Cluster and Cloud Computing Assignment1
HPC Twitter GeoProcessing

Ao Li 798657

1. Introduction

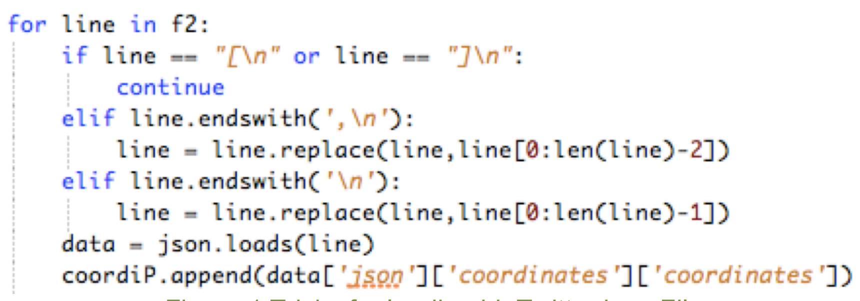
With the number of Twitter users increasing rapidly, it's very difficult for someone to calculate the number of users in different region for getting specific information in some scenario. This report would describe how to use MPI to implement this scenario. The platform tools and data sets are as below:

1) Test Platform and Tools:
- SPARTAN (Unimel HPC facility)
- Python + MPI4py

2) Main data sets:
- bigTwitter.json (Involve the data for tweets coordinates)
- melbGrid.json (Involve the data for grid boxes coordinates)

2. Brief Description of the Methods

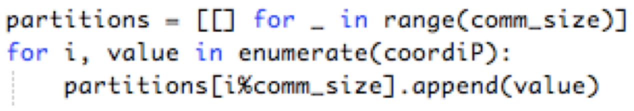
1) Load Grid Box Coordinates: Each process should load the grid box coordinates information from melbGrid.json.
2) Set Master Node: Generally, setting the process whose rank is 0 as master process. It needs to do more jobs than other processes. (Note: rank is like an ID for each process in MPI)
3) Load Tweets Coordinates: Only master process needs to load the tweets information from bigTwitter.json. But there is a little tricky here, it shouldn't load the whole file at once into memory on a master process, which will make the memory exceed, instead to load line-by-line, extract and store only the coordinate information for each tweet. These part of codes are shown in Figure.1.



```
for line in f2:
    if line == "[\n" or line == "]\n":
        continue
    elif line.endswith(',\n'):
        line = line.replace(line,line[0:len(line)-2])
    elif line.endswith('\n'):
        line = line.replace(line,line[0:len(line)-1])
    data = json.loads(line)
    coordiP.append(data['json']['coordinates']['coordinates'])
```

Figure.1 Tricky for loading bigTwitter.json File

4) Scatter Tweets Coordinates evenly to Each Node: Master process partition and scatter tweets coordinates to all the processes. For the sake of improving the parallel processing performance, use the method in Figure.2 for even partition.

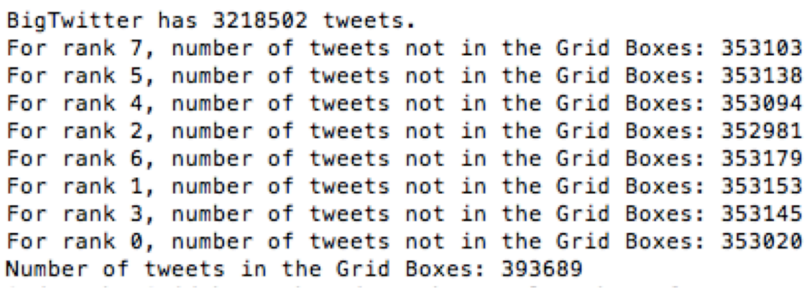


```
partitions = [[] for _ in range(comm_size)]
for i, value in enumerate(coordiP):
    partitions[i%comm_size].append(value)
```

Figure.2 Tricky for Partition

Set a list of list, the number of list is the number of process, and each tweets coordinates will give one list row by row. Then scatter each list to each process.

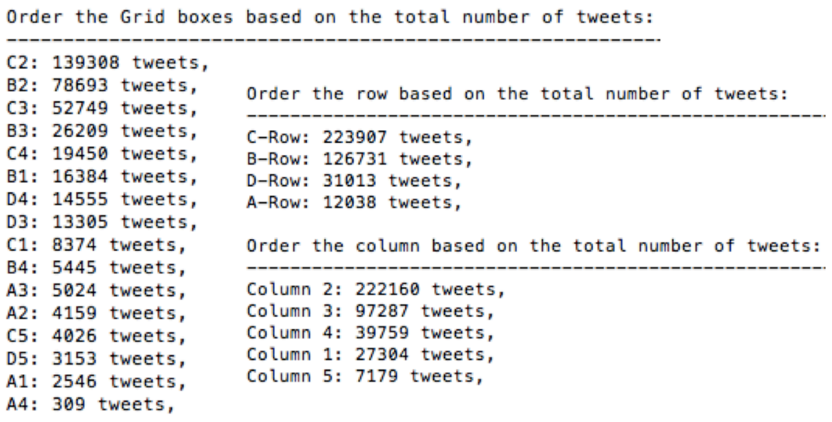
5) Calculate the Number of Tweets in Each Grid Box by Different Process: Each process calculates locally with the tweets sent by the master process. There is a problem here is whether there are any tweets on the boundaries between two grid boxes. Since there are many boundaries between the grid boxes, it's tedious to use some logic control the boundaries. So, use some methods to print out the total number of tweets, the number of tweets not in the grid boxes and using simple logic ($\leq$ and $\geq$) calculating the number of tweets in the grid boxes. It's easy to find find that there are no tweets on the boundaries through Figure.3 (Accumulated all numbers equal to total).



```
BigTwitter has 3218502 tweets.
For rank 7, number of tweets not in the Grid Boxes: 353103
For rank 5, number of tweets not in the Grid Boxes: 353138
For rank 4, number of tweets not in the Grid Boxes: 353094
For rank 2, number of tweets not in the Grid Boxes: 352981
For rank 6, number of tweets not in the Grid Boxes: 353179
For rank 1, number of tweets not in the Grid Boxes: 353153
For rank 3, number of tweets not in the Grid Boxes: 353145
For rank 0, number of tweets not in the Grid Boxes: 353020
Number of tweets in the Grid Boxes: 393689
```

Figure.3 No Tweets on the Grid Boxes Boundaries

6) Gathering Each Grid Box and Order: Master process gather the calculate results from all processes, then use some basic accumulate methods to order the grid boxes The results for running bigTwiiter are depicted as Figure.4 and give an example for the slurm script for 2 nodes 8 cores in Figrue.5.

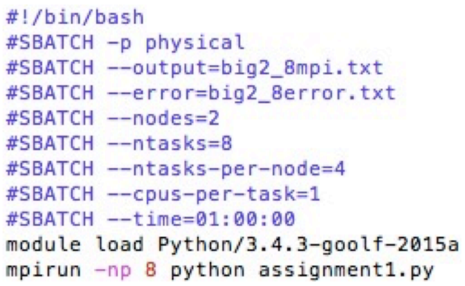


```
Order the Grid boxes based on the total number of tweets:
----------------------------------------------------------
C2: 139308 tweets,
B2: 78693 tweets,
C3: 52749 tweets,
B3: 26209 tweets,
C4: 19450 tweets,
B1: 16384 tweets,
D4: 14555 tweets,
D3: 13305 tweets,
C1: 8374 tweets,
B4: 5445 tweets,
A3: 5024 tweets,
A2: 4159 tweets,
C5: 4026 tweets,
D5: 3153 tweets,
A1: 2546 tweets,
A4: 309 tweets,
```

```
Order the row based on the total number of tweets:
----------------------------------------------------
C-Row: 223907 tweets,
B-Row: 126731 tweets,
D-Row: 31013 tweets,
A-Row: 12038 tweets,

Order the column based on the total number of tweets:
----------------------------------------------------
Column 2: 222160 tweets,
Column 3: 97287 tweets,
Column 4: 39759 tweets,
Column 1: 27304 tweets,
Column 5: 7179 tweets,
```

Figure.4 Order the Grid Boxes for BigTwitter



```
#!/bin/bash
#SBATCH -p physical
#SBATCH --output=big2_8mpi.txt
#SBATCH --error=big2_8error.txt
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=1
#SBATCH --time=01:00:00
module load Python/3.4.3-goolf-2015a
mpirun -np 8 python assignment1.py
```

Figure.5 Slurm Script Example 2 Nodes 8 Cores

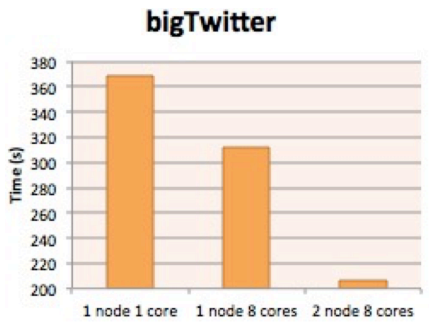
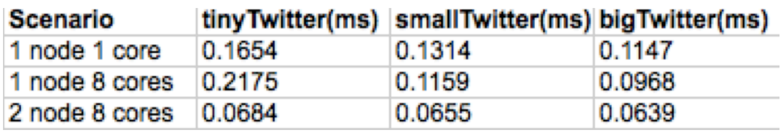


Figure.6 Running Time(s) for BigTwitter

3. Evaluation and Conclusion

After implementation, to all the files' running time, trend is like：

- Running time for file: 2 nodes 8 cores ≤ 1 node 8 cores ≤ 1 node 1 core. Shown in Figure.6.
- Unit of running time for each tweet: big ≤ smallTwitter ≤ tinyTwitter. Shown in Figure.7.



Scenario	tinyTwitter(ms)	smallTwitter(ms)	bigTwitter(ms)
1 node 1 core	0.1654	0.1314	0.1147
1 node 8 cores	0.2175	0.1159	0.0968
2 node 8 cores	0.0684	0.0655	0.0639

Figure.7 Unit of Running Time (ms) for Calculating Each Tweet

We can find that parallel programming can speed up a lot for executing calculation (1 node 8 cores better than 1 node 1 core), all of CPUs in a node evenly receive the data so that they can make fully use of the resources. Besides, I think there are many reasons for multiple nodes is better than 1 node, For example, the cost for read and write to a single memory in a node must be pretty high. Also find the unit time for bigTwitter is the smallest, it confirms a fact that parallel programming is more efficient than just cut down the files into small pieces and run in serial.