

CS5300 Projec 2: Fast Convergence PageRank in MapReduce

Giri Kuncoro (gk256), Yihui Fu (yf263), Shibo Zang(sz428)

1. Overall Structure

In this project, we implemented the following versions of computing PageRank values:

- Simple Computation of PageRank.
- Jacobi Blocked Computation of PageRank.
- **[Extra credit]** Gauss-Seidel Computation of PageRank
- **[Extra credit]** Random Block Partition

2. Preprocessing

2.1 Filtering Edges

The netID used to compute rejectMin and rejectLimit is gk256 with the reversed value being 0.652. The rejectMin is 0.5868 and the rejectLimit is 0.5968. In total, 7,524,304 edges are selected in the graph. The pre-filtered edges file is stored in our S3 bucket: https://s3-us-west-2.amazonaws.com/edu-cornell-cs-cs5300s16-gk256/processed_edges.txt. To download the file, type

```
aws s3 cp s3://edu-cornell-cs-cs5300s16-gk256/processed_edges.txt <target_file_name>
```

in the terminal. It's the same format as the given edges.txt, i.e,

```
src-node-number dest-node-number floating number .
```

The Python code used for filtering is in `preprocess/process_edges.py`. To run it, `cd` to the directory where `process_edges.py` is, put `edges.txt` in the same directory, then run

```
python process_edges.py <input-file> <output-file> .
```

2.2 Converting Edges to Nodes

Additionally, we processed the edge file into a node-centric one so that it can be an appropriate input for mapper. It's also stored in our S3 bucket: <https://s3-us-west-2.amazonaws.com/edu-cornell-cs-cs5300s16->

[gk256/nodepairs.txt](#). The format of a node entry is

```
nodeID-blockID pagerank-value neighborNodeID-blockID [...]
```

To download the file, type `aws s3 cp s3://edu-cornell-cs-cs5300s16-gk256/nodepairs.txt <target_file_name>` in the terminal. Although some edges were rejected in the filtering, all the nodes in these edges were still retained.

The python code for this is in `preprocess/process_nodes.py`. To run it, `cd` to the directory where `process_nodes.py` is, put `blocks.txt` and the filtered edge file in the same directory, then run `python process_nodes.py <edge-file> <block-file> <output-file>`.

2.3 Random Block Partition

To compute a bad partition for the graph, we assigned a node to a block based on the value of `nodeID % totalBlocks` where `totalBlocks` is 68. This ensures that blocks are uniformly sized. The processed file is store in the S3 bucket as well:<https://s3-us-west-2.amazonaws.com/edu-cornell-cs-cs5300s16-gk256/nodepairsRandom.txt>. To download the file, type

```
aws s3 cp s3://edu-cornell-cs-cs5300s16-gk256/nodepairsRandom.txt <target_file_name>
```

in the terminal.

The python code for this is in `preprocess/process_nodes_random.py`. To run it, `cd` to the directory where `process_nodes_random.py` is, put the filtered edge file in the same directory, then run `python process_nodes_random.py <edge-file> <block-file> <output-file>`.

3. Other Important Information Regarding Compilation

3.1 Helper Classes

`src/proj2/main/node/Node.java` :

- Includes meta data about a node to make extracting information easier: `nodeID-blockID`, `nodeID`, `blockID`, `currentPageRank`, `degree`, `emittedPageRank`, and an array of destination nodes
- Please include this file when compiling code Blocked or Gauss-Seidel Computation

`src/proj2/main/util/Constants.java` :

- Defines constant values used in this project, including damping factor, total node number, total block number, pass number for simple computation, convergence threshold, and enumerations for Hadoop counters
- Please include this file when compiling code for every version of computation

3.2 External Jar

Official jar files downloaded from [Hadoop distribution](#)

- hadoop/share/hadoop/common/hadoop-common-2.7.2.jar
- hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.2.jar

3.3 Project Jar File

In case the submitted source code cannot be compiled, we exported a jar file for the project and included it in the submission. To run a certain version, for example, type

```
hadoop jar <jar-file> proj2.main.simple.SimplePageRank <input-path> <output path>
```

in the terminal. You may add the number of MapReduce pass you want to run as the third argument after <output path> .

4. Simple Computation of PageRank

src/proj2/main/simple package:

- SimplePageRank.java
 - Sets the input and output path accordingly
 - Runs 5 MapReduce passes
 - Initializes a MapReduce job for each MapReduce pass
 - Computes per iteration average residual using a Hadoop Counter
- SimpleMapper.java
 - Emits < srcNodeID-blockID, node entry > for computing residual and recovering node structure
 - Emits < destNodeID-blockID, outgoingPageRank> for every outgoing edge
- SimpleReducer.java
 - Updates the PageRank value for a node based on the PageRank values of its immediate neighbors, using the formula $(1-d)/N + d * \sum (<PR^t(u)/degree(u)>)$ where $d = 0.85$ and $N = 685230$
 - Emits < nodeID-blockID, updated PageRank and reconstructed adjacency list >
 - Adds the residual for this node $|PR^t(u) - PR^{t+1}(u)| / PR^{t+1}(u)$ to a residual counter

The average residual errors in each MapReduce pass are as the following table. After 5 MapReduce passes, it's still far from convergence.

Iteration	Average Error
0	2.338036138444408
1	0.32297669467638457
2	0.19207763229971833
3	0.09406491742335128
4	0.06281398244773434

5. Jacobi Blocked Computation of PageRank

`src/proj2/main/blocked` package:

- `BlockedPageRank.java`
 - Sets the input and output path accordingly
 - Runs MapReduce passes until reaching convergence where the average relative residual error is below 0.001
 - Computes average relative residual and average number of iterations per block for each reduce task by using two Hadoop Counters
 - Reports the PageRank value for two lowest-numbered nodes in each block after the entire computation converged
- `BlockedMapper.java`
 - Emits `< src-blockID, PR node entry >` to compute residual in reducer and reconstruct node structure
 - Emits `< dest-blockID, BE srcNodeID-blockID destNodeID-blockID >` for every destination node in the same block as the source node
 - Emits `< dest-blockID, BC srcNodeID-blockID emittedPageRank destNodeID-blockID >` for every destination node not in the same block as the source node
- `BlockedReducer.java`
 - Parse the input values and store in different HashMaps accordingly
 - Performs in-block iterations until the in-block residual for the last iteration reaches convergence threshold 0.001
 - Emits `< nodeID-blockID, updated PageRank value and reconstructed adjacency list >`
 - Add the entire block reducer residual to a residual counter
 - The relative residual error reflects the difference of a node's PageRank value before and after internal block iterations: $|PR^{\text{start}}(v) - PR^{\text{end}}(v)| / PR^{\text{end}}(v)$
 - Add the number of iterations in this block to an iteration counter

The average error and average in-block iteration number per block for each MapReduce pass of this version is as the following table. It shows that Jacobi Computation can reach convergence in 6 MapReduce passes while the Simple Computation would take longer.

Iteration	Average Error	Average Number of Iterations per Block
0	2.8105539255441174	17.529411764705884
1	0.037807487029411765	7.161764705882353
2	0.023995235897058823	5.882352941176471
3	0.009842906882352941	3.8970588235294117
4	0.003830367220588235	2.514705882352941
5	9.506281323529412E-4	1.338235294117647

The PageRank values for the two lowest-numbered nodes in each block are in

`result/jacobi_pagerank_values.txt`

6. Gauss-Seidel Computation of PageRank

`src/proj2/main/gauss` package:

- GaussPageRank.java
 - This is the same as the BlockedPageRank.java
- GaussMapper.java
 - This is the same as the BlockedMapper.java
- GaussReducer.java
 -

The average error and average in-block iteration number per block for each MapReduce pass of this version is as the following table:

Iteration	Average Error	Average Number of Iterations per Block
0		
0		
0		
0		
0		
0		

The PageRank values for the two lowest-numbered nodes in each block are in

```
result/gauss_pagerank_values.txt
```

Comparison

The results show that Gauss-Seidel Computation takes less number of in-block iteration to converge, particularly for the first three MapReduce passes.

7. Random Block Partition

7.1 Convergence Performance for Jacobi Reducer using METIS partition and a random partion

Iteration	Content	Jacobi METIS	Jacobi Random
0	Average error	2.8105539255441174	
0	Average iteration/block	17.529411764705884	
1	Average error	0.037807487029411765	
1	Average iteration/block	7.161764705882353	
2	Average error	0.023995235897058823	
2	Average iteration/block	5.882352941176471	
3	Average error	0.009842906882352941	
3	Average iteration/block	3.8970588235294117	
4	Average error	0.003830367220588235	
4	Average iteration/block	2.514705882352941	
5	Average error	9.506281323529412E-4	
5	Average iteration/block	1.338235294117647	

Comparison

7.2 Convergence Performance for Gauss-Seidel Reducer using METIS partition and a random partition

Iteration	Content	Gauss-Seidel MEITS	Gauss-Seidel Random
0	Average error		
0	Average iteration/block		
1	Average error		
1	Average iteration/block		
2	Average error		
2	Average iteration/block		
3	Average error		
3	Average iteration/block		
4	Average error		
4	Average iteration/block		
5	Average error		
5	Average iteration/block		

Comparison