Cluster and Cloud Computing Assignment1-HPC Twitter GeoProcessing

Can Cui 765821
cuic3@student.unimelb.edu.au

1. User Guide

1. Upload Twitter_GeoProcessing.java, 1-node-1-core.sh, 1-node-8-core.sh, 2-node-8core.sh to Spartan
2. Use instruction “sbatch ***.sh Twitter_GeoProcessing.java” to submit the tasks to Spartan

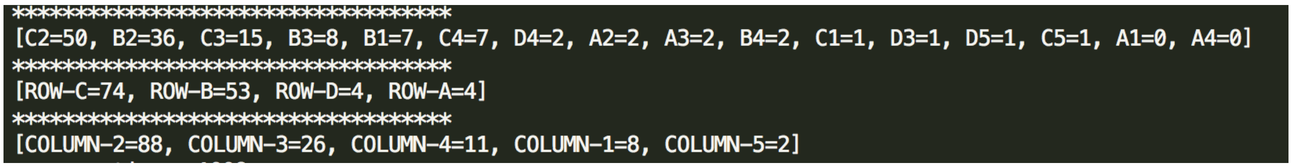
About ***.sh files, there are two mpjrun.sh instructions in each file. The reason is that if I don’t add -jar .:$MPJ_HOME/lib/starter.jar, it throws a ClassNotFoundException, but if I add -jar .:$MPJ_HOME/lib/starter.jar, it shows that starter.jar cannot be found in that path, what weird is I do find starter.jar there. I finally found a way to solve it, which is using two mpjrun instructions, although the first mpjrun throws an exception. I don’t know if this is a bug, my program works well on eclipse.

2. Implementation

I have tried two different ways to parallelise my code:
At first, I assigned the same task to each processor: read a line of tweet and then process it, store the result in an array, after finishing the process job, read the next line. After finishing all the job, slaves send results to the master process (whose rank equals 0), and the master process puts the results together. To do this, I add an accumulator to each process, processor ***i*** only processes line ***n*** which satisfies ***n%7 = i.*** However, this method needs all processor reading the whole file respectively, although they can skip some process job. It is not an efficient approach.
Therefore, I tried another way to assign tasks: this time, the master process does all the reading file job, slaves do the process job. The master read in a line of tweet and send it to one of slaves, after finishing reading the file, the master tells slaves to send the results back to it and put them together. Compared with the previous method, this method is faster and more efficient.

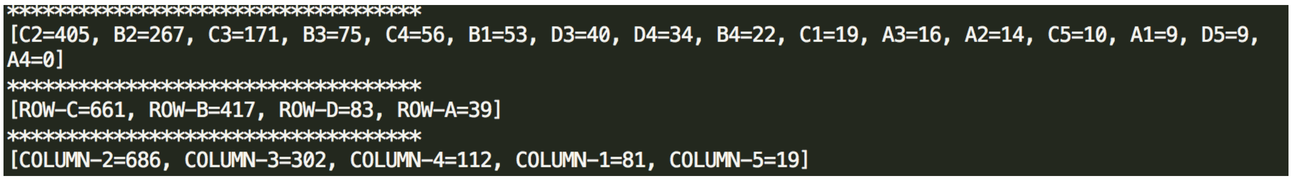
3. Results

The number of tweets shows as follows:



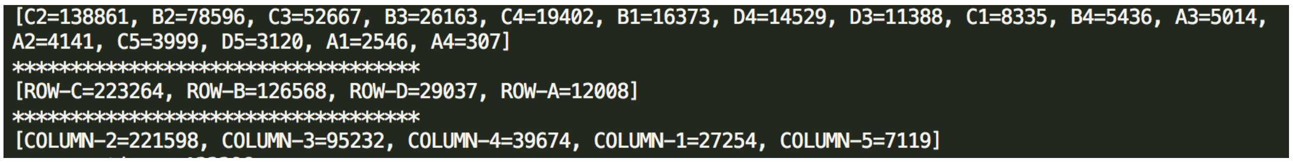
```
***********************************
[C2=50, B2=36, C3=15, B3=8, B1=7, C4=7, D4=2, A2=2, A3=2, B4=2, C1=1, D3=1, D5=1, C5=1, A1=0, A4=0]
***********************************
[ROW-C=74, ROW-B=53, ROW-D=4, ROW-A=4]
***********************************
[COLUMN-2=88, COLUMN-3=26, COLUMN-4=11, COLUMN-1=8, COLUMN-5=2]
```

Figure 1. The result of tinyTwitter



```
***********************************
[C2=405, B2=267, C3=171, B3=75, C4=56, B1=53, D3=40, D4=34, B4=22, C1=19, A3=16, A2=14, C5=10, A1=9, D5=9,
A4=0]
***********************************
[ROW-C=661, ROW-B=417, ROW-D=83, ROW-A=39]
***********************************
[COLUMN-2=686, COLUMN-3=302, COLUMN-4=112, COLUMN-1=81, COLUMN-5=19]
```

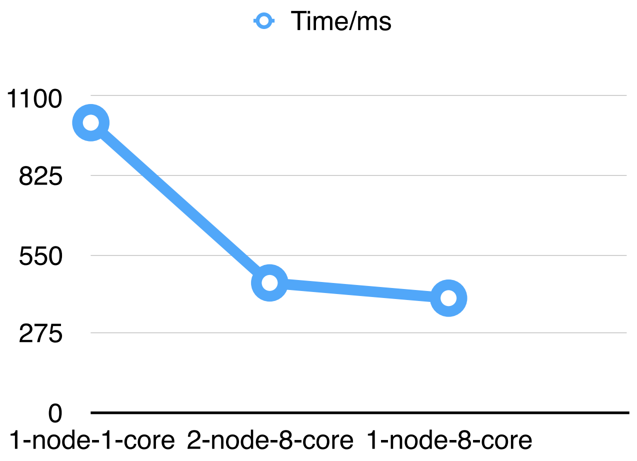
Figure2. The result of smallTwitter



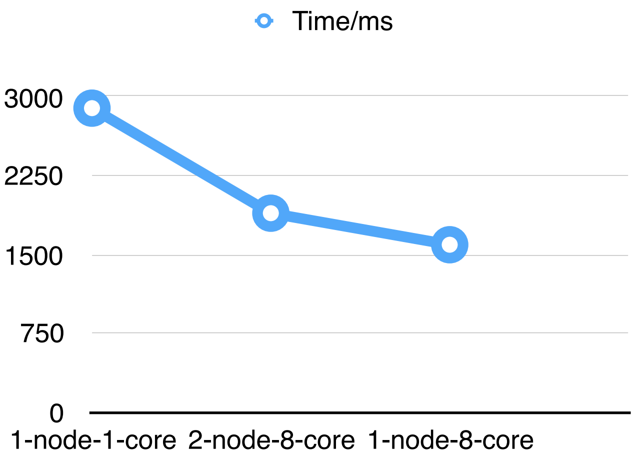
```
[C2=138861, B2=78596, C3=52667, B3=26163, C4=19402, B1=16373, D4=14529, D3=11388, C1=8335, B4=5436, A3=5014,
A2=4141, C5=3999, D5=3120, A1=2546, A4=307]
***********************************
[ROW-C=223264, ROW-B=126568, ROW-D=29037, ROW-A=12008]
***********************************
[COLUMN-2=221598, COLUMN-3=95232, COLUMN-4=39674, COLUMN-1=27254, COLUMN-5=7119]
```

Figure3. The result of bigTwitter

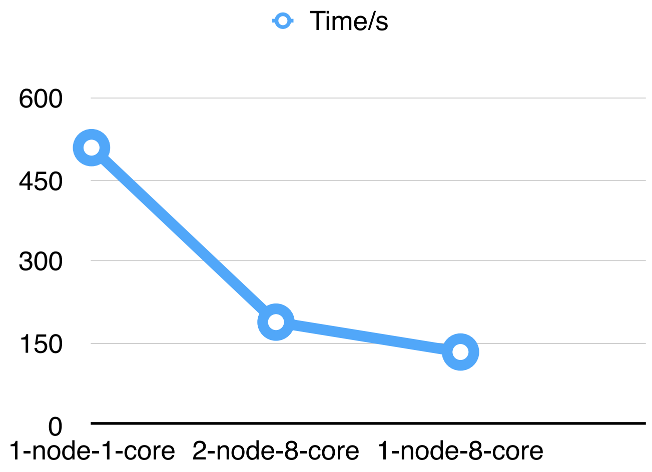
The process time of each twitter file in different mode:



Graph1. The process time of tinyTwitter



Graph1. The process time of smallTwitter



Graph1. The process time of bigTwitter

As can be seen from above graphs, compared with 1-node-1-core mode, multicore implementation increases the processing speed dramatically, about 2 times faster for tinyTwitter and smallTwitter files and 3 times faster for bigTwitter file. It is interesting to note that 2-node-8-core is a little bit slower than 1-node-8-core, I guess the reason might be that 2-node requires some extra time to transmit data between two node, by contrast, 1-node-1-core only transmits data between processors.