CMS/CS/EE144 **Gurus: Joon/Cathy**

Miniproject : Rankmaniac

Assigned: 2/1/2018 Rankings close: 2/9/2018 5:59pm
Report due: 2/12/2018 11:59pm

1 Rankmaniac [100 points]

For your first miniproject of the term, we want you to get some hands-on experience with MapReduce and PageRank! When we talked about PageRank in class, we discussed how to compute the rank of *all* nodes, but of course this is rarely how the problem is set up in practice. It's no mystery that Google cares a whole lot more about the ordering of search results on Page 1 than Page 10.

This is going to fuel Rankmaniac: your goal in this assignment is to optimize the process of computing PageRank for the practical and more manageable purpose of identifying the highest-ranked nodes.

2 Your Task

Working in groups of 2-4, your goal is to use MapReduce to determine and rank the 20 nodes of (large) graphs that have the highest PageRanks. You will be interfacing with the industry-standard Amazon Elastic MapReduce stack to compute these ranks.

We want to give you as much creative freedom as possible for your optimizations, but let's go over some basic specifications:

- Your application will consume the graph in node-adjacency form and run for at most 50 iterations. Each iteration consists of two sequential MapReduce steps (M-R-M-R).
- We will detect that your application has completed its work when it outputs the final ranks of the top 20 nodes (more on the specific output format below!).
- You will be evaluated based on the accuracy of your rankings and the computation time. We will use a damping factor (α) of 0.85 to determine "correct" ranks.
- You *must* use the MapReduce paradigm (i.e. don't run everything in one map call). While doing the latter might be an efficient approach for the graphs we're practicing on, you can imagine how infeasible this is on a Web-sized graph.

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! In fact, we're going to give you an incentive to do that (see **Grading** below).

3 Project Structure

We want you to use Amazon's Elastic MapReduce service because it's the tool of choice for many big players in industry! Unfortunately, using it programmatically requires knowledge of the Amazon Web Services API. Since we would rather you get familiar with MapReduce and PageRank than focus on the specifics of their API, we provide you with a simple Python wrapper to interface with Amazon.

At a high level, you'll be playing with the 4 python files within `data/`, where you will write the code to execute each of the 4 corresponding MapReduce steps. We've provided a Rankmaniac class in `rankmaniac.py` to spin up the MapReduce cluster on Amazon and an uploader in `uploader.py` to upload and run your local submission.

Let's take a more detailed look at what's inside:

- `uploader.py`: This file is a simple submission script, which you can use to upload the contents of your `data/` folder, run a MapReduce job, and download the results. You'll need to properly authenticate using command line arguments when running the uploader. See the section on **Authentication** below.
- `rankmaniac.py`: This file is a wrapper we have written for you to interface with Amazon. Do NOT modify this file, but you should familiarize yourself with the available methods.

  The `rankmaniac` module holds the `Rankmaniac` class, which keeps track of your connection to Amazon, as well as your 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)
  ```

- `data/rankmaniac.cfg`: This file contains your MapReduce instance configuration, which has the following parameters:
  - `num_mappers`: The number of pagerank map tasks to run. We will be running your program on 10 virtual machines, so scale this number accordingly.
  - `num_reducers`: 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 path to your process step reducer.
- `data`: Along with the configuration file above, this folder will contain the four source files for each map-reduce iteration.
- `boto`: This folder contains the Amazon Web Services Python SDK which is used by `rankmaniac.py`. You should not need to directly use these modules. Again, please do NOT touch this folder!
- `local_test_data`: This folder contains some datasets that you can use for testing. It also includes the PageRanks for the top 20 nodes of each dataset. The solutions for the sample datasets can be found in `sols/` directory.

4 Authentication

Your Amazon Web Services security credentials, which are needed by the uploader (`uploader.py`) we give you, consist of:

- A team ID [`team_id`]
- A decoding key [`decoding_key`]
- An encoded AWS access key [`enc_access_key`]
- An encoded AWS secret key [`enc_secret_key`]

Keep these details safe, as you will need them while working on this assignment. In previous years, AWS access and secret keys have ended up on public repositories, leading to hefty charges. To that end, we are provided you with *encoded* versions of the AWS credentials, which are then decoded by logic in `uploader.py`. ***The raw, decoded credential strings should not appear directly in your source code***. Instead, you'll pass them in as command line arguments to the uploader.

5 Working with Amazon EMR

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

- Read your input data into your mapper.
- Sort the output of your mapper by key.
- Stream the result of the sort into your reducer. [1]
- 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. To ensure the tasks are recognized as a Python script, you should begin the files with:

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

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, `1.0` and `0.0` are the current and previous PageRanks of this node, and `83,...,987` are nodes to which node `0` has outlinks to.

[1]You should treat this like a grouping. In particular, all (key, value) pairs for a given key will get sent to the same reducer, but you should not rely on an ordering within that group, or among key-groups. See **Additional notes** below.

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.cfg`, 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 or reducers, as long as your final output is in the required format and your program produces the final output within 50 iterations of execution.

The final output, which lists the top 20 nodes in descending order, should have the following (tab-delimited) format:

```
FinalRank:9.4\t90
FinalRank:8.7\t81
```

During grading, we will detect and terminate your application when any iteration (i.e. the second reduce step) produces output in the above format.

6 Testing your submission

For this project, there are two ways to test your code on known datasets: locally on your computer and on the actual MapReduce cluster. We ask that you:

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

6.1 Testing Locally

You *must* test your map and reduce routines locally on your computer for correctness. To simulate a single iteration of the two sequential steps per iteration, you can simply pipe output from your mapper to a sorter which sorts by the key to your reducer. Note that Amazon EMR doesn't do a genuine sort but just groups (key, value) pairs to the same reducer, so don't rely on this sorting in your logic! On the command line, a sample iteration might look like:

```
python pagerank_map.py < input.txt | sort | python pagerank_reduce.py |
python process_map.py | sort | python process_reduce.py > output.txt
```

Some notes about this:

- The bar sends the previous program's output as input to the next program in the line.
- To create `input.txt`, just copy and paste a dataset from the `local_test_data/` directory into a text file named `input.txt` in the `data/` directory.
- You probably want to write a shell script that repeats these commands until your process reducer returns the required final output.

6.2 Testing on Amazon

When you are satisfied with your program, you may use the `uploader.py` script provided to test performance and compatibility with Amazon Elastic MapReduce. You can do this by running:

```
python uploader.py
```

Your submission should contain your code for your pagerank step mapper and reducer and your process step mapper and reducer. It should also contain your configuration file `rankmaniac.cfg`. These files must be placed in your `data/` directory.

You might find that testing on Amazon takes significantly longer than testing locally on your computer. Your actual testing time will clearly depend on your implementation, but each iteration could take anywhere from 2-10 minutes for the sample datasets. This is normal and is due to the overhead of provisioning new instances on Amazon. Hence, you might consider doing most of your algorithm testing locally and using Amazon only to check for compatibility. Also, when testing on Amazon, you might want to reduce the maximum number of iterations so that you can see output is produced sooner. You can do this by overriding the `max_iter` parameter in `uploader.py`. Do NOT create text files in your submission to Amazon!

You might find the files under the `results/` directory (created by `uploader.py` script) useful for debugging your submissions. Especially, try to look under `results/job_logs/` which contains `syslog` and `stderr` for EMR jobs.

7 Scoring your submission

To submit your code, use the `uploader.py` script we provide. There are two scoring datasets: the small dataset has between 5,000 and 10,000 nodes and the big dataset has between 100,000 and 500,000 nodes.Here's how we will score a submission:

- Your total score will be the sum of the runtime on both datasets. Runtimes are calculated by summing the difference of end and start timestamps for each step, so don't worry about Amazon's ramp-up time. Note, of course, that you can choose to output final rankings in an earlier iteration than the 50th: this will conclude the timing and execution.

- You will be penalized in runtime for incorrect ordering of the top 20 nodes (we don't care about the PageRank values). The penalty will be based on the sum of square differences between the correct rank and your rank for each of the 20 nodes.[2]. Then, we will multiply this sum by 30 seconds to get the final penalty. As expected, one or two small transpositions will be much less costly than large mistakes.

  If your ranking is 1, 2, 4, 3, 5, 100, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 (where each node number is also its rank), then your penalty is $(1^2 + 1^2 + 94^2) \cdot 30 \approx 74$ hours, since node 4 was ranked 3rd ($|4 - 3|^2 = 1^2$), node 3 was ranked 4th ($|3 - 4|^2 = 1^2$), and node 100 was ranked 6th ($|100 - 6|^2 = 94^2$).

7.1 Leaderboard & Schedule

We will accept submissions continuously up until the due date and maintain a leaderboard of the most recent and best submission scores. The leaderboard at `http://cs144-rankmaniac.com` will contain both

[2]The penalty per node is capped at 1000^2. If you output an invalid node or don't output enough nodes, you will receive the max penalty for each invalid or missing node.

your best score and the score of your latest submission. We will run your submission twice a day throughout the project.

From Feb. 5 (Monday) to Feb. 8 (Thursday), we will run your submission twice a day (noon and midnight) to update the leaderboard, so you should **submit by 11:59am and 11:59pm**. **On Feb. 9 (Friday)**, we will run your submission at noon and 6pm, so you should **submit by 11:59am and 5:59pm**.

Your upload directory is automatically cleared every time you upload to Amazon, which means you will not be able to perform multiple runs simultaneously. For this same reason, please refrain from submitting code within 20 minutes after the daily deadlines. If you do not have a submission to be graded, don't upload anything.

8 Grading

(a) **Report [55 points]:** Your group must turn in one detailed report describing the steps taken to optimize your calculations and why you think your approach works. The report must describe the contribution of each team member to your team's effort and 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 is due February 12 at 11:59pm, on Moodle.** Please have *each team member* submit an identical copy of your report as a single PDF file on the Moodle submission page.

(b) **Milestone [35 points]:** A leaderboard of the most recent and best submission scores (for each team) will be maintained at `http://cs144-rankmaniac.com`. There are two milestones ("Milestone 1" and "Milestone 2" that you should aim to beat by the end of the competition. You will receive 20 points for the first milestone and 15 points for the second milestone.

(c) **Early Bird [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 by Wednesday, which means having working code submitted to Amazon by **Tuesday night (Feb. 6 at 11:59pm)**. Note that this does not mean you need to submit code that accurately implements the PageRank algorithm; you just need to submit code that will successfully post any score on the scoreboard. Starting early is crucial to the assignment because it will ensure that you have enough time to work on clever optimizations!

(d) **Class leaderboard [Bragging rights]:** At the end of the project, at 6pm (plus a few hours to grade the final submission) on Feb. 9, the team with the best score will be crowned "Rankmaniacs", and be listed in perpetuum on the course website.

(e) **Non-working submissions [-20 points]:** We reserve the right to **deduct up to 20 points** for submissions that have clear not been tested locally using the sample datasets and for submissions with algorithms that do not use MapReduce. Such submissions may waste a lot of money on Amazon instances. So *please* test your submission locally before you submit.

9 Sample datasets

We have provided you with two sample datasets on which you may test your code. The smaller dataset is a $G(N = 100, p = 0.05)$ Erdös-Rényi random graph; the bigger one is a subset of Enron's email communication network with 1000 nodes. You can use both datasets for local testing. These datasets should 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. To aid in your testing, we've included the PageRanks of the top 20 nodes in the two sample datasets:

```
G(n,p) Results              EmailEnron Results
--------------              ------------------
FinalRank:9.122179 21       FinalRank:6.822899 16
FinalRank:2.135749 31       FinalRank:6.658854 83
FinalRank:1.927447 72       FinalRank:6.213688 17
FinalRank:1.723596 63       FinalRank:6.157889 47
FinalRank:1.667057 9        FinalRank:6.134326 27
FinalRank:1.631539 78       FinalRank:5.835673 225
FinalRank:1.611656 92       FinalRank:5.676395 84
FinalRank:1.509583 71       FinalRank:5.534920 135
FinalRank:1.440428 45       FinalRank:5.525422 272
FinalRank:1.386811 95       FinalRank:5.368599 22
FinalRank:1.331227 27       FinalRank:5.157754 76
FinalRank:1.329219 86       FinalRank:5.048887 12
FinalRank:1.313974 37       FinalRank:4.995004 178
FinalRank:1.270373 83       FinalRank:4.654378 342
FinalRank:1.266456 88       FinalRank:4.536737 219
FinalRank:1.226960 1        FinalRank:4.345334 91
FinalRank:1.210430 15       FinalRank:4.291703 222
FinalRank:1.204049 70       FinalRank:4.243678 78
FinalRank:1.183045 80       FinalRank:4.199803 229
FinalRank:1.179873 99       FinalRank:4.149635 305
```

Although we have provided you with two datasets to help you validate correctness, it will likely be beneficial for you to acquire other datasets in order to optimize your algorithm. There are a variety of datasets that can be found quite easily online such as the SNAP repository at Stanford (`http://snap.stanford.edu/data/index.html`).

10 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 will have as to whether your code works with our datasets, as well as its performance. Don't miss out on easy points for the assignment because of procrastination.
- In general, when writing your map-reduce code, you should view `sort` as nothing more than a mechanism to group keys into contiguous blocks (where each block of key-value pairs with the same key is sent to a single reducer). Do not rely on the sorting order in your code!

- **You should write all your code in Python 2.6.9!** Amazon uses this version of Python, and all of the provided code is compatible with it. You might want to install this older version of Python while doing local testing.
- **Hadoop Subtleties:** An important note on how Hadoop handles the parameter that specifies the number of map/reduce tasks: the number of map tasks is only a hint–the Hadoop cluster software does not always follow your recommendation. Hence, when writing your process mapper/reducer, if you require a step in which the entire dataset is processed by a single task, you should write that code in the process reducer. See `http://wiki.apache.org/hadoop/HowManyMapsAndReduces` for background on how Hadoop handles these parameters.
- Refrain from using `break` statements to process std, as it will interfere with Hadoop streaming.
- You may consider modifying the format of the input during subsequent iterations, e.g. adding a new column to represent the iteration number.
- We encourage you to investigate papers on MapReduce optimizations; 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 the hints we talked about with taking advantage of the sparsity of the underlying graph and the fact that heavy-tailed degrees lead to heavy-tailed PageRanks.
- If you are curious about the Python SDK for Amazon Web Services, the complete reference is at `http://boto.readthedocs.org/en/latest/index.html`.
- If you are curious about Hadoop streaming, you can find a good reference at `http://hadoop.apache.org/docs/stable1/streaming.html`.