Cluster and Cloud Computing Assignment 1 Report

Chi Zhang 1067750
Qianjun Ding 1080391

March 29, 2022

1 Introduction of Dataset and Pre-processing

1.1 Datasets

Three given Twitter Datasets, 'tinyTwitter.json,' 'smallTwitter.json' and 'bigTwitter.json' contain details of tweets, such as languages, contents and geographic coordinates. Only 'bigTwitter.json' with size 20.8GB is used for submission on Spartan and the other two files are only used for local test.

Grid's dataset 'sydGrid.json' includes the longitudes and latitudes of 16 gridded boxes around Sydney.

External 'language.csv' is downloaded via datahub.io with 186 languages and corresponding abbreviation. CSVs from different websites may contain different languages, which might affect the final results.

2 Data Processing

2.1 Sydney Grid Polygons

It is loaded by using 'json.load()' method as its size is only 9.20 KB which would not affect our memory usage significantly. It is worth noting that the sequence of id number of the grid is out of order. Thus, the parsed dictionary firstly is sorted ascendingly according to the value of ids in an attempt to follow the gridded box example from the specification, i.e., From A1, B1, C1 ... to D4 and save their corresponding polygons into a list. Moreover, to satisfy the cell selection algorithm from the Spec, the previous resultant list is further resorted in groups of 4, i.e., Change from [A1, B1, C1, D1] to [D1, C1, B1, A1], in a left to right, bottom to the top order.

2.2 Twitters

After appropriately reading the JSON file line by line described in the "Algorithm" parts, we filter out each tweet by firstly checking whether its location information is none or not. If not, the coordinate is compared against each polygon according to the order from the previous polygon list. If the point is within or intersects with one polygon, and its tweet language exists in the language.csv, the iteration will be stopped immediately. Its corresponding language index and the polygon grid code will be saved. Otherwise, the tweet information will be discarded. This filtering algorithm fulfills the spec requirements. For example, if a location lines between grid B2 and C2, the location will be directly compared to C2 only, and the iteration will terminate. Also, if the location is between B1/ B2 cells, only B1 will be checked first as it is on the left.

3 Algorithm

3.1 Library

Three main libraries are applied to achieve the primary goal of this project, with the inbuilt 'json' library, which is used for reading in and processing the json data. Two external libraries, 'mpi4py', which is used to allow distributed computing, and "Shapely" is applied, which help generate polygons with Sydney's grid data and generate 'Point' for each Twitter with valid data coordinates.

3.2 Main Algorithm

By first setting the MPI in the code, the distributed computing is applied for proceeding the data with the given number of node(s) and core(s) via the command line in Slurm. Sydney's grid data and a list of language codes are pre-loaded and pre-processed for further usage of collecting information of each region's Twitter information.

By splitting the big data file into pieces with size divided by the number of applied core(s), the index (Pointer) value for each piece can be generated in each core ("processor" or "rank") accompanied by its rank name (e.g., Number from 0-7 with eight cores applied). Each processor then take the piece of data (captured by the start and end pointer value) with its rank name and start processing. To avoid loading all data into the memory before processing, which will cause memory overflow error as excess input is loaded into limited RAM (approximate 20.8GB for the "bigTwitter.json" and limited memory in Spartan with 8GB per core by default), the File I/O with read-only (with open(file, 'r') as f) and readline() method are applied to cap the inflow of memory when processing the data.

With the readline() method in each processor, the allocated piece of JSON data in each core (processor) can extract an object (one line) each time, which ensure the occupied memory when proceeding data is always capped at the size of one JSON object, and pend its validation of geo-coordinates and Twitter content's language type before storing the valid information. The readline() method will cease till the end of the data piece, and all the valid JSON object (Twitter) is stored in a python dictionary with key-value as Sydney's grid name. The try and Exception method is applied during the process to ensure that the invalid decoding of an incomplete JSON object (caused by splitting the data with file size) and the undefined language which cannot be recognised would not break the code.

After all the processors have finished processing, the 'gather' command is applied to retrieve outputs from all cores and presented in the processor with rank == 0. By finalising data in the gathered list from all processors, the result of amount and language of Twitter in each Sydney's grids is presented in the table in Figure 1.

Cell	#Total Tweets	#Number of Languages Used	#Top 10 Languages & #Tweets
A1	21	2	('English-20', 'French-1')
A2	22	2	('English-21', 'Japanese-1')
A3	11	3	('English-9', 'Thai-1', 'Indonesian-1')
A4	85	4	('English-80', 'Turkish-2', 'French-2', 'Spanish-1')
B1	184	5	('English-179', 'Tagalog-2', 'Danish-1', 'Catalan; Valencian-1', 'Haitian-1')
B2	499	11	('English-482', 'Italian-4', 'Japanese-3', 'German-2', 'Tagalog-2', 'Portuguese-1', 'Finnish-1', 'Spanish-1', 'Hindi-1', 'Estonian-1')
B3	636	11	('English-610', 'Japanese-5', 'Indonesian-5', 'Romanian; Moldavian; Moldovan-4', 'Spanish-3', 'Portuguese-3', 'Catalan; Valencian-2', 'Tagalog-1', 'Danish-1', 'Thai-1')
B4	469	10	('English-450', 'Japanese-9', 'Portuguese-3', 'Italian-2', 'Indonesian-1', 'German-1', 'Spanish-1', 'Turkish-1', 'Dutch; Flemish-1', 'French-1')
C1	46	4	('English-42', 'French-2', 'German-1', 'Haitian-1')
C2	94	5	('English-84', 'Indonesian-6', 'Spanish-2', 'Romanian; Moldavian; Moldovan-1', 'French-1')
C3	4467	29	('English-4186', 'Japanese-56', 'Spanish-42', 'Indonesian-32', 'Catalan; Valencian-24', 'Portuguese-15', 'Tagalog-13', 'German-12', 'Romanian; Moldavian; Moldovan-11', 'Italian-9')
C4	962	21	('English-888', 'Haitian-17', 'Japanese-6', 'Indonesian-6', 'Romanian; Moldavian; Moldovan-6', 'Spanish-5', 'Estonian-5', 'Italian-5', 'Tagalog-4', 'Catalan; Valencian-3')
D1	97	6	('English-90', 'Portuguese-2', 'Welsh-2', 'Spanish-1', 'Catalan; Valencian-1', 'Lithuanian-1')
D2	33	2	('English-31', 'Spanish-2')
D3	160	7	('English-154', 'Spanish-1', 'Estonian-1', 'Portuguese-1', 'Thai-1', 'Tagalog-1', 'Catalan; Valencian-1')
D4	3	1	('English-3')

Figure 1: Grid Region's Twitter Result

4 Slurm Command

The jobs will be submitted three times via "1node1core.slurm", "1node8core.slurm" and "2node8core.slurm" on Spartan. From each document, "SBATCH –nodes" and "SBATCH –ntasks" indicate the number of nodes and cores required to run the task. "SBATCH –time=0-12:00:00" clarifies that the job will run for 12 hours as maximum. Moreover, "SBATCH –output" helps save the output in a text file with corresponding names.

All required python modules are loaded in advance by using "module load" command. As 'Shapely' module is not available in the Spartan software list, "module load python/3.7.4" and "pip install Shapely" should be typed in Terminal before running the slurm documents.

Finally, "time srun -n x python3 twitterProcesser.py" is to run our python program by using x cores. The command "time" here is to record the running time this program takes. The Below figure (Figure 2) presents the '1 node 8 cores' parallelisation slurm command as example for reference.

```
$ 1node8core.slurm
1   #!/bin/bash
2   #SBATCH --nodes=1
3   #SBATCH --ntasks=8
4   #SBATCH --time=0-12:00:00
5   #SBATCH --output=1node8core.txt
6
7   # Load required modules
8   module load python/3.7.4
9   module load mpi4py/3.0.2-timed-pingpong
10  module load numpy/1.18.0-python-3.7.4
11
12  time srun  -n 8 python3 twitterProcessor.py
```

Figure 2: 1 Node 8 Cores Slurm Command

5 Performance on Different Data Resource Distributed Computing

With the same results table generated by the three different data resource distributed computing, the consistency of the algorithm's output is guaranteed, and time consumption (performance of varying data resource computing) is then considered. Presented by the table of the exact time spent processing the data (Table 1) and the chart (Figure 3), the '1 node 1 core' processing consumed approximately 7.6 times as '1 node 8 cores' and '2 nodes 8 cores' (with four cores per node). It indicates that parallel computing is successfully applied, significantly decreasing time consumption for reaching outputs by distributing jobs to cores from the same / different node(s).

Trial	1 Node 1 Core	1 Node 8 Cores	2 Nodes 8 Cores
Trial 1	488.715s	64.682s	64.462s
Trial 2	482.302s	62.868s	64.028s

Table 1: Time Consumption

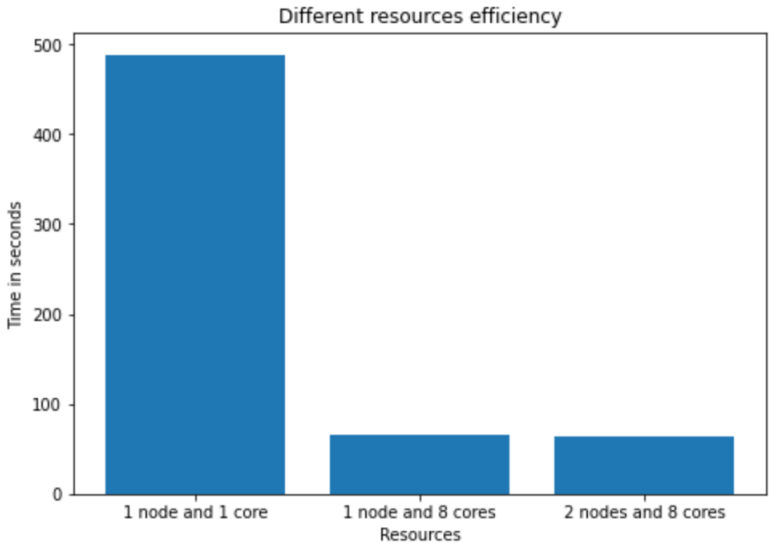

Figure 3: Time Consumption in Trial 1

However, as the total amount of workload is fixed for processing data and the operation for loading in Sydney's grids, language CSV is repeated in each processor (non-parallelisable part) similar to a '1 node 1 core' process, the performance of parallelisation would not be exact eight times as a single core process's performance. Besides, with randomness from the performance difference in each core during the parallelisation, the consumed time within the repeated test of the same data resource or between different data resources with the same amount of cores will differ. Furthermore, as the primary process of this project does not require dependence / interactive jobs, which involves communication between cores within the same node or between separated node(s), the performance difference between '1 node 8 cores' and '2 nodes 8 cores' is negligible.