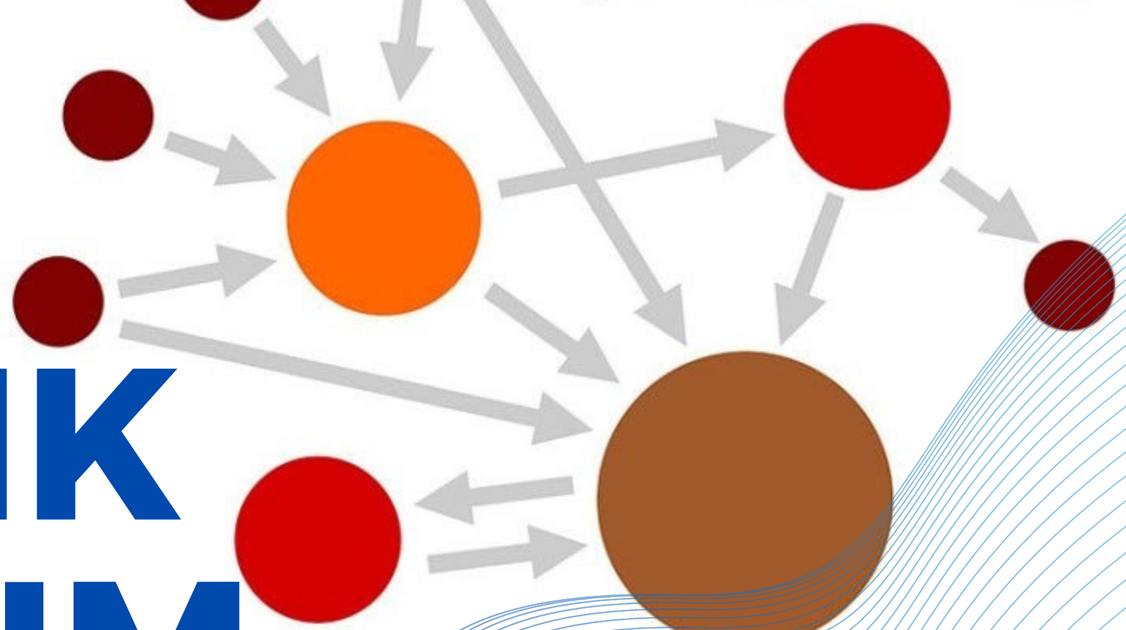


PAGERANK ALGORITHM ON DISTRIBUTED SYSTEMS

Team 19

Parag Sharma - 202103004

Parv Agarwal - 202103010

Darshak Zankat - 202103041

MOTIVATION

This project on implementing the PageRank algorithm in distributed computing using MR Job and Spark offers a unique blend of academic exploration and practical relevance by tackling the scalability challenges of large-scale networks and providing insights into distributed computing frameworks, it not only advances algorithmic research but also holds significance for industries relying on efficient graph processing with applications in web search.

OBJECTIVE

- Integration of PageRank into Distributed Systems
- Iterative computation and addressing convergence challenges
- Comparison of MR Job and Spark frameworks
- Study of varying damping factor

PageRank Introduction

$$P = \left(\frac{\epsilon}{n}\right) J + (1 - \epsilon)Q$$

- Graph Representation:** G = (V, E)
- Random Walk:** A specialized random walk on the graph.
- Reset Probability:** ε (constant) - probability of starting the random walk from a random node.
- PageRank Vector:** Stationary distribution vector π from the random walk.
- Random Walk Steps:**
- With probability ε, start from a randomly chosen node.
- With probability 1-ε, move to a randomly selected outgoing edge/neighbor.
- PageRank Transition Matrix (State Space V):**
- P = (ε/n) J + (1-ε) Q
- J: Matrix with all entries being 1.
- Q: Transition matrix of a simple random walk on G.

PAGERANK MASS

Conceptualization:

- Probabilistic distribution reflecting the transfer of probability mass through outbound connections.
- Nodes recalculate PageRank by amalgamating contributions from neighboring nodes.
- Iterative process: Diffusion of PageRank mass along outgoing edges.

Mapper to Reducer Transition:

- Careful consolidation of emitted PageRank mass during transition from mappers to reducers.
- Cumulative sum of updated PageRank values in reducers must form a valid probability distribution.
- Emission includes values and node IDs for respective neighbors.

Handling Dangling Nodes:

- Challenge in conserving total PageRank mass with graphs containing dangling nodes.
- Dangling nodes pose a challenge during summation.
- Solution involves equitable redistribution of "lost" PageRank mass from dangling nodes to all graph nodes.

-Addressing Challenges:

- Various methodologies available for allocating absent PageRank mass due to dangling nodes.
- Ensuring conservation of total PageRank mass is crucial for algorithm accuracy.

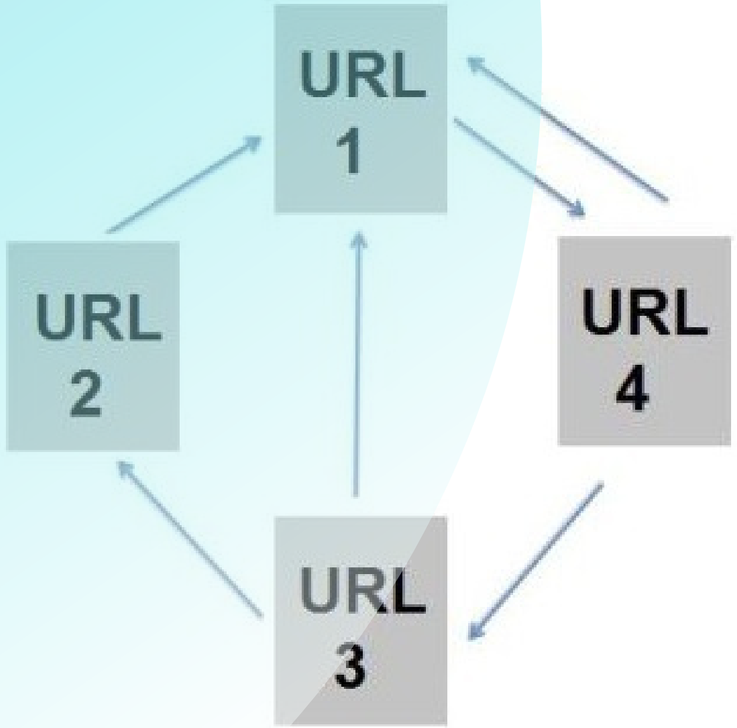
CONTENT SUMMARY OF THE STUDY

Algorithm 1 PageRank MapReduce Algorithm

```
1: class Mapper
2:     method Map(nid n, node N)
3:     p ← N.PageRank / |N.AdjacencyList|
4:     Emit(nid n, N)                          ▷ Pass along graph structure
5:     for all nodeid m ∈ N.AdjacencyList do
6:         Emit(nid m, p)                      ▷ Pass PageRank mass to neighbors
7:     end for
8: class Reducer
9:     method Reduce(nid m, [p1, p2, ...])
10:    M ← ∅
11:    for all p ∈ counts [p1, p2, ...] do
12:        if IsNode(p) then
13:            M ← p                           ▷ Recover graph structure
14:        else
15:            s ← s + p                       ▷ Sum incoming PageRank contributions
16:            M.PageRank ← s
17:            Emit(nid m, node M)
18:        end if
19:    end for
```

PAGERANK IN SPARK

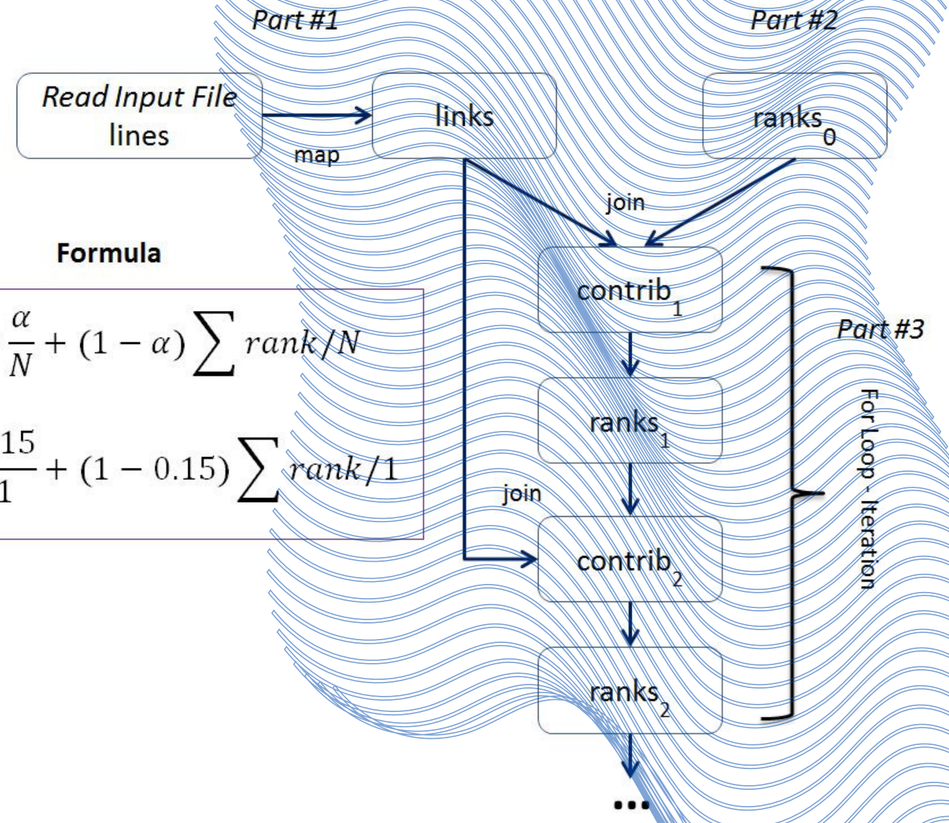
url_1	url_4
url_2	url_1
url_3	url_2
url_3	url_1
url_4	url_3
url_4	url_1



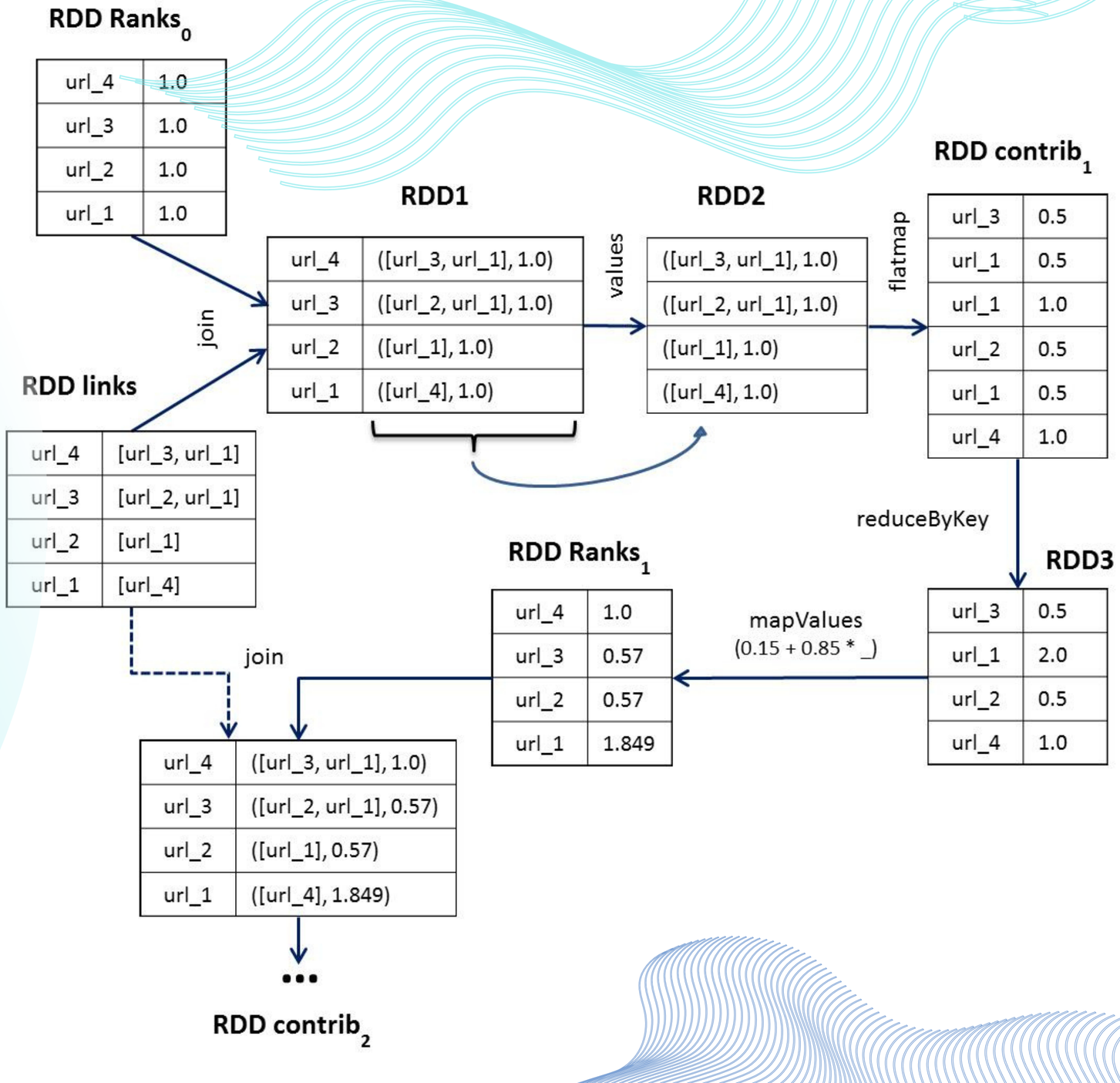
Formula

$$\text{rank} = \frac{\alpha}{N} + (1-\alpha)\sum rank/N$$

$$\text{rank} = \frac{0.15}{1} + (1-0.15)\sum rank/1$$



Demonstration



SPARK ALGORITHM

Algorithm 1 Page Rank Algorithm

1: **Input:** Links RDD, ITERATIONS
2: **Output:** Updated ranks RDD
3: *links* ← `spark.textFile(...).map(...).persist()`
4: *ranks* ← *// RDD of (URL, rank) pairs*
5: **for** i **in** 1 **to** $ITERATIONS$ **do**
6: *// Build an RDD of (targetURL, float) pairs*
7: *// with the contributions sent by each page*
8: $contribs \leftarrow links.join(ranks).flatMap\{(url, (links, rank)) \Rightarrow$
9: $links.map(dest \Rightarrow (dest, rank/links.size))\}$
10: *// Sum contributions by URL and get new ranks*
11: $ranks \leftarrow contribs.reduceByKey((x, y) \Rightarrow x + y)$
12: **end for**

LEARNING OUTCOMES

We successfully implemented a PageRank random walk with Markovian properties using MR Job and Spark as our primary goal. Additionally, we achieved our secondary goal by implementing a Monte Carlo simulation based on random walks for PageRank.

1 Convergence Dynamics

- PageRank converged with more iterations, plateauing after a point.
- Additional iterations had minimal impact on PageRank scores.

2 MR Job vs. Spark

- Consistent PageRank results in both MR Job and Spark validated algorithm reliability
- Spark outperformed MR Job in execution times, especially with larger datasets

3 Damping Factor Impact

- Damping factor 0 led to uniform PageRank scores, emphasizing random jumps.
- Higher damping reduced random jump impact, emphasizing random walks for convergence.

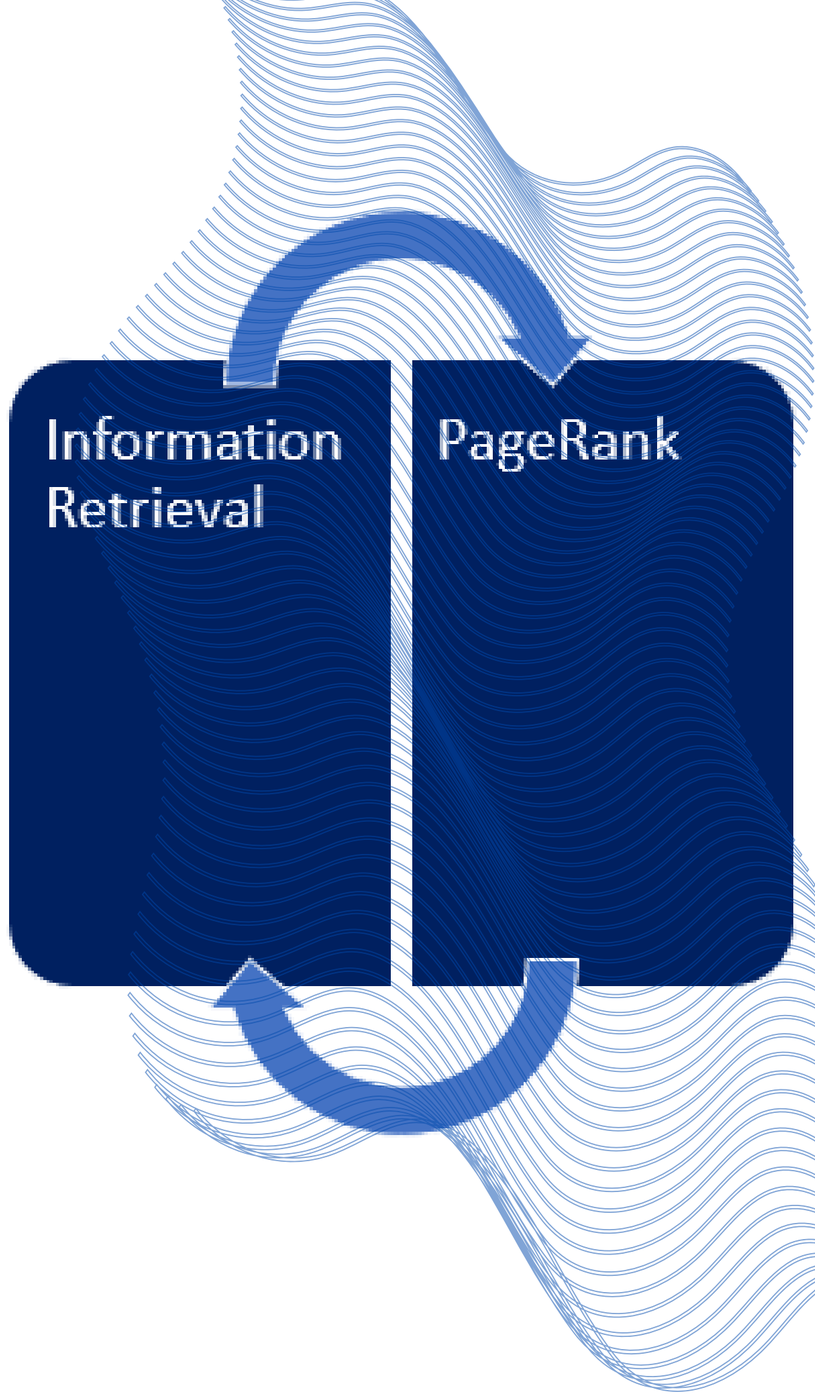
3 Dangling Nodes Treatment

- Dangling nodes handled by redistributing lost PageRank mass
- Ensured conservation of total PageRank mass within the graph.

FUTURE WORK

Thus far we have discussed the PageRank computation with a teleport operation in which the surfer jumps to a random web page chosen uniformly at random. We now consider teleporting to a random web page chosen non-uniformly. In doing so, we are able to derive PageRank values tailored to particular interests.

For instance, a sports aficionado might wish that pages on sports be ranked higher than non-sports pages.



REFERENCES

https://arxiv.org/pdf/1208.3071.pdf

(PDF) Application of Markov Chain in the PageRank Algorithm

PDF | Link analysis algorithms for Web search engines determine the importance and relevance of Web pages. Among the link analysis algorithms, PageRank... | Find, read and cite all the research you need on ResearchGate

ResearchGate