AMS 598 Project 3 Report

Tinghui Wu 114719023

Oct 25, 2022

1 Introduction

We built a pagerank process from scratch using taxation method in this project.

2 Methods

2.1 Initial Mapper

Following the course document, first we parsed the data files and generated a map that shows all the relationship between each web page and their destination. Namely, we'd transform the data files from (URL, next URL) to a list of (URL, (PR_{init}, [next URL1, next URL2, $\cdots$), where PR_{init} should be the initial page rank, here we simply entered 1 and will change later. In our code, we used dictionary to store the information.

Since we can use multiple processes and there are more than 1 data files, each process could read files separately.

Also, we recorded all the URLs, including the first web page and the destination, and stored them into a set to obtain a total set of all web pages.

- Input: original data file with a number of (URL, next URL)
- Output 1: dictionary storing (URL, (PR_{init}, [next URL1, next URL2, $\cdots$])
- Output 2: a set of all URLs

2.2 Middle Process

Then, we would (1) store the dictionary to a json file then record the file names to all processors and (2) use root node to collect the total set of the URLs then distribute evenly to all processors, those distributed list of URLs would be handled only by the corresponding processor from now on.

2.3 Initial Reducer

Although the class document used the identity function for reducer in phase 1, we used the initial reducer to merge all the dictionaries since we got n separated dictionaries from the initial mapper, where n is the number of processes. Each processor would read all the stored json files but only collected the URLs

We used the set of total URLs collected from Initial Mapper and distributed to all processes so that each process would only need handle a proportion of URLs. Also, since we had collected the set of all URLs, we can generate the initial page rank as $\frac{1}{\text{number of all URLs}}$.

- Input 1: json file name
- Input 2: the list of the URLs that the processor is in charge
- Input 3: dictionary storing (URL, (PR_{init}, [next URL1, next URL2, $\cdots$]). In the first loop this would be an empty dictionary, then the current dictionary so that we can keep updating it.
- Input 4: N, number of all URLs, so that we can feed the PR_{init} as $\frac{1}{N}$
- Output: updated dictionary storing (URL, (PR_{init}, [next URL1, next URL2, $\cdots$])

2.4 Mapper

After we collected the complete mapping dictionary we would process the page rank distribution. We calculated the new page rank by

$$PR_{new} = \frac{PR}{\text{length of the list [next URL1, next URL2, } \cdots]}$$

and stored the PR_{new} to all the next URL. To make the process quicker, here we also stored as the dictionary format, and add on all the PR_{new} for the same next URL.

- Input: dictionary storing (URL: (PR, [next URL1, next URL2, $\cdots$])
- Output 1: dictionary with (next URL: PR_{new})
- Output 2: dictionary storing (URL: [next URL1, next URL2, $\cdots$]) to keep the original mapping directions, where the keys are the same as the URLs that the processor is in charge

After running the mapper, we would also save the output as a json file and share the file name to all the processors.

2.5 Reducer

In the reducer, each processor would sum up the URLs that they were in charge of by reading all the json file. At the end, we would also modify the final page rank to

$$pr = \beta * pr + \frac{1-\beta}{N}$$

for the taxation methods.

- Input 1: the list of json file names
- Input 2: dictionary storing (URL: [next URL1, next URL2, $\cdots$]) to keep the original mapping directions, where the keys are the same as the URLs that the processor is in charge
- Input 3: beta, the parameter for taxation
- Output: updated dictionary storing (URL, (PR, [next URL1, next URL2, $\cdots$])

3 Results

3.1 Hyper-parameters

- beta: the β for the texation methods
- k: the top k websites with higher rank we should report
- loops: the loops running over mapper (2.4) and reducer (2.5)

3.2 Main Output

After running the code, we got the following results in Table 1. We used 4 processes to run the code with 5 loops for the mapper and reducer. It a total running time 52 minutes. We found out that there were a total of 10 millions websites.

Web page	Rank
1278445	6.538144026909136e-05
5043239	6.312796874993215e-05
69917	6.069727509340634e-05
8937112	6.006467784136152e-05
6971730	5.927261340160495e-05
7716978	5.733743577214907e-05
4751585	5.492513127991553e-05
3891296	4.801526843672319e-05
2088650	4.5237907510942405e-05
9933087	4.151525671874937e-05

Table 1: Top 10 websites

4 Discussion

4.1 Time Consuming

Since our data is a little bit large (100 data files with 10 millions web pages in total), although we were using parallel computing it still took almost 1 hour to finish the process. Here are some possible improvement to optimize the process:

- Omit the web pages with lower rank: since we only need the top 10 web pages out of 10 millions, it is nearly impossible for those web pages with lower rank at the first iteration rising to top 10 after several iterations.
- Change some summation and multiplication to numpy array process: it is possible to speed up the calculation

4.2 Other Experiments

Saving and reading files also took lots of time to process. We tried several way to share the information between processors and found out that

- Using MPI alltogather directly share the dictionary took longest time
- Saving the dictionary to pickle file took longer time, but the file size was smaller
- Saving the dictionary to json file was fastest in our case, but the file size was larger