COMP 322: Fundamentals of Parallel Programming

Lecture 31: TF-IDF and PageRank Algorithms using Map-Reduce Parallelism

Instructors: Vivek Sarkar, Mack Joyner
Department of Computer Science, Rice University
{vsarkar, mjoyner}@rice.edu

http://comp322.rice.edu/

Worksheet #30 solution: Variant of Word Count

```
1.  JavaRDD<String> file = context.textFile(inputFile);
2.  // Change w.r.t. slide 13: replace s by s.length()
3.  JavaPairRDD<???, Integer> counter =
4.    file.flatMap(s -> Arrays.asList(s.split(" ")))
5.        .mapToPair(s -> new Tuple2<>(s.length(), 1))
6.        .reduceByKey((a, b) -> a + b);

8.  counter.collect().forEach(System.out::println);
```

a) In the space below, indicate what type should be provided instead of ??? in line 3.

Integer

b) Also, explain what this program computes.

The frequencies of word lengths

Background for TF-IDF algorithm (can be implemented as M-R jobs in Hadoop or Spark)

- Goal: Given a document, D_0, find most similar documents in a corpus of documents, D_1, …, D_N
- Approach: model each document as a multiset of terms ("bag of words") and use word frequencies to guide similarity search. Let TERM_1, TERM_2, … represent all the terms across all documents
- Definitions
  - —TF(i,j) = total frequency (count) of TERM_i in document D)j
    - Measure of significant terms in a document
  - —DF(i) = number of documents that contain TERM_i
  - —IDF(i) = N / DF(i)
    - Measure of how common or rare a term is across all documents
  - —Commonly used weight of TERM_i in D_j = TF(i,j) * log (IDF(i))
- See https://en.wikipedia.org/wiki/Tf–idf for more background

Map-Reduce Job 1: Computing TF

- Map task
  - Input: (D_i, TERM_j) pairs for all terms in documents (including duplicates)
  - Output: ((D_i, TERM_j), 1) for each input pair
- Reduce task
  - Use SUM as reduce operator
  - Outputs ((D_i, TERM_j), TF(i,j))

Map-Reduce Job 2: Computing DF

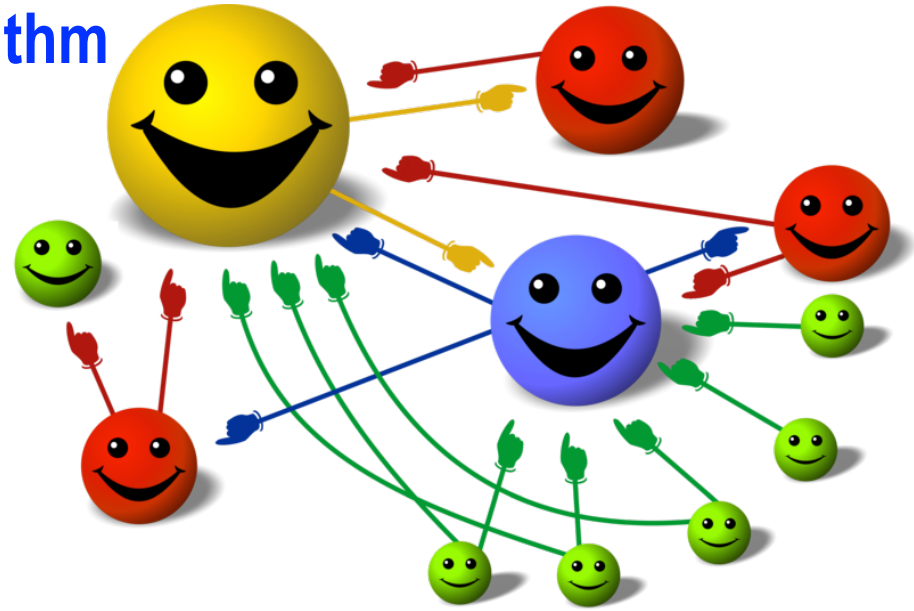
- Map task
  - Input: (D_i, TERM_j) pairs for all terms in documents (without duplicates)
  - Output: (TERM_j, 1) for each occurrence of TERM_j in a document
- Reduce task
  - Use SUM as reduce operator
  - Outputs (TERM_j, DF(j))
- IDF can be easily computed from DF using a map task

Background for PageRank algorithm and its implementation in Spark

- Give pages ranks (scores) based on links to them
  - —Links from many pages → high rank
  - —Link from a high-rank page → high rank

- Needs an iterative map-reduce algorithm

- Good match for Spark's in-memory processing capabilities



Acknowledgment: slides for this topic were taken from "Parallel Programming With Spark" lecture by Prof. Matei Zaharia, Stanford University

Algorithm

Start each page at a rank of 1

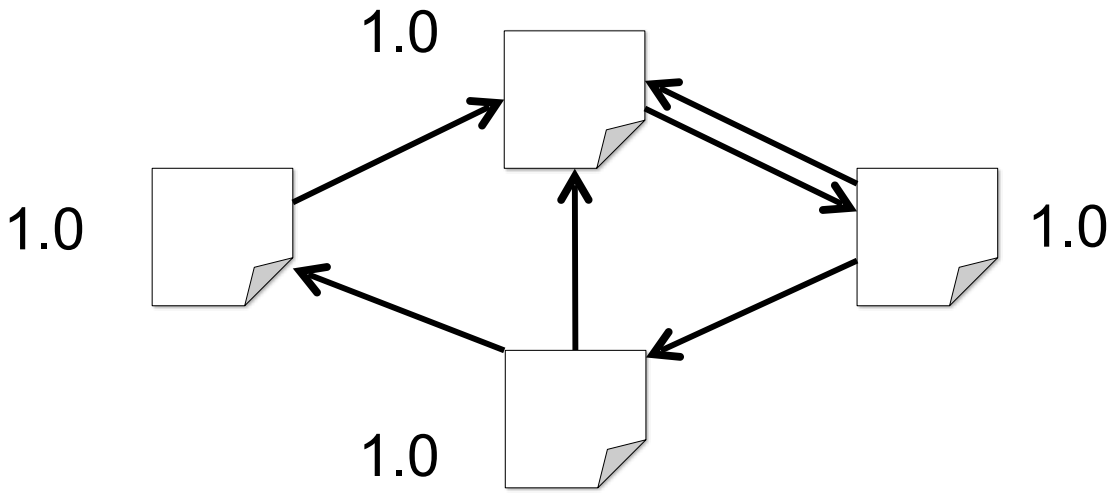
FOR (iter = …) {

1. On each iteration, have each page A *contribute* to the rank of B when there is a link (edge) from A to B

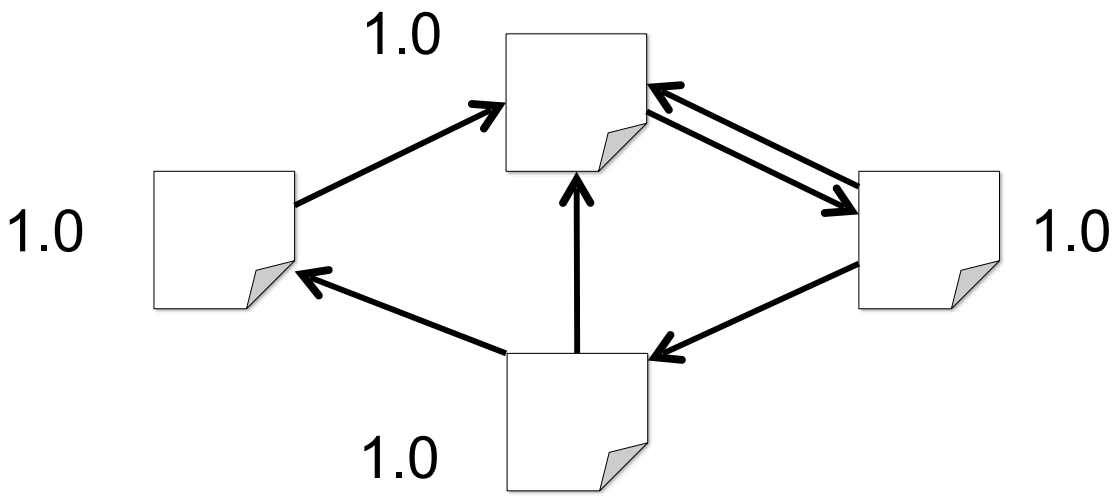
   CONTRIBS(B) += RANK(A) / DEST_COUNT(A)

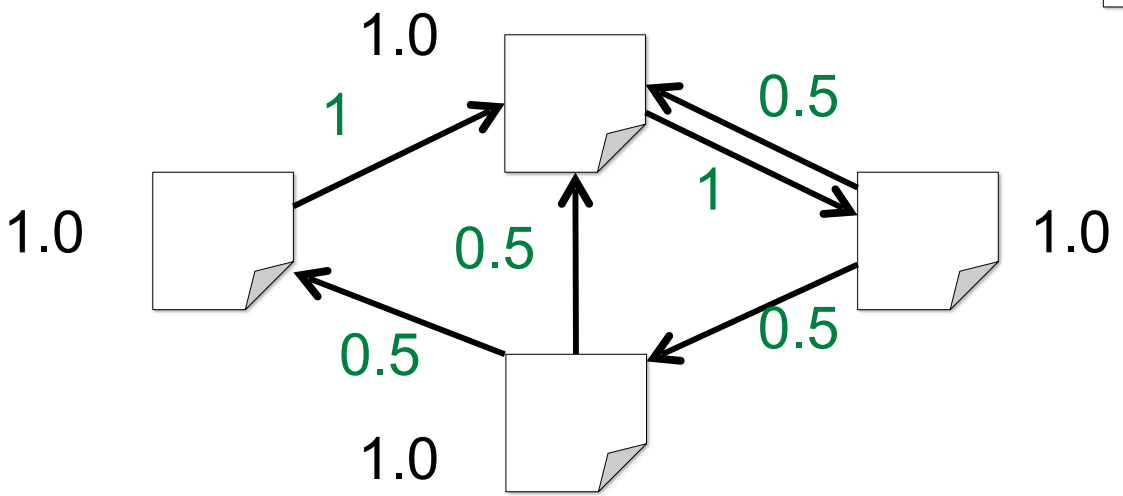
2. Update all page ranks to RANK(B) = 0.15 + 0.85 × CONTRIBS(B)

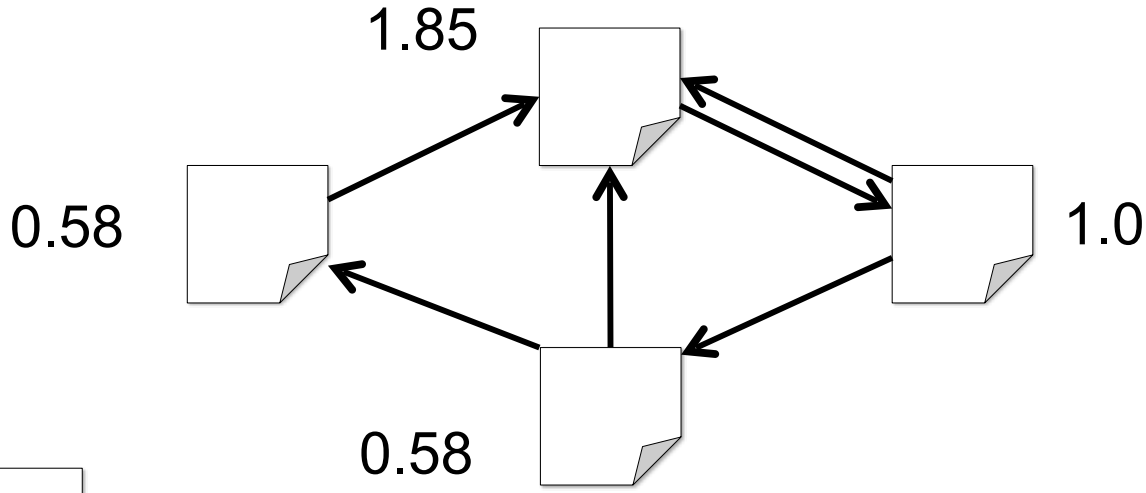
}



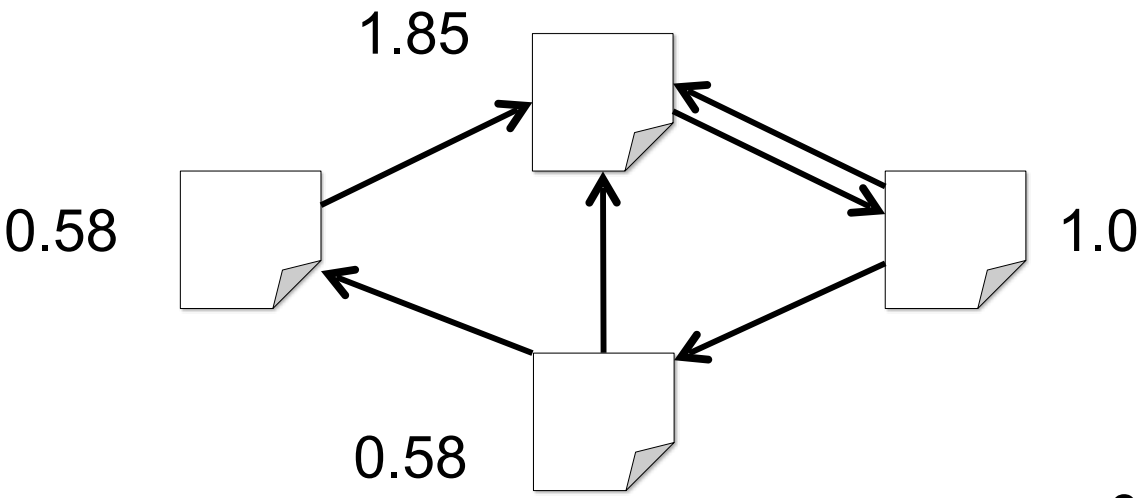
Example: First Iteration

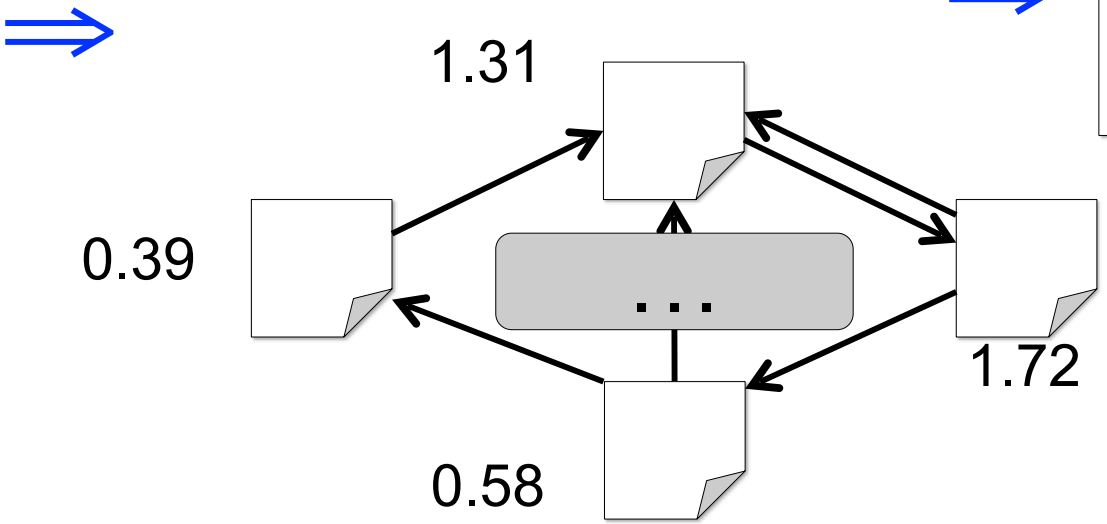




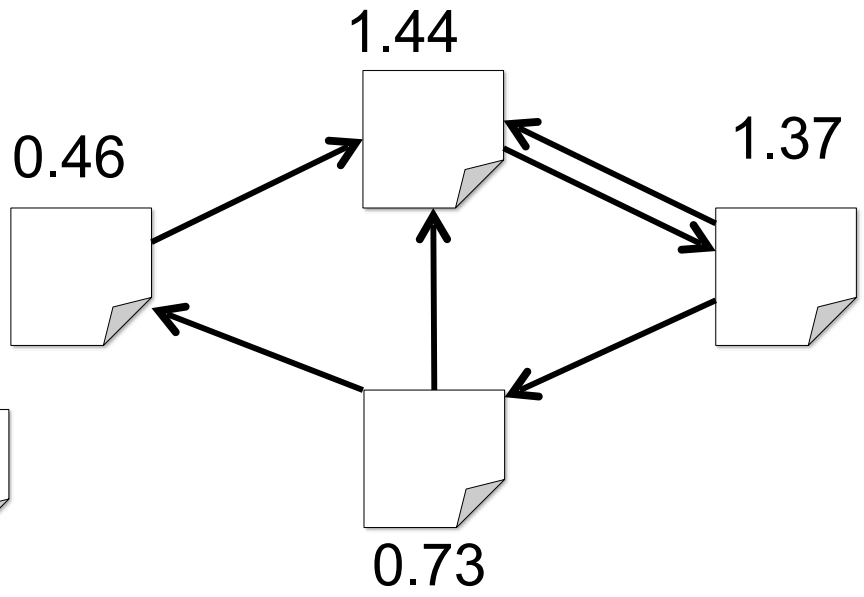


Example: Successive Iterations





Final state:



Scala 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 {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```


Announcements and Reminders

- Checkpoint-1 for Homework 4 is due by 11:59pm this Wednesday (April 5th)
- There will be a lab this week (at 7pm on Wednesday) as usual
- Quiz for Unit 8 is due by April 7th
- Quiz for Unit 9 is due by April 14th (last quiz!)
- Final exam (Exam 2) is scheduled at 9am - 12noon on Tuesday, May 2nd (scope of exam is limited to lectures 18 - 38)