15-319 / 15-619
Cloud Computing

Recitation 10

Tuesday, October 29, 2019

Overview

- **Last week's reflection**
  - Spark OPE
  - Team Project Phase 1
  - OLI Unit 4: Modules 15, 16, 17
  - Quiz 8
- **This week's schedule**
  - Project 4.1, due on Sunday, November 3
  - OLI Unit 4: Module 18
  - Quiz 9, due on Friday, November 1
- **Twitter Analytics: The Team Project**
  - Phase 2 Live Test, due on Sunday, November 10

Conceptual Content on OLI

- OLI, Unit 4
  - Module 18
    - Introduction to Distributed Programming for the Cloud

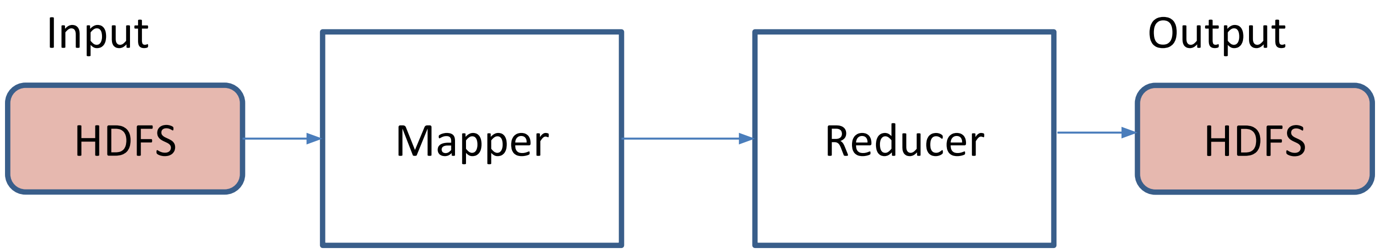
- Quiz 9
  - Friday, November 1, 2019

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

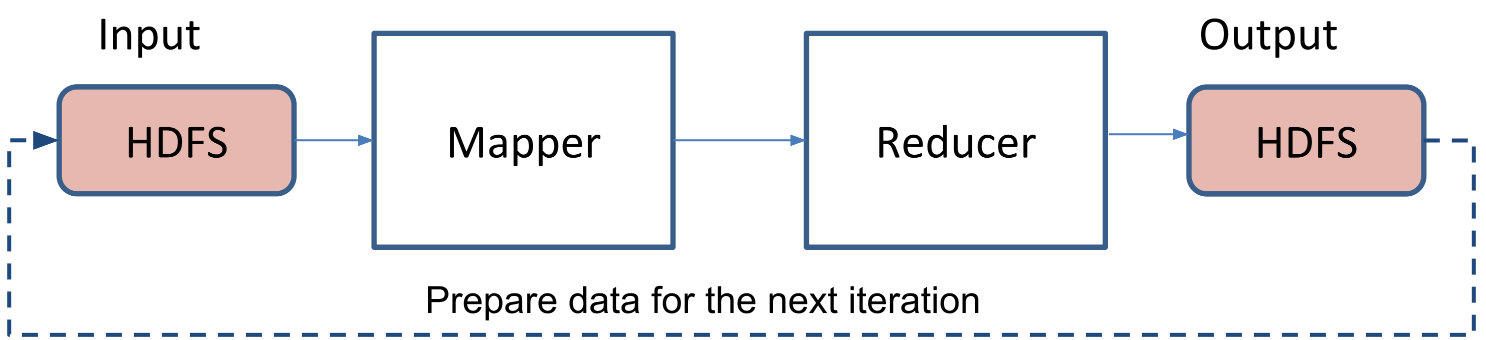
- Simplistic view of a MapReduce job



- 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

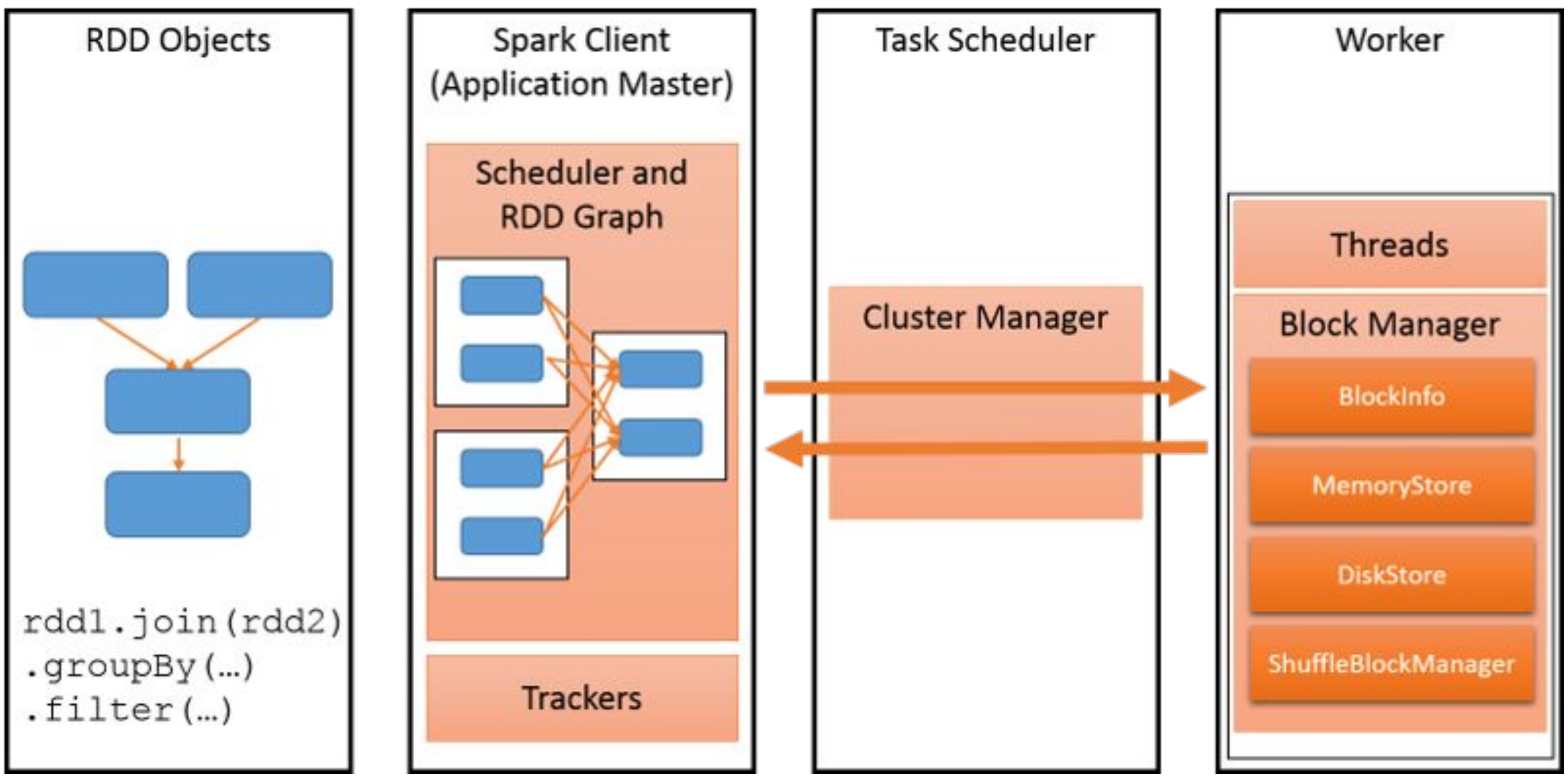
- Some applications require iterative processing
- E.g., Machine Learning



- 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?

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

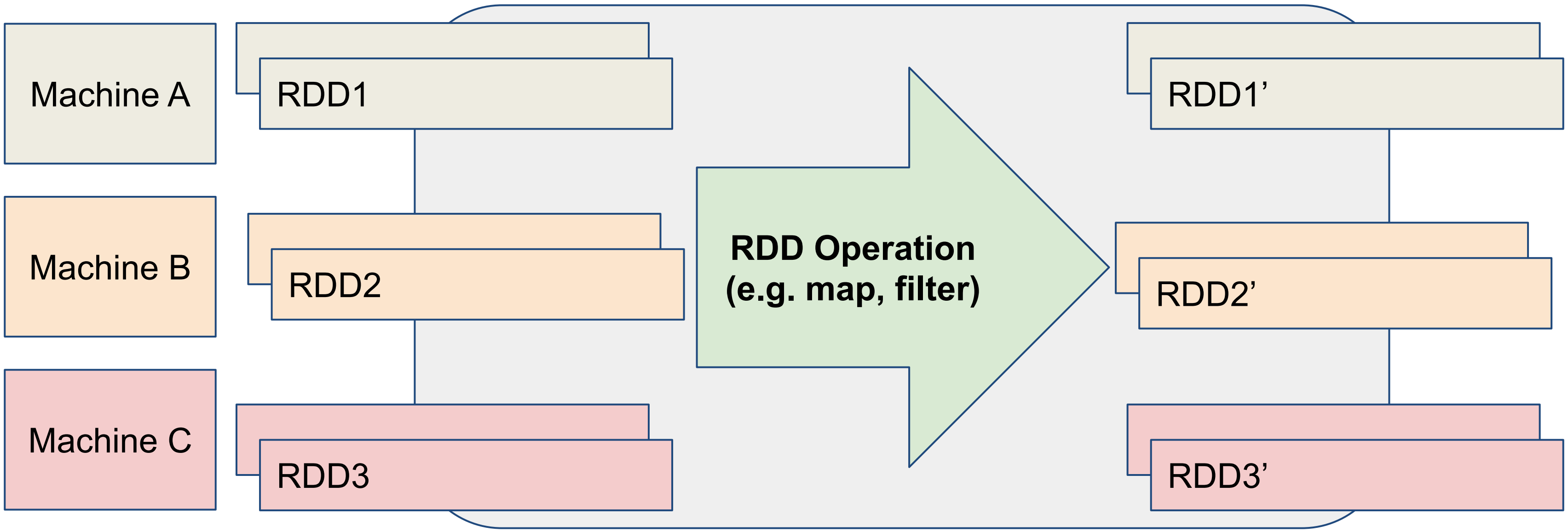
Apache Spark APIs

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

Resilient Distributed Dataset (RDD)	DataFrame	Dataset
• Distributed collection of JVM objects • Functional operators (map, filter, etc.)	• Distributed collection of Row objects • Expression-based operations • Fast, efficient internal representations	• Internally rows, externally JVM objects • Type safe and fast • Slower than dataframes

Resilient Distributed Datasets

- 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



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

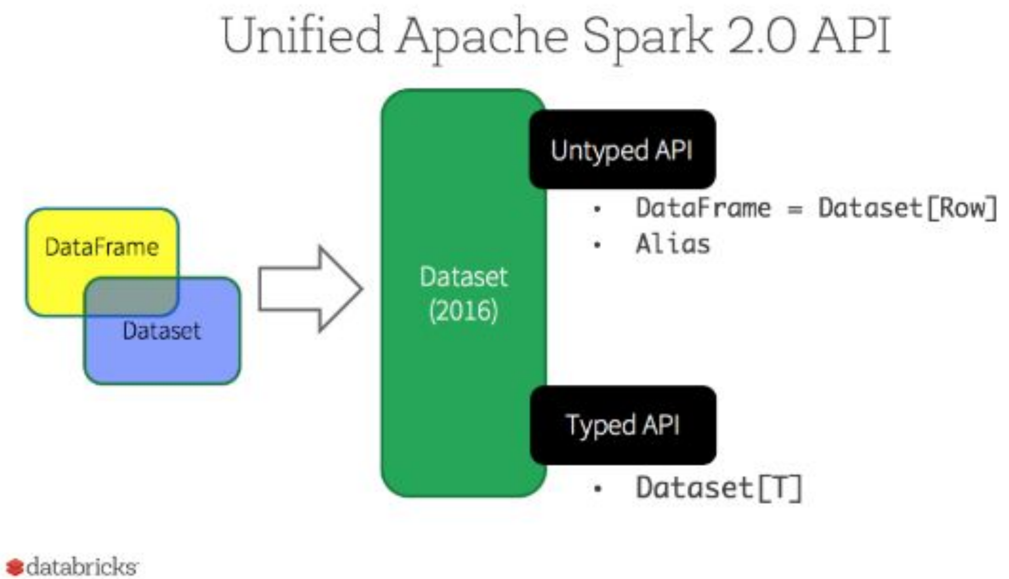
```
>>> output.saveAsTextFile(“hdfs:///output”)
```

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

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 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.filter($"age" > 20).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

- Save data to file

```
df.filter($"age" > 20).select(“name”).write.format(“parquet”).save(“output”)
```

Note: Parquet is a column-based storage format for Hadoop. You will need special dependencies to read this file

Project 4.1

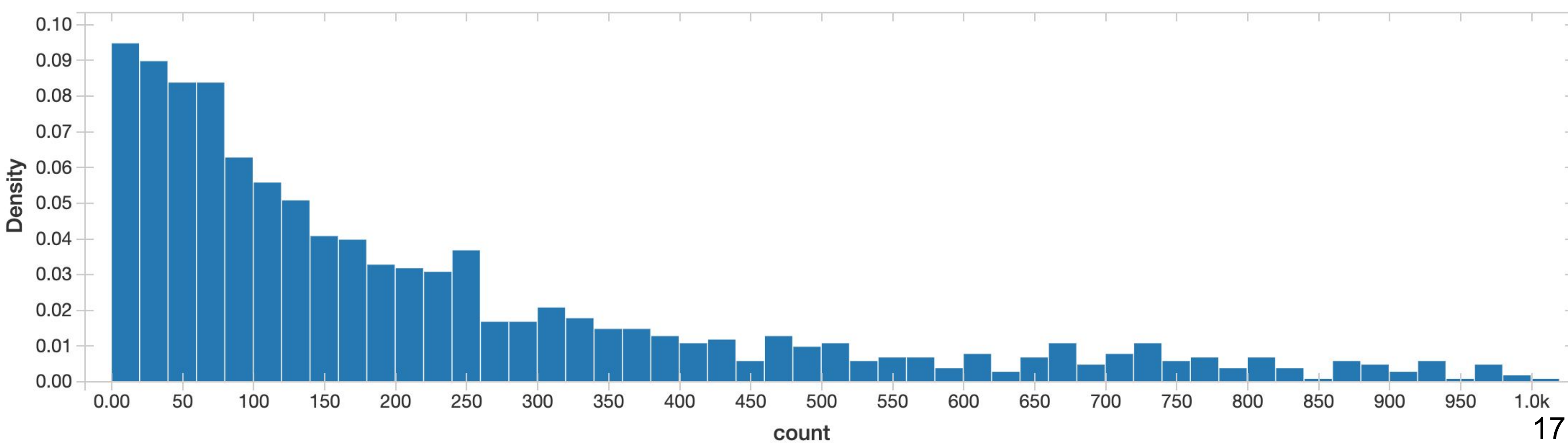
Task	Points	Description	Language
Spark OPE	5	Implement a simple inverted index application	Scala
Task 1	20	Data exploratory analysis on a Twitter graph dataset	Scala
Task 2	30 + 30	Write efficient Spark program to calculate the influence of users in the Twitter graph dataset	Scala
Bonus task	10	Use Azure Databricks to run the pagerank algorithm in Task 2	Scala
Reflection & discussion	5		-
Manual grading	5		-
Code Review	5		

Project 4.1

- **Spark OPE:** Implement a simple 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

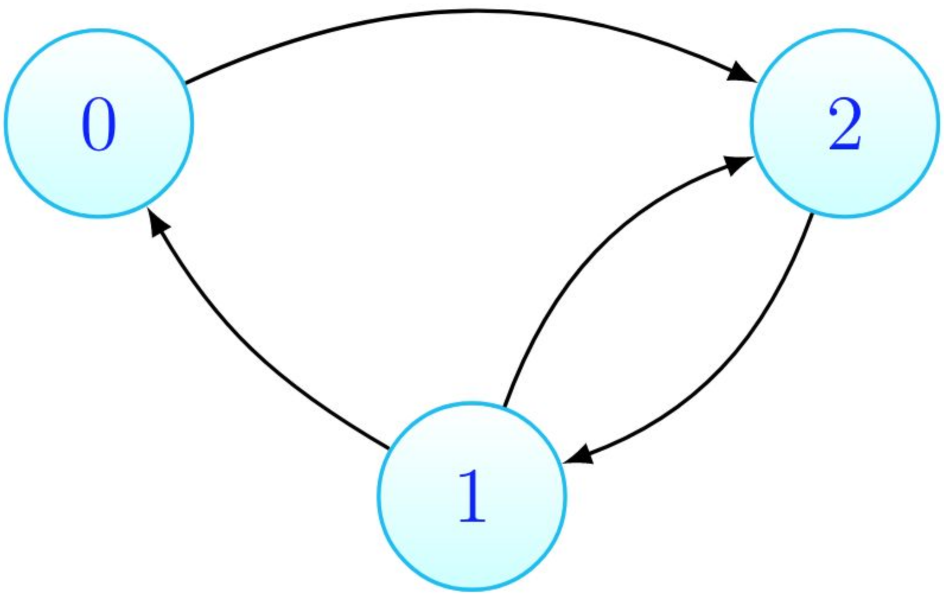
- Two parts to Task 1
  a. Counting using Jupyter notebook
     - Find the number of edges
     - Find the number of vertices
  b. Find top 100 most-popular users
     - RDD API
     - Spark DataFrame API

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 ⇒ high rank
  - A high-ranking node ⇒ (slightly less) high rank
- In Task 2, we will implement pagerank to find the top 100 influential vertices (*meaning accounts*) in the Twitter social graph

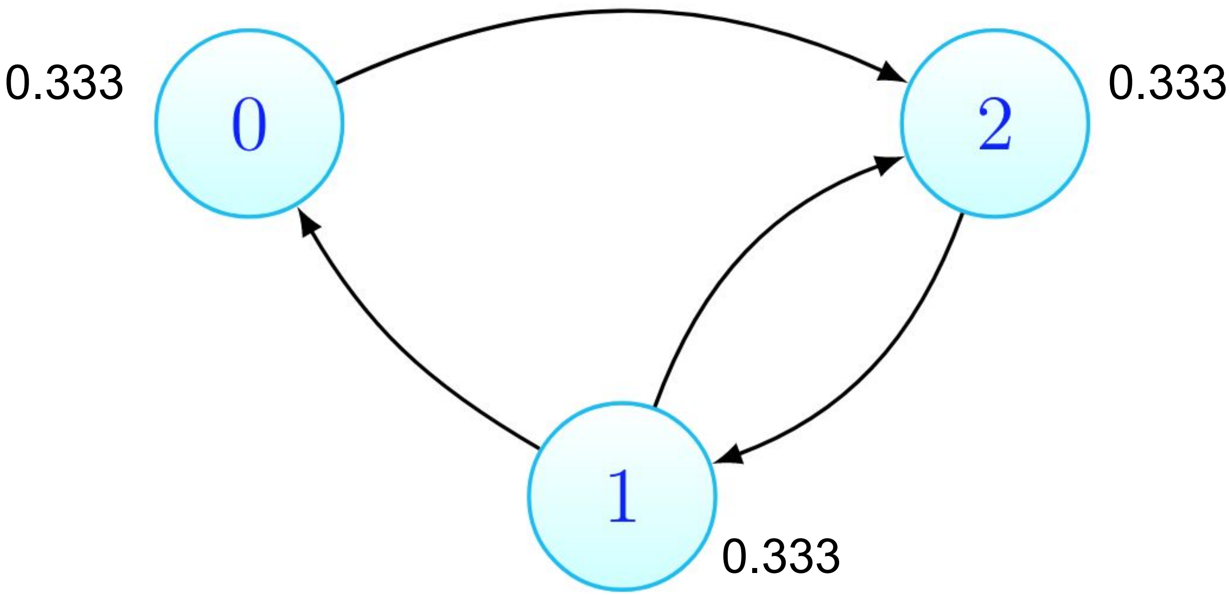
Basic PageRank

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



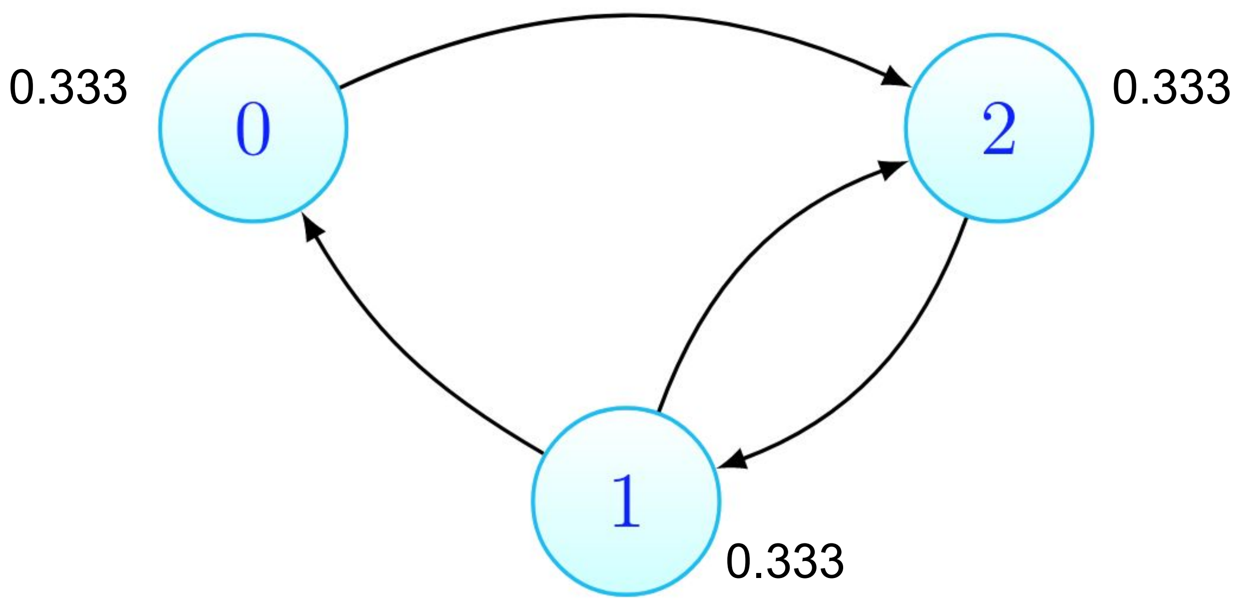
Basic PageRank

- Influence scores are initialized to `1.0 / # of vertices`



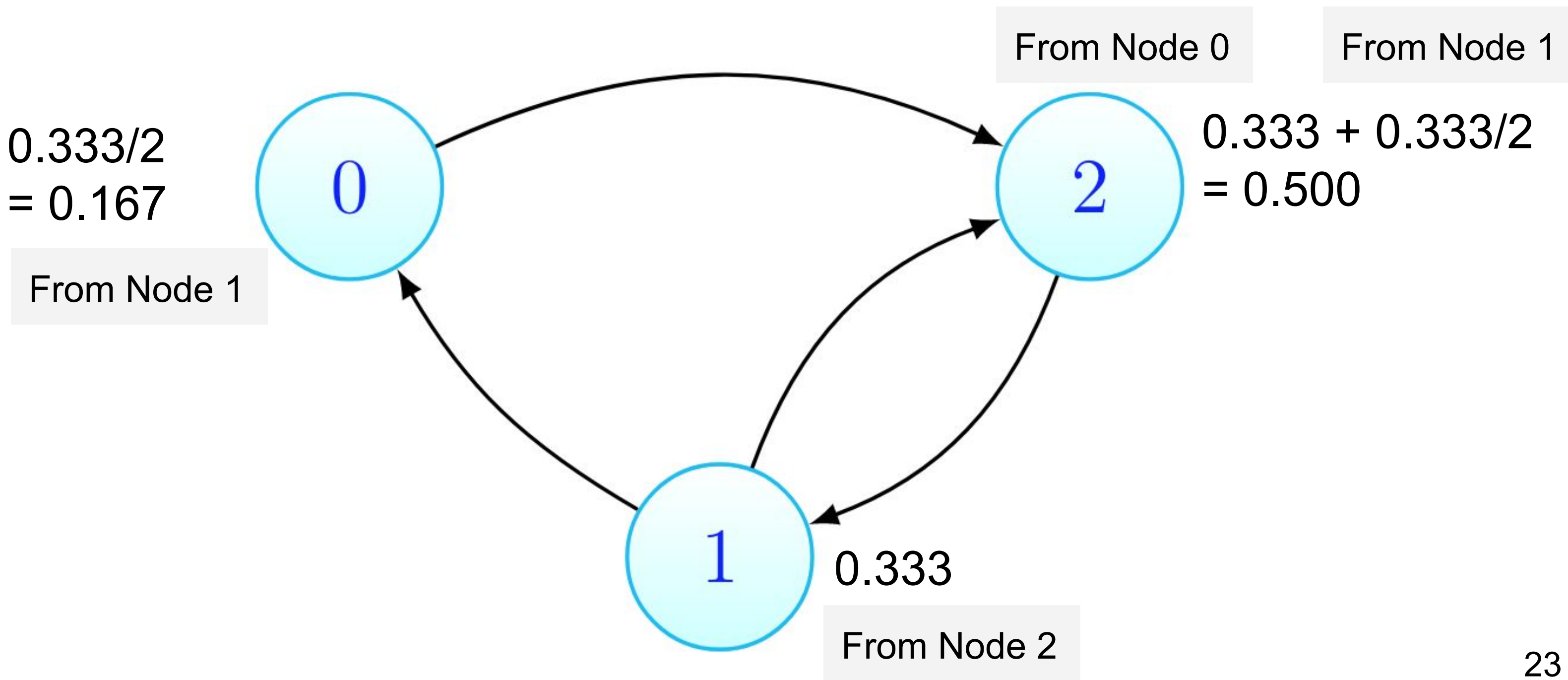
Basic PageRank

- Influence scores are initialized to `1.0 / # of vertices`
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



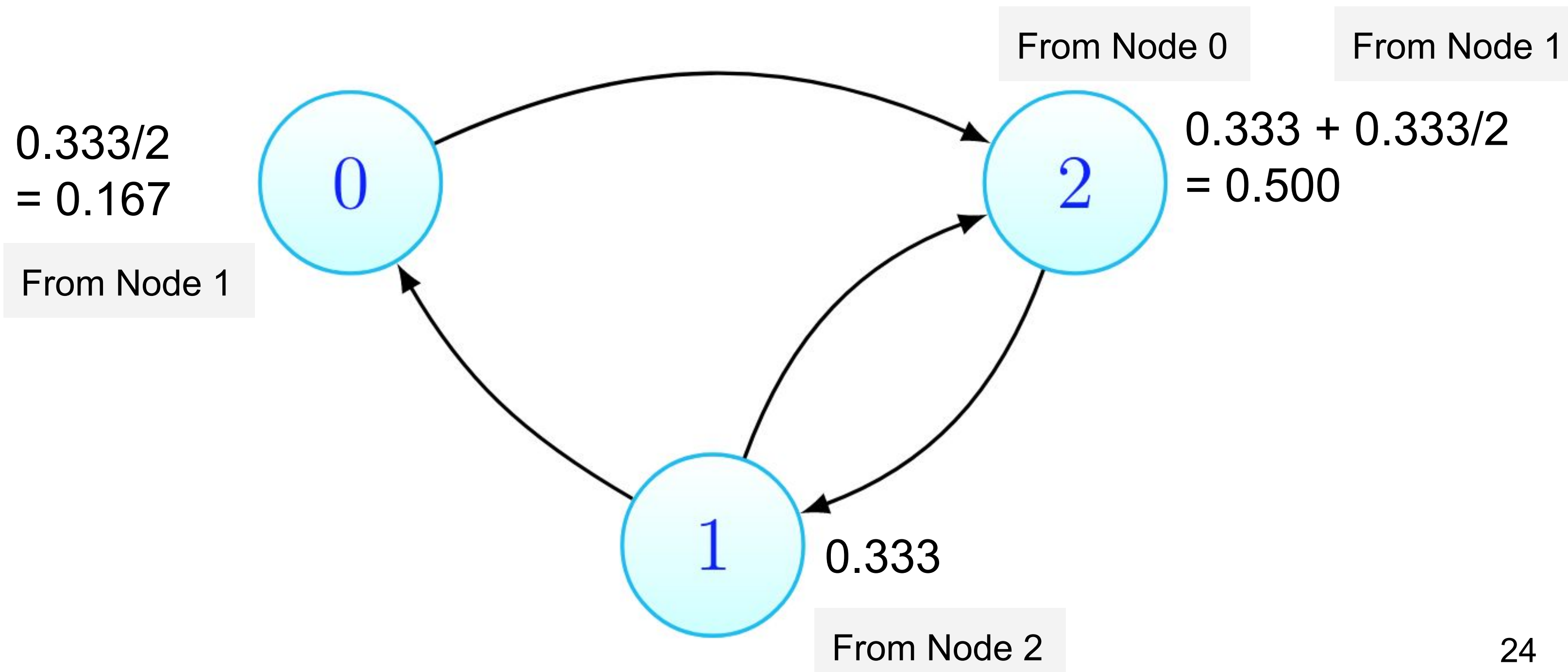
Basic PageRank

- Influence scores are initialized to `1.0 / # of vertices`
- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



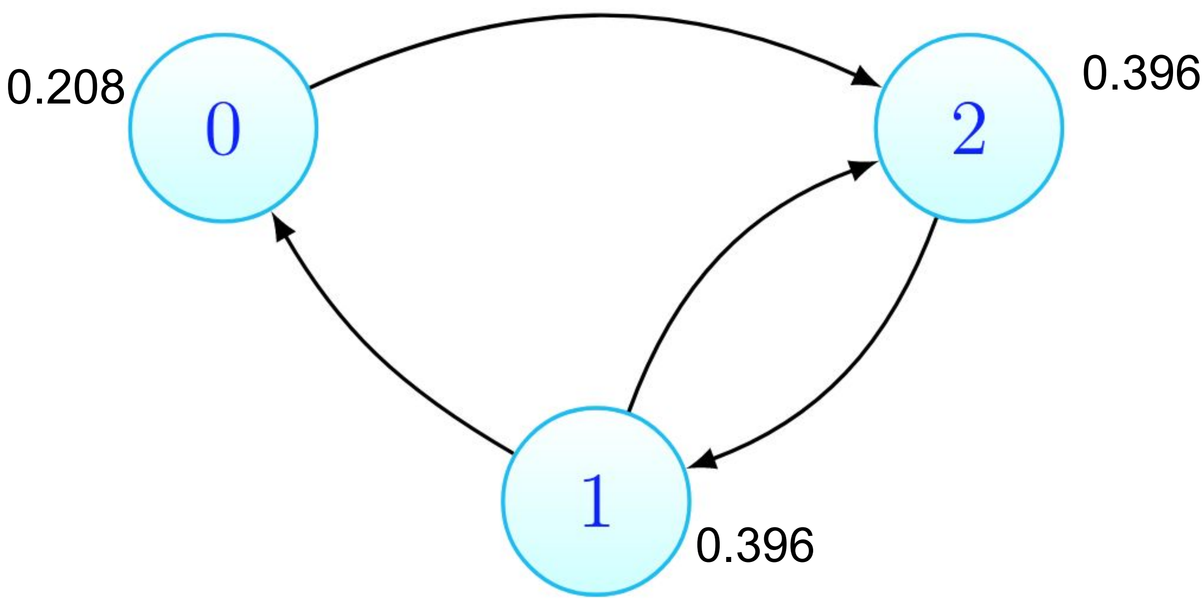
Basic PageRank

- Influence scores are initialized to `1.0 / # 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 / # 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 Pseudocode

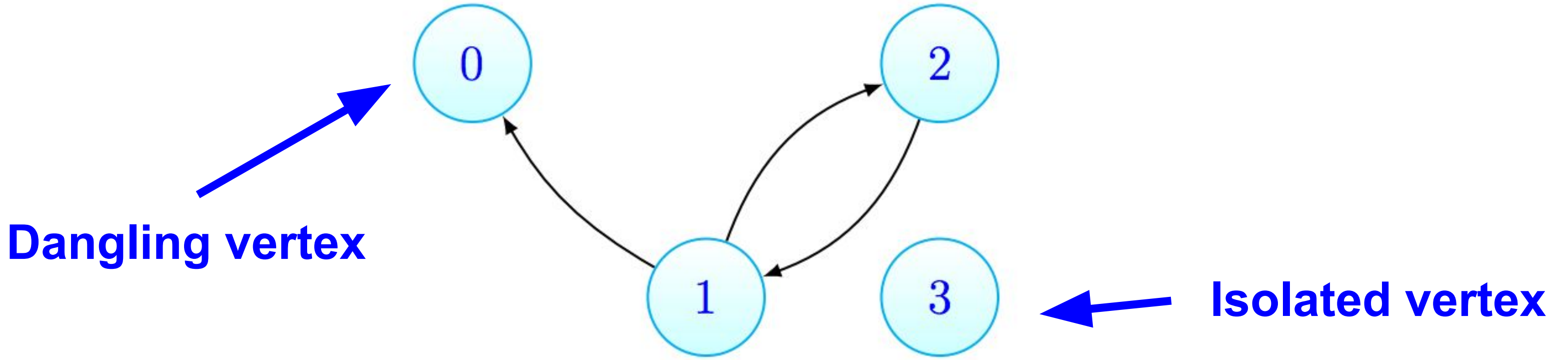
(Note: This does not meet the requirements of Task 2)

```
val links = spark.textFile(...).map(...).persist()
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
    {
        (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }

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

Reference: https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/SparkPageRank.scala

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 $d = 0.85$



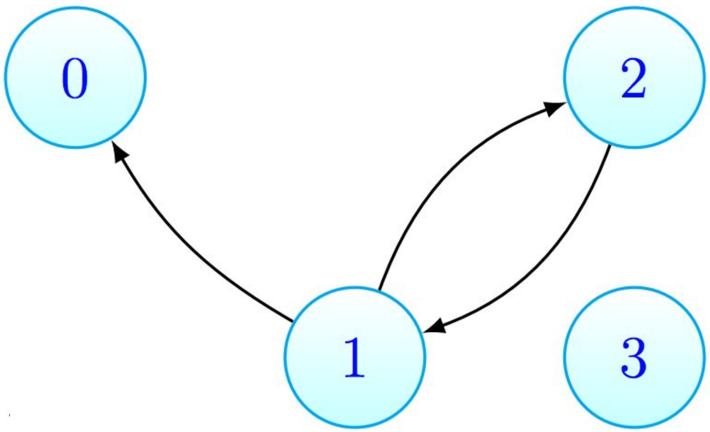
Visualizing Transitions

- Adjacency matrix:

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

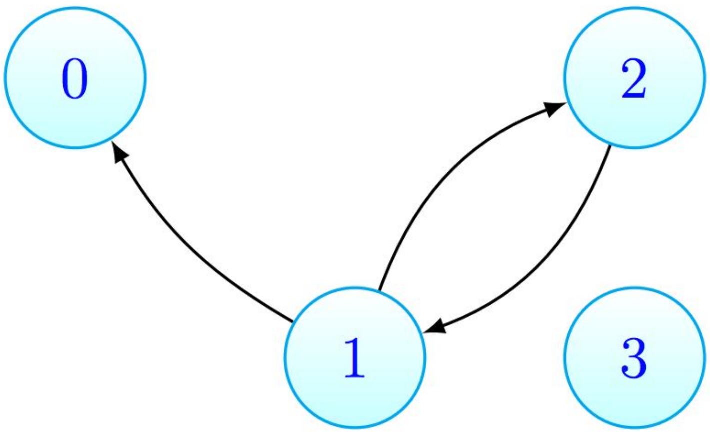
- Transition matrix: (rows sum to 1)

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



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

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

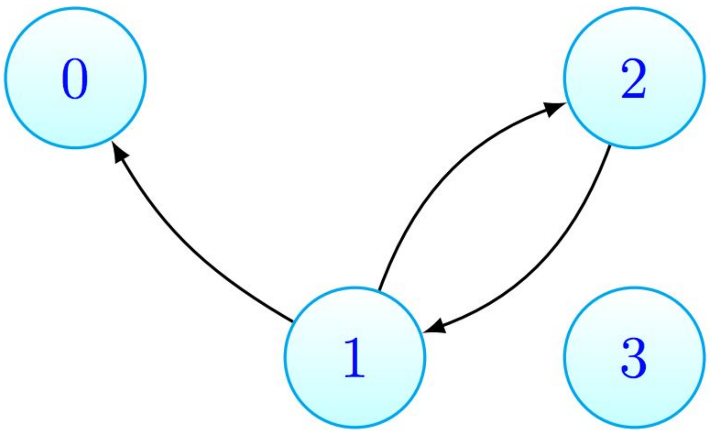
$$r_0^{(1)} = d(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

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

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

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

Task 2: PageRank



Formula for calculating rank

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

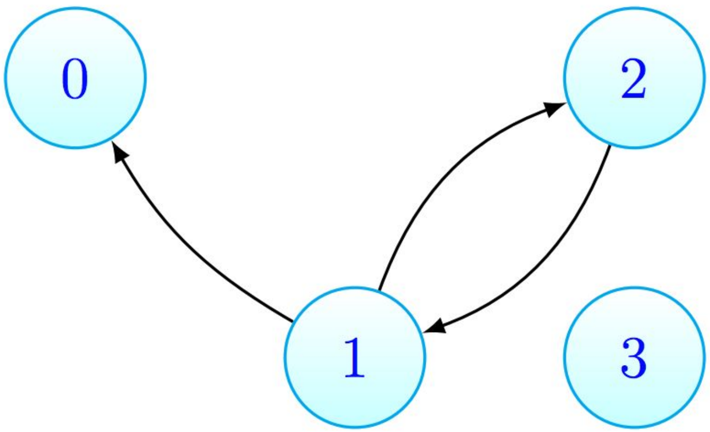
d = 0.85

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



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

This simplifies the formula to

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

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

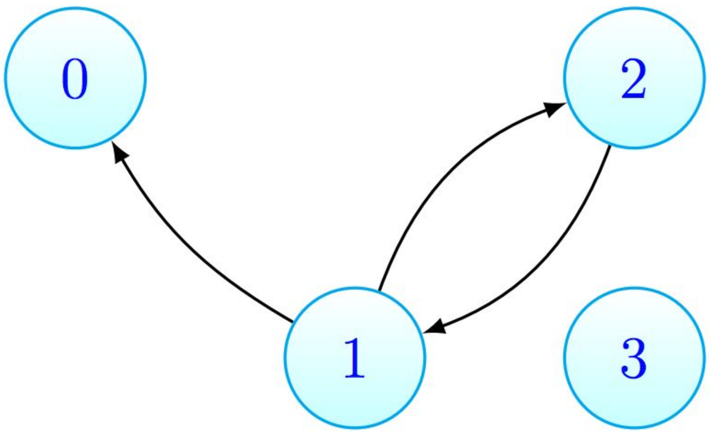
$$r_2^{(1)} = d\frac{r_1^{(0)}}{2} + \epsilon + (1-d)\frac{1}{n}$$

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

Let

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

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

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

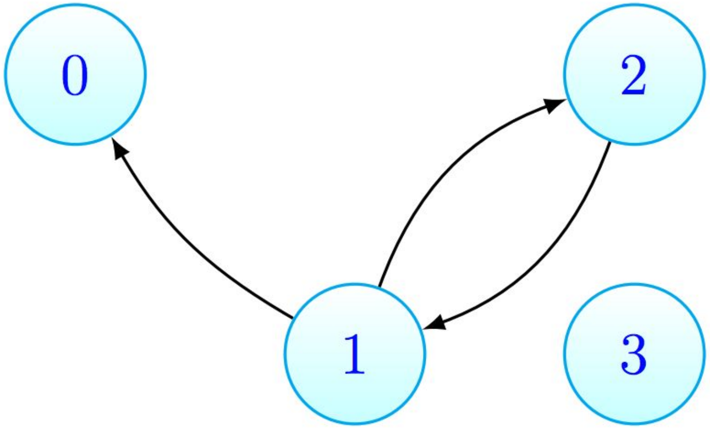
$$r_0^{(1)} = 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25$$

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

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

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

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.
  - It's a **good practice to close SparkSession at the end of application.**
- 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 45 min - 1 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

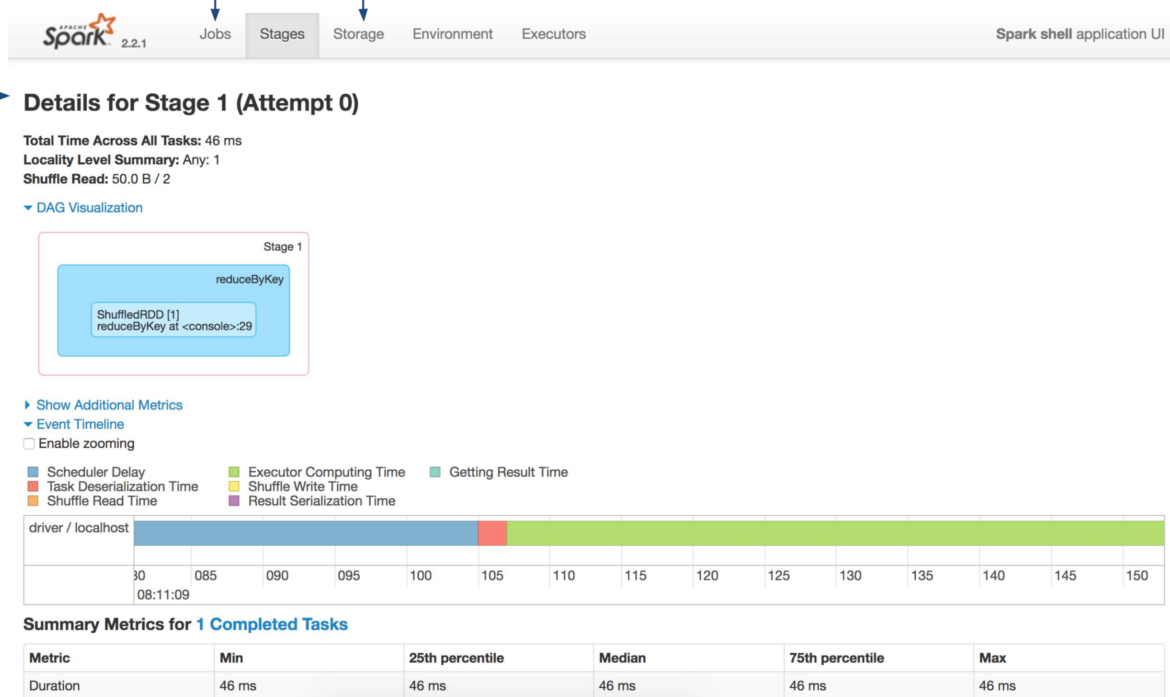
YARN UI

- 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 Yarn 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? (Thrashing)
  - Memory errors - check container logs
  - Parameter tuning applied successfully?
  - Exponential increase in partitions? - Read about HashPartitioner in Spark
- 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?

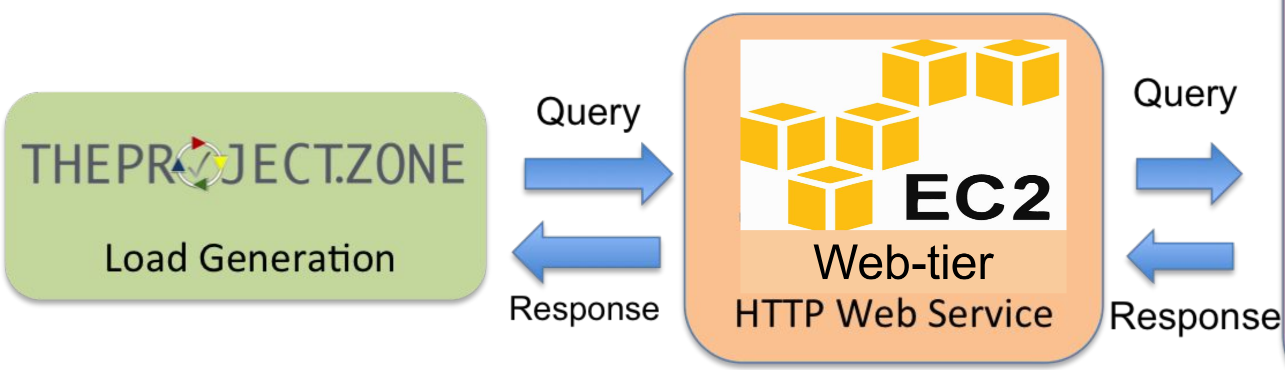
```
object PageRank {
  def calculatePageRank(inputGraphPath: String, outputPath: String, iterations: Int, isLocal: Boolean): Unit = {
    val spark = SparkUtils.getSparkSession(isLocal, appName = "PageRank")
    val sc = spark.sparkContext


    val inputGraph = "wasb://spark@cmuccpublicdatasets.blob.core.windows.net/Graph"
    val outputPath = "wasb:///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)
  }
}
```

TEAM PROJECT
Twitter Data Analytics

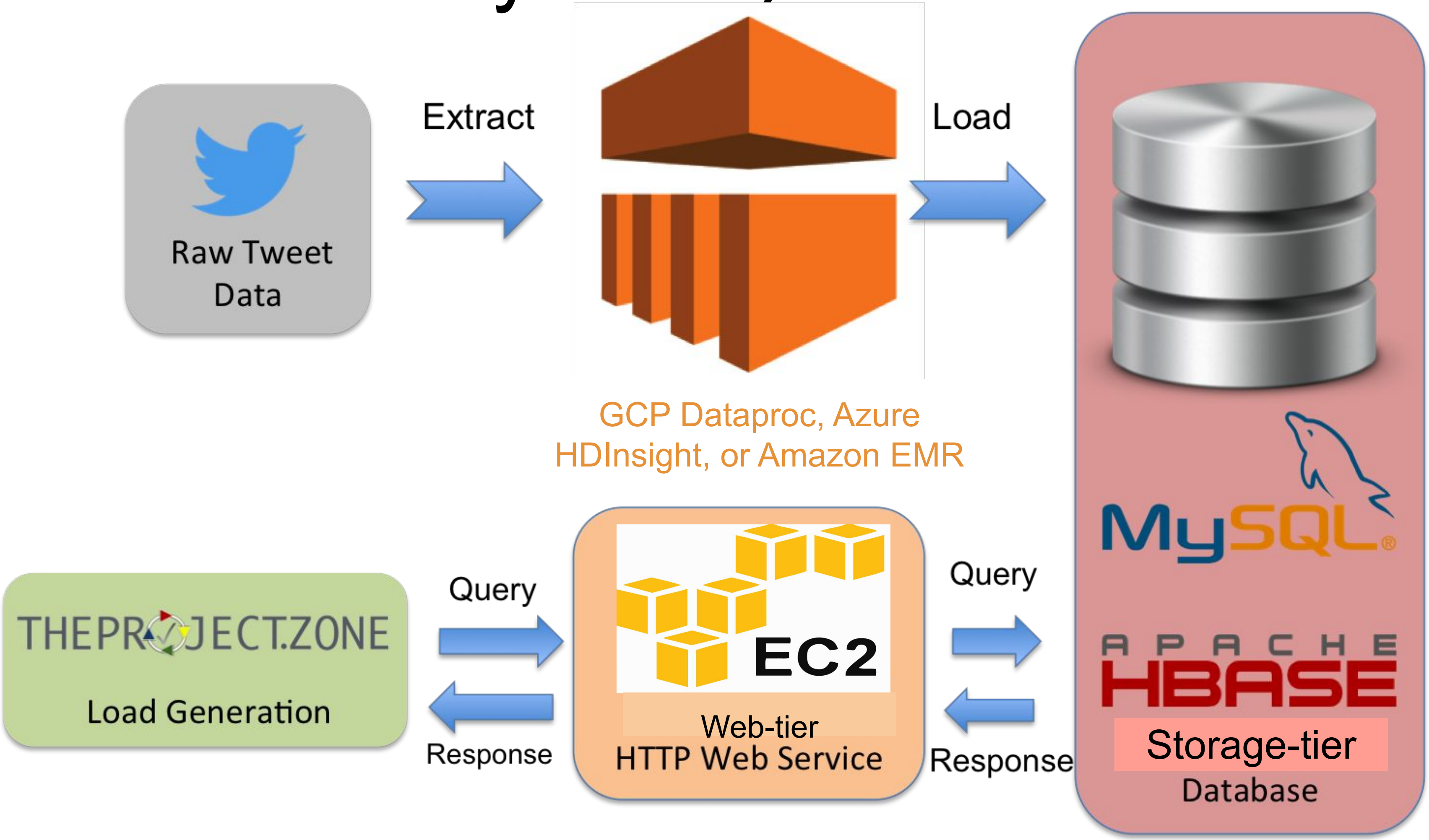
Team Project

Twitter Analytics Web Service

- Given ~1TB of Twitter data
- Build a performant web service to analyze tweets
- Explore web frameworks
- Explore and optimize database systems



Twitter Analytics System Architecture



- Web server architectures
- Dealing with large scale real world tweet data
- HBase and MySQL optimization

Team Project

- Phase 1:
  - Q1
  - Q2 (MySQL **AND** HBase)

- Phase 2
  - Q1
  - Q2 & Q3 (MySQL **AND** HBase)

- Phase 3
  - Q1
  - Q2 & Q3 (MySQL **OR** HBase)

Scoreboard: Phase 1 Phase 1: High Performance Web Service for Data Retrieval

Submitter	Score	Q1 Score (10)	Q1 Effective Throughput	Q1 Checkpoint (5)	Q2 Effective Throughput	Q2 Score (50)	Q2 Checkpoint(10)
zban@andrew.cmu.edu	74	9.14	31992.70	5.00	19418.00	50.00	10.00
MakeTwitterGreatAgain	15	10.00	45309.10	5.00	11749.02	0.00	-
Team Rocket	15	10.00	54620.60	5.00	11309.00	0.00	-
Team Mellon	15	10.00	43387.95	5.00	11004.39	0.00	-
StayUpForCC	15	10.00	37718.70	5.00	10417.34	0.00	-
YiQiGanCC	15	10.00	35997.72	5.00	10135.53	0.00	-
abc123	15	10.00	37137.18	5.00	6647.94	0.00	-
INI OG	15	10.00	36160.00	5.00	6381.85	0.00	-
WGW	15	10.00	41129.47	5.00	6055.08	0.00	-
Tritter	15	10.00	45881.96	5.00	5229.94	0.00	-
BareMetalAlchemist	15	10.00	37226.90	5.00	1042.53	0.00	-
YJZ	15	10.00	41602.20	5.00	203.47	0.00	-
ThreeStrangers	15	10.00	61883.50	5.00	0.00	0.00	-

Team Project Deadlines

- Phase 2 milestones:
  - Phase 2, Live test: on **Sunday, Nov 10**
    - HBase:
      - Q1/Q2/Q3/mixed
    - MySQL:
      - Q1/Q2/Q3/mixed
  - Phase 2, code, scripts and report:
    - due on **Tuesday, Nov 12**

Team Project Time Table

Phase (and query due)	**Start**	**Deadlines**	**Code and Report Due**
Phase 1 ● Q1, Q2	Monday 10/07/2019 00:00:00 ET	Checkpoint 1, Report: Sunday 10/13/2019 23:59:59 ET Checkpoint 2, Q1: Sunday 10/20/2019 23:59:59 ET Phase 1, Q2: Sunday 10/27/2019 23:59:59 ET	Phase 1: Tuesday 10/29/2019 23:59:59 ET
Phase 2 ● Q1, Q2,Q3	**Monday 10/28/2019 00:00:00 ET**	**Sunday 11/10/2019 15:59:59 ET**	
Phase 2 Live Test (Hbase **AND** MySQL) ● Q1, Q2, Q3	**Sunday 11/10/2019 17:00:00 ET**	**Sunday 11/10/2019 23:59:59 ET**	**Tuesday 11/12/2019 23:59:59 ET**
Phase 3 ● Q1, Q2, Q3 (Managed services)	Monday 11/11/2019 00:00:00 ET	Sunday 11/24/2019 15:59:59 ET	
Phase 3 Live Test ● Q1, Q2, Q3 (Managed services)	Sunday 11/24/2019 17:00:00 ET	Sunday 11/24/2019 23:59:59 ET	Tuesday 11/26/2019 23:59:59 ET

Live Test Schedule - setup

Submit DNS for Live Test

Time	Task	Description
4:00 pm	HBase	Submit your DNS for the HBase Live Test before the deadline
4:00 pm	MySQL	Submit your DNS for the MySQL Live Test before the deadline
5:30 pm - 5:31 pm	HBase DNS Validation	Validate your HBase DNS. Last chance to update your DNS for the HBase Live Test
5:33 pm - 5:34 pm	MySQL DNS Validation	Validate your MySQL DNS. Last chance to update your DNS for the MySQL Live Test

Live Test Schedule - HBase

HBase Live Test

Information

Time	Value	Target	Weight
6:00 pm - 6:25 pm	Warm-up (Q1 only)	0	0%
6:25 pm - 6:50 pm	Q1	35000	6%
6:50 pm - 7:15 pm	Q2	10000	10%
7:15 pm - 7:40 pm	Q3	2000	10%
7:40 pm - 8:05 pm	Mixed Reads(Q1,Q2,Q3)	9000/2500/500	4+5+5 = 14%

Half-time Break

Information

Time	Value
8:05 pm - 8:30 pm	Time to relax and prepare for the MySQL Live Test

Live Test Schedule - MySQL

MySQL Live Test

Information

Time	Value	Target	Weight
8:30 pm - 8:55 pm	Warm-up (Q1 only)	0	0%
8:55 pm - 9:20 pm	Q1	35000	6%
9:20 pm - 9:45 pm	Q2	10000	10%
9:45 pm - 10:10 pm	Q3	2000	10%
10:10 pm - 10:35 pm	Mixed Reads(Q1,Q2,Q3)	9000/2500/500	4+5+5 = 14%

Budget Reminder

- Your team has a total AWS budget of **$50** for Phase 2
- Your web service should cost ≤ **$0.89/hour**, including:
  - EC2
    - We evaluate your cost using the On-Demand Pricing towards **$0.89/hour** even if you use spot instances.
  - EBS & ELB
  - Ignore data transfer and EMR cost
- Phase 2 - Live Test Targets:
  - Query 1 - 35000 rps
  - Query 2 - 10000 rps (for both MySQL and HBase)
  - Query 3 - 2000 rps (for both MySQL and HBase)
  - Mixed - 9000/2500/500 rps (for both MySQL and HBase)

Phase 2, Query 3

- **Problem Statement**
  - Given a time range and a user id range, which tweets have the most **impact** and what are the **topic words**?

- Impact score and topic words (see the write up for details)
  - Impact of tweets: Which tweet is “important”? Calculate using the effective word count, favorite count retweet count and follower count.
  - Topic words: In this given range, what words could be viewed as a “topic”? Done using TF-IDF.

- Request/Response Format
  - Request: Time range, uid range, #words, #tweets
  - Response: List of topic words with their topic score, as well as a list of tweets (after censoring)

Phase 2, Query 3 FAQs

<u>Question 1</u>: How to calculate the topic score?

For word ***w*** in the given range of tweets, calculate:

- Calculate the Term Frequency of word *w* in tweet $t^{(i)}$
- Calculate Inverse Document Frequency for word *w*
- Calculate Impact Score of each tweet
- Topic Score for word w =

  $$\sum_{i}^{n} TF(w,\, t^{(i)}) \;\bullet\; IDF(w) \bullet ln(Impact(t^{(i)}) \,+\, 1)\,,$$

  for n tweets in time and uid range

Phase 2, Query 3 FAQs

Question 2: When to censor? When to exclude stop words?

- Censor in the Web Tier or during ETL. It is your own choice.
  - If you censor in ETL, consider the problem it brings to calculating the topic word scores (two different words might look the same after censoring).
- You should count stop words when counting the total words for each tweet in order to calculate the topic score.
- Exclude stop words when calculating the impact score and selecting topic words.

Hints

- Completely understand every AssessMe question
- Completely understand the definition of a **word**. This is different for text censoring and calculating scores.
- A query contains two ranges. Log some requests to get an idea on the range of user_id and timestamps.
- Optimization is time-consuming. Before ETL, please
  - Think about your schema design
  - Think about your database configuration

Hints

- For HBase, you’re not restricted to just 1 master node. The two sample setups below are both permitted.
    - 1 x (1 master + 5 slaves)
    - 2 x (1 master + 2 slaves)
- Understand and keep an eye on
    - **EC2 CPU Credits** and burstable performance
    - **EBS volume I/O Credits** and Burst Performance

EC2 CPU Burst Credits

- One CPU credit is equal to one vCPU running at 100% utilization for one minute.
- Other combinations of number of vCPUs, utilization, and time can also equate to one CPU credit.
- For example, one CPU credit is equal to:
    - one vCPU running at 50% utilization for two minutes, or
    - two vCPUs running at 25% utilization for two minutes.

Hints for the live test

- The request pattern will differ for Phase 2 submission test and the live test so your solution should handle all types of load.
- Monitor your system during the live test to recover in case of a system crash.
- Be prepared with your monitoring consoles setup
- Lookup what commands you can use to learn about the aspects of your web service health.
- Your Phase 2 budget should take into account the cost for the live test.
- Take cloudwatch snapshots

Warning

- NEVER open all ports to the public (0.0.0.0) when using instances on a public cloud.
- For your purposes, you likely only need to open port 80 to the public. Port 22 should be open only to your public IPs.
- Port 3306 (for MySQL) and HBase ports should be open only to cluster members if necessary.

Questions?