

15-319 / 15-619

Cloud Computing

16th March , 2021

Overview

- **Last week's reflection**
 - Spark OPE
 - OLI Unit 3: Modules 10, 11, 12
 - Quiz 5
- **This week's schedule**
 - Project 4.1, due on Sunday, March 21st
 - OLI Unit 3: Module 13
 - Quiz 6, due on Thursday, March 18th
- **Twitter Analytics: The Team Project**
 - Query 1 Final, due on March, 21st
 - Query 2 Early bonus due on March, 21st

Modules to Read

- UNIT 3: Virtualizing resources for the cloud
 - **Module 13: Storage and Network virtualization**

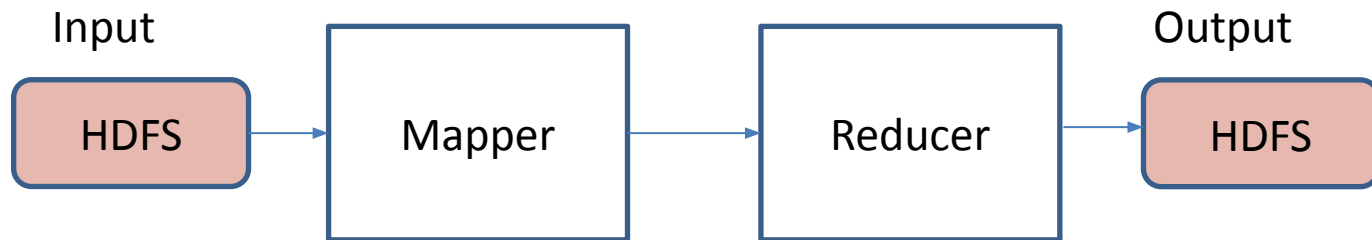


Project 4, Frameworks

- Project 4.1
 - Iterative Batch Processing Using Apache Spark
- Project 4.2
 - Machine Learning on the Cloud
- Project 4.3
 - Stream Processing with Kafka and Samza

Typical MapReduce Batch Job

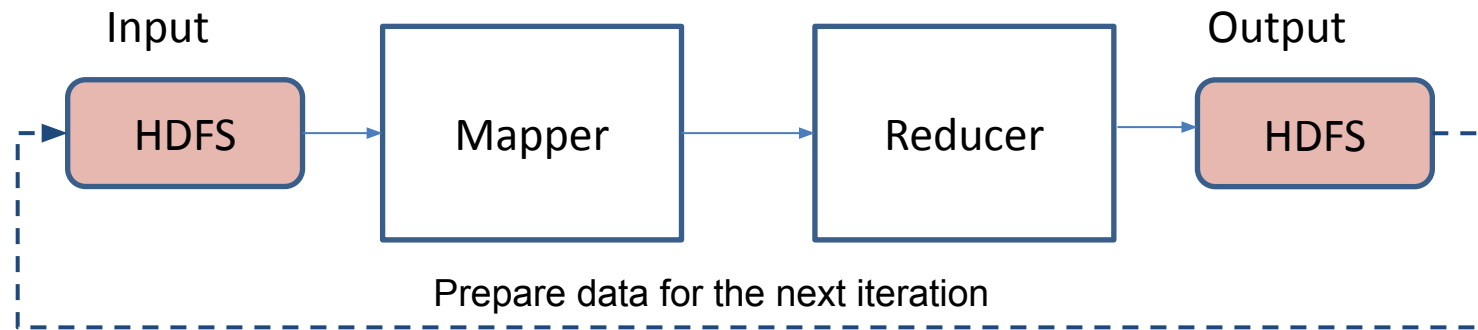
- Simplistic view of a MapReduce job



- You write code to implement the following classes
 - Mapper
 - Reducer
- Inputs are read from disk and outputs are written to disk
 - Intermediate data is spilled to local disk

Iterative MapReduce Jobs

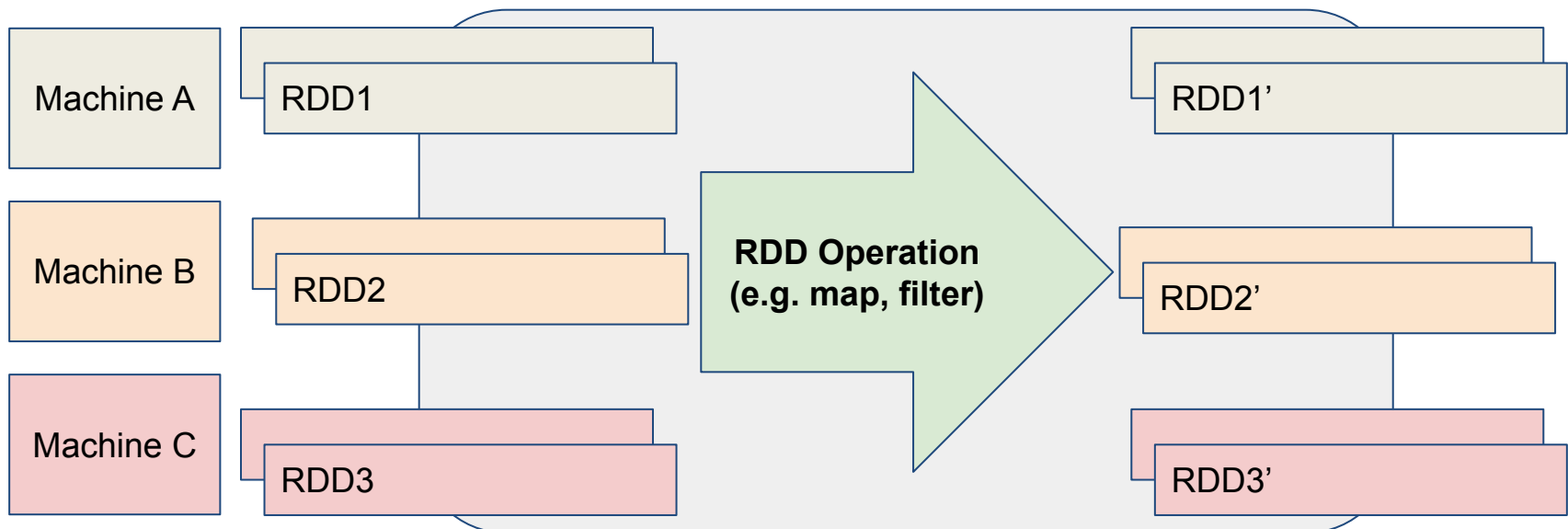
- Some applications require iterative processing
- E.g., Machine Learning



- MapReduce: Data is always **written** to disk
 - This leads to added overhead for each iteration
 - Can we keep data in memory? Across Iterations?
 - How do you manage this?

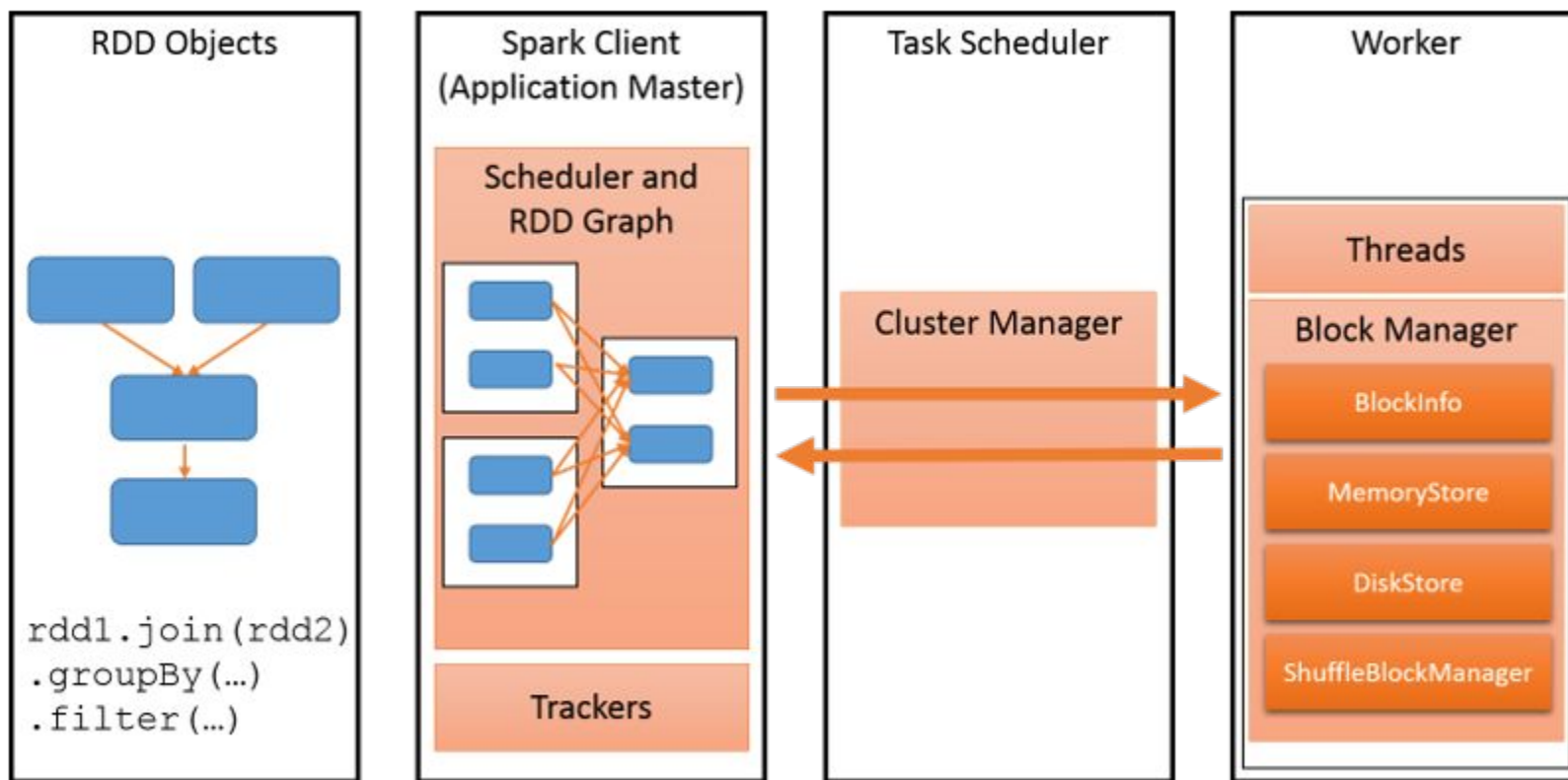
Key to Apache Spark - RDDs

- Resilient Distributed Datasets (RDDs)
- Can be in-memory or on disk
- Read-only objects
- Partitioned across the cluster based on a range or the hash of a key in each record



Apache Spark

- General-purpose cluster computing framework
- APIs in Python, Java, Scala and R
- Runs on Windows and UNIX-like systems



Spark Ecosystem

- [Spark SQL](#)
 - Process structured data
 - Run SQL-like queries against RDDs
- [Spark Streaming](#)
 - Ingest data from sources like Kafka
 - Process data with high level functions like map and reduce
 - Output data to live dashboards or databases
- [MLlib](#)
 - Machine learning algorithms such as regression
 - Utilities such as linear algebra and statistics
- [GraphX](#)
 - Graph-parallel framework
 - Support for graph algorithms and analysis

Apache Spark APIs

- There exists 3 sets of APIs for handling data in Spark

Resilient Distributed Dataset (RDD)

- Distributed collection of JVM objects
- Functional operators (map, filter, etc.)

DataFrame

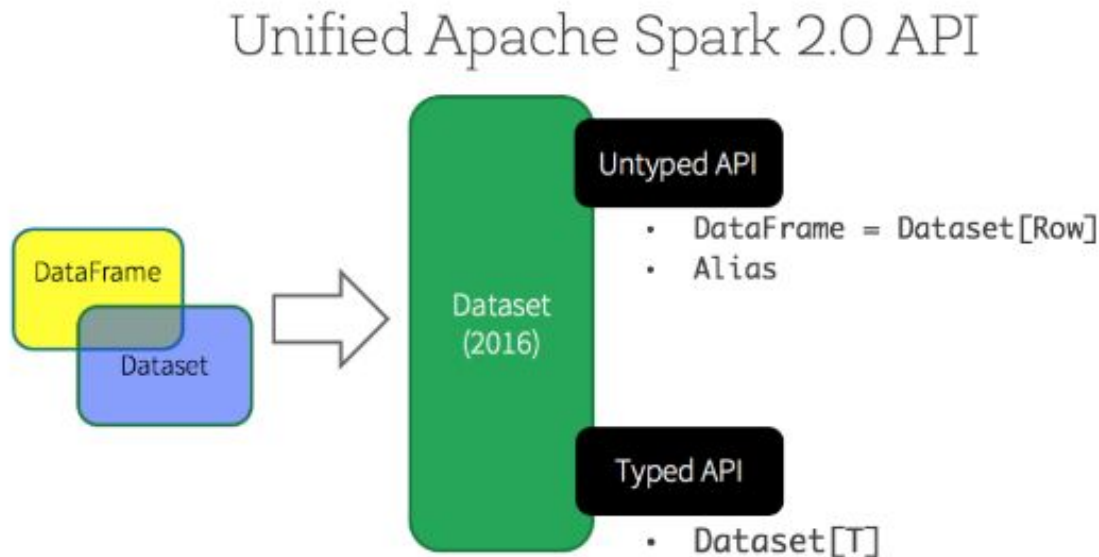
- Distributed collection of Row objects
- No compile time type safety
- Fast, efficient internal representations

Datasets

- Compile time type-safe
- Fast

DataFrames and Datasets

- A DataFrame is a collection of rows
 - Tabular
 - Organized into named columns, like a table in a relational DB
- A dataset is a collection of objects
 - Domain specific
 - Object oriented



Operations on RDDs

- Loading data

```
>>> input_RDD = sc.textFile("text.file")
```

- Transformation

- Applies an operation to derive a new RDD
- Lazily evaluated -- may not be executed immediately

```
>>> transform_RDD = input_RDD.filter(lambda x: "abcd" in x)
```

- Action

- Forces the computation on an RDD
- Returns a single object

```
>>> print "Number of “abcd”:" + transform_RDD.count()
```

- Saving data

```
>>> output.saveAsTextFile("hdfs:///output")
```

RDDs and Fault Tolerance

- Actions create new RDDs
- Uses the notion of lineage to support fault tolerance
 - Lineage is a log of transformations
 - Stores lineage on the driver node
 - Upon node failure, Spark loads data from disk to recompute the entire sequence of operations based on lineage

Operations on DataFrames

- Suppose we have a file `people.json`

```
{"name":"Michael"} {"name":"Andy", "age":30} {"name":"Justin", "age":19}
```

- Create a DataFrame with its contents

```
val df = spark.read.json("people.json")
```

- Run SQL-like queries against the data

```
val sqlDF = df.where($"age" > 20).show()
```

```
+---+-----+
```

```
|age|name|
```

```
+---+-----+
```

```
| 30|Andy|
```

```
+---+-----+
```

- Save data to file

```
df.where($"age" > 20).select("name").write.parquet("output")
```

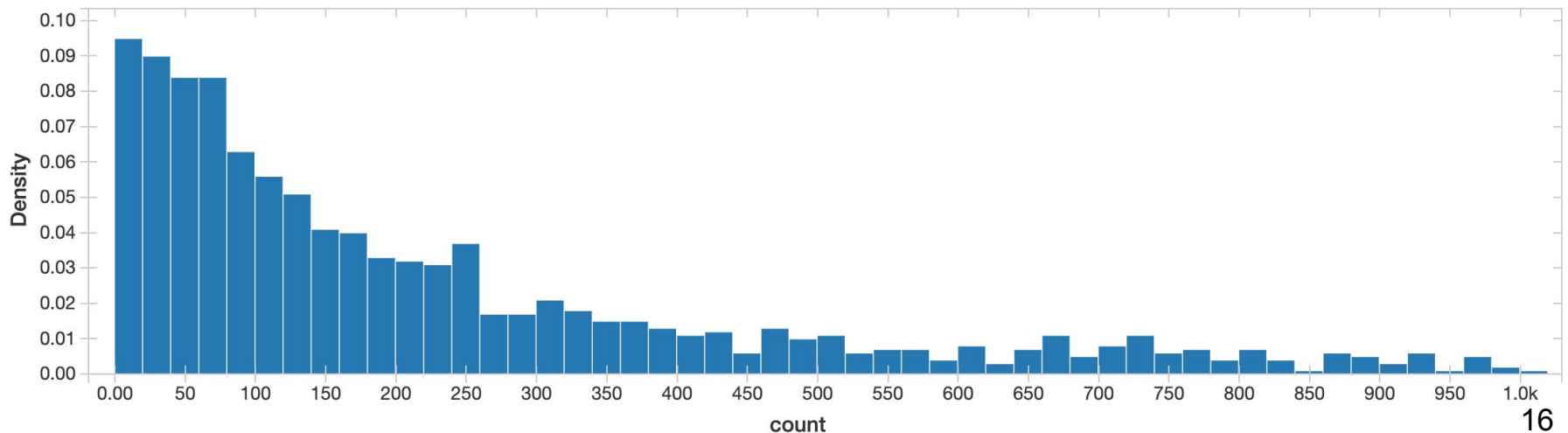
Note: Parquet is a column-based storage format for Hadoop.

Project 4.1

- **Spark OPE:** Implement a TF-IDF inverted index
- **Task 1:** Exploratory Analysis on a graph based dataset
- **Task 2:** Create an efficient Spark program to calculate user influence
- **Bonus:** Use Azure Databricks to run Task 2

Twitter Social Graph Dataset

- tsv format
- Appx. 10GB of data (**do not download**)
- Edge list of (follower, followee) pairs
 - Directed
- # of followers distribution → power tail



Task 1 Exploratory Data Analysis

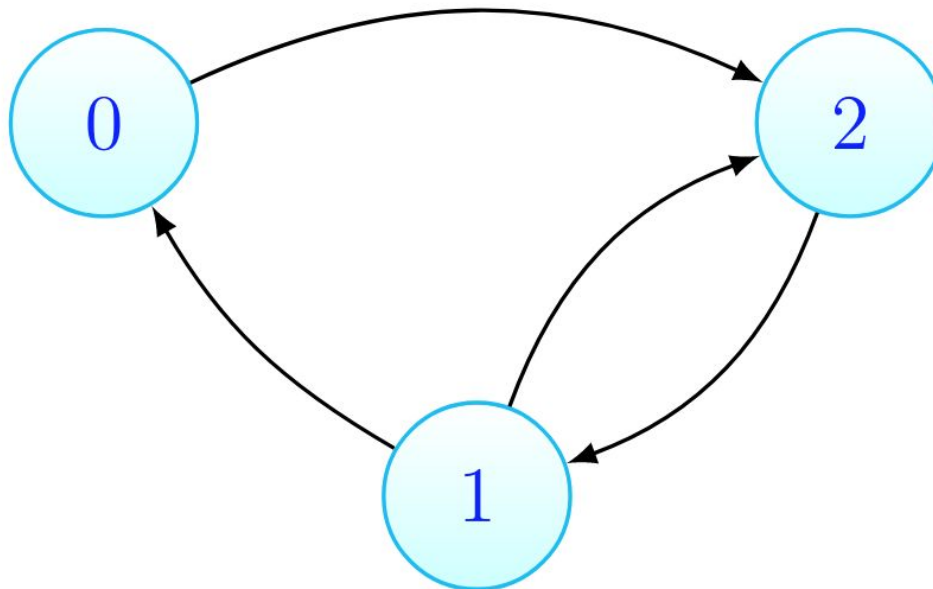
- Two parts to Task 1
 - a. Counting using Zeppelin notebook
 - Find the number of edges
 - Find the number of vertices
 - b. Find top 100 most-popular users
 - RDD API
 - Spark DataFrame API

Task 2: PageRank

- Started as an algorithm to rank websites in search engine results
- Assign ranks based on the number of links pointing to them
- A page that has links from
 - Many nodes $\Rightarrow$ high rank
 - A high-ranking node $\Rightarrow$ (slightly less) high rank
- In Task 2, we will implement pagerank to find the rank of each user

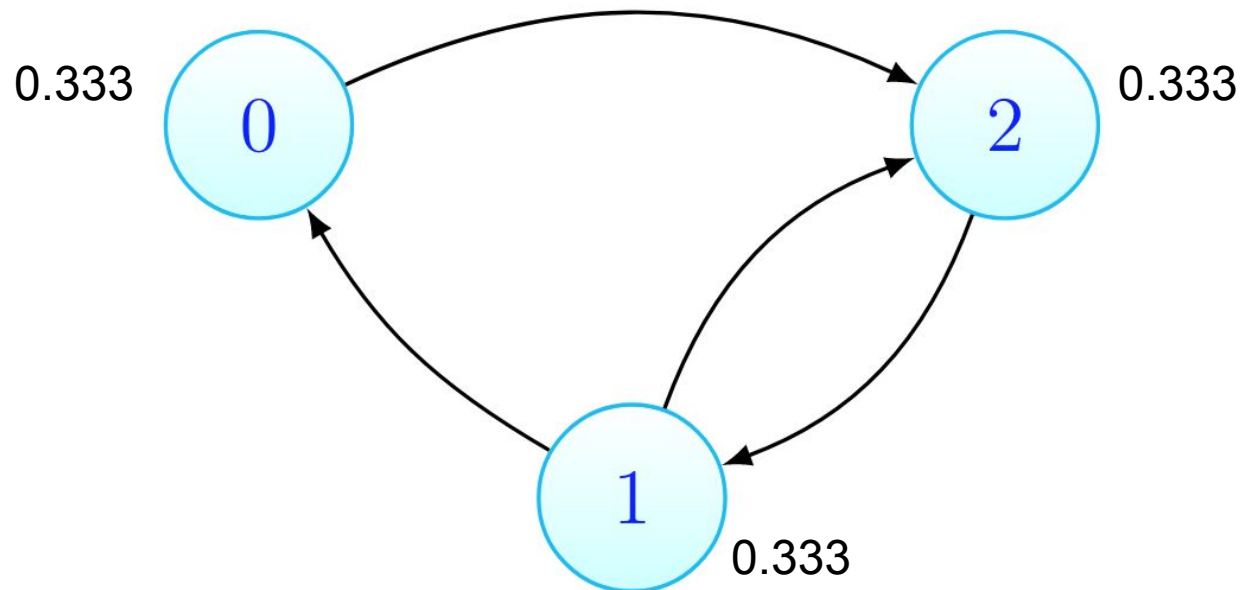
Basic PageRank

- How do we measure influence?
 - Intuitively, it should be the node with the most followers



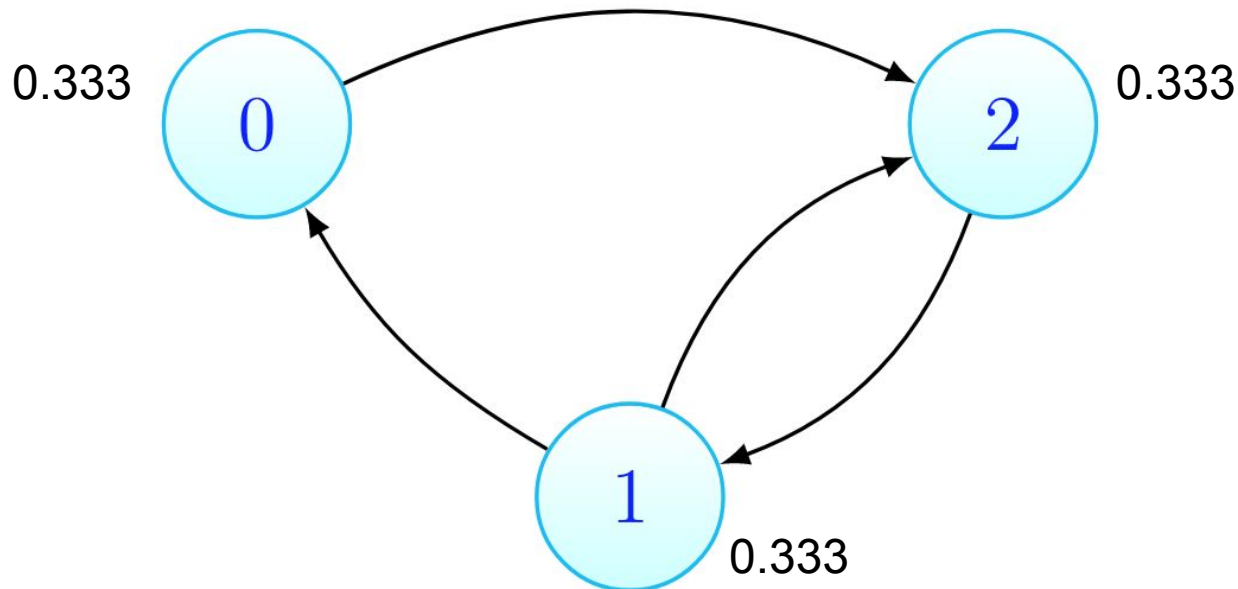
Basic PageRank

- Influence scores are initialized to $1.0 / \# \text{ of vertices}$



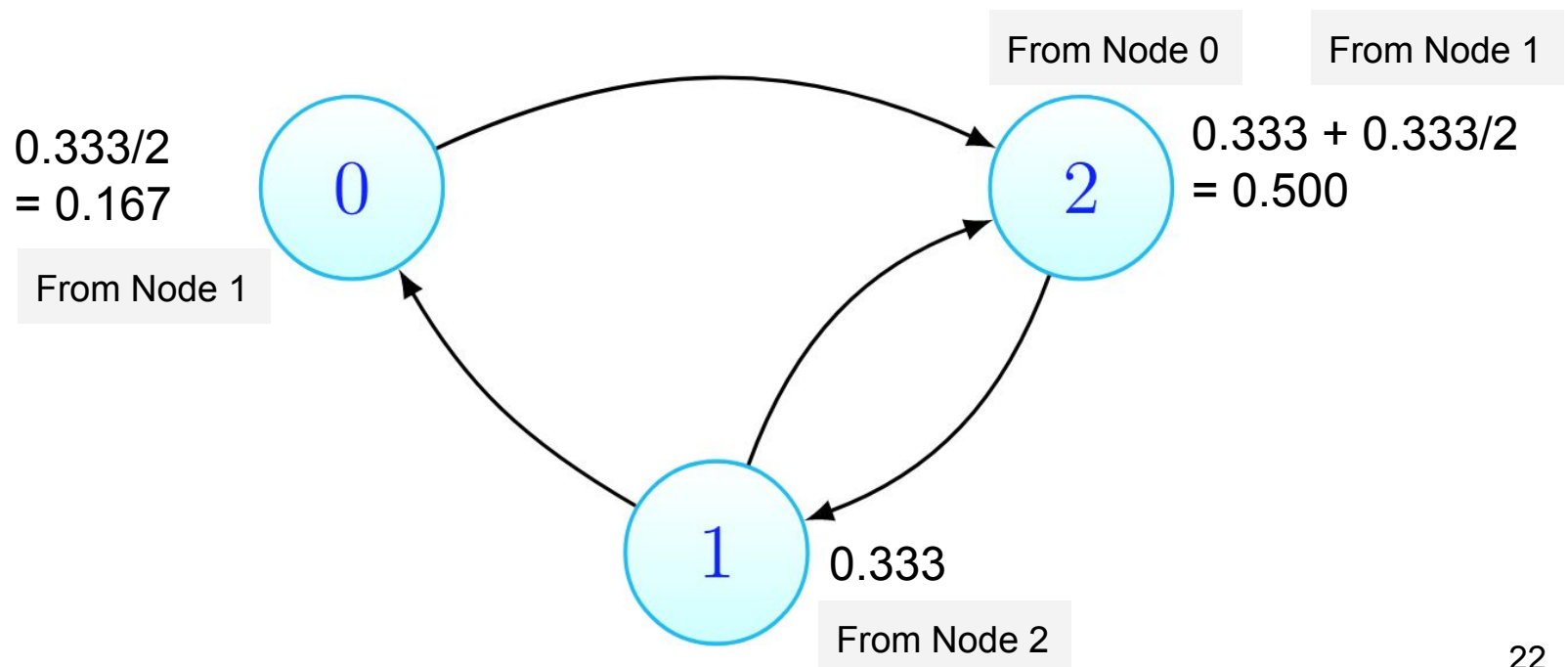
Basic PageRank

- Influence scores are initialized to $1.0 / \# \text{ of vertices}$
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



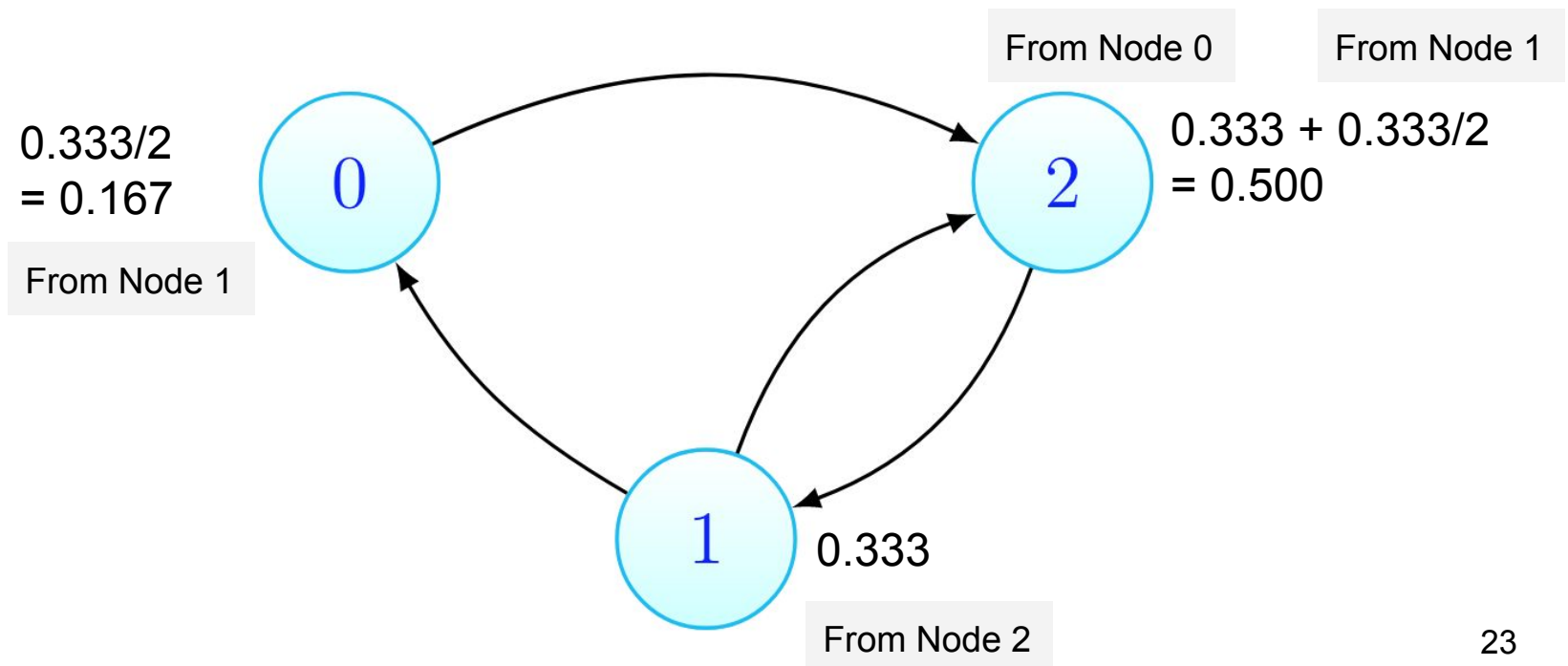
Basic PageRank

- Influence scores are initialized to $1.0 / \# \text{ of vertices}$
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



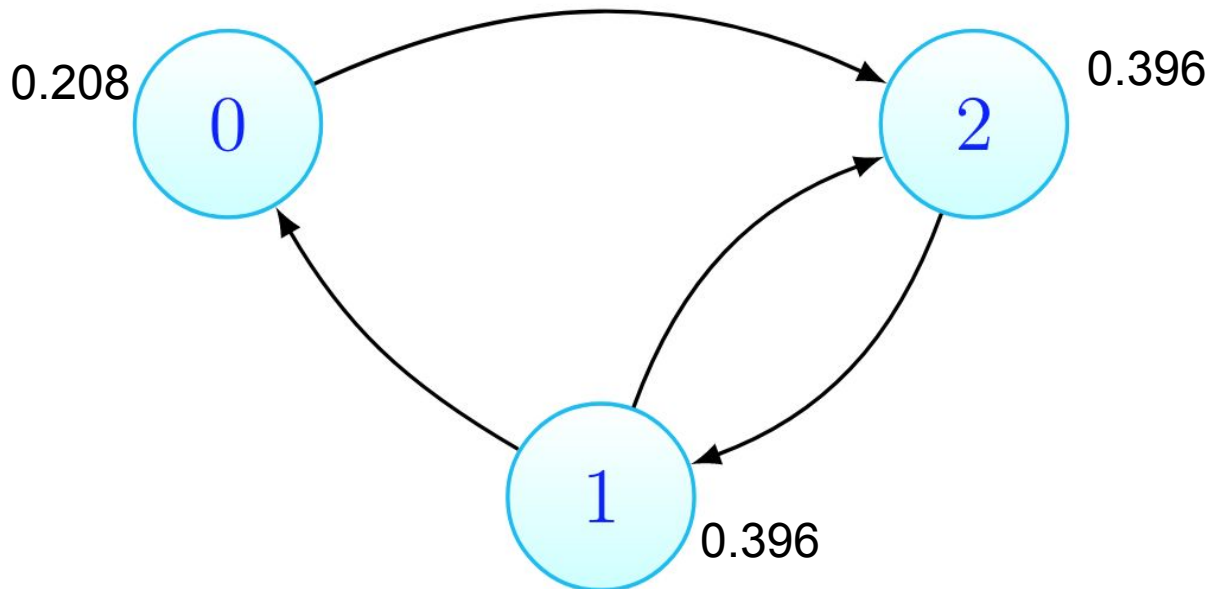
Basic PageRank

- Influence scores are initialized to $1.0 / \# \text{ of vertices}$
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following
- Convergence is achieved when the scores of nodes do not change between iterations
- PageRank is guaranteed to converge



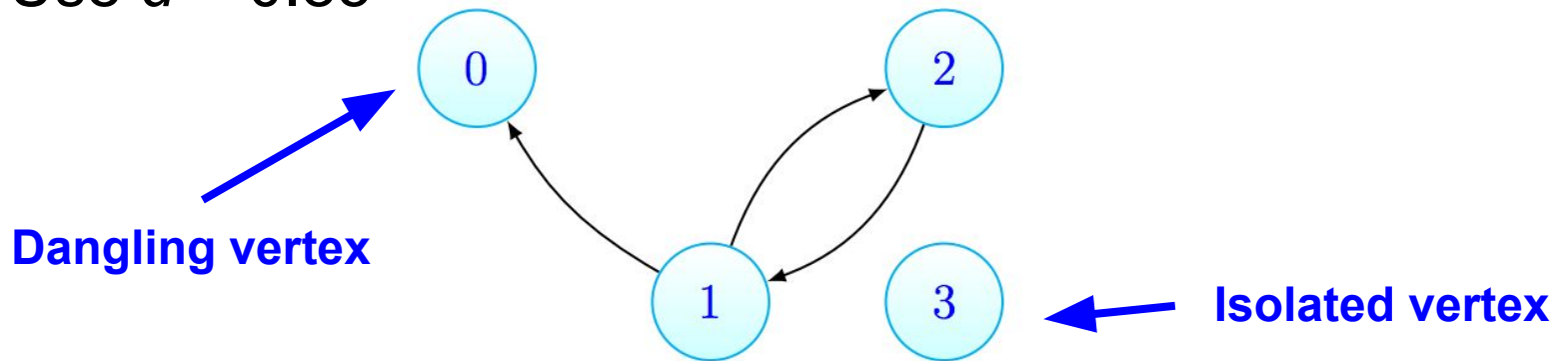
Basic PageRank

- Influence scores are initialized to $1.0 / \# \text{ of vertices}$
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following
- Convergence is achieved when the scores of nodes do not change between iterations
- PageRank is guaranteed to converge



PageRank Terminology

- **Dangling or sink vertex**
 - No outgoing edges
 - Redistribute contribution equally among all vertices
- **Isolated vertex**
 - No incoming and outgoing edges
 - **No isolated nodes in Project 4.1 dataset**
- **Damping factor d**
 - Represents the probability that a user clicking on links will continue clicking on them, traveling down an edge
 - Use $d = 0.85$



Basic PageRank Pseudocode

(Note: This does not meet the requirements of Task 2)

```
val links = spark.textFile(...).map(...).cache()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS)
{
    // Build an RDD of (targetURL, float) pairs
    // with the contributions sent by each page
    val contribs = links.join(ranks).flatMap
    {
        case (url, (links, rank)) =>
            links.map(dest => (dest, rank/links.size))
    }

    // Sum contributions by URL and get new ranks
    ranks = contribs.reduceByKey(_ + _)
                    .mapValues(sum => a/N + (1-a)*sum)
}
```

Visualizing Transitions

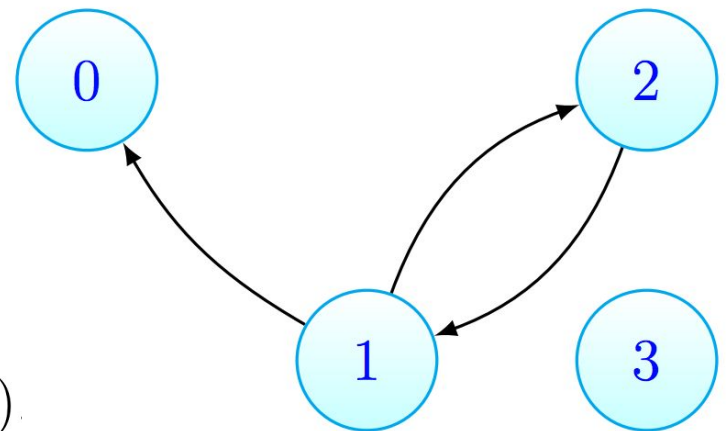
- Adjacency matrix:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

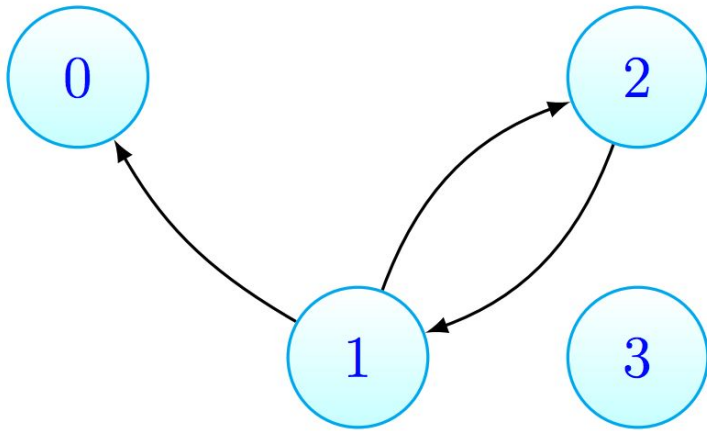
- Transition matrix: (rows sum to 1)

$$\mathbf{M} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

$$M_{ij} = \frac{G_{ij}}{\sum_{k=1}^n G_{ik}} \left(\text{when } \sum_{k=1}^n G_{ik} \neq 0 \right)$$



Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1 - d)r_i^{(0)}$$

$$d = 0.85$$

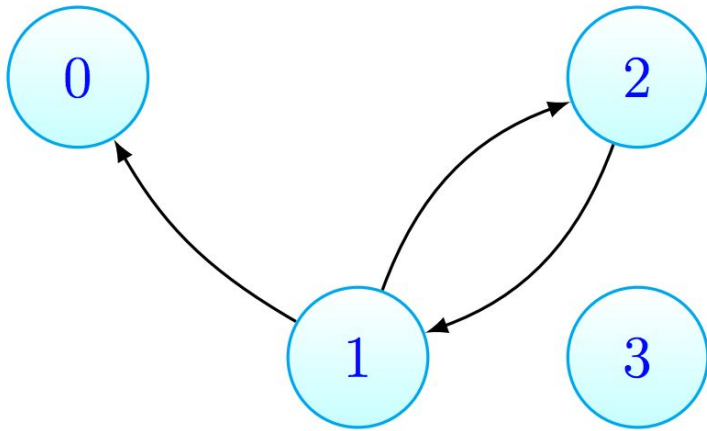
$$r_0^{(1)} = d\left(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right) + (1 - d)\frac{1}{n}$$

$$r_1^{(1)} = d\left(\frac{r_2^{(0)}}{1} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right) + (1 - d)\frac{1}{n}$$

$$r_2^{(1)} = d\left(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right) + (1 - d)\frac{1}{n}$$

$$r_3^{(1)} = d\left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right) + (1 - d)\frac{1}{n}$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1 - d)r_i^{(0)}$$

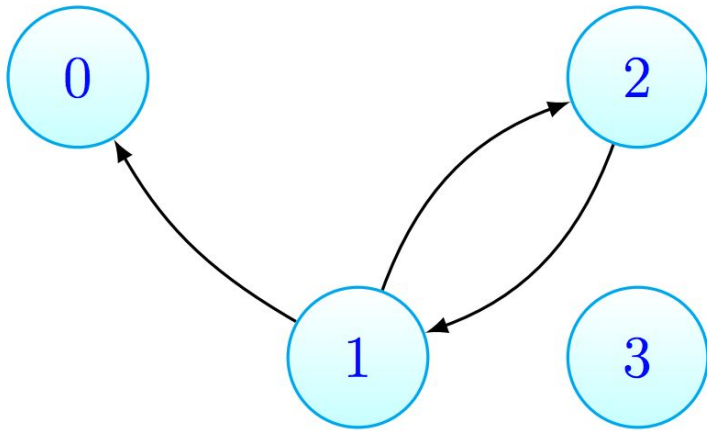
$$d = 0.85$$

Note: contributions from isolated and dangling vertices are constant in an iteration

Let

$$\epsilon = d \left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4} \right)$$

Task 2: PageRank



This simplifies the formula to

$$r_0^{(1)} = d \frac{r_1^{(0)}}{2} + \epsilon + (1 - d) \frac{1}{n}$$

$$r_1^{(1)} = d \frac{r_2^{(0)}}{1} + \epsilon + (1 - d) \frac{1}{n}$$

$$r_2^{(1)} = d \frac{r_1^{(0)}}{2} + \epsilon + (1 - d) \frac{1}{n}$$

$$r_3^{(1)} = \epsilon + (1 - d) \frac{1}{n}$$

Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1 - d) r_i^{(0)}$$

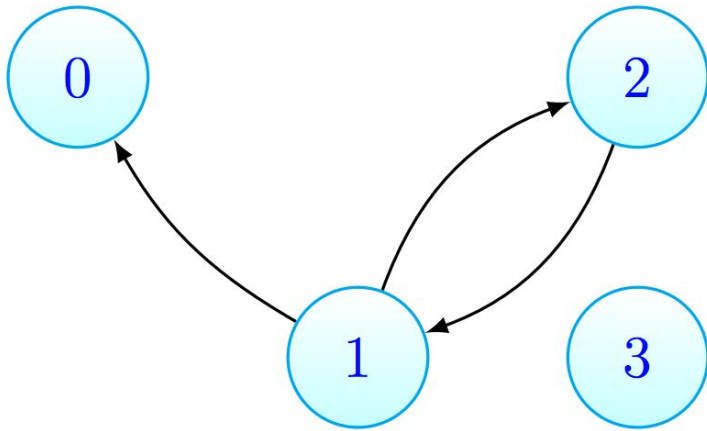
$$d = 0.85$$

Note: contributions from isolated and dangling vertices are constant in an iteration

Let

$$\epsilon = d \left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4} \right)$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1 - d)r_i^{(0)}$$

$$d = 0.85$$

$$\epsilon = 0.85 \times (0.25/4 + 0.25/4) = 0.106$$

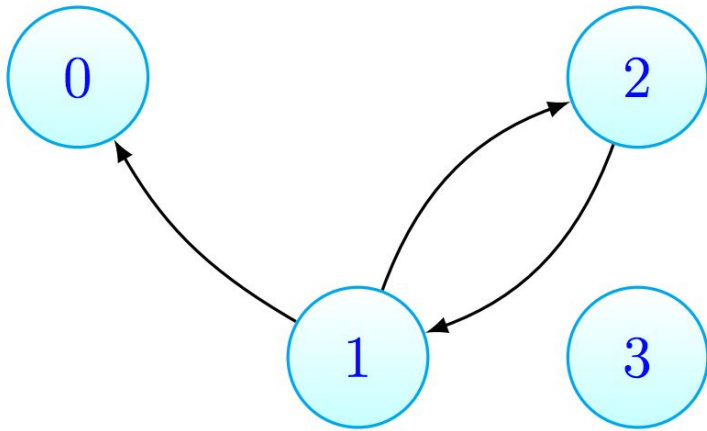
$$r_0^{(1)} = 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25$$

$$r_1^{(1)} = 0.85 \times 0.25 + 0.106 + 0.15 \times 0.25 = 0.356$$

$$r_2^{(1)} = 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25$$

$$r_3^{(1)} = 0.106 + 0.15 \times 0.25 = 0.144$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1 - d)r_i^{(0)}$$

$$d = 0.85$$

$$r_0^{(k)} = 0.2656$$

$$r_1^{(k)} = 0.3487$$

$$r_2^{(k)} = 0.2656$$

$$r_3^{(k)} = 0.1199$$

What you need to do for Task 2

- Run your page rank application on a 10GB graph data for *10 iterations*.
- Using HDInsight cluster on Azure:
 - Use the Terraform template provided
 - **Very expensive - 2.6USD per hour**
- Scoring for Task 2 has 2 components:
 - 100% correctness for page rank - 30 points
 - Performance optimization (runtime within 30 minutes) - 30 points

General Hints

- Starter code:
 - SparkUtils.scala - Use this for creating SparkSession objects.
- Test out commands on a Zeppelin notebook (refer to the Zeppelin primer)
- Test Driven Development (TDD):
 - Starter code contains a small graph test.
 - **Develop and test locally first!**
 - **Develop and test locally first!**
 - **Develop and test locally first!** HDInsight clusters are expensive
 - Add more test cases to check robustness.
 - Each submission can take anywhere from 6 min to an hour to run on the cluster.
- When in doubt, read the docs!
 - [SparkSQL](#)
 - [RDD](#)
- Don't forget to include in your submission
 - Updated references file
- Arguably the hardest P4 project. Start early!

Pagerank Hints

- Ensuring correctness

- Make sure total scores sum to 1.0 in every iteration
- Understand closures in Spark
 - Do not do something like this

```
val data = Array(1,2,3,4,5)
var counter = 0
var rdd = sc.parallelize(data)
rdd.foreach(x => counter += x)
println("Counter value: " + counter)
```
- Graph representation
 - Adjacency lists use less memory than matrices
- More detailed walkthroughs and sample calculations can be found [here](#)

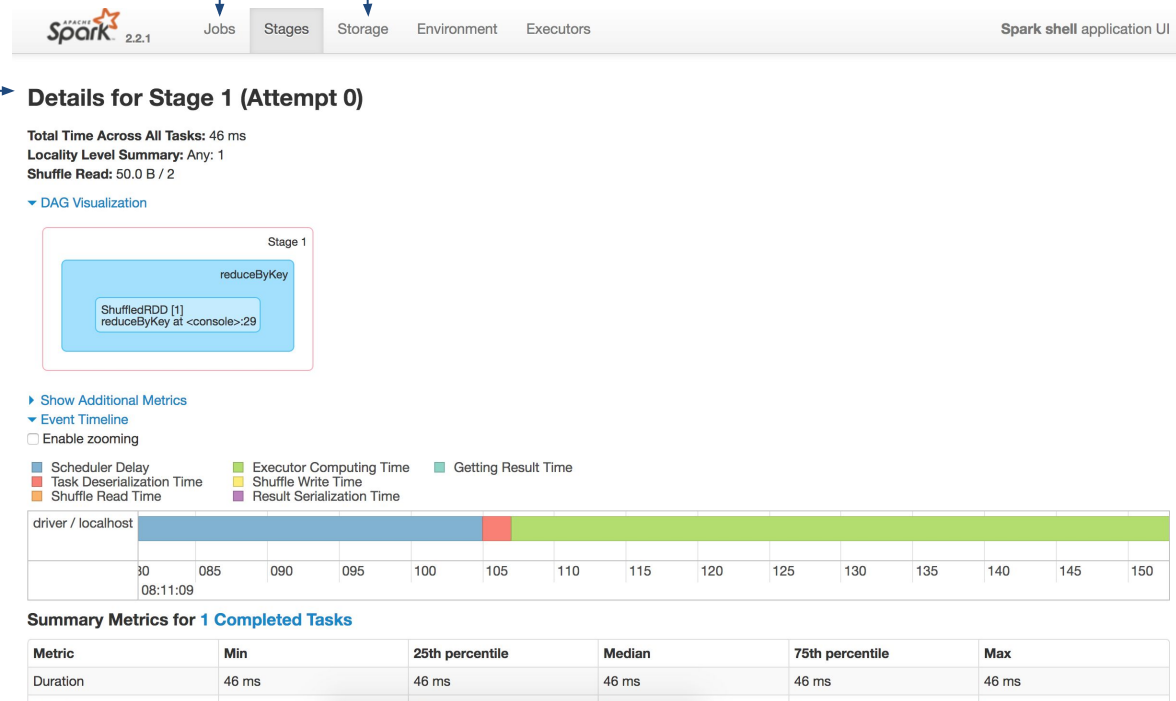
Spark UI

- Provides useful information on your Spark programs
- You can learn about resource utilization of your cluster
- Is a stepping stone to optimize your jobs

Status of RDD
actions being
computed

Info about cached
RDDs and
memory usage

In-depth job info



Optimization Hints

- Understand RDD manipulations
 - Actions vs Transformations
 - Lazy transformations
- Use the Ambari UI
 - Are you utilizing your cluster completely? How can you change that? Refer optimization hints in the writeup.
- Use the Spark UI
 - Are your RDDs cached as expected?
 - Memory errors - check container logs
 - Parameter tuning applied successfully?
 - Exponential increase in partitions?
- How do you represent the node IDs? Int/String/Long?
- **Many more optimization hints in the writeup!**

Bonus Task - Databricks

- Databricks is an Apache Spark-based unified analytics platform.
- Azure Databricks is optimized for Azure
 - Software-as-a-Service
- One-click setup, an interactive workspace, and an optimized Databricks runtime
- Optimized connectors to Azure storage platforms for fast data access
- Run the same PageRank application (in Task 2) on Azure Databricks to compare the differences with Azure HDInsight

What you need to do for bonus?

- You can only get bonus (10 points) when:
 - 100% correctness
 - Runtime under 30 minutes on Databricks
- Copy your code to a Databricks notebook:
 - **Do not** create or destroy SparkSession objects
 - Change the output to DBFS instead of WASB
- Create a cluster and job using databricks-setup.sh
- Submitter takes in a job ID
- Don't forget to destroy resources after you are done!

How to change your code?

```
object PageRank {  
  def calculatePageRank(inputGraphPath: String, outputPath: String, iterations: Int, isLocal: Boolean): Unit = {  
    val spark = SparkUtils.getSparkSession(isLocal, appName = "PageRank")  
    val sc = spark.sparkContext  
  
    ... Your implementation goes here ...  
    graphRDD = sc.textFile(inputGraphPath)  
    graphRDD.map(...)  
  
    spark.close()  
  }  
  
  def main(args: Array[String]): Unit = {  
    val inputGraph = "wasb://spark@cmuccpublicdatasets.blob.core.windows.net/Graph"  
    val outputPath = "wasb:///pagerank-output"  
    val iterations = 10  
  
    calculatePageRank(inputGraph, outputPath, iterations, isLocal=false)  
  }  
}
```

How to change your code?

```
object PageRank {  
  def calculatePageRank(inputGraphPath: String, outputPath: String, iterations: Int, isLocal: Boolean): Unit = {  
    val spark = SparkUtils.getSparkSession(isLocal, appName = "PageRank")  
    val sc = spark.sparkContext  
  
    val inputGraph = "wasb://spark@cmuccpublicdatasets.blob.core.windows.net/Graph"  
    val outputPath = "dbfs:/pagerank-output"  
    val iterations = 10  
    ... Your implementation goes here ...  
    graphRDD = sc.textFile(inputGraphPath)  
    graphRDD.map(...)  
  
    spark.close()  
  }  
  
  def main(args: Array[String]): Unit = {  
    calculatePageRank(inputGraph, outputPath, iterations, isLocal=false)  
  }  
}
```

Deadlines

- **This week's schedule**
 - Project 4.1, due on Sunday, March 21st
 - OLI Unit 3: Module 13
 - Quiz 6, due on Thursday, March 18th
- **Twitter Analytics: The Team Project**
 - Query 1 Final, due on March, 21st
 - Query 2 Early bonus due on March, 21st

Questions?

