Churn

John Li, Albert Ge, Timothy Chou, Kevin Chang

Rankmaniac Report

1 Overview

During this RankManiac project, our group "churn" implemented PageRank on the given network via Map Reduce through Hadoop on Amazon AWS Clusters. Two sample datasets, "GNPn100p05" and "EmailEnron", were distributed to us for the purpose of local testing. Our Rankmaniac score was determined by our PageRank's elapsed time and the correctness on a larger, hidden dataset on these Amazon AWS clusters. This report's purpose is to illustrate the process for our implementation of PageRank.

2 Initial Framework

Development

The data that was given to us was in the form:

NodeId:[NODEID] [CURRENT_RANK],[PREVIOUS_RANK],[ADJACENCY_LIST]

After a basic understanding of the structure of the project, our initial framework of MapReduce was as follows:

- PAGERANK_MAP:
  The input of this mapping will either be in the above format, or the output of PROCESS_REDUCE, which prepends a [ITERATION] value in front of the current values. This mapping function will take in the key/value pair, increment [ITERATION] by 1, and map the key/value pairs to be either:

  [NODEID],[FLAG],[CURRENT_RANK],[ITERATION] [ADJACENCY_LIST]

  or

  [NEIGHBORID],[FLAG],[CURRENT_RANK],[ITERATION] [NEIGHBOR_RANK]

  NOTE: [FLAG] represents a string with the values : "rank" or "list ". This is specify which type of key/value pair was sent in (first format is with the "list" flag, while the second format is with the "rank" format).

  The first format is used to pass in the appropriate adjacency list. The second format passes in a value that is equal to the current node's rank divided by its outgoing degree:

$$[\text{NEIGHBOR\_RANK}] = \frac{[\text{CURRENT\_RANK}]}{[\text{ADJACENCY\_LIST}].\text{size}()}$$

The reason why we do this is twofold. First, it will avoid matrix multiplication, which is a very time-consuming process. Secondly, the PageRank value for each node in the graph is equal to:

$$r_j = \sum_{(j,i) \in E} \frac{r_i}{d_i}$$

By passing this value into the reduce function, we assume the sort would allow all of the NodeId's to be grouped together so that PageRanks can be calculated for each iteration. Furthermore, since the adjacency matrix is passed in as well, the idea is to have the adjacency matrix sorted at the beginning of each node block.

- PAGERANK_REDUCE: The input of this PageRank reduce function will be format denoted in the PAGERANK_MAP section. We also expect that the Amazon machine will sort based on NodeId, so the implementation that we have is to reset aggregate rankings and adjacency lists whenever a new NodeId is encountered. Since there are two different input styles, I will state what the reduce function will do for each type of input.
  - **Rank**:
    We expected the nodes all to be grouped up based on NodeId. So, with the knowledge of sort sorting "link" in front of "rank", we assumed that whenever we saw a "rank" in the key, then all subsequent keys for that node would also include "rank". Since this key/value pair would include a rank increment, we aggregated all of these rankings, and set that aggregated ranking to be equal to the new ranking of the node of that block.
  - **Link**:
    Since "link" is less than "rank" when you compare them, we assumed that for a particular NodeId group, the adjacency list will come first. Thus, if the reduce function receives a key/value pair with the key containing "link", it would simply store the adjacency matrix.

    There is one edge case: the adjacency matrix is empty. This means there are no outgoing edges, so instead of aggregating the rank starting at 0, we initialize the rank to be the current rank of that node (also passed in via key), and aggregate ranks from the "rank" key/value pairs.

  After retrieving the adjacency matrix and setting the current rank to be the aggregated rankings for a particular NodeId, the PageRank reduce function will output a key/value pair in the following format:

  NodeId:[NODEID] [ITERATION],[CURRENT_RANK],[PREVIOUS_RANK],[ADJACENCY_LIST]

  This output will then be passed into the PROCESS_MAP function.

- PROCESS_MAP:
  The process map function will take in key/value inputs with the above format, and from there, keep track of the top 20 rankings. This function stores a global variable known as "top", which stores the

top 20 rankings. For each key/value pair, the input node is compared with the rank 20 node, and if the input node has a higher ranking (or the top 20 list isn't fully populated), it is added to the list (and removing the 20th position if the list was previously populated). The list is then resorted in decreasing order. In the case that the node has been added to this top 20 list, the function will output another key/value pair for the reduce to handle:

A:[NODEID] [CURRENT_RANK]

Note: The reason why "A:" is added is for the sorting to keep these into account first. This will pose problems in the future, which we will list out in future sections.

If the input node has not been added, then we simply output the input line (so it would be the identity function).

- PROCESS_REDUCE:
  The process reduce function is pretty similar to the process map function. Like the process map function, there is a global list of the top 20 scores, except we add ranks into that list whenever we receive a key starting with "A:". The inputs for this function have keys that start with either: "A:" or "NodeId:". Since we originally assumed sorted/grouped keys for this, (which is wrong), this is what we had planned the program to do:
  - **A:**
    For each iteration, the sorting would cause this reduce function to process these first. Whenever a key starts with "A:", we would either append it to the list (if it wasn't originally populated), or replace the 20th rank with the input rank, and resort the list.
  - **NodeId:**
    If this were the case, the reduce function would either serve as the identity function (if the iteration count is not maximum), or we would print out its final rank (if the iteration count is maximum).

  Therefore, at the very end of the simulation, the output for reduce should be:

  FinalRank:[CURRENT_RANK] [NODEID]

Problems

3 Refactored Framework

Refactoring

Optimizations

4 Testing

5 Contributions

6 References