CS/EE 144 **Gurus: Max, Angela**

Miniproject 1: Rankmaniac

Assigned: 2/5/2014 Due: 02/13/14 9:00am (Competition)
02/14/14 5:00pm (Report)

1 Rankmaniac [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 not all nodes are created equal. In reality, determining the first page of results is far more important than understanding exactly what the PageRank values are. Motivated by that, your goal in this assignment will be to optimize the PageRank calculation to take advantage of the fact that what you really care about is only identifying the nodes with the highest PageRanks.

Specifically, your goal is to use MapReduce to determine and rank the 20 nodes of (large) graphs that have the highest PageRanks. Furthermore, you'll be competing with each other to see who can do this in the shortest amount of time.

Your task: In groups of 2-3, your objective will be to create a MapReduce application that uses the Amazon Elastic MapReduce stack to compute and rank the 20 nodes of a graph that have the highest PageRanks. Your application will take as its input a graph dataset in node adjacency form and run for at most **50** iterations. In each iteration, your application will utilize two sequential map/reduce steps. Upon completion, your application will output the top 20 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]:** 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 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 is due Friday, February 14 at 5pm, by email to the TAs (csee1442014caltech@gmail.com).**

(b) **Working code [15 points]:** Your MapReduce application will be graded using two datasets. These grading datasets will **not** be provided to you. You will get 7.5 points for each dataset that your application runs and for which it returns the top 20 nodes 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 most recent and best submission scores (for each team) will be maintained at `http://anjoola.com/rankmaniac/`. 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 by Friday, which means having working code submitted to Amazon by **Thursday night (11:59pm on February 6)**. Starting early is crucial to the assignment because it will ensure that you have enough time to work on clever optimizations!

(e) **Class competition [Extra Credit]:** At the end of the competition at 9:00am on February 13, winning teams will receive additional points:

Place	Additional Points
1st	4
2nd	2
3rd	1

(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 later.)

What we provide: Once you have emailed the TAs (csee1442014caltech@gmail.com) with your team name and team members, your team will receive via email Amazon Web Services security credentials, which consists of:

- A team ID [`team_id`]
- An AWS access key [`access_key`]
- An AWS secret key [`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 provide you with a simple Python wrapper to interface with Amazon. This package is available for download at `http://courses.cms.caltech.edu/cs144/homeworks/rankmaniac-lib.zip`

The package contains the following files and folders:

- `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 **must** modify the `team_id`, `access_key`, and `secret_key` variables with the ones provided via email.

- `rankmaniac.py`: This file is a wrapper we have written for you to interface with Amazon. It is unlikely that you will need to 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`: This folder will contain all your data (i.e. input data and source code for the mappers and reducers). The wrapper only supports a flat file structure in this directory, so you should not create additional subdirectories.

- `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.

- `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.

Elastic MapReduce on Amazon: 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 each map/reduce step, 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.
- 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 highly suggest using Python (we will not be able to help you with any other language). To ensure the tasks are recognized as a Python script, you should begin the files 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, `1.0` and `0.0` are the current and previous PageRanks of this node, and `83,...,987` are nodes to which node `0` links to.

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. We expect each line of the **final output** to have the following (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 output in the above format.

Scoring: We will accept submissions continuously up until the due date and maintain a leaderboard of the most recent and best submission scores. To submit your code, use the `uploader.py` script we provide (see the *Submission* section for more details). We will run your submission nightly to update the leaderboard, so you should **submit by 11:59pm**. During the final days of the competition, we will run twice a day, so submissions should be in by 1:59pm and 11:59pm. **Note that your upload directory is automatically cleared every time you upload to Amazon, which means you will not be able to perform multiple runs simultaneously.** If you do not have a submission to be graded, just upload an empty directory.

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. Assuming a correct ordering of the top 20 nodes, your total score will be the sum of the runtime on both datasets. 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 iterations, you may choose to output a final ranking in an earlier iteration, which would conclude the timing and execution.

Penalties: Only incorrect final rankings will be penalized (your final PageRank values are disregarded). You will be penalized based on the sum of square differences in the correct rank and your rank for the top 20 nodes you output. (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).

The resulting sum is then multipled by 30 seconds to get the final penalty, which is added to the runtime of your algorithm. This is done in a way such that one or two minor mistakes will not incur a large penalty, but large mistakes would cause serious penalties.

For instance, 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 ($|100 - 6|^2 = 94^2$).

The leaderboard at `http://anjoola.com/rankmaniac/` 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. **Do NOT rely on the sorting order of the keys for your code!** 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
```

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 `uploader.py` script provided to test performance and compatibility 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 `uploader.py` script, so please do not change this value).

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`.

Submission: To submit your code for scoring and testing on Amazon, execute the `uploader.py` script 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/` folder.

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 datasets.

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`).

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. If you do not have a score on the leaderboard by the start of the weekend, you are behind...
- 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!

- **Note that Amazon uses Python 2.5.2, so write your code accordingly!** All of the provided scripts are compatible with this version of Python, so you want to simply install the older version while doing your local testing as well.
- **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.
- 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...
- Feel free to modify the code we have provided as you see fit, given that you follow our testing guidelines.
- 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`.