Using MapReduce to Calculate PageRank

Group #3 : Ravish, Sourabh, Xialin Yan

Agenda

- Project Objective
- Hadoop, MapReduce and PageRank
- Apache Giraph
- Experiments
- Results
- Discussion

Project Objective

- Gaining working experience with hadoop and MapReduce.

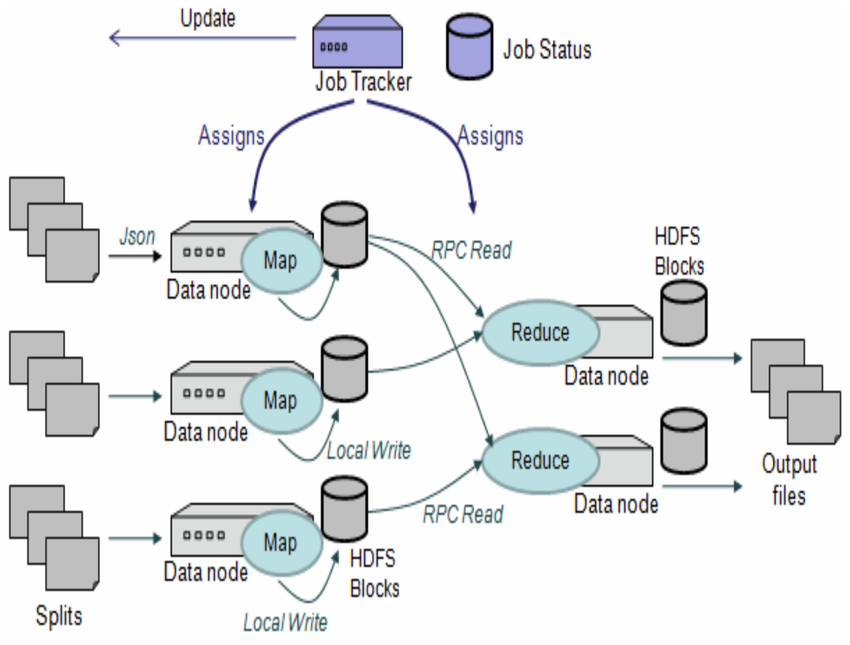
- Understanding graph algorithms such as PageRank and implementation using MapReduce.

- Running PageRank using Apache Giraph, and other algorithms that rely on MapReduce.

- Comparing results and performance between direct implementation in MapReduce vs Giraph.

Hadoop

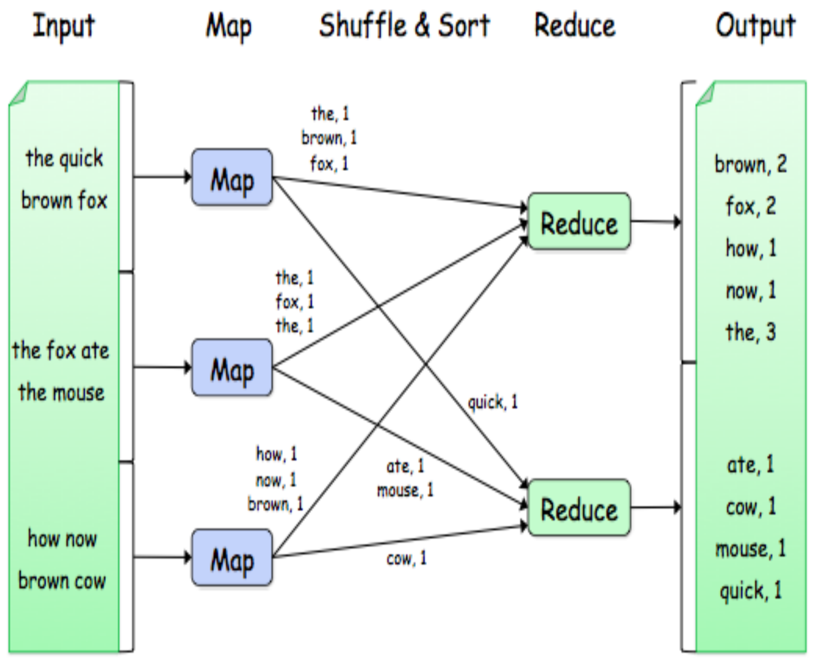
An apache open source project. It is an application framework that helps integrate HDFS (hadoop distributed file system) and run MapReduce jobs.



MapReduce

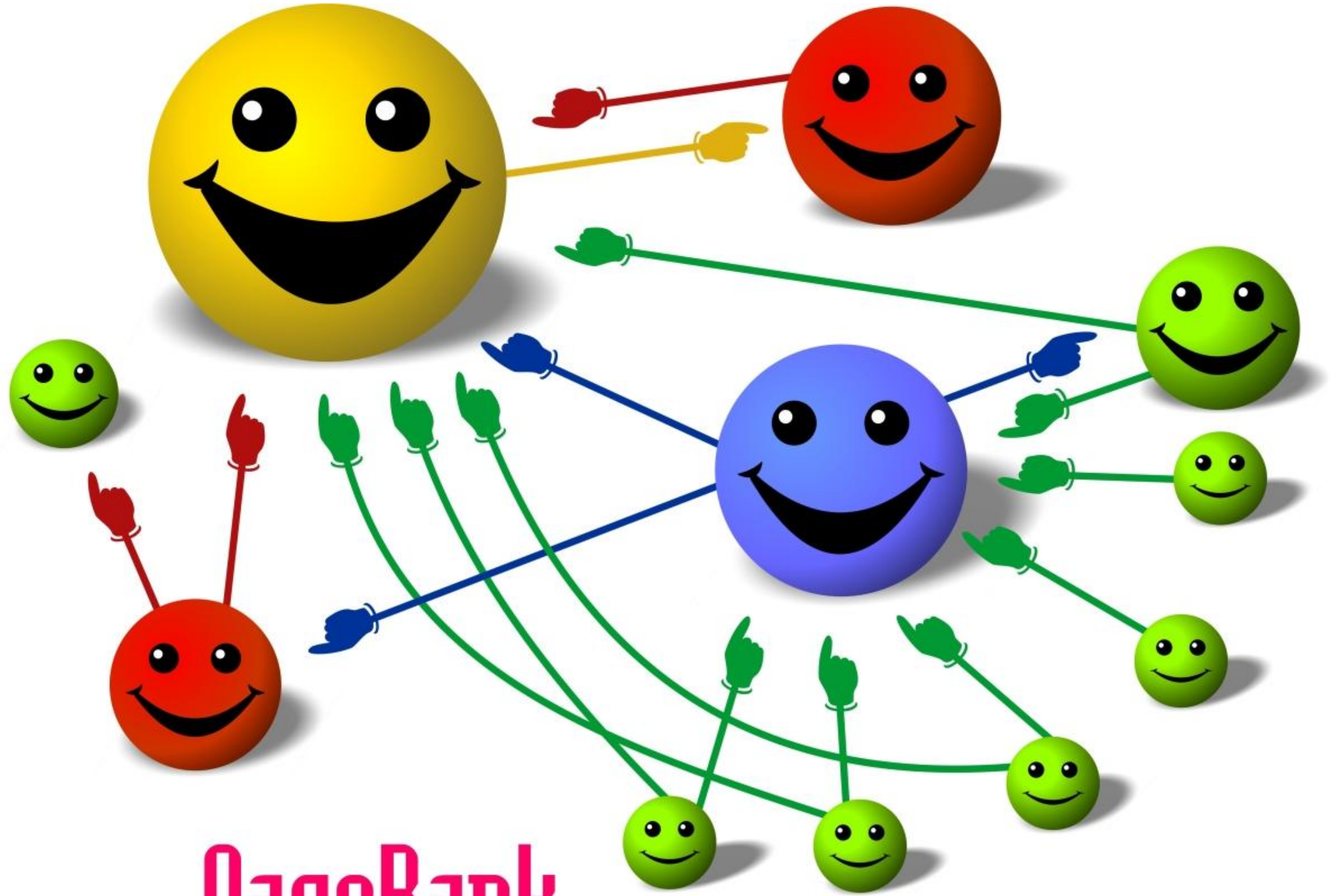
A computational technique of dividing up application into small independent tasks which can be executed on any node in the cluster.

Map -> Shuffle -> Reduce



What is PageRank?

- Measures the relative importance of a hyperlinked set of documents (web-pages)
- Assumes that more important websites will have more links from other websites
- Each web-page is a node and the hyperlink between web-pages are edges
- A hyperlink to a page is a count of support with a weighted value based on importance



PageRank

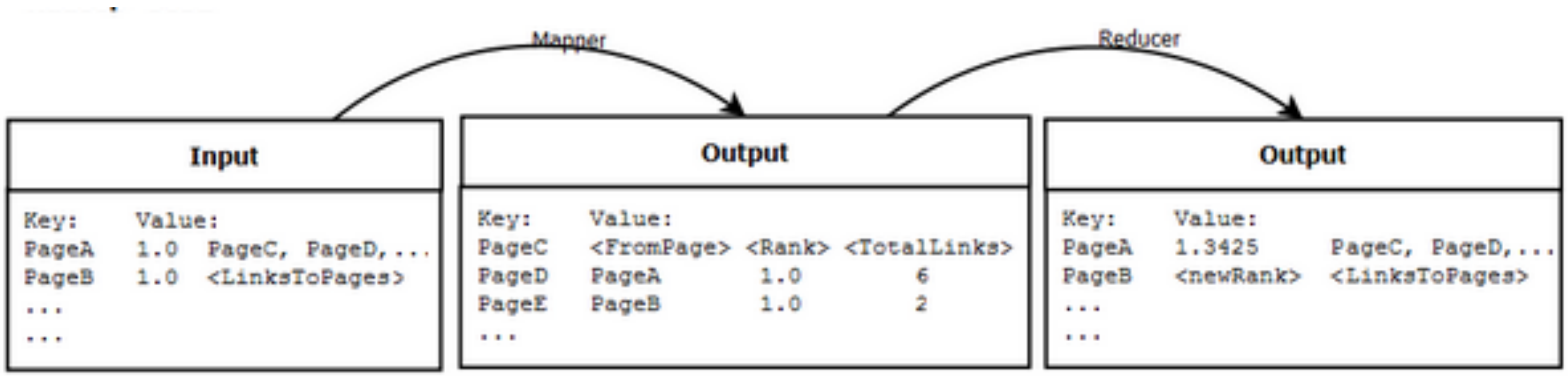
PageRank Algorithm

- Start with uniform initialization of all pages
- Simple Algorithm: $PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$
- Damping Factor : $PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$ (d=0.85)

where d is the damping factor (0.85), $M(p_i)$ is the set of pages linking to p_i and $L(p_j)$ is the cardinality of p_j.

- Iterate until convergence or for a fixed number of iterations.

PageRank using MapReduce



Input

```
Key:     Value:
PageA    1.0   PageC, PageD,...
PageB    1.0   <LinksToPages>
...
...
```

Output

```
Key:     Value:
PageC    <FromPage> <Rank> <TotalLinks>
PageD    PageA         1.0         6
PageE    PageB         1.0         2
...
```

Output

```
Key:     Value:
PageA    1.3425      PageC, PageD,...
PageB    <newRank>   <LinksToPages>
...
...
```

Implementation

Job 1: Parse the data

- XML for wikipedia and text for Twitter Memetracker

Job 2: PageRank MapReduce job

- Complete 15 iterations or till convergence of output

Job 3: Sort the pages based on PageRank

Apache Giraph

Apache Giraph is an iterative graph processing system built for high scalability. It is currently used at Facebook to analyze the social graph formed by users and their connections.

PageRank can be implemented using Giraph due to the message passing and is efficient because of graph processing optimization.

Result (English Wikipedia)

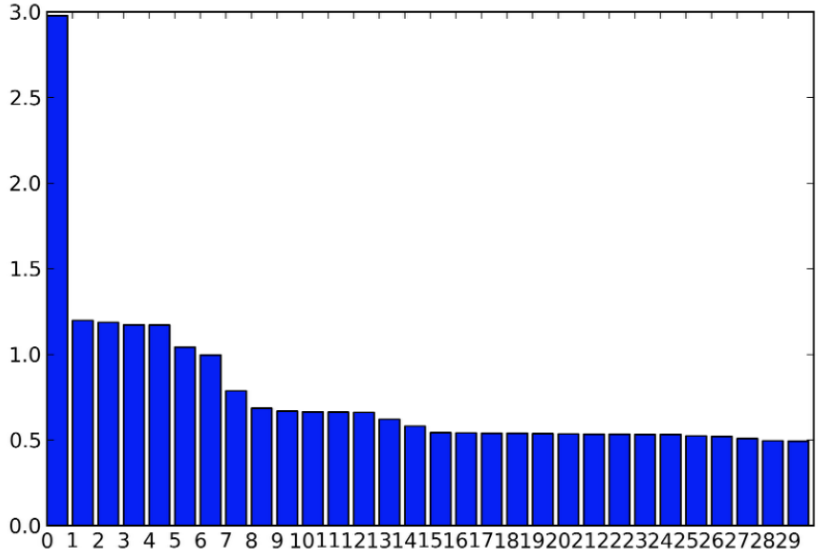
- Tests conducted on the english wikipedia dataset.
- In the top 30 pages (all same), only two positions did not match between the results from our implementation and Giraph.

Computation Time	: Giraph is 10x faster than our MR.
Average Error	: 19.842
Kendall Tau	: 0.9479

Result (English Wikipedia)

Top Pages:

1. United States
2. Genomics Institute of the Novartis Research Foundation
3. Category:Living people
4. England
5. Canada
6. France
7. United Kingdom
8. Germany
9. India
10. Australia



Result (Memetracker)

- Memetracker dataset from : http://www.memetracker.org/

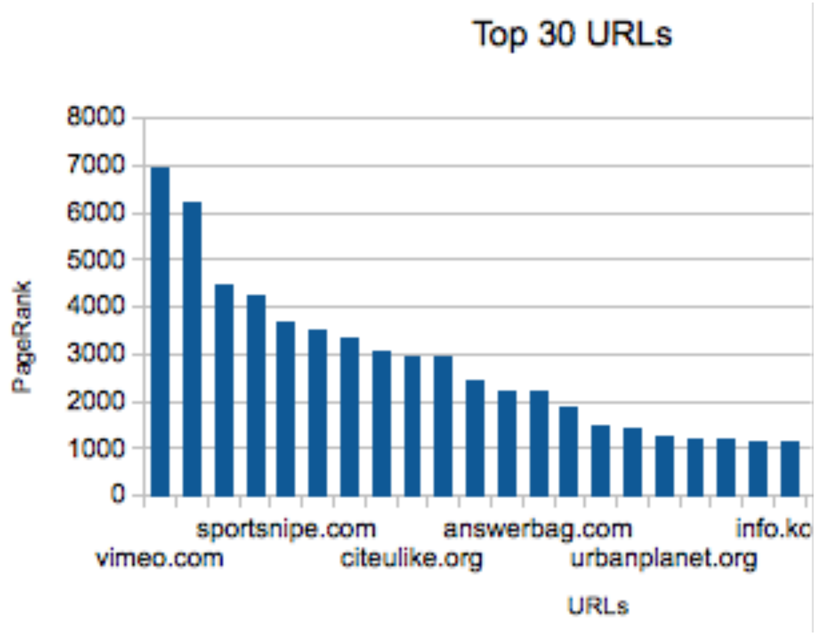
- Top URL : vimeo.com
- Bottom URL : uscapital.wordpress.com

- Blogs with many wordpress URLs are among the lowest ranked websites
- PageRank value decays really fast over webpages.

Result (Memetracker)

Top URLs:

1. Vimeo.com
2. slideshare.net
3. answerbag.com
4. news.chosun.com
5. cnn.com
6. rss.feedportal.com
7. uk.news.yahoo.com



Giraph Evaluation

- Giraph is significantly faster in comparison to our implementation of PageRank on vanilla hadoop.
- Memory requirements by Giraph are significantly higher.
- Bugs in Giraph create issue with Text class from Hadoop.
- Heavy data cleaning was required for Giraph.
- Results from the two implementations are very similar as Kendall Tau observed is 0.9479 for wikipedia.

Memetracker Exploration

- Ran different algorithms on the Memetracker dataset
  - Word Count
  - Phrase Count
- Explored the dataset for interesting findings.

Word Count and Phrase Count

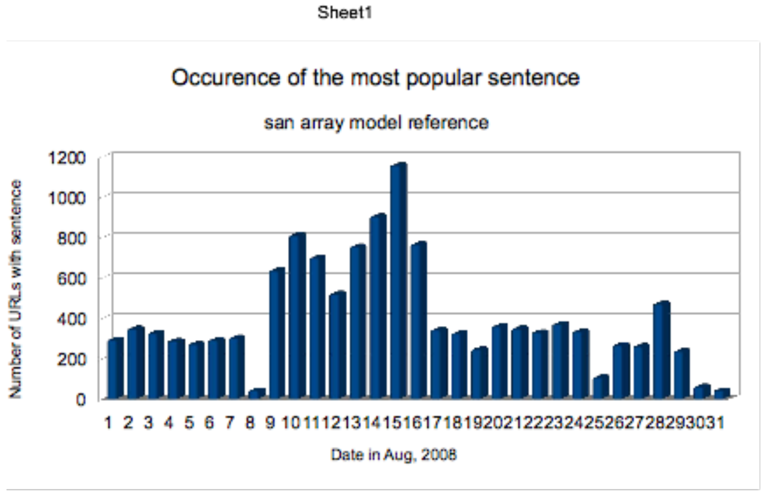
Algorithm:

- Read the data line by line
- Tokenize the sentence.
- Mapper emit (Word / Phrase , 1).
- Reducer emits (Word / Phrase, sum of input values).

Sentence count

- The most popular sentence by count was: “San array model reference”.

- Second most popular was “Gang of ten”



Sentence Count Evaluation

- On close observation we realize the most popular phrase “San array model reference” was only present in one website and it just self cites it to increase the count.
- Gang of ten for the New Reforms was announced in Aug10 which is the time duration for our analysis.

Areas for Optimization

Current Implementation	Solution
PageRank is called one after another when run	Synchronize the jobs so that as a current job finishes reducing its data, the second job can start on mapping its data. This reduces the total effective running time
Naïve iterative PageRank algorithm	Do not map node to itself Take into account factors such as damping factor Give more weights to credible websites
Large intermediate results of output cannot be committed to disk by reducer	Reduce the partition sizes of output

What we learned from the project

- How Hadoop works, and how HDFS is structured. The data manipulated was stored, pulled, and restored directly into HDFS, so we learned commands and how certain things can be implemented in Hadoop (such as specifying our own Mapper and Reducer classes to run on datasets).
- How a single node cluster, and multi node clusters (such as EmuLab) are set up.
- How MapReduce works in Hadoop, the general implementation of Mapper and Reducer classes, and how each can be used to aggregate data and organize it using specific functions.
- How general algorithms can be run on datasets, with specified functions for mapping the data and reducing it. Different algorithms require different implementations, such as differences in implementations of the Reducer classes for the PageRank algorithm and Word Count.
- How efficient and useful Hadoop is, when working with large datasets.

Problems Faced

- Different versions of Hadoop created problems. Hadoop 1.2 and 2.2 are different and the same code didn’t work on both.
- Giraph the developer branch is more useable than the stable release. Stable release is full of bugs.
- Support for Giraph is really low, hard to find good online tutorials.
- TextWritable class in Hadoop, worked perfectly in plain hadoop but not in Giraph (without some data cleaning).
- Wikipedia we had to remove non-existent links. Memetracker is really ugly in terms of data, each entry spans over multiple lines. Memetracker URLs can contain special characters and other language charters. To work on Giraph we had to filter them out.

References

[1] http://www.google.com/competition/howgooglesearchworks.html

[2] http://infolab.stanford.edu/~backrub/google.html

[3] http://dumps.wikimedia.org/enwiki/latest/

[4] http://giraph.apache.org/

[5] http://hadoop.apache.org/docs/stable/mapred_tutorial.html

[6] https://github.com/10gen-interns/big-data-exploration/wiki/_pages

[7] http://snap.stanford.edu/data/memetracker9.html

[8] http://my.fit.edu/~akhademzadeh2011/thesis/Thesis/10.1.1.175.113.pdf

[9] http://www.undercloud.org/?p=408

[10] http://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/