Cluster and Cloud Computing Assignment 1
HPC Twitter Processing

Yuming Lin 883717 **Yunxin Chen 1065812**

1 Introduction

In HPC Twitter Processing problem, the most ten popular hashtags and languages will be extracted from bigTwitter.json by using the High Performance Computing (HPC).

2 Related Work

2.1 Spartan

Spartan is a HPC system at the University of Melbourne. The key indicators are as follows (Lev, Greg, Linh & Bernard,2016).

Partition	Nodes	Cores	Memory	Peak Performance (DP TFlops)
Cloud	165	12	100GB	63.5
Physical	9	12	254GB	2.28
	5	32	508GB	2.15
	12	72	1540GB	83
	Processor			
cloud	Intel(R) Xeon(R) Gold 6138 CPU 2.00GHz			
Physical	Intel(R) Xeon(R) CPU E5-2643 v3 3.40GHz			
	Intel(R) Xeon(R) CPU E5-2683 v4 2.10GHz			
	Intel(R) Xeon(R) Gold 6154 CPU 3.00GHz			

Table 1: Key indicators

Physical and cloud are two different execution environment of spartan. HPC Twitter Processing problem will be executed in both of them.

2.2 MPI Package: mpi4py

The mpi4py package is base on the principle of MPI, which is a cross-language communication protocol. In HPC Twitter Processing problem, it is expected to gather the data from the other processors to one specific processor.

2.3 Slurm Scripts

Slurm is an independent Linux cluster job schedulingsystem. In this project, nodes, cores and tasks are assigned in slurm scripts. Commands and parameters are shown in Table 2. Besides, Python module is loaded with command: $module load Python/3.7.3-spartan\_gcc-8.1.0$. Finally, slurm jobs are submitted by bash command $sbatch\ xx.slurm$, and queried status by $squeue\ --job[id]$.

3 Implementation

The architecture of this project is shown in Figure 1. The raw data is split into multiple chunks with same size. Each processor, which from rank 0 to n, handles the task in parallel. Then, the results from each processor gather at the rank 0. The processor with rank 0 summary those results serially.

Command	Parameter
–partition	cloud/physical
–node	1/2
–ntasks-per-node	1/8/4
–cpus-per-task	1
–time	1-0:0:00
–output	FileName.output
–mail-user	Mail@address
–mail-type	BEGIN/END/FAIL

Table 2: Explanation of slurm commands

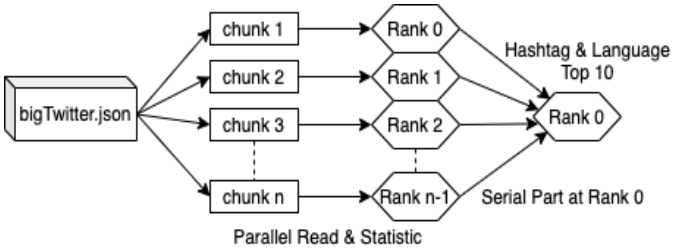

Figure 1: Flow chart of HPC Twitter Processing

3.1 Message Passing Interface

3.1.1 MPI initialization

It is the first step of a MPI procedure.

$$comm = MPI.COMM\_WORLD \quad (1)$$

$$rank = comm.Get\_rank() \quad (2)$$

$$size = comm.Get\_size() \quad (3)$$

The communicator, comm in (1), defines the collection of which processes may communicate with each other. Rank in (2) identify the process within the communicator. Size in (3) represents the number of processes contained within that communicator. This is a point-to-point communication which all communication takes place within a communicator.

3.1.2 MPI Communication

$$comm.gather(myHashtag, root = 0) \quad (4)$$

$$comm.gather(myLanguage, root = 0) \quad (5)$$

A processor with a rank of 0 collects the hashtag statistical results in (4) and the language statistical results in (5) from the other processors.

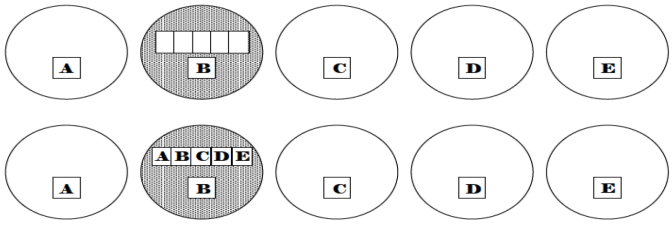

Figure 2: MPI gather, before & after

Processor B in Figure 2 ("ARCHER » Message-Passing Programming with MPI", 2014) plays the role with a rank of 0 in the process of MPI communication.

3.2 Parallel Part: Reading & Statistics

3.2.1 Pseudocode

algorithm 1 Parallel read & count the JSON file

Input: JSON file $file$, size of the file $size$, number of processors n

Output: the statistic of hashtags on each processor $statL\ 0\ to\ n$, the statistic of languages on each processor $statH\ 0\ to\ n$

```
1:  function ParallelRC(file, size, n)
2:      avgSize ← size/n
3:      offsetList ← list of length n
4:      for i = 0 → n − 1 do
5:          offsetList[i] ← avgSize·i
6:      end for
7:      for i = 0 → n − 1 do in parallel
8:          processor i does
9:              statH i ← accumulateHashtags(file
10:             offsetList[i], file offsetList[i + 1])
11:             statL i ← accumulateLanguages(file
12:             offsetList[i], file offsetList[i + 1])
13:     end for
14:     for i = 0 → n − 1 do in parallel
15:         processor i does
16:             statH i ← collections.Counter(statH
17:             i)
18:             statL i ← collections.Counter(statL
19:             i)
20:     end for
21:     return statH 0 to n, statL 0 to n
22: end function
```

3.2.2 Handle the Non-standard Format

Some lines do not conform to the JSON standard; for example, they end with illegal symbols. At this time, the processor will discard the illegal characters, convert them into a standard format line and load them.

3.2.3 Handle text & retweeting & quoting

Considering that retweeting and quoting are both important ways for users to express, even in some cases, users will only consider using retweeting and/or quoting instead of the text. So not only the text is counted in 3.2, but also retweeting and quoting are counted. Only in this way can we collect hashtags and languages more accurately.

3.2.4 Handle Non-initial Position Problem

Since offset is an average value, after using the $seek()$ function, each processor will have a high probability of falling to the middle position of each row, instead of being lucky to fall to the initial position of each row. To solve this problem, each processor will determine whether the current line is in a standard JSON format. Obviously, those processors that fall in the middle of each line will not get a standard format line, so they must discard the line and read the next line. Those processors that are lucky enough to fall into the initial position of each line can start reading from the beginning of the line. Finally, all processors will read the last position of the bytes in the file stream marked by the $tell()$ function. Then, they stop.

3.3 Serial Part: summary results

3.3.1 Pseudocode

algorithm 2 Summary other processors' results

Input: a list of statistical hashtag results from other processors $listH$, list of statistical language results from other processors, $listL$

Output: the most 10 popular hashtags $top10H$, the most 10 popular languages $top10L$

```
1:  function SummaryResults(listH, listL)
2:      hashtag ← listH[0]
3:      language ← listL[0]
4:      for result in listH do
5:          hashtag ← mergeCounter(hashtag
6:          , result)
7:      end for
8:      for result in listL do
9:          language ← mergeCounter(language
10:         , result)
11:     end for
12:     sortedH ← sort(hashtag)
13:     10stHashtag ← sortedH[9]
14:     duplicatedH ← 0
15:     for i = 10 → n − 1 do
16:         if 10stHashtag == sortedH[i] then
17:             duplicatedH ← duplicatedH + 1
18:         else
19:             break loop
20:         end if
21:     end for
22:     sortedL ← sort(language)
23:     10stLanguage ← sortedL[9]
24:     duplicatedL ← 0
25:     for i = 10 → n − 1 do
26:         if 10stLanguage == sortedL[i] then
27:             duplicatedl ← duplicatedL + 1
28:         else
29:             break loop
30:         end if
31:     end for
32:     top10H ← sortedH[0: 10+duplicatedH]
33:     top10L ← sortedL[0: 10+duplicatedL]
34:     return top10H, top10L
35: end function
```

3.3.2 Handle Special Cases

If there are other hashtags with the same statistics as the tenth most popular hashtag, these hashtags are still output until a statistic is encountered that is less than the tenth most popular hashtag. After that, execute the same operations to languages.

4 Output of Top 10

Hashtag Top10	Language Top10
#auspol,37307	English(en),5025807
#coronavirus,18870	Undefined(und),365068
#มาพ่องเพิ่งอะไร,15060	Thai(th),265244
#firefightaustralia,13123	Portuguese(pt),177772
#oldme,12777	Spanish(es),122250
#grammys,10136	Japanese(ja),75327
#assange,9531	French(fr),70308
#sportsrorts,9475	Tagalog(tl),67867
#scottyfrommarketing,8815	Indonesian(in),58981
#เป๊กผลิตโชค,8671	Korean(ko),34081

Figure 3: Output of HPC Twitter Processing

5 Output of Performance Parameters

5.1 Physical Execution Environment

HW	Total	Serial	Parallel	Cmpi	Impi
1n1c	460.873	0.045	459.099	0.084	3.8×10^{-5}
1n8c	62.138	0.161	59.377	2.567	0.2×10^{-5}
2n8c	62.132	0.173	59.32	3.297	3.7×10^{-5}

Total = Total program running time
Serial = Serial part running time
Parallel = Parallel part running time
Cmpi = MPI communicate time
Impi = MPI initial time

Table 3: Time(sec) of different performance measures, running by different hardwares on physical environment

5.2 Cloud Execution Environment

HW	Total	Serial	Parallel	Cmpi	Impi
1n1c	655.213	0.067	649.022	0.165	1.7×10^{-5}
1n8c	101.813	0.187	96.912	3.365	0.8×10^{-5}
2n8c	95.639	0.213	89.102	3.105	1.2×10^{-5}

Total = Total program running time
Serial = Serial part running time
Parallel = Parallel part running time
Cmpi = MPI communicate time
Impi = MPI initial time

Table 4: Time(sec) of different performance measures, running by different hardwares on cloud environment

6 Performace Analysis

6.1 Maximum Speedup

According to Amdahl 's Law (Amdahl, 1967),

$$Speedup = \frac{ts}{f \cdot ts + \frac{(1-f) \cdot ts}{p}} = \frac{p}{1 + (p-1) \cdot f}$$

Where ts = sequential running time, p = the number of processors, f = fraction of the time that cannot be paralized.

For a large p, maximum speed up = 1 / f. Substituting measurements from Table 3, Table 4, we get:

	Physical	Cloud
1n8c	386	555
2n8c	360	450

Table 5: Maximum speed up for a large p

It can be seen that, because 1n8cCloud has the lowest serial part running time, its maximum speedup is also the largest for a large p.

6.2 Theoretical Speedup

Eight processors participate in parallel to solve the HPC Twitter Processing problem, substituting p = 8 and related data from Table 3 and Table 4 into Amdahl 's Law formula in 6.1.

$$Speedup = \frac{ts}{f \cdot ts + \frac{(1-f) \cdot ts}{p}} = \frac{8}{1 + (8-1) \cdot f}$$

We get:

	Physical	Cloud
1n8c	7.86	7.90
2n8c	7.85	7.88

Table 6: Theoretical Speedup

6.3 Actual Speedup

$$Actual\ Speedup = \frac{actual\ sequential\ time}{actual\ parallel\ time}$$

Substituting related data from Table 3 and Table 4, we get:

	Physical	Cloud
1n8c	7.42	6.44
2n8c	7.42	6.85

Table 7: Actual Speedup

Therefore, it can be inferred from the results that no matter it is 1n8c, 2n8c, or physical, cloud, the actual speedup is less than the speedup. However, they are very close in the physical execution environment.

6.4 Speedup Error Analysis

6.4.1 Delay on Nodes: Context Switch Cost

According to Figure 4, the total time of 1n8c-PH is 1.60% faster than that of 2n8c-PH, and the total time of 1n8c-CL is 6.46% faster than that of 2n8c-CL. This shows that the cost of switching context between nodes can affect speedup.

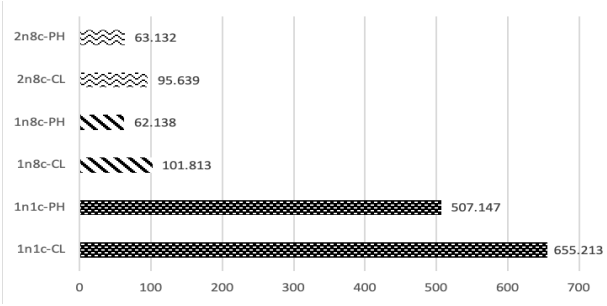

Figure 4: Total time(sec)

6.4.2 Delay on Physical & Cloud Environments

In Table 7, the speedup will have a significant change only by changing the execution environment. The speedups of 1n8c and 2n8c in physical execution environment are increased by 15.22% and 8.32% than in the cloud partition. Therefore, the execution environment has a dramatic impact on the speedup.

6.4.3 Fraction of Parallel & Serial

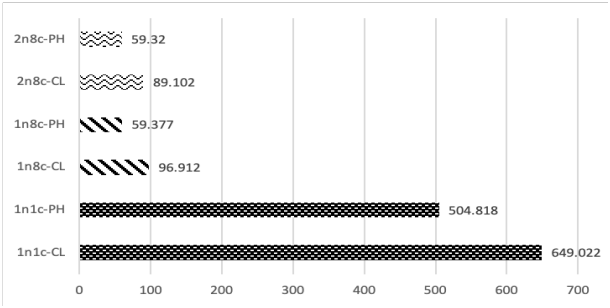

Figure 5: Parallel execution time(sec)

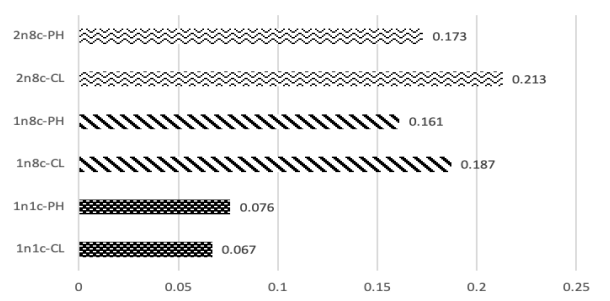

Figure 6: Serial execution time(sec)

The fraction of parallel & serial in the four groups of models are shown below:

1n8c-PH	2n8c-PH	1n8c-CL	2n8c-CL
369 : 1	343 : 1	518 : 1	418 : 1

Table 8: Parallel execution time : Serial execution time

The proportion of the execution time of the serial part is almost negligible, so it hardly affects the speedup.

6.4.4 MPI initialization delay

Measurements in Table 3 and Table 4 show that the time to initialize MPI has almost no effect on speedup.

6.4.5 MPI communication delay

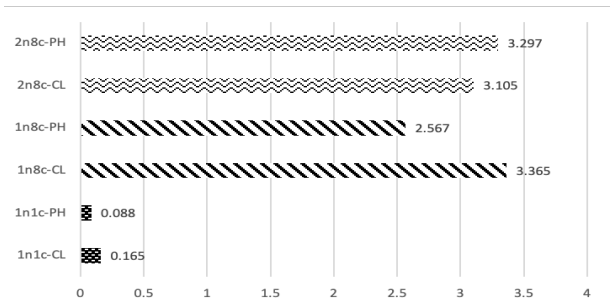

Figure 7: MPI communication time(sec)

The communication time as a percentage of the total execution time is as follows:

1n8c-PH	2n8c-PH	1n8c-CL	2n8c-CL
4.17%	5.26%	3.33%	3.23%

Table 9: MPI communication as a percentage of the total

Therefore, the impact of MPI communication time on speedup cannot be ignored.

7 Critical Reflection

HPC Twitter Processing problem is a data-parallel problem divides the input data into several entirely independent parts. The same computation is undertaken on each part. Assuming that the size of each completely independent part is the same, then the load divided by each processor are all equal, which means that as long as it is a problem that meets the above characteristics, its speedup will have a linear positive correlation relationship with the number of processors. In other words, as the processor increases, the speedup will increase until the maximum speedup is reached.

8 Conclusion

In the HPC Twitter Processing problem, the error of the speedup mainly comes from execution environments and MPI communication.

The speedup of the data-parallel problem is positively correlated with the number of processors, but the speedup will not exceed 1 / f, where f is the fraction of the time that cannot be parallelized.

The HPC Twitter Processing problem is a data-parallel problem.

References

Amdahl, G. M. (1967, April). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference* (pp. 483-485).

ARCHER » *Message-Passing Programming with MPI.(2014)* from http://archer.ac.uk/training/course-material/2014/07/MPI_Edi/

Lev Lafayette, Greg Sauter, Linh Vu, Bernard Meade, *"Spartan Performance and Flexibility: An HPC-Cloud Chimera"*, OpenStack Summit, Barcelona, October 27, 2016. doi.org/10.4225/49/58ead90dceaaa