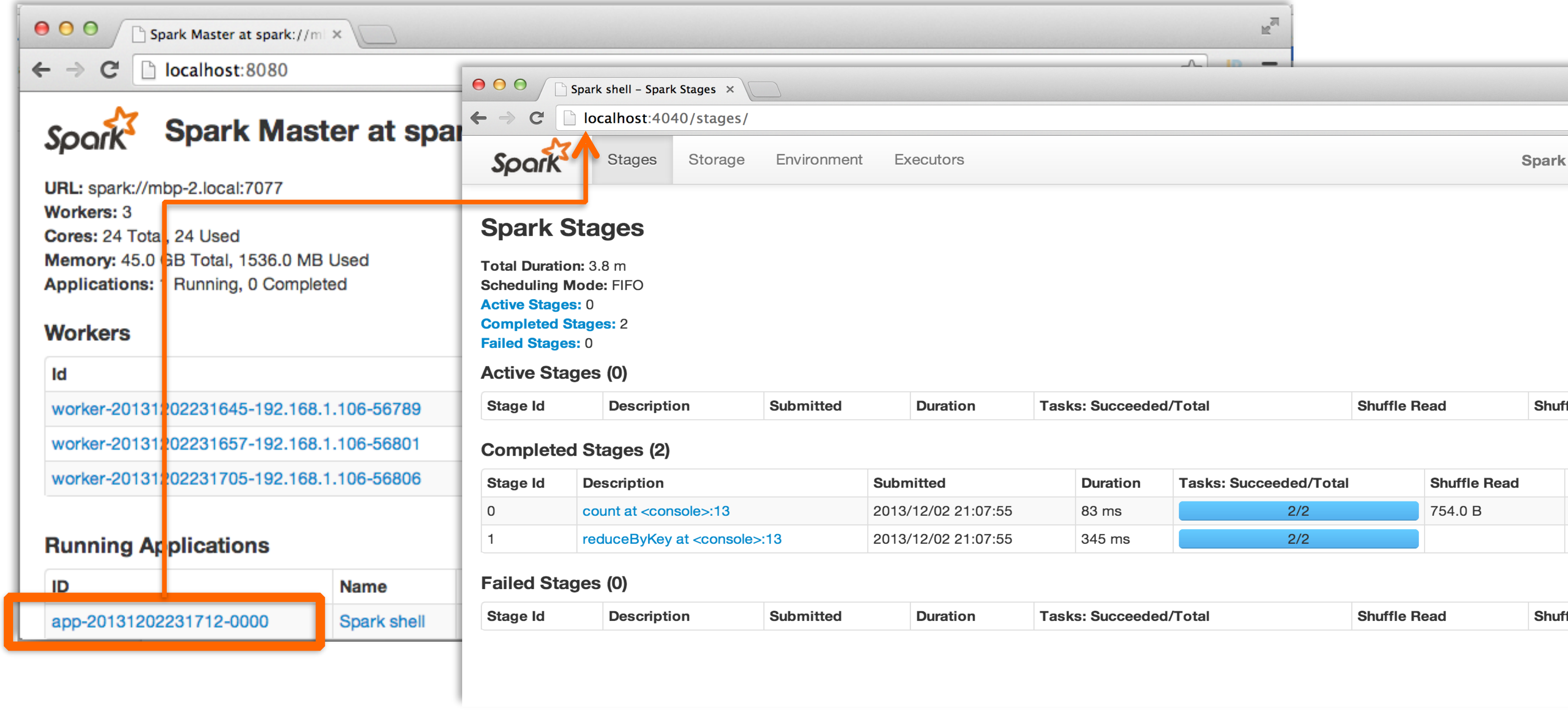
Interactive Shell

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



Administrative GUIs

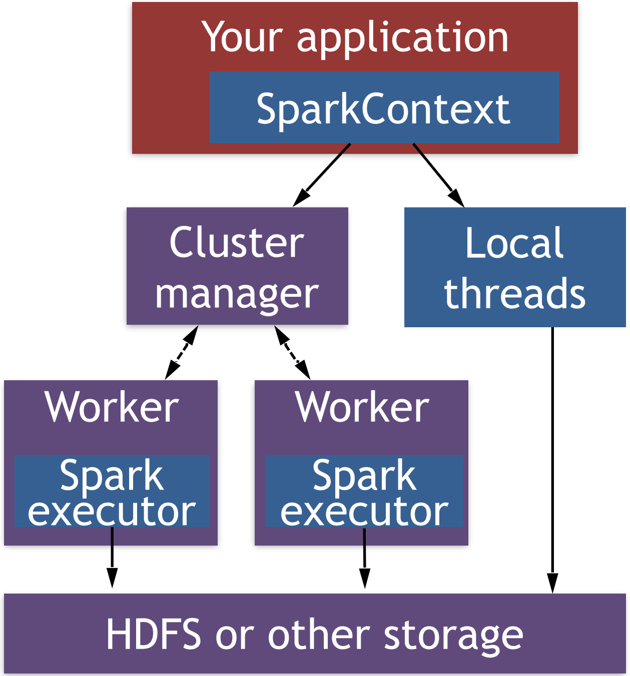
http://<Standalone Master>:8080 (by default)



JOB EXECUTION

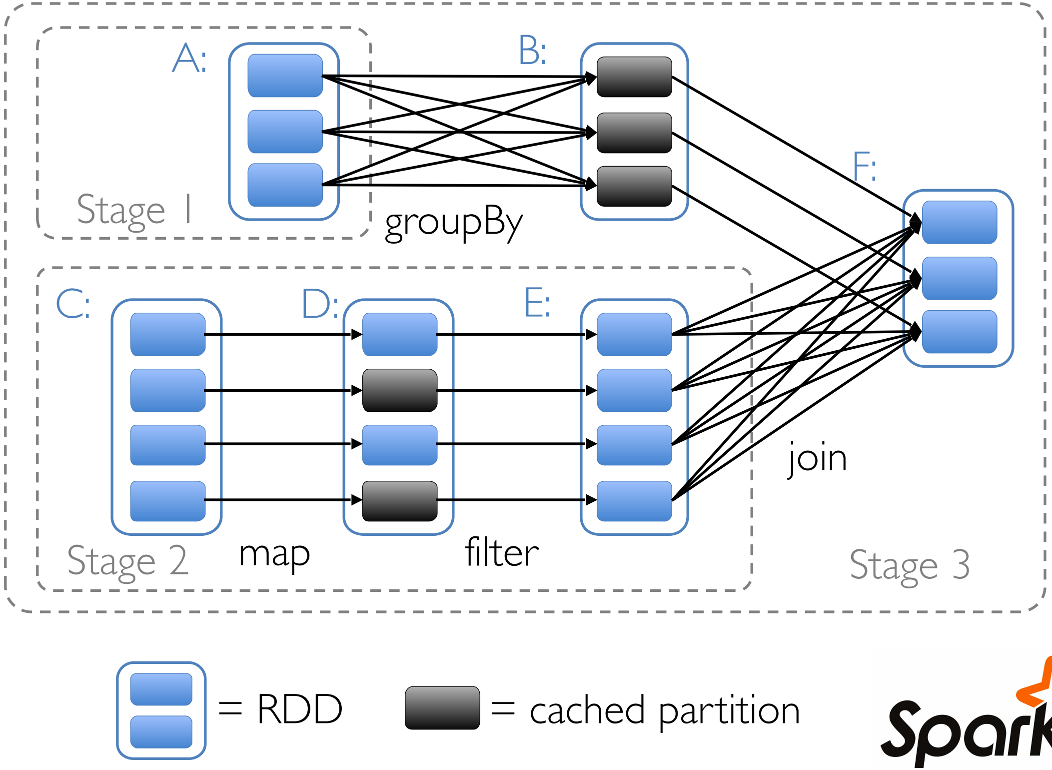
Software Components

- 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!

WORKING WITH SPARK

Using the Shell

Launching:

spark-shell

pyspark (IPYTHON=1)

```
cloudera-5-testing — root@ip-172-31-11-254:~ — ssh — 85×22
root@ip-172-31-11-254:~    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 avaiable 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()
...
101
```

Modes:

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

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
```

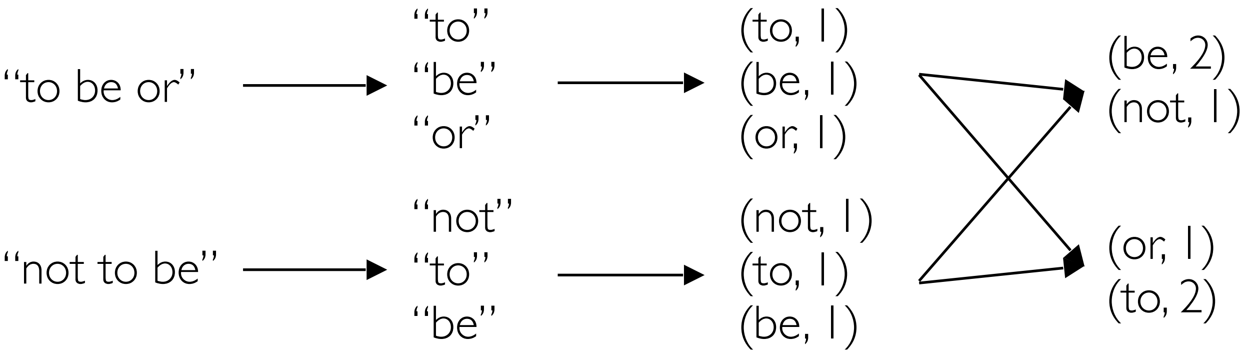
Some Key-Value Operations

```
> 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

This is a problem:

```
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

More RDD Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

sample

take

first

partitionBy

mapWith

pipe

save ...

CREATING SPARK APPLICATIONS

Add Spark to Your Project

- Scala / Java: add a Maven dependency on

groupId:	org.spark-project
artifactId:	spark-core_2.9.3
version:	0.8.0

- Python: run program with our pyspark script

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.a

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

Python

```
from pyspark import SparkContext

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

Complete Spark App in Python

```
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])
```

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

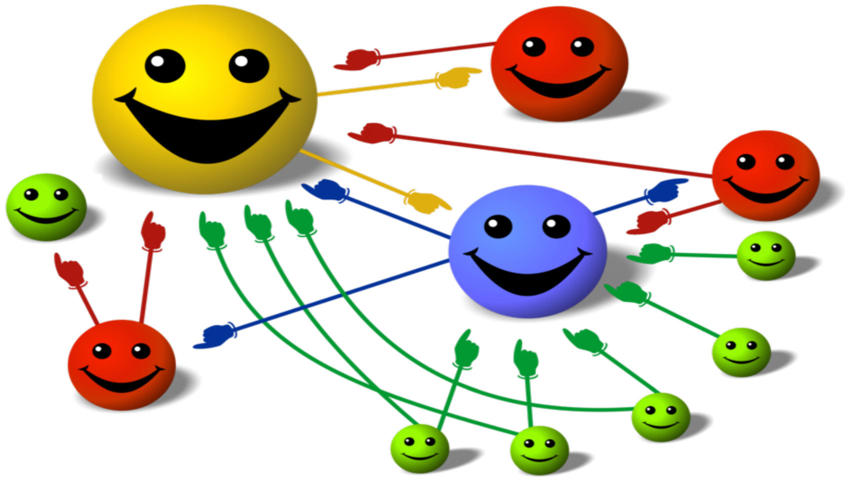
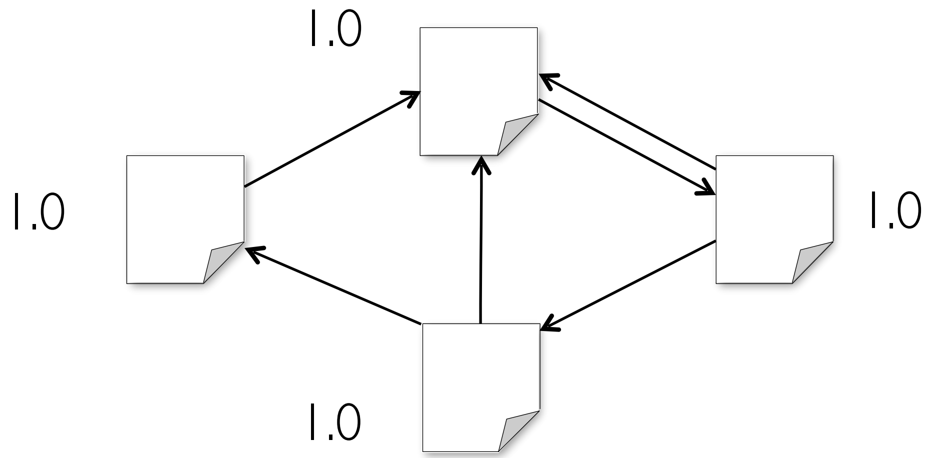


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

Spark

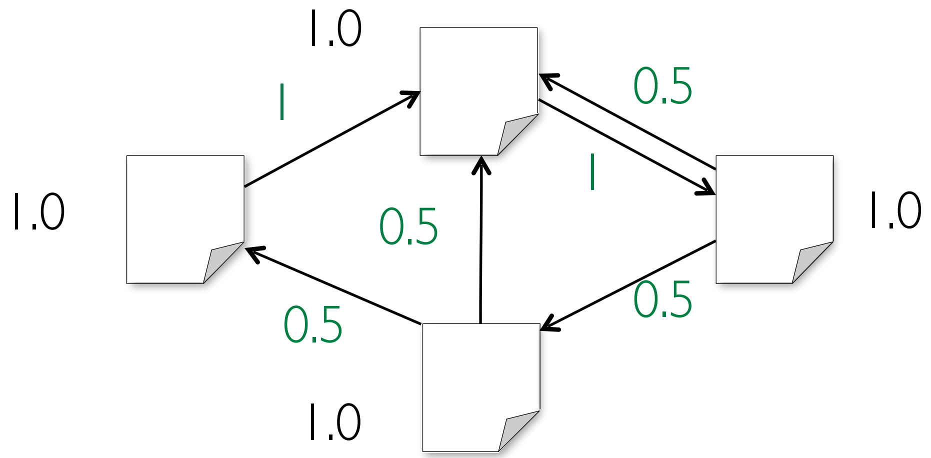
Algorithm

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 $0.15 + 0.85 \times contribs$



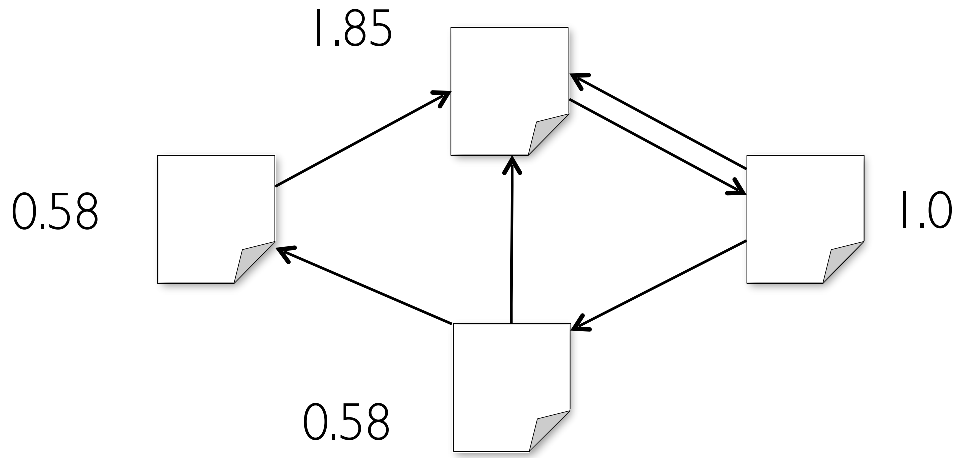
Algorithm

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 $0.15 + 0.85 \times contribs$



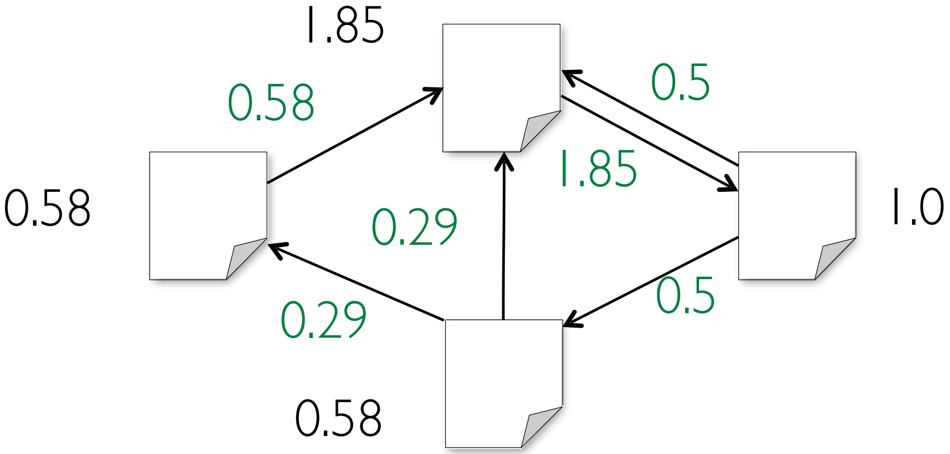
Algorithm

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 $0.15 + 0.85 \times contribs$



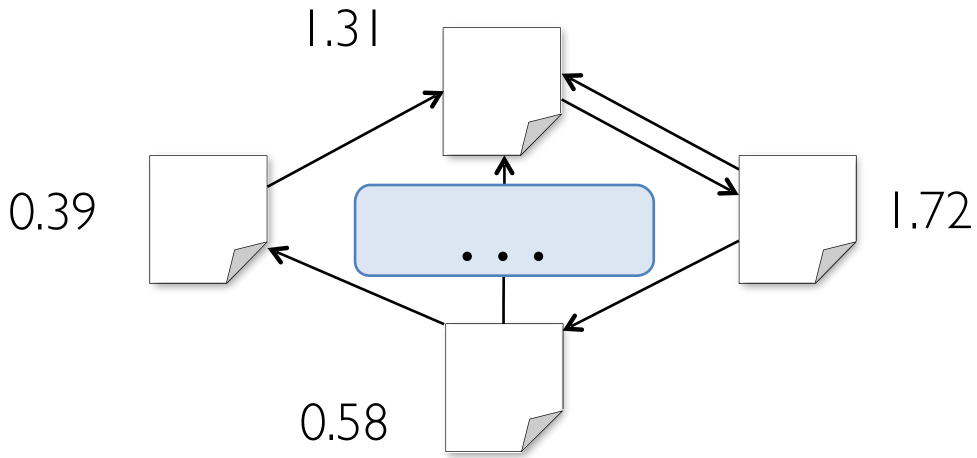
Algorithm

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

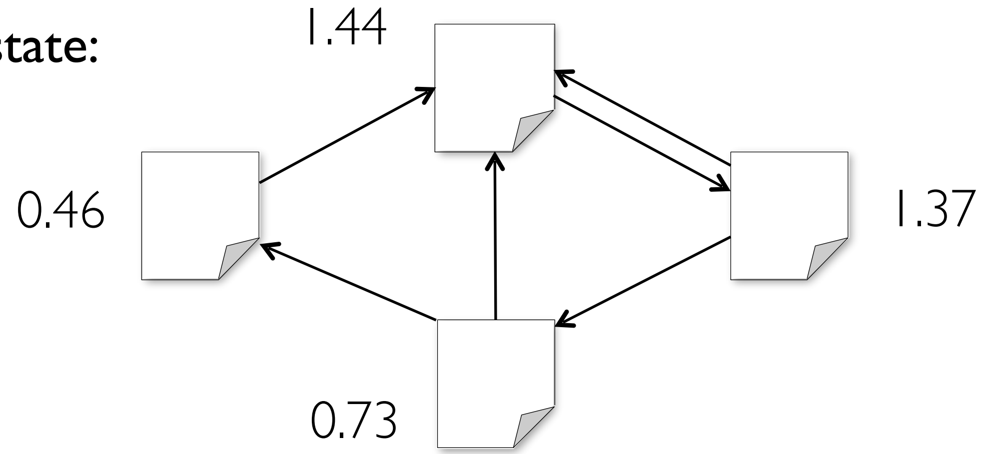
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times contribs$



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 $0.15 + 0.85 \times contribs$

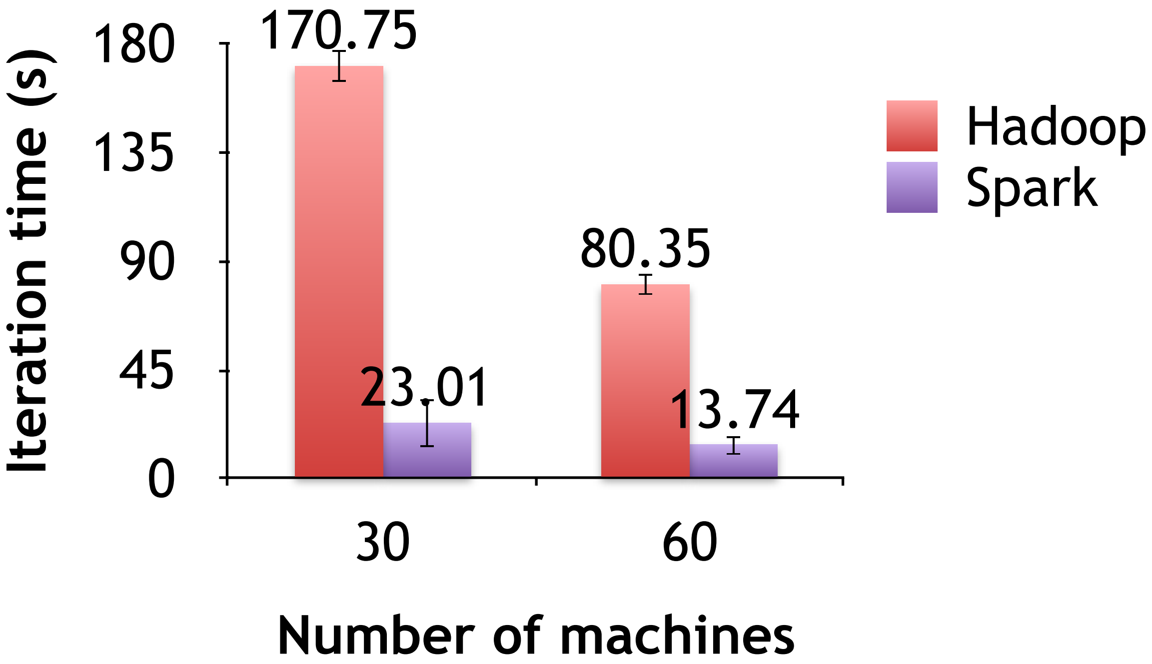
Final state:



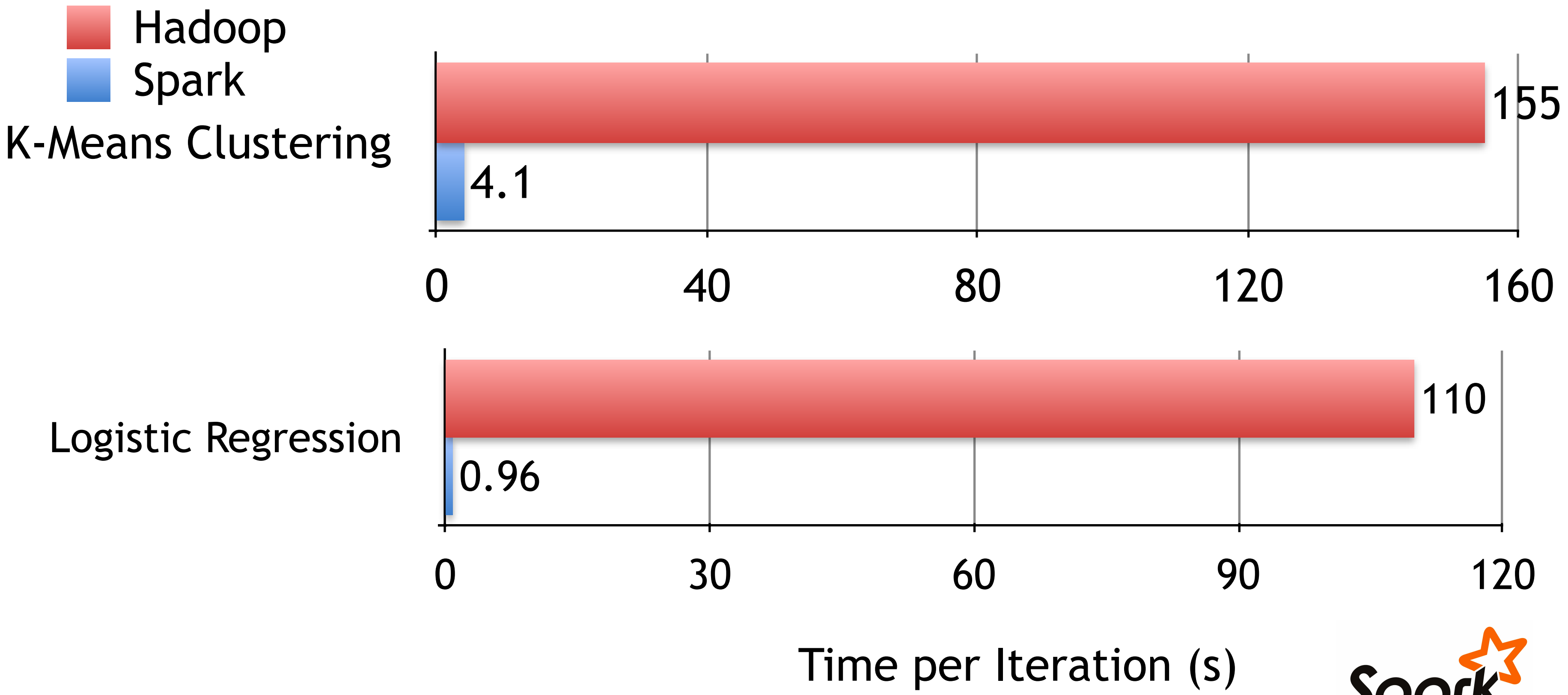
Scala Implementation

```
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



CONCLUSION

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 25+ companies contributing

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