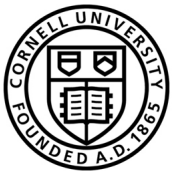

Project 2: Fast Convergence PageRank in Hadoop

Alice Renegar arr233
Spencer Steel shs257
Garth Bjerk gab225

May 5, 2015

1 Solution Structure

1.1 Overall Structure

Our code is organized into 4 packages, `node`, `block`, `extra`, and `common`. `Common` contains enums, and other useful functions that we used in all of our MapReduce jobs. `Node`, `block`, and `extra` each contain a main class that runs each different job i.e. PageRankMain (Node by Node), BlockMain (Block), and GaussMain (Gauss-Seidel Reducer). Each package also contains a mapper and reducer class for pass 0 which processes input files, and then mapper and reducer classes for subsequent jobs or passes to actually process the data. The MapReduce classes will be discussed more in the next section.

1.2 MapReduce Passes

Each main method discussed above follows the same flow. Pass 0 takes in an input file and processes the inputs to be more usable the program. Then the main method enters a loop and either loops until a specific residual sum is reached, or in the case of node by node, we have looped 5 times.

1.2.1 Node by Node

Pass 0 mapper takes the pre-processed edges.txt and outputs a list of edges (fromNode, toNode) and decides what edges to use (based on netid, see section below). Pass 0 reducer then combines all the edges and creates an output similar to (fromNode -¿ list of toNodes, PR) All subsequent passes the mapper gets each node id from the list of 'toNodes' and calculates a PR value (toNode, PR/totalToNodes) and (fromNode -¿ list of toNodes, PR). Each reducer pass, it calculates residual and emits (fromNode -¿ toList, UpdatePR, residualValue)

1.2.2 Block

Pass 0 looks at all edges and determines which to keep and what not to keep. It also assigns the nodes to blocks and outputs (fromBlock, fromNode -¿ toNode). Reducer pass 0 combines all info from mapper0

and emits (fromBlock, nodesList, innerEdgeList, outerEdgeList). All subsequent passes run the same mapper and reducer. The mapper gets all the outgoing edges and emits (blockID, toNode on edge, PR on edge) and (fromBlock, nodesList, innerEdgeList, outerEdgeList). The reducer collects PR of outgoing edges. It then will loop all nodes in the data given and calculate PR and residual until that 'block' is converged. Once converged calculate residual per node based on starting PR and ending PR. Then outputs the updated (blockID, nodesList, innerEdgeList, outerEdgeList). This process continues until the termination requirement is met.

1.3 Edges Filtering and Parameters

To filter our edges we used Spencer's netID - shs257
Therefore we had the following filter parameters:

- idValue = .752
- rejectMin = 0.6768000000000001
- rejectLimit = 0.6868000000000001
- Edges Selected = 7524631
- Percentage of edges selected = 99.00055193047387%

1.4 Implementation breakdown

for further details please see javadocs included at: docs/index.html

2 Amazon EMR

2.1 Setup Procedure

To get our application running on Amazon EMR follow the steps below:

1. From IDE or command line create Jar file from source code.
2. Upload jar to s3.
3. From the Amazon EMR console create a new cluster with desired specifications. Our code is using Hadoop 2.4.
4. Add a step. Selecting Custom Jar. Point it to uploaded s3 jar.
5. Add the correct arguments. First argument is the entry point or main class. For our project specify PageRankMain/BlockMain/GaussMain. Second argument is input location, third location is output location.

6. Create cluster with the custom step.
7. EMR will perform the necessary MapReduce notify when it is done.
8. After it completes output will be available on s3 or EMR console.

3 Results

3.1 Node by Node Results

Below are the results from the Node by Node MapReduce.

Round: 1
Residual sum (across all nodes): 1602356.88339 avg: 2.338421965456854

Round: 2
Residual sum (across all nodes): 221255.24082 avg: 0.3228919352917998

Round: 3
Residual sum (across all nodes): 131590.70527 avg: 0.19203873921165157

Round: 4
Residual sum (across all nodes): 64607.58472 avg: 0.09428598385943406

Round: 5
Residual sum (across all nodes): 43005.68138 avg: 0.06276094359558104

3.2 Block Results

Below are the results from the Block MapReduce. We were able to get the error down within two rounds which was a huge improvement from Node by Node.

Round: 1
Inner block rounds total: 4124 avg 60.64705882352941
Residual sum (across all nodes): 1964547.51313 avg: 0.12923642906443508

Round: 2
Inner block rounds total: 2772 avg 40.76470588235294
Residual sum (across all nodes): 8766.39452 avg: 5.766913327180306E-4

3.3 Gauss-Seidel Results

We did implement the Gauss-Seidel reducer and we got the error down within two rounds, but as you can see it was done in less inner block rounds.

Round: 1

Inner block rounds total: 2190 avg 32.205882352941174
Residual sum (across all nodes): 1964491.28713 avg: 0.129232730275064

Round: 2
Inner block rounds total: 1590 avg 23.38235294117647
Residual sum (across all nodes): 8775.27978 avg: 5.772758435359337E-4

3.4 Random Partition Results