Final Project Report

Perceptron Clustering on Wikipedia Data

Name: Junbo Li, Nicolas Valdiviezo

CS-345 Parallel System & Programming
(Spring, 2017)

Denison University, USA

6th May 2017

Contents

1 Overview

The motivation of this project is to truly understand the complexity of Big Data. In this month-long final project, we truly understand the three "V"'s of the Big Data: Volume, Velocity, and Variety.

We have been closely working with the Wikipedia related dataset. By using multiple parallel platforms, the ultimate goal for this project is to be able to answer the following questions:

What is the most generally preferred topics that people look up in Wikipedia?

Is there a pattern by time of the day? the month? the year?

Our working space is in Olin Hall, Denison University. And the devices that we have been using include Olin 219 machines and a Beowulf cluster. There are three total programming systems we used in order to drive to our final result: Hadoop, MPI (Message Passing Interface), and Pthreads.

The big picture of involving those abovementioned parallel systems for downloading data, wrangling data, and analyzing data is displayed in the following table:

Hadoop	MPI	Pthreads
	Download raw data .gz	
Unzip the jar of raw files		
Clean the raw data (delete non-English search)		
	Job 1	Job 1
Parse each category list		
Job 2	Job 2	
Run data analysis		

In order to better collaborate and have controls of different versions of file, we used GitHub as our shared working space. All of our codes, working history and each stage's records can be found at https://github.com/JunboLarryLi/getWiki_MPI. Notice, the link of our repository is not our final submission of the projects. It contains a lot of intermediate attempts and results for the sake of our project tracking. For example, the directory of *Wikiletics* is a MapReduce job we created for conducting Job 1 (querying categories from the web), but, the job ran forever because we did not realize the restriction of the bandwidth due to its private network setup, where only the namenode hadoop2 is able to communicate with the outside world.

2 The Dataset: WikiMedia

2.1 Dataset Selection

Most of people might think data selection part is very easy, as there are a lot of available both raw and cleaned datasets in the Internet. However, in order to find an interesting, meaningful, and huge dataset, it becomes trick. Since we are going to use Hadoop as our parallel tool, we set our limit of the size of the dataset as at least in the unit of Tera-bytes.

The processing of deciding topics and finalizing our datasets took roughly four days. And the final dataset we picked is called *Wikimedia*. Initially found at Amazon Web Service, but later on we realized there is a free access directly through Wikipedia's servers. However, the downside of grabbing data from this place is to the host adds a quota, where they have rate limited down-loaders and they are capping the number of per-IP connections to 2 in order to ensure that everyone can access the files with reasonable download times.

Because of the pressure of the deadline plus we are not malicious students, we decided to utilize all of 219 labs to download pieces of the entire set. The reason we are going to do so is because each 219 machines has its own public IP address. In this way, we can have relatively faster downloading rate overall.

2.2 Downloading Techniques

As mentioned above, we utilized all of the 219 machines downloading power to download the data. In order to better manage, we incorporated the MPI as our approach. The downloading MPI script is named as *getWiki_MPI.c*.

In order to not mess up with our school's local file system, we specify our downloading path into */tmp/...* locally instead of flood our student's local file system.

After the download finishes, we transport the data from each local 219 machines to Hadoop2, and then transport from Hadoop2 to HDFS.

2.3 Dataset Content

The content of the data is well-organized. The Wikimedia project released 9 years of pagecount views. And the layout of content of the data is shown in Figure 1.

A more detailed explanation such as their methodology of collection of can be found at `https://dumps.wikimedia.org/other/pagecounts-raw/`.

```
en Leonard_beaumont 1 13493
en Leonard_cohen 5 311842
en Leonard_nimoy 3 210321
en Leonard_of_Mayfair 1 0
en Leonard_of_Veroli 1 29464
en Leonard_prokoficff 1 6508
en Leonard_susskind 3 75813
en Leonard_swett 1 11282
en Leonard_swidler 1 15493
en Leonard_v._Pepsico,_Inc. 5 85360
```

Figure 1: A snippet of what the data looks like.

```
▼<property>
  ▼<name>
     yarn.nodemanager.disk-health-checker.max-disk-utilization-per-disk-percentage
   </name>
   <value>90.0</value>
   <source>yarn-default.xml</source>
 </property>
```

Figure 2: A yarn configuration that sets the maximum disk utilization.

3 Data Clean-up: Phase One

The first stage of cleaning happened in Hadoop.

3.1 Unzip the jar of raw files

After transporting all of the .gz files, we need to unzip our files. The unzip script is the file named *unzip_ Wiki.c*. However, during the unzipping process, we encountered our first problem.

Up to this point, we have only 1 year's tar date: 2015, in which each file, represented as a pagecount for every hour, takes roughly 80MB. After Unzipping, the amount of data will be generated for this year will be approximately:

$$24 \times 365 \times 450 \approx 3.8TB$$

.

Nevertheless, the capacity of our HDFS is only 7.06 TB, and we exceeded the limit, which led all 16 datanodes become healthy.

The way we solved this problem is to look at the logs and the YARN configurations, and we found out the following setup:
And the way we fixed, along with the help from Dr. Bressoud included the following two steps:

(1) Changed that threshold to 97%. But this is a only a temporal fix, as we start running into many other OS and systems issues and the system can

become really fragile. But at least this allows us run hadoop jobs until we can reduce the disk usage.

(2) Changed the HDFS replication to 1 from 2. So we will consume resources at a slower rate and can, after we do some clean up, increase our capacity.

3.2 Delete non-English Search

Since our later sections of language processing will only recognize the English part, hence, we decided to get rid off the non-English rows by using a MapReduce job.

The project we created is named as *cleandata*. In the first subdirectory of *cleandata*, we added a script to constantly call the hadoop command to run our reduce job. The output of the reducers will be all of the rows whose first columns equal to "en", "En", "En.d", or "en.n".

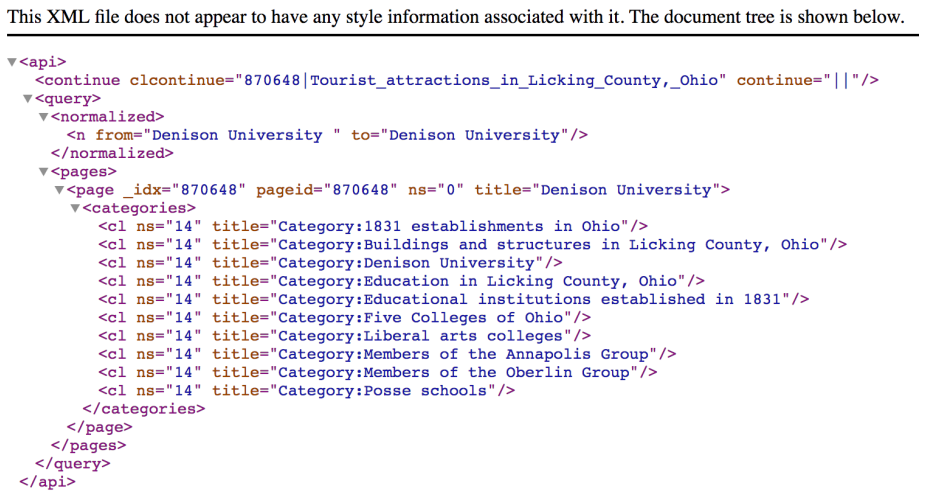

Figure 3: Queried result of the word “Denison University".

4 Job 1

4.1 Motivation

The reason we need job 1 is because we need something more general than a page name. Hence, we want to find the categories associated with each page.

4.2 Approach

We luckily found out that there is a Wikipedia query API website. The way it works is that given a search topic, the API will return a XML format file which contains all the categories such key word fall into. For example, if we want to search the key word “Denison University”, we can put the word “Denison University” after the “title=” from the link `en.wikipedia.org//w/api.php?action=query&format=xml&prop=categories&titles=DenisonUniversity&clshow=!hidden&cllimit=10`.

And the corresponding opened and queried result will look Figure 3:

4.3 Parallel Querying

As you might have noticed, the query still requires the Internet power. Hence, we adopted the technique of using MPI with 219 machines again since each one will have a public IP. Moreover, since each request only takes a small portion of the entire download and upload bandwidth, thus, we decided to incorporate multiple threads within each one of those machines in order to promote the performance. For each computer, we created 8 threads, and we achieved roughly 8 times speed-up for each machine.

In order to achieve this, it demands a careful manage of the data movement and a complex designing to cooperate those hybrid parallel systems. And Figure

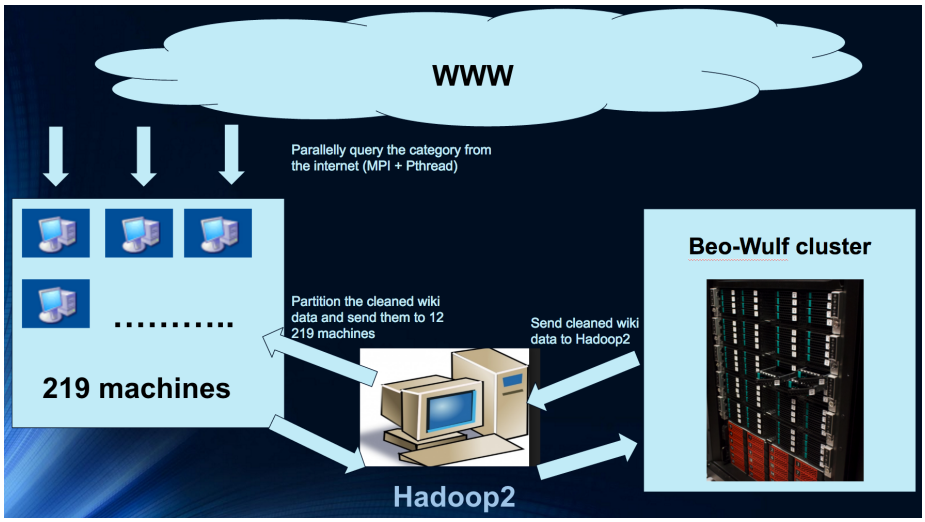

Figure 4: The data movement in Job 1.

4 gives a general picture of how the data movement looks like.

The corresponding codes are *GetCategories.c*, *thread_ wiki.c*, and *query_ wiki.c*. Basically *GetCategories.c* contains the MPI calling section, *thread_ wiki.c* contains the Pthreads section, and *query_ wiki.c* is a helper function to query the data.

5 Data Clean-up: Phase Two

In section, we simply parsed the already queried and downloaded raw xml content into a category list in which each category is separated by a tab.

We created a MapReduce job to do so, and the corresponding project is called *parsecategory*.

6 Job 2

Job 2 is the Perceptron clustering. We had several difficulties with the perceptron. The largest one was to create an appropriate dictionary that resembled the top most relevant words on the seven topics we were analyzing. First, we used the Oxford Learner Dictionary. However, it was very limited and too specific. We found later a webpage called: *relatedwordsto.org*. It retrieve a list of words that were related to another one (input). We made a topic dictionary of 950 words, some of them overlapped. With a Python notebook ($PerceptorDB_Dictionary$) we generated a database with a table to insert all words. We assigned lesser value to the words that appear multiple times - in many categories or topics. The values are specified in the python notebook.

Because ideally this database is in a distributed file system and then a larger dataset can be used to do the analysis, like we first intended, we have a java file that creates a HashMap of this Perceptron Dictionary and serialize it. The idea is that we can via serialized cache send this serialized version to all reducers and then "de-serialize" into a HashMap in every Mapper or Reducer. So the analysis also occurs in parallel. Due to time difficulties of job 1, and the direct dependence between these two jobs, we didn't fully succeed on these aspect. We serialize it, and wrote a jave to run in MapReduce. However, we had some difficulties with the execution of these code.

We then proceed to a lesser but still effective version of this job 2. We use python to open the results of job 1 locally on hadoop2 and we use the perceptron analysis. For each word in the list of categories we query the .db file we created before. This is effectively a dot product of the database and the words that appear in the list of categories. We obtain the "scores" or "weights." Then, we add the resulting tuples getting an overall score tuple for a list of categories (for every line of output from job 1 clean data). The index of the maximum score in the tuple represents the highest category. WE will only write output to a resulting file, if the maximum score is at least 50% the maximum possible score that it could have obtained on that category/topic.

Then output then becomes again similar to the clean data from job 1, minus the list of categories. Instead we now write the overarching topic that those categories might pointed towards.

The result is strongly dependent on the dictionary. A much intense project would have been doing training sets on mapReduce to get a good perceptron

database.

7 Analysis and Result

This section is very partial. During job 1 execution process we estimated that 219 machines with MPI, running using 8 threads each would have taken about 15 days. Just to query wikipedia. We do not recommend run the MPI version since, just the movement of data is extremely expensive. (9 hrs to move data to 219). Due to this huge time difficulty, we killed the MPI application after 1 day of execution and run the worker nodes part on 2 computers in the 219lab. Each was running at $\approx$ 220% CPU. Another reason why we had to stop the MPI was that 271 and 173 students were complaining about the slow performance of their computers. We decided to run it on 219k and 219j, for the months of December and November. Using $\approx$ 220% CPU 24hrs for 5 days, we successfully queried 4GB ($8 \times 500MB$) of categories to wikipedia per computer. Approximately, 8GB overall.

After cleaning the data we obtained 1.3GB of good categorizable data on which we run our analysis.

The analysis shows that most people look up politics and sports related articles. The are based only on the first hours of November 1st and December 1st. With more time we could have analyzed more data; however, time constrains are always the hardest to overcome.