HPC Twitter GeoProcessing Report

Zichun Zhu, Xiaoyue Ma

1 Introduction

2 Methodology

2.1 HPC Parallelism

In general there are two approaches to parallelize a program: divide work by data, and divide work by steps. The chosen is highly depend on the specification of work. In geo processing, the number of steps per data entity is relatively small, while the size of whole data could be very large. Thus dividing work by data, and assigning working on a piece of data to different processes is more tractable.

2.2 MPI

There are different ways to classifiy parallel computers. The widely used classification, Flynn's taxonomy distinguishes computer architecture along instruction stream and data stream. According its definition, most current High Performance Computer(HPC) is in Multiple Instruction, Multiple Data(MIMD) type. Every process can work concurrently on different data stream, but those data and processors could be physically located in different place. To share the data between the processors, we applied MPI in the implementation.

Message Passing Interface(MPI) is an useful open source tool in multi-core programming which contains many functions to share message in processes. The main difference between OpenMP and MPI is that the processes using MPI does not shared memory between each other, while OpenMP is a shared memory multiprocessing programming library. MPI therefore able to be used in wide application regardless the physical architecture of platform.

Considering that the size of data could be up to 15 GB, even though divided into chunks it is still a heavy load message transmission. So rather than reading the whole data and then sending out pieces of raw data, the master process scans and records the line indexes, and then scatters the indexes instead. A simple calculation can imply the efficiency and sufficiency. In $smallTwitter.json$, there are about 7000 lines, each is one tweet, and the size of the file is about 25 MB. If we assume tweets density is even, a 15 GB file may contains:

$$tweet\_num = 25MB/15GB * 7000 \approx 4.2million$$

If those tweets are represented in indexes, and each takes size of int, then 15 GB tweets file can be abstracted to:

$$indexes\_size = 4.2million * 24bytes \approx 100MB$$

The load of transmission could be largely reduced by representing tweets in form of indexes.

Speedup(s)	Real	User	Sys
1 Node 1 Task	342.685	328.387	12.025
1 Node 8 Task	122.154	961.023	13.059
2 Node 8 Task	140.352	376.636	181.978

Table 1: Runtime

Runtime(s)	Speed up
1 Node 1 Task	1
1 Node 8 Task	2.81
2 Node 8 Task	2.44

Table 2: Speedup

3 Result

Appendices

A Slurm Script

A.1 1 Node 1 Task

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
```

```
#SBATCH --output="1node1task.out"

module load Python/3.5.2-goolf-2015a

time mpiexec python3 __main__.py
```

A.2 1 Node 8 Task

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --output="1node8task.out"

module load Python/3.5.2-goolf-2015a

time mpiexec python3 __main__.py
```

A.3 2 Node 8 Task

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --output="2node8task.out"

module load Python/3.5.2-goolf-2015a

time mpiexec python3 __main__.py
```