IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Scalable Data Science

Lecture 18: Spark

Sourangshu Bhattacharya
Computer Science and Engineering
IIT KHARAGPUR

In the previous lectures:

- Outline:
  - Scala
    - Var and Val
    - Classes and objects
    - Functions and higher order functions
    - Lists

In this Lecture:

- Outline:
  - Spark
    - Motivation
    - RDD
    - Actions and transformations
    - Examples:
      - Matrix multiplication
      - Logistic regression
      - Pagerank

SPARK

Spark

Spark is an In-Memory Cluster Computing platform for Iterative and Interactive Applications.

http://spark.apache.org

Spark

- ❑ Started in AMPLab at UC Berkeley.
- ❑ Resilient Distributed Datasets.
- ❑ Data and/or Computation Intensive.
- ❑ Scalable – fault tolerant.
- ❑ Integrated with SCALA.
- ❑ Straggler handling.
- ❑ Data locality.
- ❑ Easy to use.

Background

- Commodity clusters have become an important computing platform for a variety of applications
  - **In industry:** search, machine translation, ad targeting, …
  - **In research:** bioinformatics, NLP, climate simulation, …

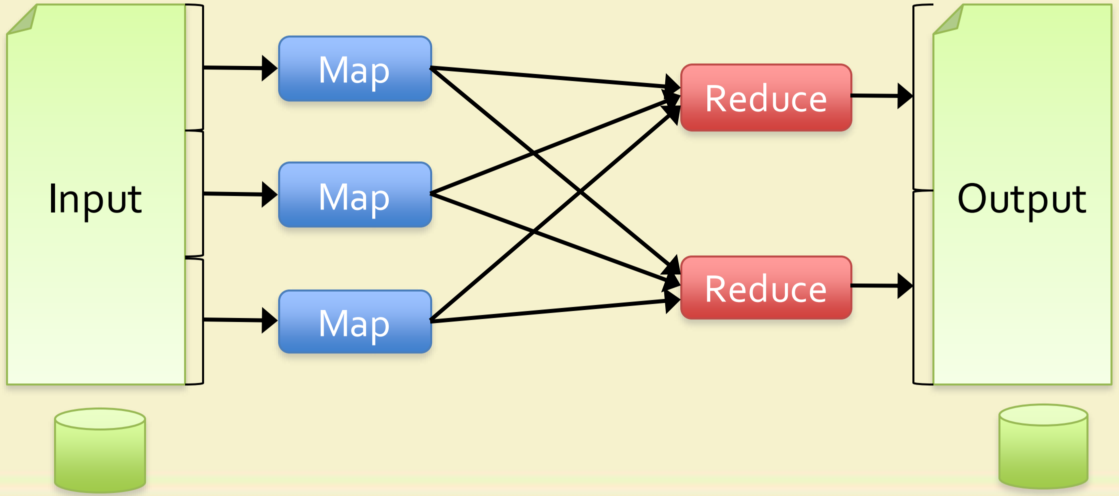
- High-level cluster programming models like MapReduce power many of these apps

- *Theme of this work: provide similarly powerful abstractions for a broader class of applications*

Motivation

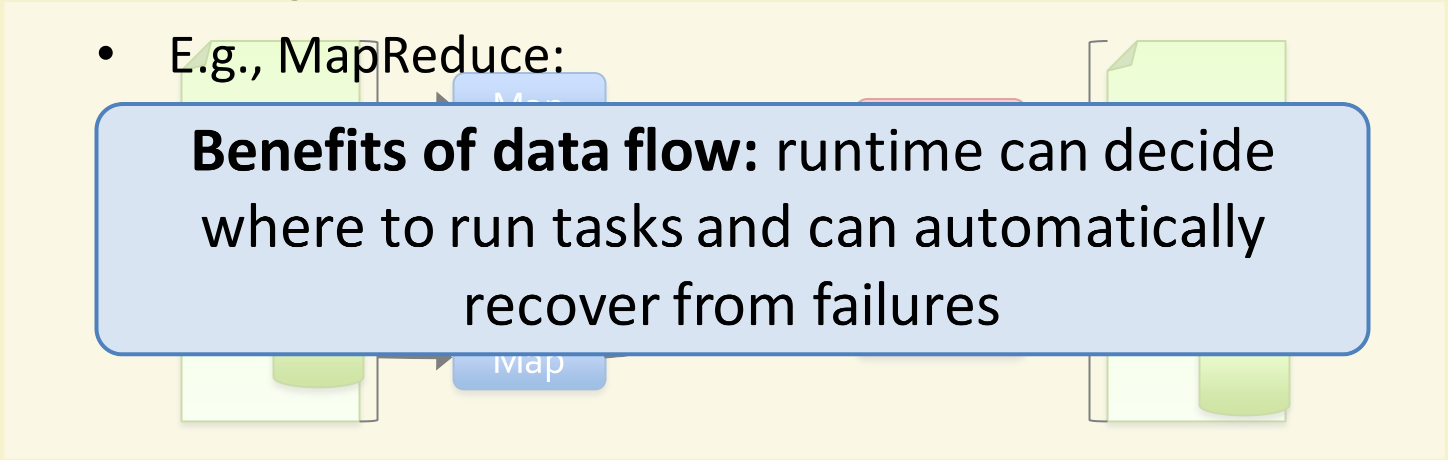
Current popular programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:



Motivation

- Current popular programming models for clusters transform data flowing from stable storage to stable storage
- E.g., MapReduce:



Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Motivation

- Acyclic data flow is a powerful abstraction, but is not efficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (many in machine learning)
  - **Interactive** data mining tools (R, Excel, Python)
- Spark makes working sets a first-class concept to efficiently support these apps

Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- Retain the attractive properties of MapReduce:
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

Solution: augment data flow model with "resilient distributed datasets" (RDDs)

Resilient Distributed Datasets

- Immutable distributed SCALA collections.
  - Array, List, Map, Set, etc.

- Transformations on RDDs create new RDDs.
  - Map, ReducebyKey, Filter, Join, etc.

- Actions on RDD return values.
  - Reduce, collect, count, take, etc.

- Seamlessly integrated into a SCALA program.
- RDDs are materialized when needed.
- RDDs are cached to disk – graceful degradation.
- Spark framework re-computes lost splits of RDDs.

RDDs in More Detail

- An RDD is an immutable, partitioned, logical collection of records
  - Need not be materialized, but rather contains information to rebuild a dataset from stable storage
- Partitioning can be based on a key in each record (using hash or range partitioning)
- Built using bulk transformations on other RDDs
- Can be cached for future reuse

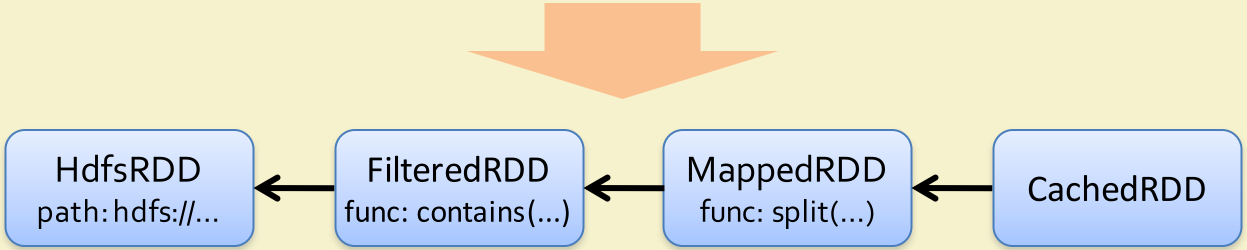
RDD Operations

Transformations (define a new RDD)	Actions (return a result to driver)
map	reduce
filter	collect
sample	count
union	save
groupByKey	lookupKey
reduceByKey	…
join	
cache	
…	

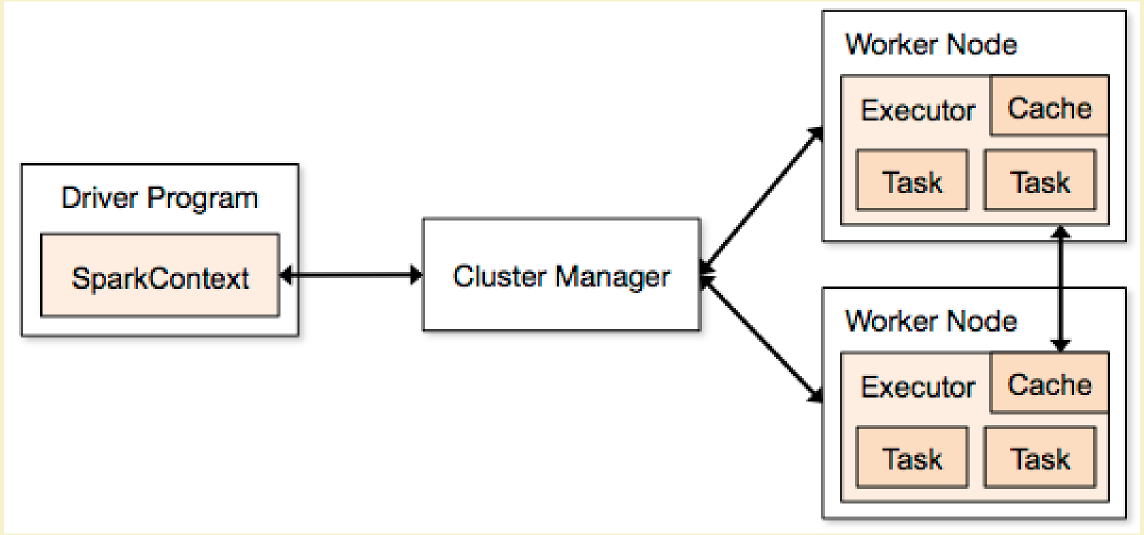
RDD Fault Tolerance

- RDDs maintain *lineage* information that can be used to reconstruct lost partitions
- Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```



Spark Architecture



Example: MapReduce

- MapReduce data flow can be expressed using RDD transformations

```
res = data.flatMap(rec => myMapFunc(rec))
          .groupByKey()
          .map((key, vals) => myReduceFunc(key, vals))
```

Or with combiners:

```
res = data.flatMap(rec => myMapFunc(rec))
          .reduceByKey(myCombiner)
          .map((key, val) => myReduceFunc(key, val))
```

Word Count in Spark

```
val lines = spark.textFile("hdfs://...")

val counts = lines.flatMap(_.split("\\s"))
                  .reduceByKey(_ + _)

counts.save("hdfs://...")
```

Example: Matrix Multiplication

Matrix Multiplication

- Representation of Matrix:
  - List <Row index, Col index, Value>
  - Size of matrices: First matrix (A): m*k, Second matrix (B): k*n
- Scheme:
  - For each input record: If input record
- Mapper key: <row_index_matrix_1, Column_index_matrix_2>
- Mapper value: < column_index_1 / row_index_2, value>
- GroupByKey: List(Mapper Values)
- Collect all (two) records with the same first field multiply them and add to the sum.

Example: Logistic Regression

Logistic Regression

- Binary Classification. *y ε {+1, -1}*
- Probability of classes given by linear model:

$$p(y \mid x, w) = \frac{1}{1 + e^{(-yw^T x)}}$$

- Regularized ML estimate of w given dataset (x_i, y_i) is obtained by minimizing:

$$l(w) = \sum_i \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2} w^T w$$

Logistic Regression

- Gradient of the objective is given by:

$$\nabla l(w) = \sum_i (1 - s(y_i w^T x_i)) y_i x_i - \lambda w$$

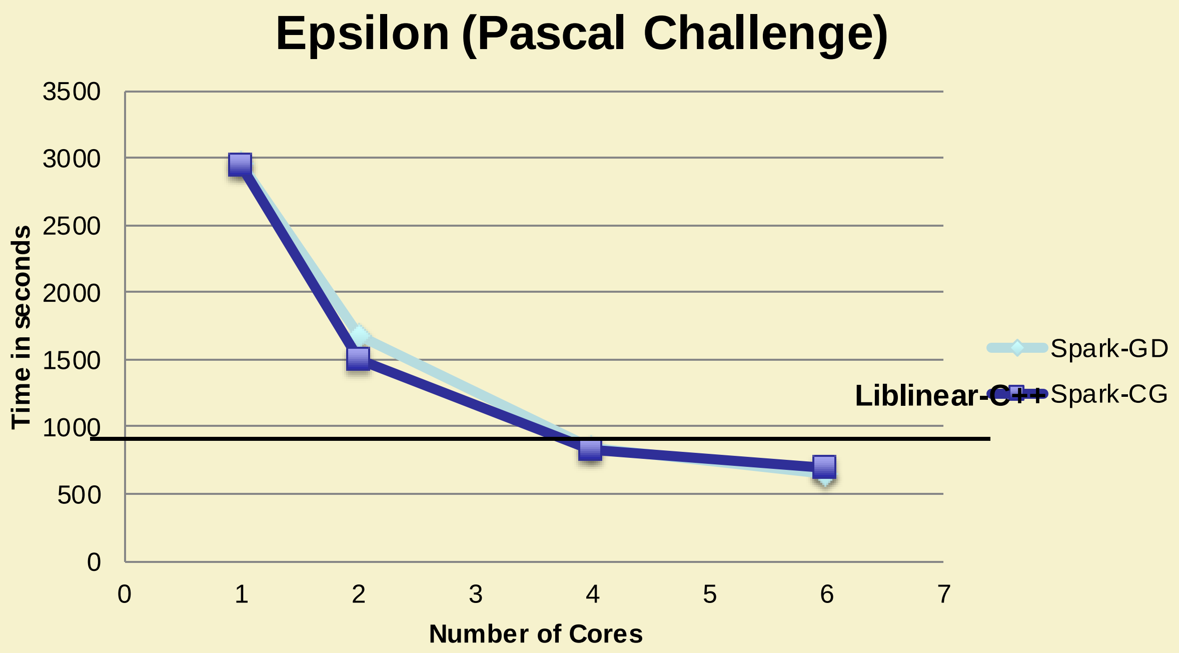
- Gradient Descent updates are:

$$w^{t+1} = w^t - s \nabla l(w^t)$$

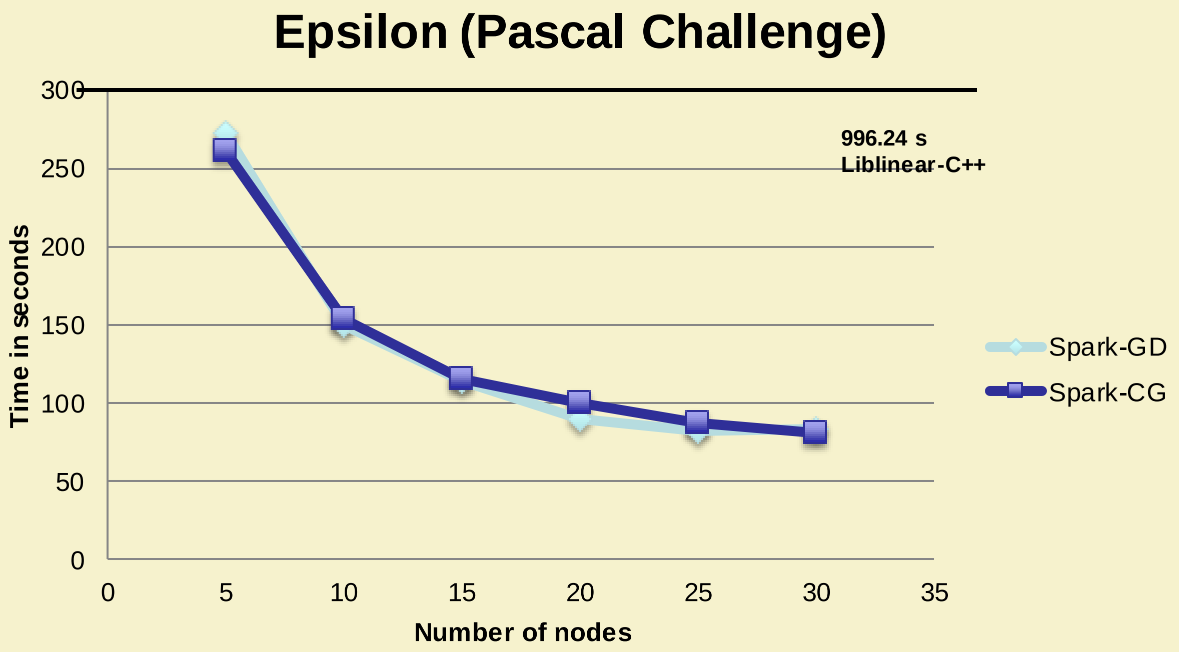
Spark Implementation

```
val x = loadData(file) //creates RDD
var w = 0
do {
//creates RDD
val g = x.map(a => grad(w,a)).reduce(_+_)
s = linesearch(x,w,g)
w = w - s * g
}while(norm(g) > e)
```

Scaleup with Cores



Scaleup with Nodes



Example: PageRank

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

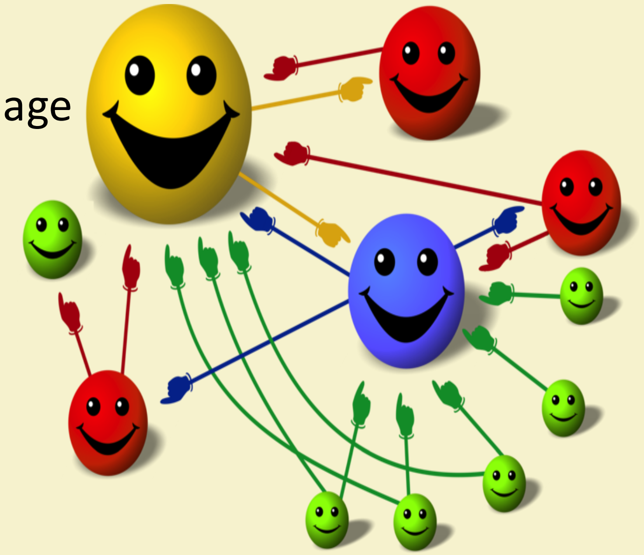
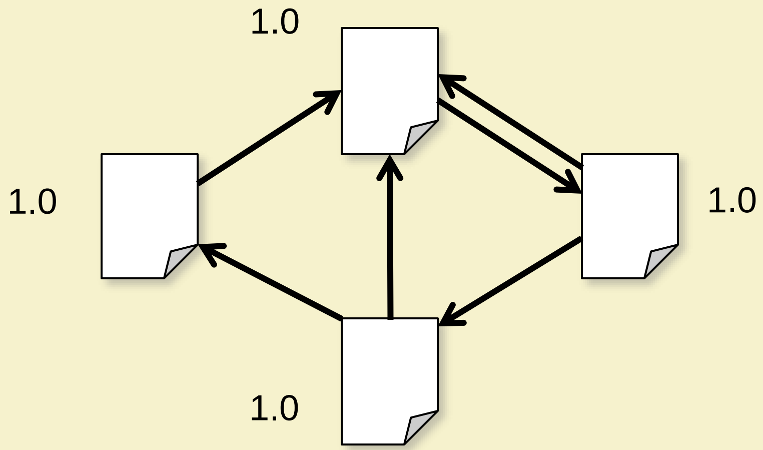


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

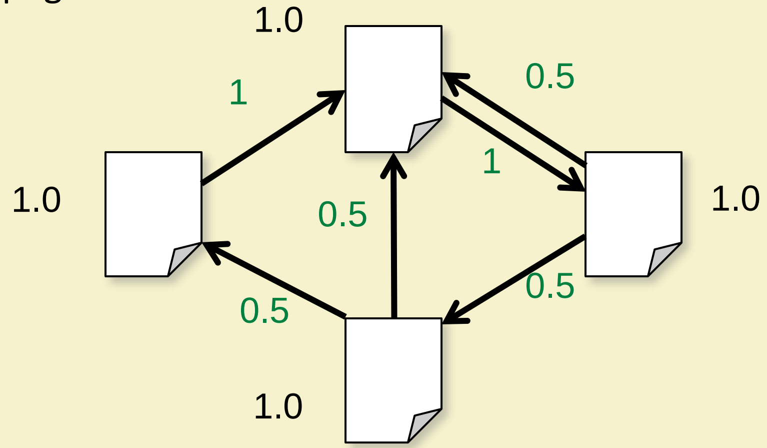
Algorithm

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



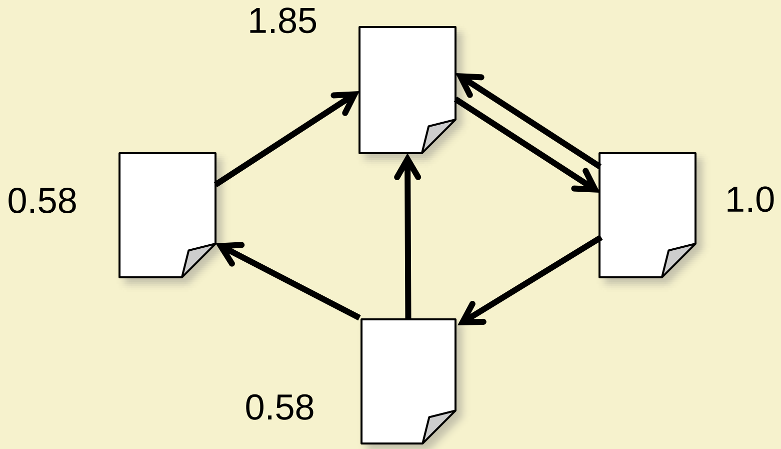
Algorithm

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 $0.15 + 0.85 \times contribs$



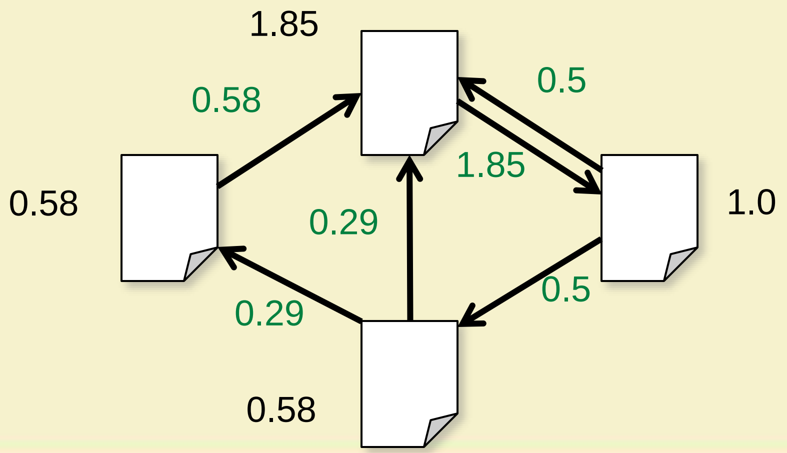
Algorithm

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



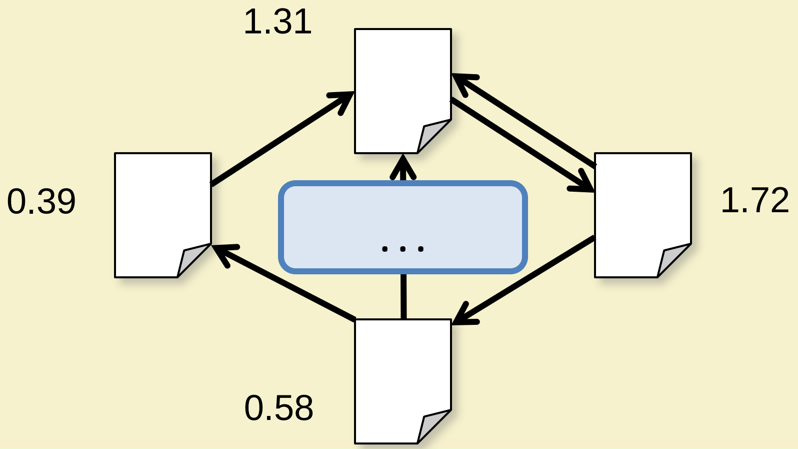
Algorithm

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



Algorithm

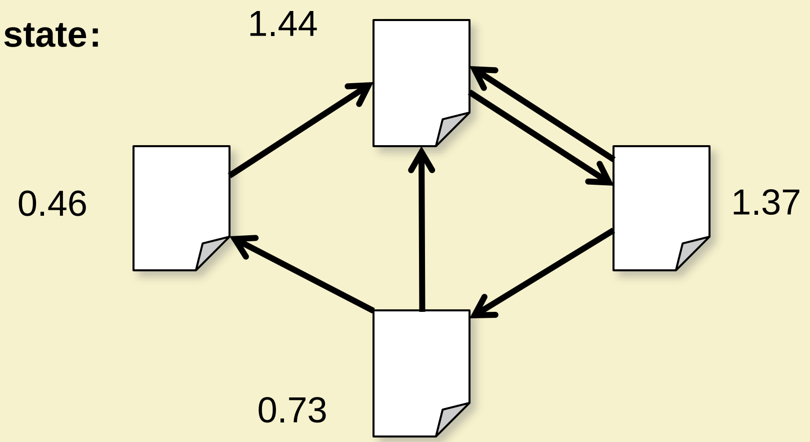
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to $0.15 + 0.85 \times$ contribs



Algorithm

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

Final state:



Spark Implementation

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

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

ranks.saveAsTextFile(...)
```

Conclusion:

- We have seen:
  - Spark
    - Motivation
    - RDD
    - Actions and transformations
    - Examples:
      - Matrix multiplication
      - Logistic regression
      - Pagerank

References:

- Learning Spark: Lightning-Fast Big Data Analysis. Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia. O Reilly Press 2015.

- Any book on scala and spark.

Thank You!!