CS/EE 144 **Guru: Joe**

Rankmaniac Reloaded

Assigned: 2/6/2013 Due: 2/15/2013 at 9:30am/5pm

1 Rankmaniac Reloaded [100 points]

In this exercise, you'll get hands-on experience writing a MapReduce application – you'll be using MapReduce to calculate the PageRanks of large graphs.

When we talked about PageRank in class, we always talked about computing *all* the PageRanks of a graph...but of course all nodes aren't created equal. In reality, determining the first page of results is far more important than understanding exactly what the PageRanks are. Motivated by that, your goal in this assignment will be to optimize PageRank calculation to take advantage of the fact that what you really care about is identifying the nodes with the highest PageRank.

Specifically, your goal is to use MapReduce to determine and rank the 10 nodes of a (large) graph that have the highest PageRanks. Further, you'll be competing with each other to see who can do this the fastest.

Your task: Working in groups of 3-4, your objective will be to create a MapReduce application that uses the Amazon Elastic MapReduce stack to compute and rank the 10 nodes of a graph that have the highest PageRanks. Your application will take as its input a graph dataset in node adjacency form and complete running within 50 iterations. In each iteration, your application will utilize two sequential map/reduce steps. Upon completion, your application will output the top 10 nodes in descending order. You will be evaluated based on the accuracy of your rankings and your computation time. We don't care how you compute the nodes and in fact we hope to see some creative approaches to calculating PageRanks...

We hope that this is a fun (and practical) assignment – but it will also be a time consuming one, so you should start as early as possible!

Grading:

(a) **Report (45 points):** Each group must turn in one detailed report describing the steps taken to optimize your calculations, and why you think your approaches will work. The report must describe the contribution of each team member to your team's effort, and the report must include citations to any papers or web sites that contained ideas for optimizations that were used in your implementations. A good report will show that you put significant effort and thought into figuring out how to take advantage of network structure (sparsity, heavy-tails, clustering, etc.) in order to improve the speed of your PageRank algorithm. **This report is due 5pm, Feb. 15 by email to Joe.**

(b) **Working code (15 points):** Your MapReduce application will be graded using two datasets. These grading datasets will **not** be provided to you. You get 10 points for each data set that your application runs and returns the top 10 nodes correctly (not necessarily in the correct order). The "correct" ranks will be determined based on a damping factor (α) of 0.85.

(c) **Beat the TAs (30 points):** A leaderboard of the scores of all submissions will be maintained throughout the assignment at `http://zhao.caltech.edu/scoreboard.html`. The TAs will take a shot at this assignment along with you. You will receive 15 points for each TA team you beat. There will be

two teams: "grad TAs" and "undergrad TAs". See the later section on "scoring" for more information about how scores are computed.

(d) **Start early (10 points):** To encourage you to start early we will give you 10 points if you get a solution that manages to place a score on the scoreboard on Tuesday, which means having working code submitted to Amazon by Monday night. Starting early is crucial to the assignment since it will allow us to quickly find any bugs in our submission process and because it will ensure that you have time to work on clever optimizations...

(e) **Class competition (Bonus points):** At the end of the competition at 9:30am on Feb. 15, the team in first place will receive 10 additional points, the team in second place will receive 7 additional points, the team in third place will receive 5 points, the team in fourth place will receive 3 points, and the team in fifth place will receive 1 point.

(f) **Non-working submissions (-20 points):** We reserve the right to **deduct up to 20 points** for submissions that clearly have not been tested locally using the sample datasets. Such submissions may waste a lot of money on Amazon instances. So *please* test your submission locally before you submit. (Local testing is described below.)

What we provide: Hopefully you have already emailed Shiyu and/or Joe with your team name and composition. Once you have done this, your team will receive via email an Amazon Web Services security credential, which consists of:

- A team ID
- An AWS access key
- An AWS secret key

Keep these details safe, as you will need them while working on this assignment. Usage of the Amazon Elastic MapReduce service requires knowledge of the Amazon Web Services API. However, since we would rather you get familiar with MapReduce and PageRank than the API, we will provide you with a simple Python wrapper to interface with Amazon. The package you will download contains the following files and folders:

- `rankmaniac.py`: The wrapper we have written for you to interface with Amazon.
- `data/rankmaniac.conf`: This file contains your MapReduce instance configuration.
- `data`: This folder will contain all your data (i.e. input data and codes for mappers and reducers). The wrapper only supports a flat file structure in this directory, so you should not create additional subdirectories beneath this directory.
- `boto`: This folder contains the Amazon Web Services Python SDK which is used by the Python wrapper. You should not need to directly use this module.
- `local_test_data`: This folder contains some data sets that you can use for testing. It also includes the PageRanks for the top ten nodes in each data set.

The package is available at `http://courses.cms.caltech.edu/cs144/homeworks/rankmaniac-lib.zip`

Elastic MapReduce on Amazon: Amazon Elastic MapReduce uses the Hadoop streaming API. Your mapper and reducer routines must take input via `stdin` and to write output to `stdout`. Each line of input or output is a single key-value pair, delimited by a tab character. For instance, the key-value pair (1, 100) should be output as `1\t100`. For each map/reduce step, Amazon Elastic MapReduce will:

- Read your input data into your mapper.
- Sort the output of your mapper by key in ascending order.
- Stream the result of the sort into your reducer.
- Write the output of your reducer.

In the case of multiple mapper and reducer tasks, each mapper/reducer will receive a subset of the data to process and output. Although the mapper and reducer may be written in any language Amazon supports, we recommend using Python. To ensure a script is recognized as a Python script, you should begin your file with:

```
#!/usr/bin/env python
```

Specifications: The input data that we will use has the following (tab delimited) form for each line:

```
NodeId:0\t1.0,0.0,83,212,302,475,617,639,658,901,902,916,937,987
```

where 0 is the current node identifier, 83,...,987 are nodes to which node 0 links to, and 1.0 and 0.0 are the current and previous PageRanks of this node.

For each iteration, you are to provide **two map/reduce steps** to run in sequence. We call the first step the "pagerank" step and the second step the "process" step. The "pagerank" step will run with a variable number of map and reduce tasks that you may specify in `rankmaniac.conf`, while the "process" step will always run with a **single** map and a **single** reduce task. There are no restrictions as to how you use these steps or format your data in between your mappers, reducers, as long as your final output is in the required format and your program produces the final output within 50 iterations. We expect each line of the **final output** to have the (tab delimited) form:

```
FinalRank:9.4\t90
```

where 9.4 is the final calculated PageRank, and 90 is the node identifier to which the PageRank belongs. During grading, we will terminate your application when any step produces data in the above format.

Scoring: We will accept submissions continuously up till the due date and maintain a leaderboard of the scores of all submissions. To submit your code, simply use the upload function we provide in `rankmaniac.py` (see the "submission" section for more details). We will run your submission nightly to update the leaderboard. In order to help us cut down time, we ask, unless you are in the process of testing code, that you remove extraneous data from your uploaded directory (e.g. sample datasets, output from previous job runs, etc). Note that your upload directory is automatically cleared at the beginning of each call to `Rankmaniac.upload`. If you do not have a submission to be graded, just upload an empty directory.

There are two scoring datasets: the small dataset is about 7,000 nodes and 100,000 edges; the big dataset is about 300,000 nodes and 1,500,000 edges. Assuming a correct ordering of the top 10 nodes, your total score will be the sum of the runtime on both datasets (so ironically, the team with the least score is the

winner). Because runtimes are calculated by summing the difference of end and start timestamps for each step, Amazon's ramp-up overhead for each map/reduce step is not included in the total time. While there is a maximum of 50 steps, you may choose to output a final ranking in an earlier step, which would conclude the timing at that point.

Only incorrect final rankings will be penalized (your final pagerank values are disregarded). For incorrectly ordered nodes in the top 10, you will be penalized by the sum-of-squared-difference of rank times 30 seconds. For instance, if the correct ranking is node 1, 2, 3, ..., 8, 9, 10, but your output is 1, 2, 3, ..., 7, 10, 8, 9, you will be penalized for $30(2^2 + 1^2 + 1^2) = 180$ sec $= 3$ min. (You output node 10 as the 8th node instead of the 10th node, hence difference of 2 with the actual ranking; difference of 1 for node 8 and node 9, so sum of difference for your ranking is $2^2 + 1^1 + 1^1$). Another example would be, if your program outputs the ranking in the order of node 10, 2, 3, ..., 8, 9, 1, you will be penalized for $30(9^2 + 9^2) = 30 \cdot 162$ sec $= 81$ min. Any nodes not in the top 10 list are regarded as the 11th rank, so if your program's output is node 1, 2, ..., 7, 8, 42, 10, node 42 is considered as rank 11 and you have swapped a rank-9 node with a rank-11 node, and will receive penalty of $30(2^2 + 2^2) = 4$ min. Generally, one or two minor mistakes will not incur a large penalty, but large mistakes would cause serious penalties.

The leaderboard will contain both your best score and the score of your latest submission.

Testing: You must test your map and reduce routines locally on your computer for correctness. To simulate a single iteration of the two sequential steps (assuming one map task and one reduce task), you can simply pipe output from your mapper to a sorter which sorts by the key to your reducer. On a Linux command line, a sample iteration might look like:

```
./pagerank-map.py < input.txt | sort -k 1,1 | ./pagerank-reduce.py
  | ./process-map.py | sort -k 1,1 | ./process-reduce.py > output.txt
```

You might want to write a script that repeats these commands until your process reducer returns the required final output. When you are satisfied with your program, you may use the `rankmaniac` module that we provide to test performance and integration with Amazon Elastic MapReduce. Given that Amazon Elastic MapReduce costs money for the amount of time and number of virtual machine instances used, we ask that you:

- Test on Amazon for a maximum of **2 hours per day**. (Of course you can test much longer locally if you wish.)
- Test on Amazon with only a **single instance** (this is built-in to the `rankmaniac` module, so please do not change this value).

You might find that testing on Amazon takes longer than testing locally on your computer. This is normal and is due to the overhead of provisioning instances on Amazon. Hence, you might consider doing most of your algorithm testing locally and using Amazon only to check for compatibility.

The `rankmaniac` module implements the Rankmaniac object. This object will keep track of your connection to Amazon, as well as the current running job on Elastic MapReduce. We recommend you carefully read through the included documentation before beginning. You may view the documentation either by viewing the source code, or typing in the Python interpreter:

```
>>> from rankmaniac import Rankmaniac
>>> help(Rankmaniac)
```

As an example, suppose the input graph dataset "input.txt" is contained in the "data" folder together with your map/reduce code. In Python, the following lines will upload the map/reduce code and input.txt:

```
>>> from rankmaniac import Rankmaniac
>>> r = Rankmaniac('myteam', 'mykey', 'mysecret')
>>> r.upload()
>>> r.submit_job('pagerank-map.py', 'pagerank-reduce.py', \
... 'input.txt', 'myoutput0/')
```

We can verify if the last step has completed via:

```
>>> r.get_job().steps[-1].state == 'COMPLETED'
```

To add another step, type:

```
>>> r.add_step('process-map.py', 'process-reduce.py', \
... 'myoutput0/', 'myoutput1/')
```

We can download the results via:

```
>>> r.download()
```

The data directory should now be populated with the results in addition to the input files. To complete your run, type:

```
>>> r.terminate_job()
```

A typical test on Amazon will involve:

1. Uploading the four map/reduce routines and the input dataset to Amazon (Rankmaniac.upload).
2. Submitting the first pagerank job (`Rankmaniac.submit_job`) and adding the process step (`Rankmaniac.add_step`).
3. Polling for the state of the last step and waiting until the process step completes (`Rankmaniac.get_job`). You might consider waiting for a bit before polling Amazon for the job state (after adding a new step) as there might be a delay between adding the process step and Amazon receiving the new job.
4. Downloading the results (`Rankmaniac.download`) and checking whether the process step has returned the top 10 nodes.
5. If there are additional iterations, add the new pagerank and process steps (`Rankmaniac.add_step`).

While you may perform all the steps manually, it might be advisable to create a Python script that runs locally on your computer to manage the jobs and to add additional steps.

Submission: To submit your code daily for the scoring on the leaderboard, use the upload function in `rankmaniac.py`. Your submission should contain your codes for your pagerank step mapper and reducer and your process step mapper and reducer. It should also contain your configuration file `rankmaniac.conf`. This file must be placed in your `data` folder. It contains the following configurable values that must be correctly set for your submission:

- `num_map`: The number of pagerank map tasks to run. We will be running your program on 10 virtual machines, so scale this number accordingly.
- `num_reduce`: The number of pagerank reduce tasks to run. We will be running your program on 10 virtual machines, so scale this number accordingly.

- `pagerank_map`: The relative path to your pagerank step mapper.
- `pagerank_reduce`: The relative path to your pagerank step reducer.
- `process_map`: The relative path to your process step mapper.
- `process_reduce`: The relative step to your process step reducer.

You **do not** have to include any datasets in your submission as we will run your code on our grading datasets. Note that you are only allowed to submit **one** version of your code at a time, i.e., you cannot use different algorithms for the different data sets.

Sample datasets: We have provided two sample datasets on which you may test your code. The smaller dataset is a $G(N = 100, p = 0.05)$ Erdös-Rényl random graph; the bigger one is a web graph obtained by crawling 1000 of Caltech domain's webpages. You can use both datasets for local testing. Both of these datasets should also be small enough that even a non-distributed matrix multiplication approach would work and get the correct PageRank. Please note that the grading datasets will be significantly larger than these sample ones. As described in the scoring section, the final grading dataset will have about 7,000 and 300,000 nodes respectively.

To aid in your testing, the PageRanks of the top ten nodes in the two sample data sets are as following:

```
Caltech Web Results
FinalRank:29.281760 89
FinalRank:23.315706 223
FinalRank:16.893070 441
FinalRank:8.054729 693
FinalRank:6.966476 45
FinalRank:6.208602 44
FinalRank:5.869953 139
FinalRank:5.744268 945
FinalRank:5.361423 338
FinalRank:4.841198 140

G(n,p) Results
FinalRank:9.123267 21
FinalRank:2.135720 31
FinalRank:1.927420 72
FinalRank:1.723573 63
FinalRank:1.667035 9
FinalRank:1.631517 78
FinalRank:1.611635 92
FinalRank:1.509562 71
FinalRank:1.440409 45
FinalRank:1.386792 95
```

Though we have provided you with two data sets to help you validate correctness, it will likely be beneficial for you to use other data sets as well in order to validate your algorithmic optimizations. There are a variety of data sets that can be found quite easily online. For example, and the SNAP repository at Stanford `http://snap.stanford.edu/data/index.html`.

Additional notes:

- Start early! This is important because you do not have access to the datasets that we will be using to test your code. So your results on the leaderboard are the only feedback you have as to whether your code works with our datasets as well as its performance.
- No really, start early! This is the first time we have run this assignment, so there are bound to be some bugs. Please start quickly and post any issues you have to Piazza so that we can resolve them as quickly as possible. If you don't have a score on the leaderboard by the end of the first weekend, you're behind...
- You can feel free to investigate papers on MapReduce optimizations. In fact, we encourage you to. The idea is that through exploring and thinking about how to improve on the calculation approaches we went over in lecture, you will learn a lot more than if we just presented you approaches for optimizations. A few starting points, remember in class the hints we talked about in class about taking advantage of the sparsity of the underlying web graph and the fact that heavy-tailed degrees lead to heavy-tailed page ranks...
- The Python wrapper has been verified to work with OS X and the CS cluster. However, unfortunately, it might not work on Windows machines. It appears to be a problem with the Python SDK itself, so we could not work around it in time. We would greatly appreciate any work-arounds to this issue that you discover. The Python wrapper is only for interfacing with Amazon so you can still test your code locally on Windows machines and simply use the CS cluster for interfacing with Amazon.
- Feel free to modify the code we have provided as fit, given that you follow our testing guidelines.
- If you're curious about the Python SDK for Amazon Web Services, the complete reference is at `http://boto.readthedocs.org/en/latest/index.html`
- If you're curious about Hadoop streaming, you can find a good reference at `http://hadoop.apache.org/docs/r0.20.2/streaming.html`